Ocean Exploration Video Challenge

Team Nira, IIITDM Kancheepuram

Introduction

Our program is designed to participate in the Ocean Exploration Video Challenge, organized by NOAA Ocean Exploration. The challenge aims to reduce the time required for a human to annotate ROV dive video and associated quality assurance and quality control by harnessing the power of machine learning/artificial intelligence to assist in video annotation.

For this purpose, we have utilized YOLOv7, a single-stage real-time object detector that is known for being the fastest and most accurate real-time object detection model for computer vision tasks. YOLOv7 is one of the latest iteration in the life cycle of YOLO models and infers faster and with greater accuracy than its previous versions. By using YOLOv7, our program is able to classify and place bounding boxes around organisms in the provided NOAA ship Okeanos Explorer remotely operated vehicle video clips, thus reducing the time required for manual annotation.

Object detection plays a crucial role in understanding and analyzing the marine environment. The task of annotating video clips, classifying organisms, and placing bounding boxes around them is time-consuming and labor-intensive when done manually. Therefore, we have employed YOLOv7 to automate this process and significantly reduce the time required for human annotation.

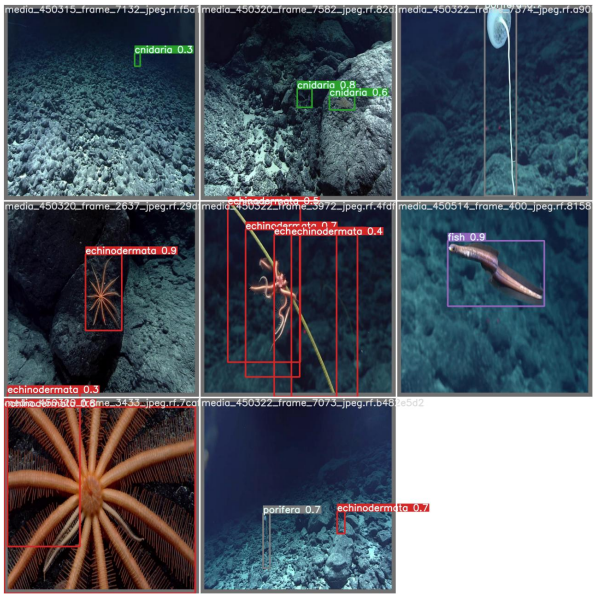


Figure 1: A Result Image of Our Program

Through this explanation, we will delve into the details of our program, including data preprocessing techniques, training procedures, and the inference process. We will provide a user guide to ensure seamless utilization of our program, enabling accurate and efficient annotation of the ROV video clips.

We hope that our program will contribute to the efforts of NOAA Ocean Exploration in exploring the unknown ocean and unlocking its potential through scientific discovery, technological advancements, partnerships, and data delivery.

Program Overview

Our program utilizes YOLOv7, a state-of-the-art deep learning model, for object detection in the Ocean Exploration Video Challenge. YOLOv7, an evolution of the YOLO (You Only Look Once) family of models, is known for its efficiency and accuracy in real-time object detection tasks. We used the official YOLOv7 repository, which is available on GitHub, to train and test our model with our custom datasets obtained from Fathomnet, Seatube v3, and Roboflow.

The architecture of YOLOv7 consists of a backbone network, usually based on a convolutional neural network (CNN), which extracts high-level features from the input images or frames. These features are then fed into detection layers that generate bounding box predictions and class probabilities.

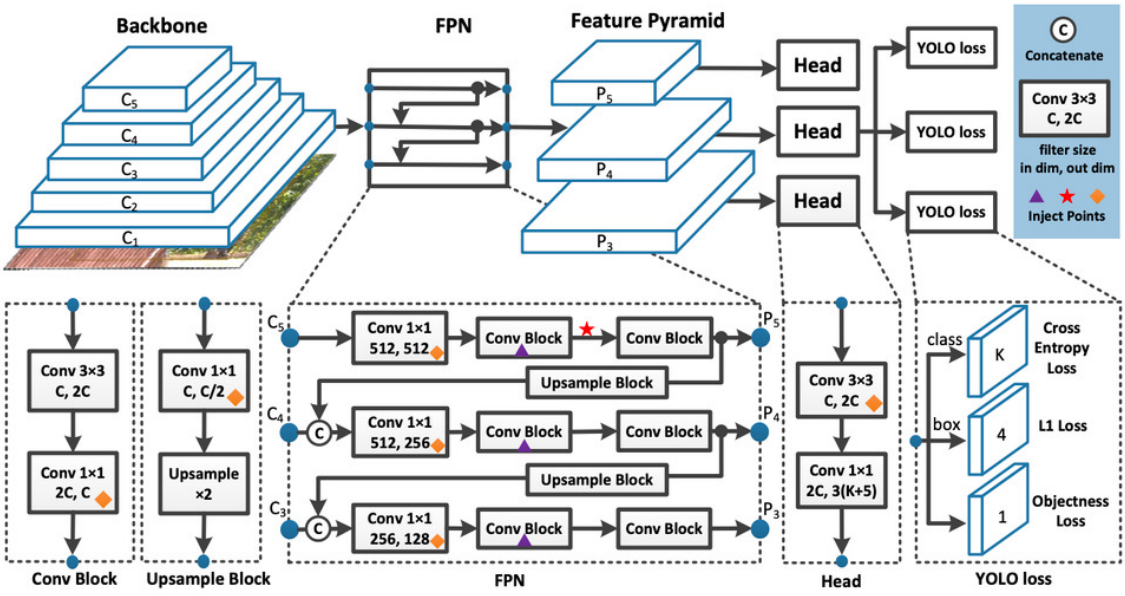


Figure 2: Flowchart of YOLOv7 Architecture

During training, the model will output the memory reserved for training, the number of images examined, total number of predicted labels, precision, recall, and mAP @.5 at the end of each epoch. We used this information to help identify when the model is ready to complete training and understand the efficacy of the model on the validation set.

YOLO works to perform object detection in a single stage by first separating the image into N grids. Each of these grids is of equal size SxS. Each of these regions is used to detect and localize any objects they may contain. For each grid, bounding box coordinates, B, for the potential object(s) are predicted with an object label and a probability score for the predicted object's presence.

This leads to a significant overlap of predicted objects from the cumulative predictions of the grids. To handle this redundancy and reduce the predicted objects down to those of interest, YOLO uses Non-Maximal Suppression to suppress all the bounding boxes with comparatively lower probability scores.

To achieve this, YOLO first compares the probability scores associated with each decision, and takes the largest score. Following this, it removes the bounding boxes with the largest Intersection over Union with the chosen high probability bounding box. This step is then repeated until only the desired final bounding boxes remain.

In the following sections, we will delve deeper into the specifics of our data preprocessing methods, the training procedures employed to train YOLOv7, and the inference and post-processing steps used to generate the annotations for the provided video clips.

Data Preprocessing

In order to prepare the data for training and testing the YOLOv7 model in the Ocean Exploration Video Challenge, we performed several data preprocessing steps. These steps ensured that the datasets obtained from Fathomnet, Seatube v3 and Roboflow were properly formatted and suitable for training the object detection model. The following is an overview of the data preprocessing steps we implemented:

Dataset Collection

We obtained datasets from Fathomnet, Seatube v3 and Roboflow.

These datasets included annotated images and their corresponding annotation files containing bounding box coordinates and class labels.

Missing Annotation Files

We checked the collected datasets to identify any missing annotation files.

If any files were missing, we omitted the corresponding images from the dataset to ensure consistency between the images and annotations.

Class Index Ranging

We ensured that the class indices used in the annotation files were correctly ranging from 0 to (num_classes - 1).

This step was important to maintain consistency between the annotations and the model's output during training and inference.

Annotation File Verification

We carefully verified the format and integrity of the annotation files.

This involved checking for any errors, missing information, or inconsistencies within the annotation files.

If any issues were found, we corrected or omitted the problematic annotations to ensure high-quality training data.

Dataset Split

We split the combined dataset, including the provided video clips and additional datasets, into training and validation sets.

The training set was used to train the YOLOv7 model, while the validation set served as a separate dataset for evaluating the model's performance during training.

We ensured an appropriate distribution of annotated images across the training and validation sets to capture the variability of organisms and environmental conditions.

By performing these data preprocessing steps, we obtained a clean and properly formatted dataset suitable for training the YOLOv7 model. This ensured that our program was trained on a diverse and representative dataset, allowing it to generalize well to unseen test data. The preprocessing steps also helped in mitigating any potential data quality issues and maintaining the integrity of the annotations throughout the training and evaluation process.

Training

The training process for YOLOv7 in the Ocean Exploration Video Challenge involved several steps to ensure the model's accuracy and robustness. We have used the official YOLOv7 training code with the following Model Configuration and Hyperparameters used in our training process:

Model Configuration

The model configuration specifies the architecture of the YOLOv7 model, including the number and type of layers, as well as other settings such as the initial weights path, data.yaml path, and hyperparameters path

Some of the more important of these parameters:

- epochs (int): the number of epochs to train for (one epoch is one full pass through the training data)
- workers (int): how many subprocesses to parallelize during training
- img_size (int): the resolution of our images
- batch_size (int): determines the number of samples processed before the model update is created

```
weights: .\runs\train\yolov7-custom\weights\last.pt
cfg: ''
data: data/custom_data.yaml
hyp: data/hyp.scratch.custom.yaml
epochs: 100
batch_size: 4
img_size:
- 640
- 640
rect: false
resume: true
nosave: false
notest: false
noautoanchor: false
evolve: false
bucket: ''
cache_images: false
image_weights: false
device: '0'
multi_scale: false
single_cls: false
adam: false
```

```
sync_bn: false
local_rank: -1
workers: 1
project: runs/train
entity: null
name: yolov7-custom
exist_ok: false
quad: false
linear_lr: false
label_smoothing: 0.0
upload_dataset: false
bbox_interval: -1
save_period: -1
artifact_alias: latest
freeze:
- 0
v5_metric: false
world_size: 1
global_rank: -1
save_dir: runs\train\yolov7-custom
total_batch_size: 4
```

Model configuration file for our program

Training Hyperparameters

Hyperparameter tuning is an important step in training a YOLOv7 custom model. It involves experimenting with different values for the hyperparameters and evaluating the performance of the model under each configuration. This process can be time-consuming and resource-intensive, but it is essential for achieving the best possible performance from an object detection model.

```
lr0: 0.01
lrf: 0.1
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.3
cls_pw: 1.0
obj: 0.7
obj_pw: 1.0
iou_t: 0.2
anchor_t: 4.0
fl_gamma: 0.0
```

```
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.2
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
paste_in: 0.0
loss_ota: 1
```

Hyperparameters file for our program

Inference and Post-processing

Once the YOLOv7 model was trained on the annotated dataset, we applied it to perform object detection on the test video clips. Here is an overview of the inference process and the post-processing steps we implemented using the official YOLOv7 repository:

Inference Process

For each frame in the test video clips, we fed the frame as input to the trained YOLOv7 model.

```
# Run inference
if device.type != 'cpu':
    model(torch.zeros(1, 3, imgsz,
↪  imgsz).to(device).type_as(next(model.parameters())))  # run once
old_img_w = old_img_h = imgsz
old_img_b = 1

for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)

        # Warmup
        if device.type != 'cpu' and (old_img_b != img.shape[0] or old_img_h !=
        ↪  img.shape[2] or old_img_w != img.shape[3]):
            old_img_b = img.shape[0]
            old_img_h = img.shape[2]
            old_img_w = img.shape[3]
            for i in range(3):
                model(img, augment=opt.augment)[0]

        # Inference
        with torch.no_grad():   # Calculating gradients would cause a GPU memory
        ↪  leak
            pred = model(img, augment=opt.augment)[0]
```

The model processed the frame and predicted bounding box coordinates and class labels for each detected organism.

Obtaining Bounding Box Coordinates and Class Labels

To obtain the bounding box coordinates and class labels for each detected organism, we extracted the relevant information from the model's output.

```
# Apply Classifier
if classify:
    pred = apply_classifier(pred, modelc, img, im0s)
```

The predicted bounding box coordinates represented the location and size of the detected organism within the frame.

The output of the model consisted of the coordinates of the bounding boxes (x, y, width, height) and the corresponding class labels.

The class labels indicated the category or type of organism detected, such as annelids, arthropods, cnidarians, etc.

These coordinates and class labels were used to create the annotations for each frame, allowing us to visualize and evaluate the model's performance.

Post-processing Steps

To refine the detections and eliminate redundant or overlapping bounding boxes, we applied post-processing techniques, such as non-maximum suppression (NMS).

```
# Apply NMS
pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes, agnostic=opt.agnostic_nms)
```

NMS helps remove duplicate detections by suppressing bounding boxes with high overlap (based on IoU) and retaining the one with the highest confidence score.

This post-processing step helped ensure that each organism was represented by a single bounding box, reducing redundancy and improving the accuracy of the final detections.

Visualization and Evaluation

After applying the post-processing steps, we visualized the final detections by drawing the refined bounding boxes on each frame of the test video clips.

```
# Process detections
for i, det in enumerate(pred):  # detections per image
    if webcam:  # batch_size >= 1
        p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
    else:
        p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)

    p = Path(p)  # to Path
    save_path = str(save_dir / p.name)  # img.jpg
    txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # img.txt
    gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
    if len(det):
        # Rescale boxes from img_size to im0 size
        det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()

        # Print results
        for c in det[:, -1].unique():
            n = (det[:, -1] == c).sum()  # detections per class
            s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

        # Write results
        for *xyxy, conf, cls in reversed(det):
            if save_txt:  # Write to file
                xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                line = (cls, *xywh, conf) if opt.save_conf else (cls, *xywh)  # label format
                with open(txt_path + '.txt', 'a') as f:
                    f.write(('%g ' * len(line)).rstrip() % line + '\n')

            if save_img or view_img:  # Add bbox to image
                label = f'{names[int(cls)]} {conf:.2f}'
                plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
```

The class labels were also displayed alongside the corresponding bounding boxes, providing a comprehensive visualization of the detected organisms.

We evaluated the performance of the model by comparing the predicted annotations with the ground truth annotations for the test video clips.

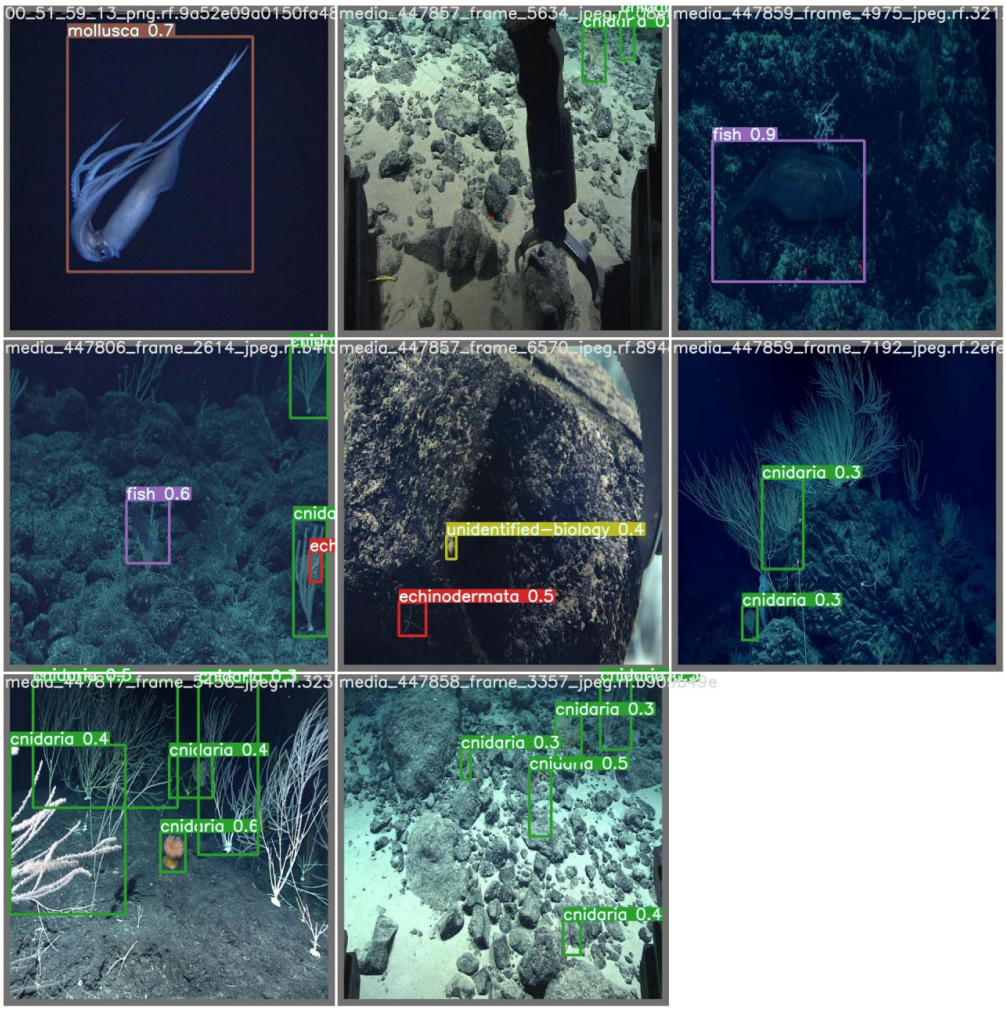


Figure 3: A Result Image showing Bounding Boxes

By following this inference process and applying post-processing techniques like non-maximum suppression, we aimed to obtain accurate and refined detections of marine organisms in the test video clips. These steps helped in reducing duplicate detections and improving the precision of the final results, enabling effective analysis and interpretation of the underwater exploration conducted by NOAA Ocean Exploration.

User Guide

Pre-requisites

Make sure you have the following installed on your system:

- Python 3
- pip

```
# Clone the official YOLOv7 repository
git clone https://github.com/WongKinYiu/yolov7.git
# Change directory
cd yolov7
```

```
# Install requirements
pip install -r requirements.txt
```

Running the Program

```
python3 detect.py \
    --weights <path/to/the/submitted/model.pt> \
    --source <path/to/the/test/video/clip.mp4> \
    --conf 0.25 \
    --img-size 640
```

Note: To test for an image or webcam, replace the `--source` argument with,

- `--source <path/to/the/test/image.jpg>`
- `--source 0` (for internal webcam)
- `--source 1` (for external webcam)

Output

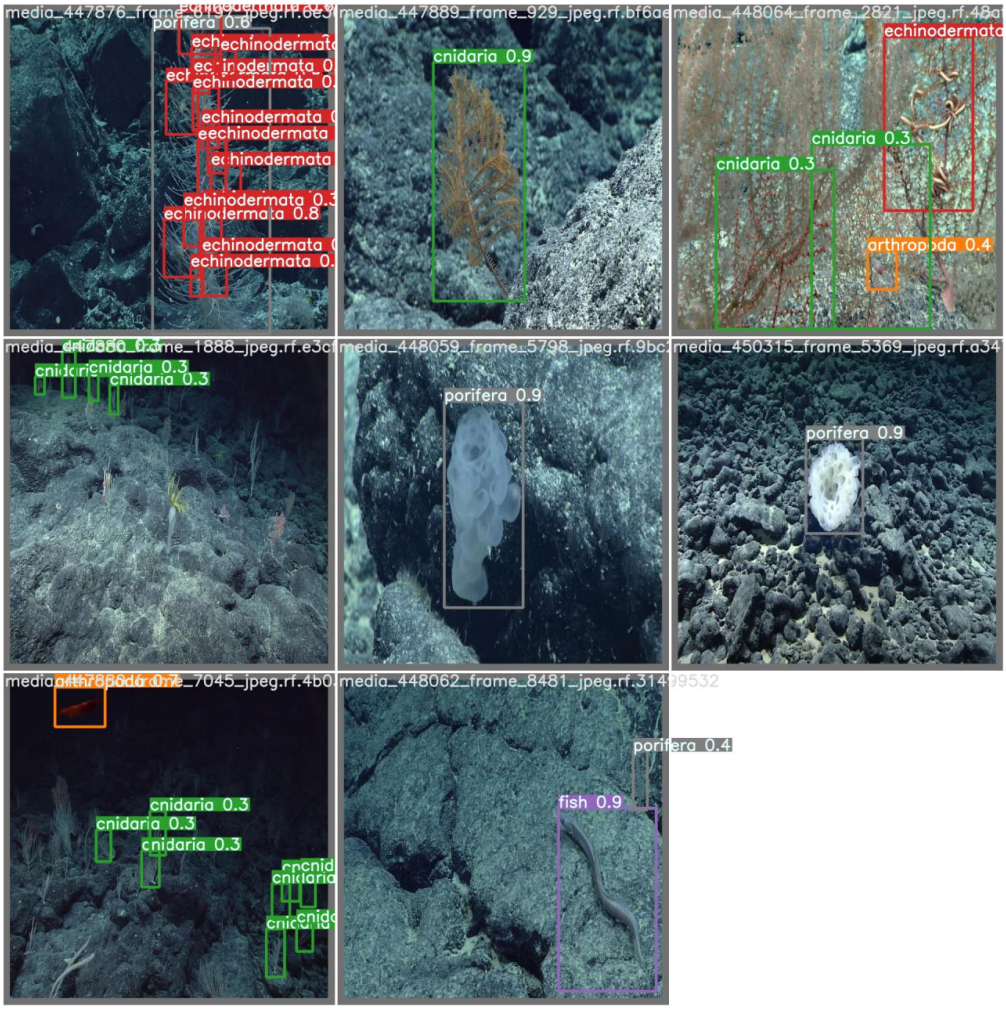
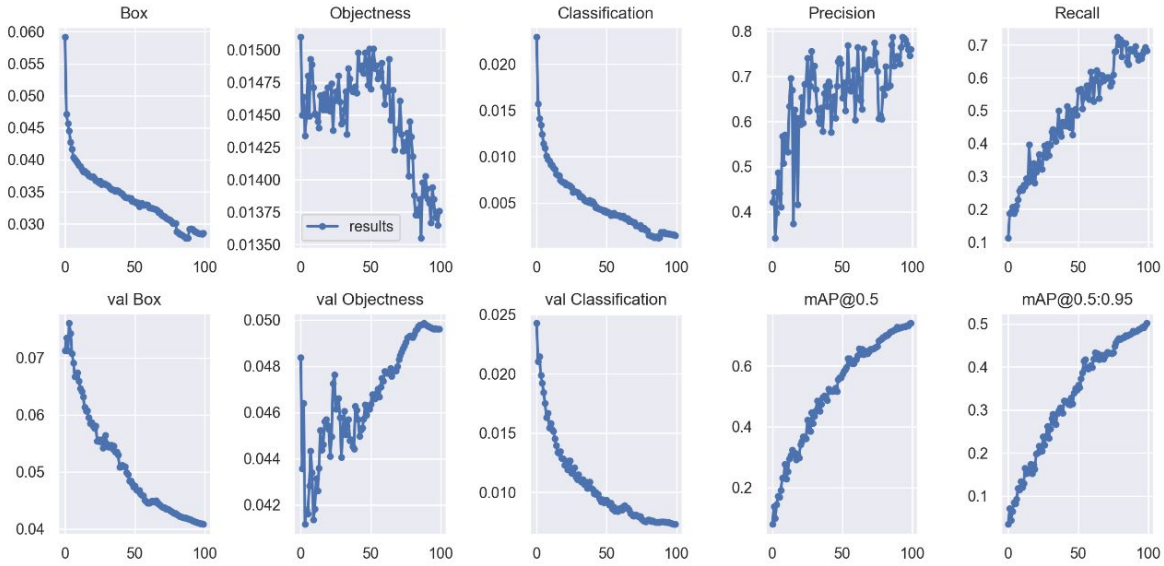


Figure 4: Sample Output Image

The output of the program will be saved in the `runs/detect` directory.

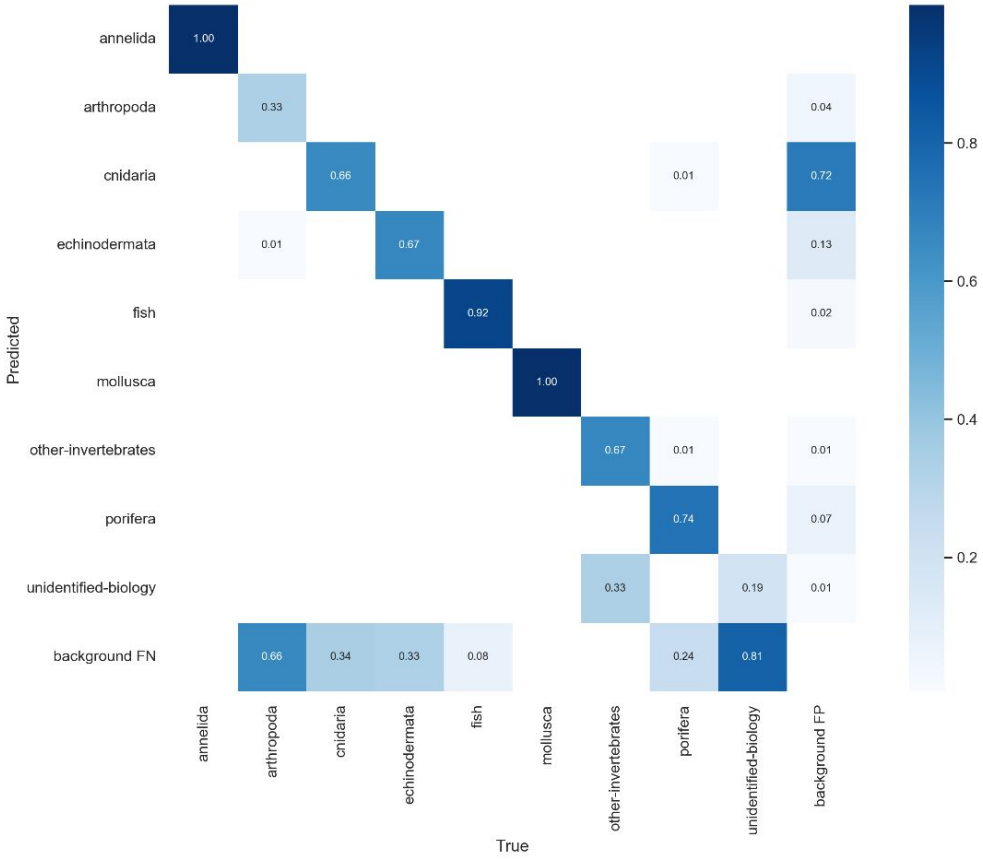
Results and Performance

The performance of the YOLOv7 model on the provided test video clips was evaluated using various metrics to assess the confidence, precision, recall, and overall performance of the object detection. Here are the performance metrics for the model:

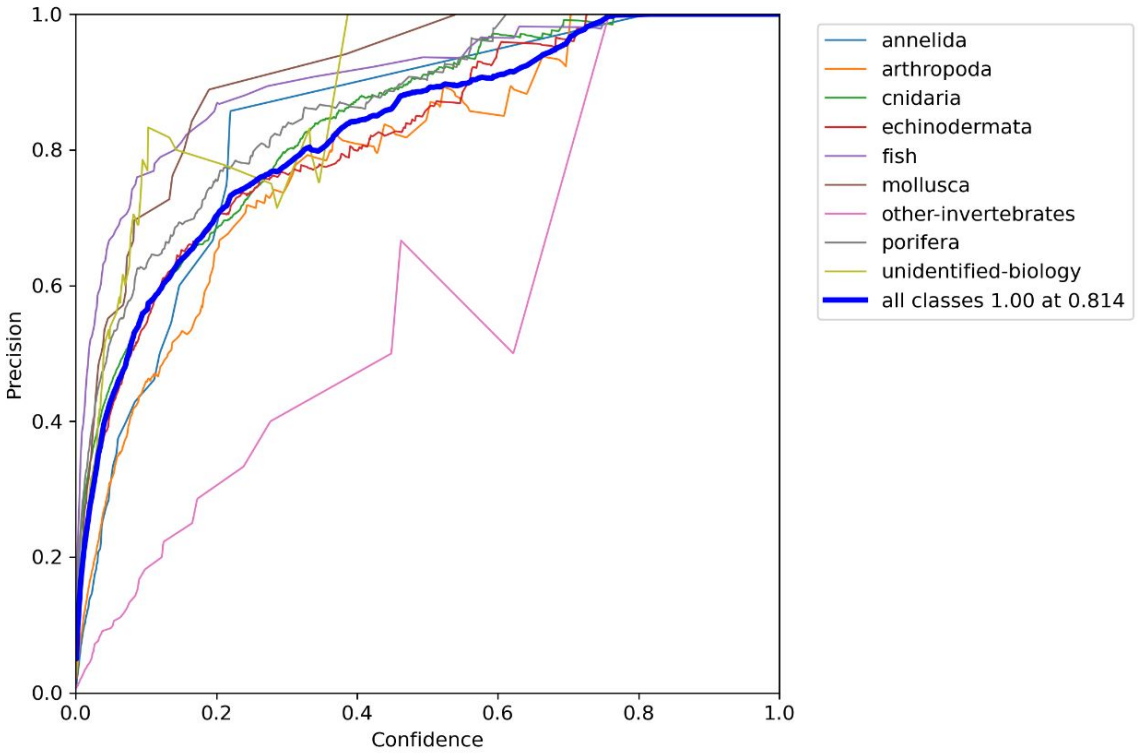


Performance Metrics

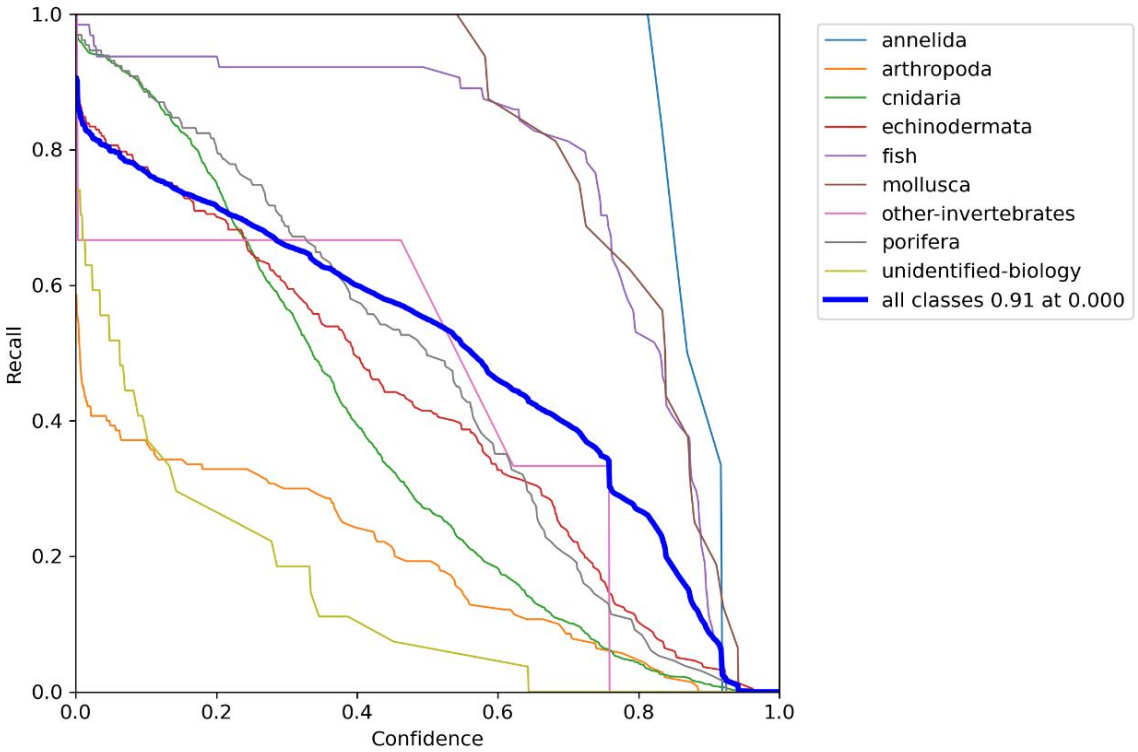
Accuracy: Accuracy measures the overall correctness of the model's predictions. It is the ratio of correctly predicted bounding boxes to the total number of bounding boxes.



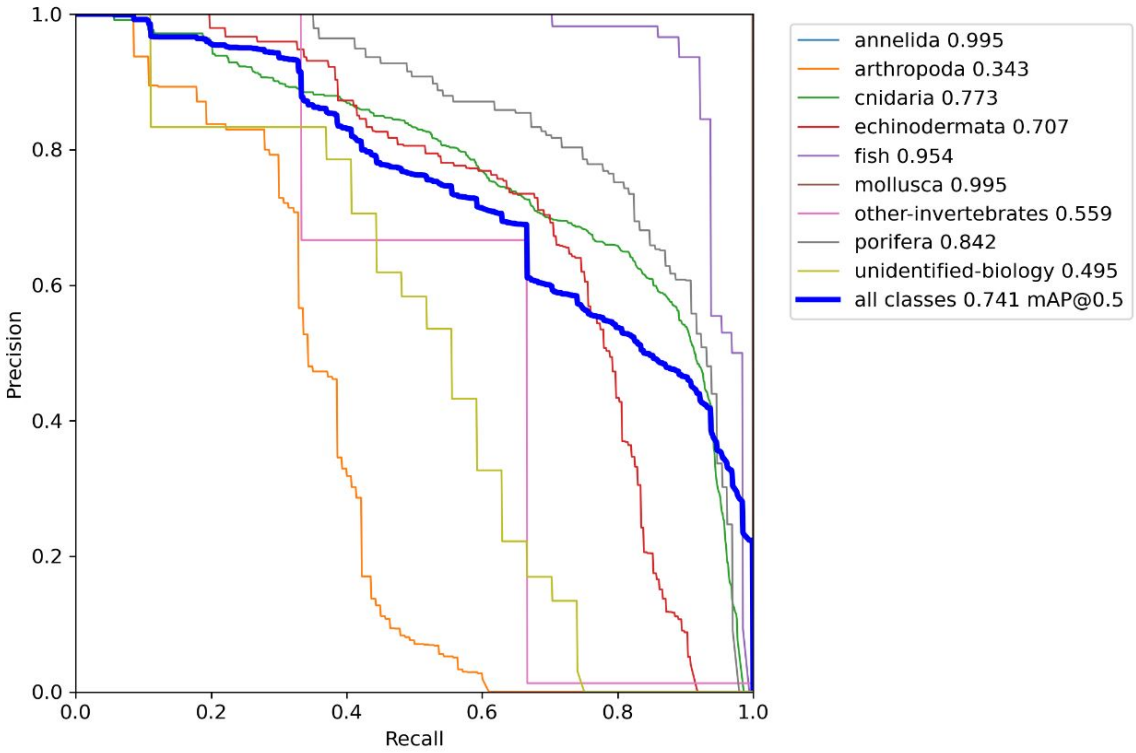
Precision: Precision measures the proportion of correctly predicted positive detections (true positives) out of all the predicted positive detections (true positives + false positives).



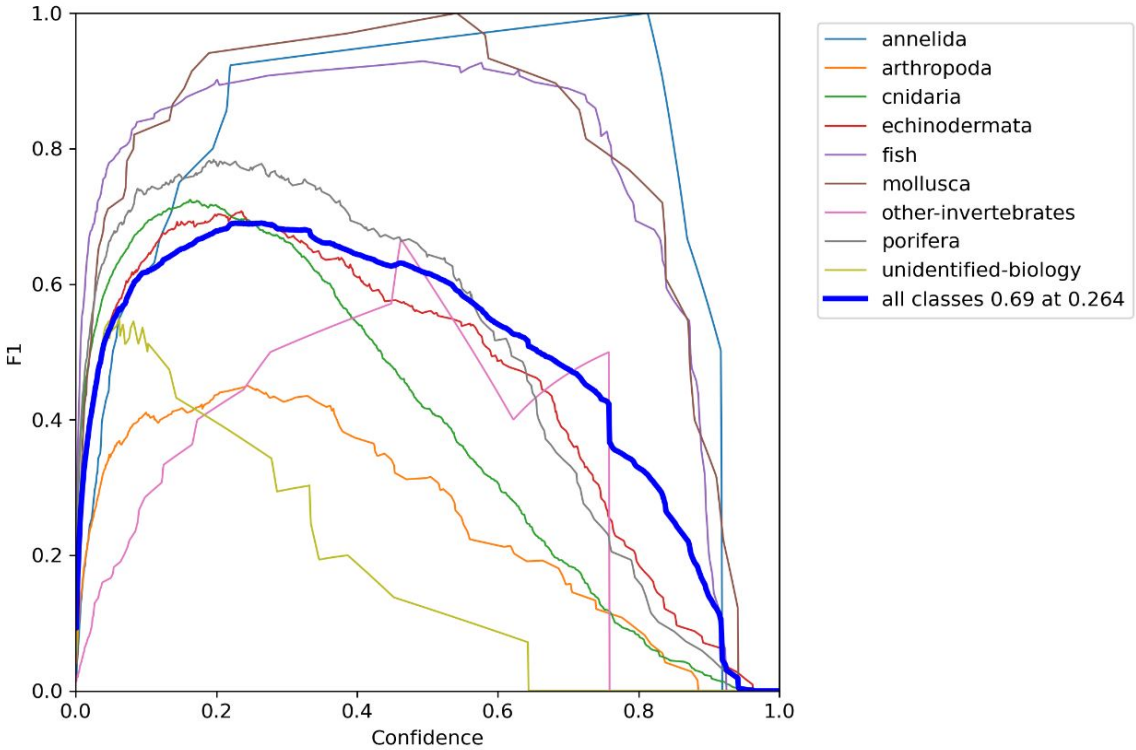
Recall: Recall measures the proportion of correctly predicted positive detections (true positives) out of all the actual positive instances (true positives + false negatives).



Precision-Recall Curve: A precision-recall curve is a plot of precision versus recall for different threshold values of a model's predicted probability, useful in cases where there is class imbalance or when the cost of false positives and false negatives is different.



F1 Score: The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's accuracy and completeness.



Limitations and Challenges

Variability in Environmental Conditions: The marine environment is known for its variability, including factors like lighting conditions, water turbidity, and occlusions. These factors can pose challenges for accurate object detection, especially if the training data does not sufficiently represent such variations.

Annotation Quality: The accuracy of the ground truth annotations used for evaluation can impact the performance metrics. Inaccurate or imprecise annotations may affect the calculation of precision, recall, and ultimately the overall performance assessment.

Organism Variability: Marine organisms can exhibit diverse appearances and behaviors, making their detection and classification challenging. Some organisms may have complex shapes or exhibit camouflage, leading to potential misclassifications or missed detections.

Conclusion

By considering the performance metrics and addressing the limitations and challenges, we aimed to develop a YOLOv7 model that could accurately detect and classify marine organisms in the test video clips, contributing to the goals of the Ocean Exploration Video Challenge.