Dataset

Aquarium Object Detection Dataset

Introduction

This dataset consists of 638 images collected by Roboflow from two aquariums in the United States. The main objective is to detect sea objects from the dataset and with a metric of detection. Even though the dataset has multiple uses, the main goal here is for coral reef conservation and give enough insight to the observer to make impact decisions for swimmer safety or even monitoring the sea animals. This extensive collection of aquatic images proves to be an asset with versatile applications, contributing to the advancement of innovation and research within the domains of computer vision and object detection. The drive behind this lies in the potential to progress the comprehension of marine ecosystems, bolster conservation initiatives, and propel innovation within the domain of computer vision. From vibrant displays of marine species to intricate tank setups, this dataset promises a captivating exploration of the underwater world, fueling curiosity and facilitating cutting-edge research.

The aquarium dataset's significance lies in its ability to bridge the gap between technology and marine science, offering a versatile resource that has the potential to shape advancements in research, education, conservation, and innovation. The aquarium dataset is important to analyze because it has advancing computer vision application and can enhance the understanding of aquatic life as well as boost its research. After the Exploratory Data Analysis (EDA) for the dataset, some conclusions can be drawn such as the number of fishes exceed the amount of any other sea creatures.

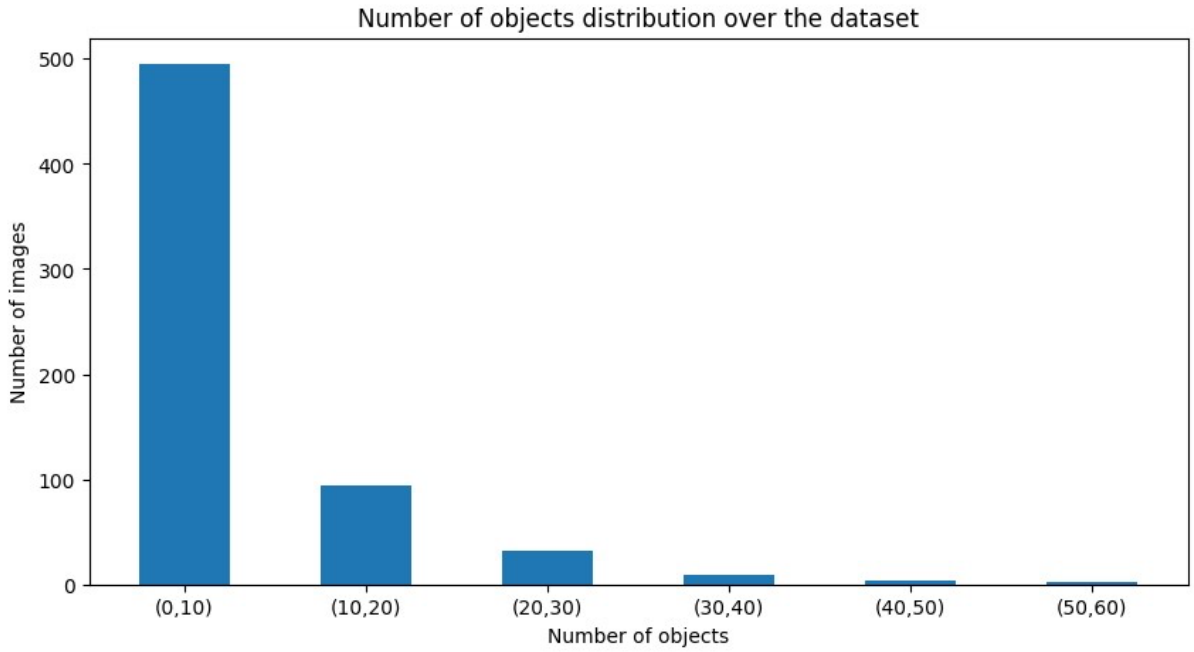
Sample Image from the dataset

EDA

Basic information about the dataset:

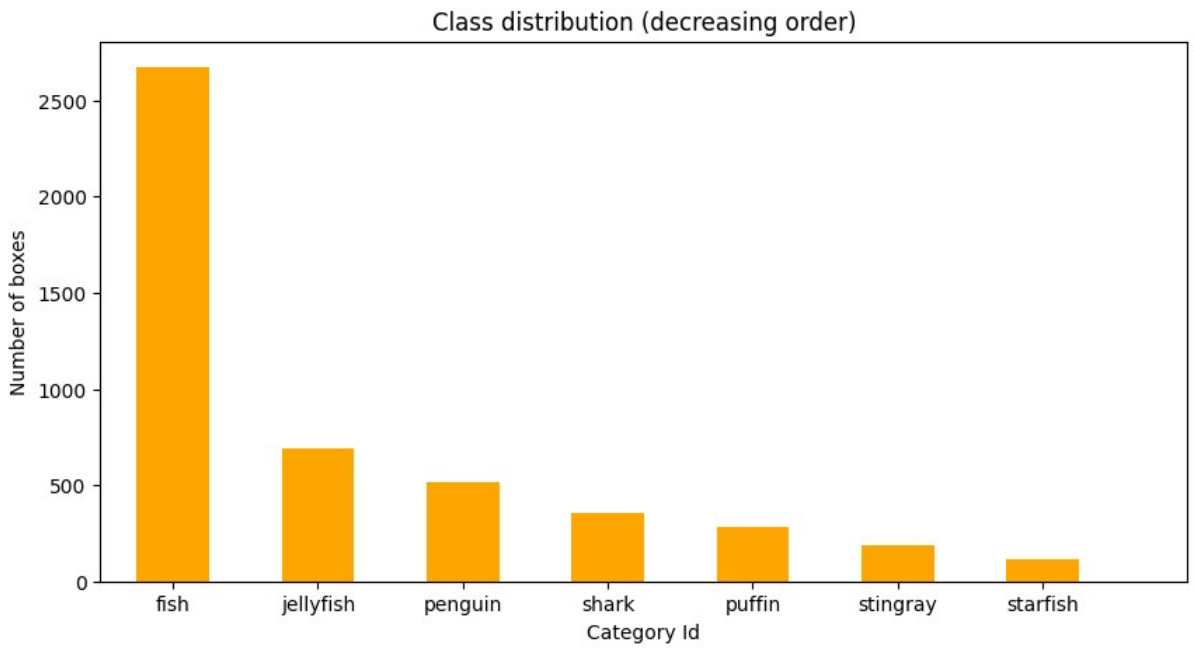
```
Dataset Name:  aquarium_objects
Number of images:  638
Number of bounding boxes:  4819
Number of classes:  8
Max min avg height:  640 640 640
Max min avg width:  640 640 640
```

Number of objects distribution across the dataset:



Number of images having at least one box: 637
Min number of boxes per image: 1
Max number of boxes per image: 56
Avg number of boxes per image: 7

Class wise distribution of objects:



Number of objects present for each class:

Number of Fish : 2673

Number of Jelly Fish : 694

Number of Penguin : 516

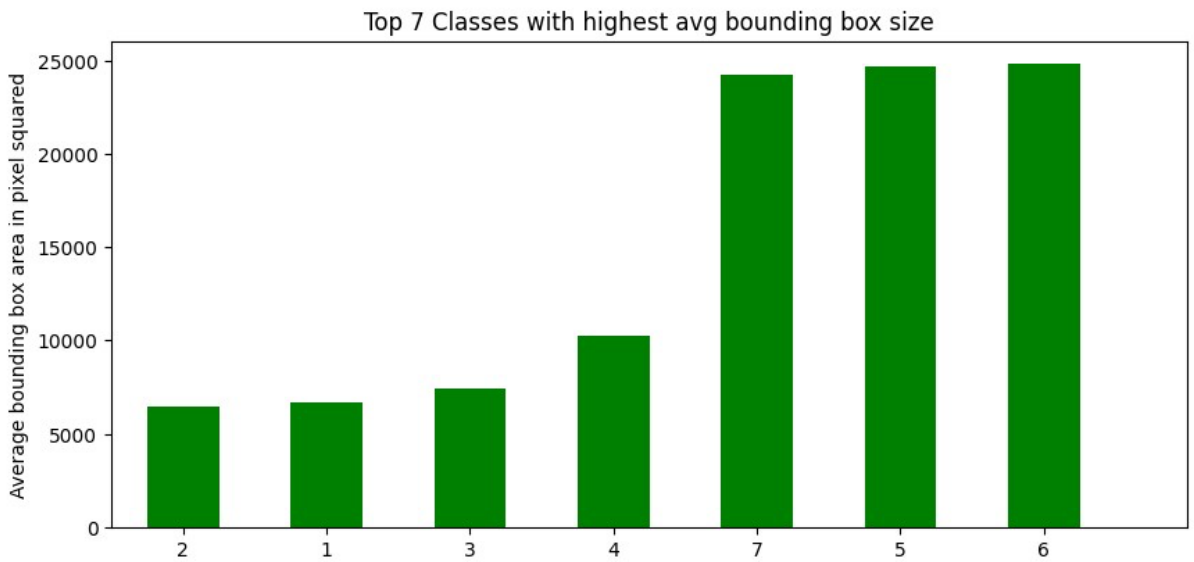
Number of Puffin : 284

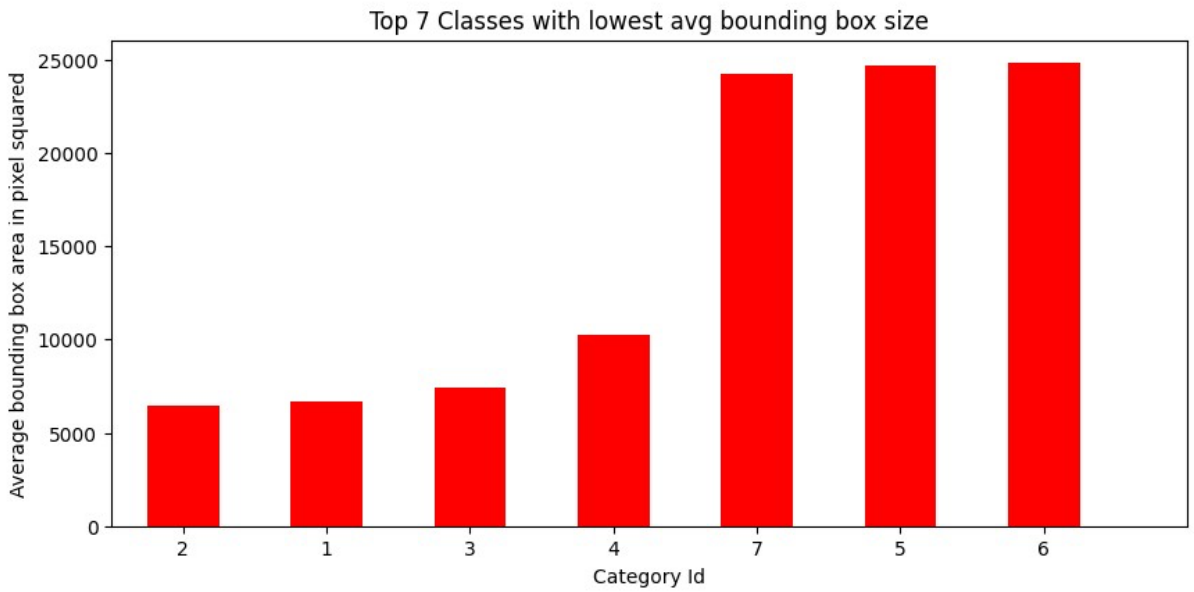
Number of Shark : 352

Number of Starfish : 116

Number of Stingray : 184

Class wise bounding box area distribution:





Class representation:

Class 0 -> creatures,

Class 1 -> fish,

Class 2 -> jelly fish,

Class 3 -> penguin,

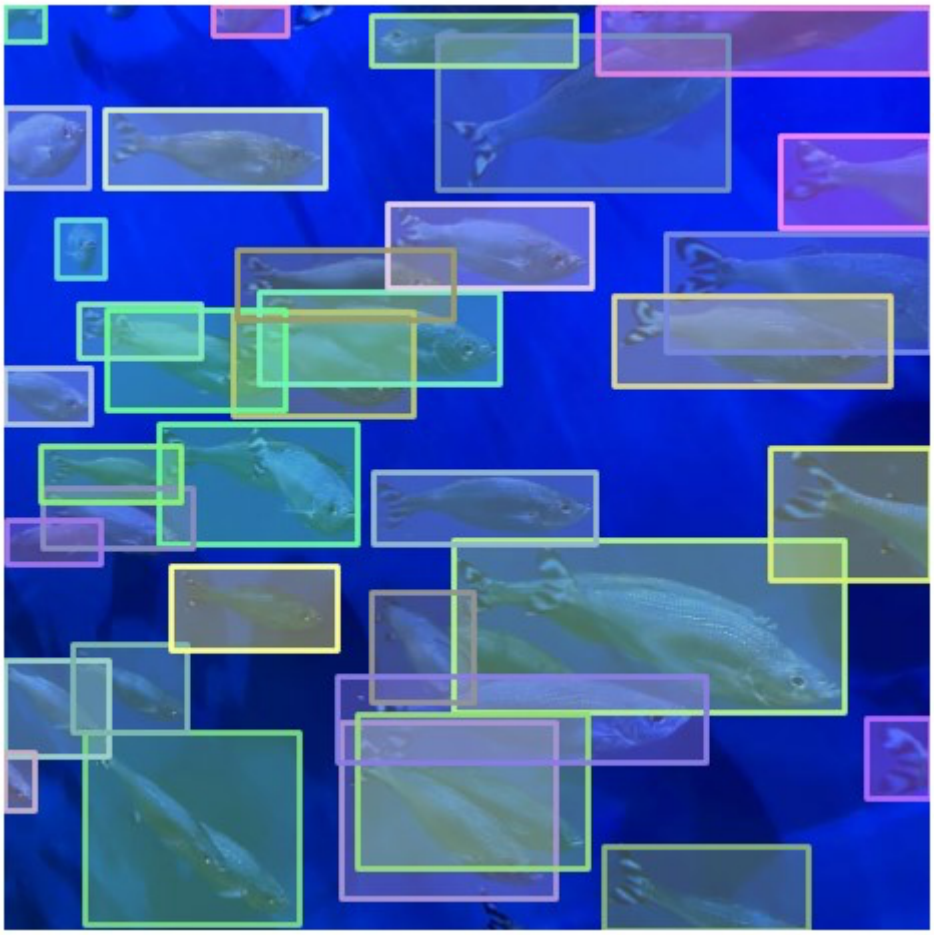
Class 4 -> puffin, Class

5 -> shark,

Class 6 -> starfish,

Class 7 -> stingray,

Annotation sample from the dataset:



Preprocessing

1. Removed images where there are no-classes present
2. Resized all the images of the dataset to size (640x640)
3. Auto oriented the images that was not oriented properly
4. Performed data-augmentation on the dataset. Tiled every one of the images to improve detection.
5. Other augmentation steps

   Flip: Horizontal, Vertical

   Rotation: Between -15° and +15°

   Bounding Box: Blur: Up to 2.5px

   Bounding Box: Noise: Up to 5% of pixels

6. The images were split into train-validation-test with 70:20:10 ratio
7. The train and valid set images were normalized between 0 and 1
8. Loaded every image with it's respective annotation file into pytorch-dataloader

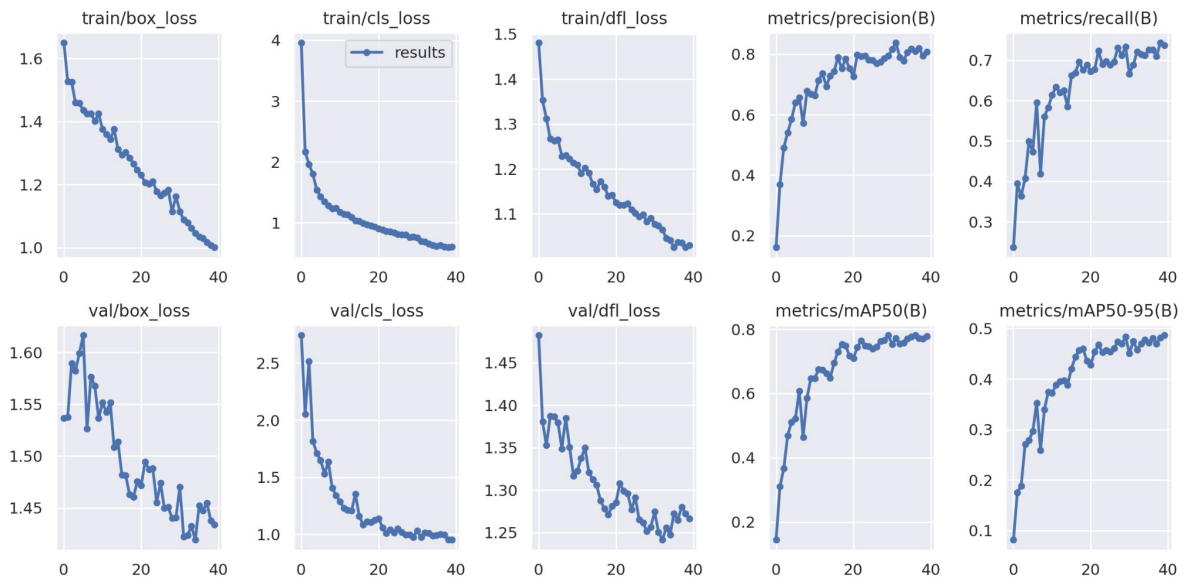
```
# read in PIL image and target in COCO format img, target = super(CocoDetection, self).__getitem__(idx)

# preprocess image and target (converting target to pytorch-dataloader format, resizing + normalization of both image and target) image_id = self.ids[idx] target = {'image_id': image_id, 'annotations': target} encoding = self.feature_extractor(images=img, annotations=target,
return_tensors="pt") pixel_values = encoding["pixel_values"].squeeze() # remove batch dimension target = encoding["labels"][0] # remove batch dimension
```

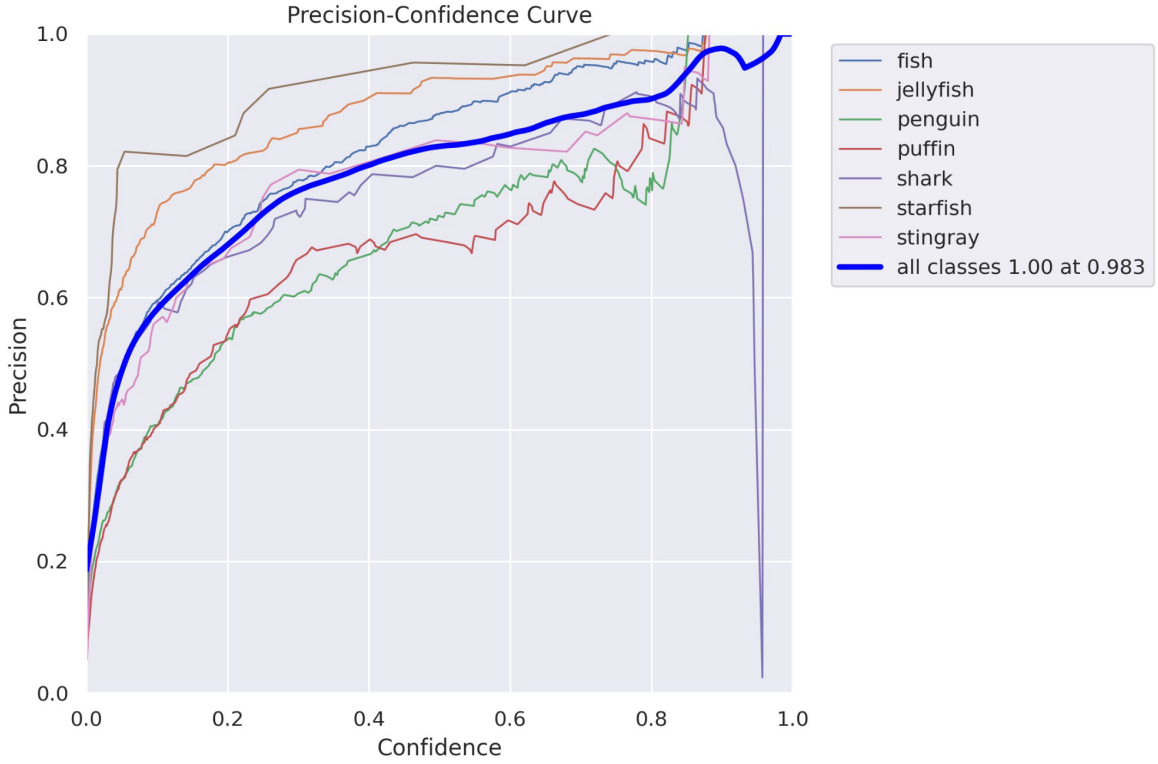
Three models were selected for the project namely YOLOv8, YOLOs and DETR object detection models. Trained every one of those models in their vanilla format with 40 epochs and then performed k-fold cross validation and hyperparameter tuning on all of them to improve their performance on selected dataset.

YOLOv8

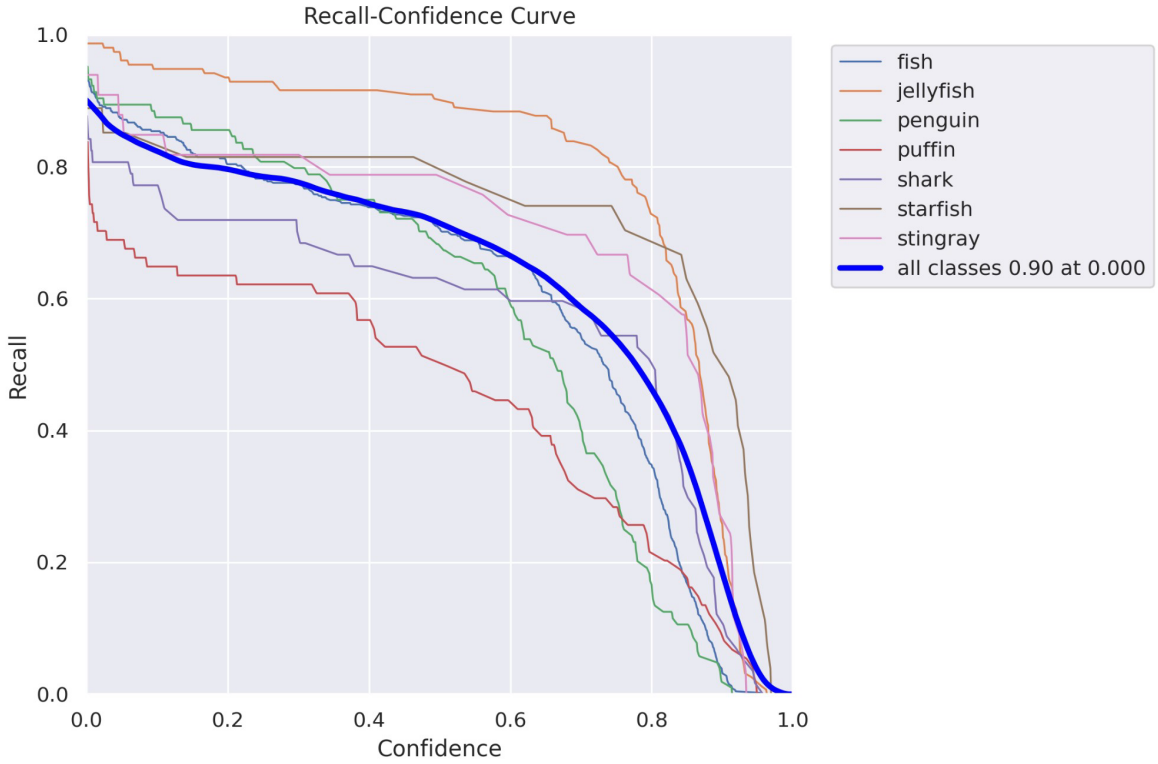
YOLOv8 is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks. YOLOv8 was developed by Ultralytics, who also created the influential and industry-defining YOLOv5 model. Also, trained YOLOv8 following the Ultralytics guide to train the model. Used the preprocessed dataset into the model to train and validate and then evaluated the model using the test set that was unseen to the model



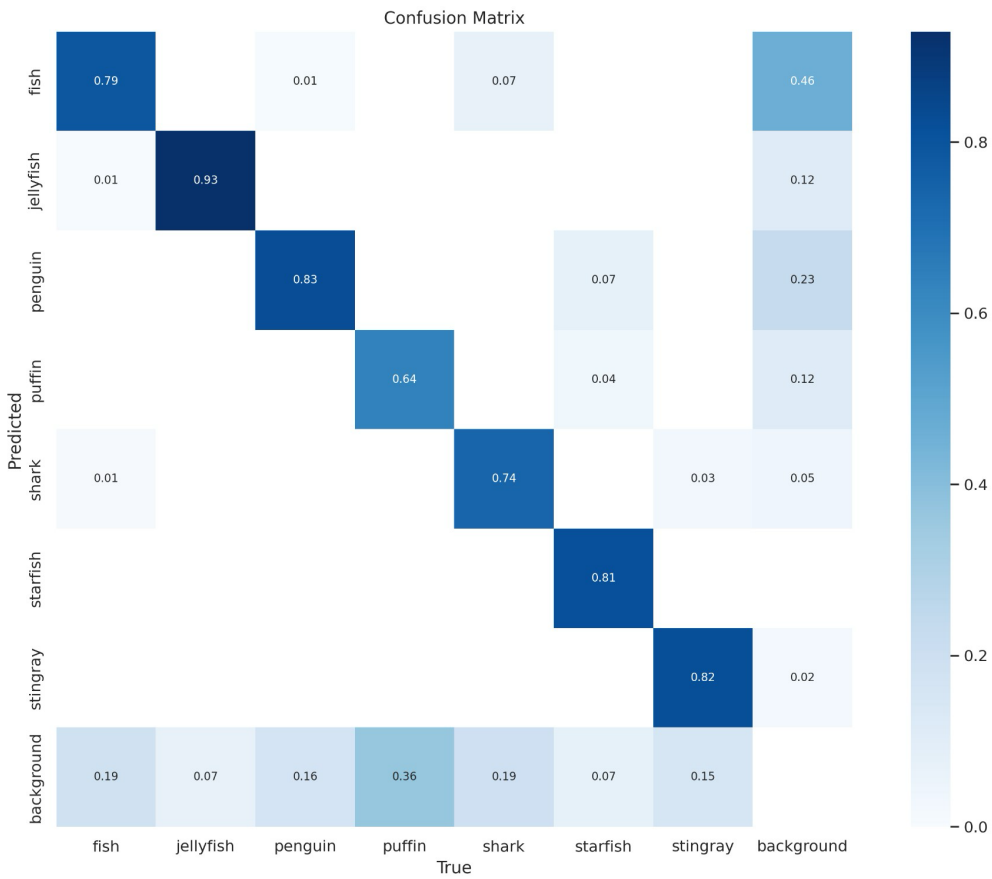
Train-validation performance



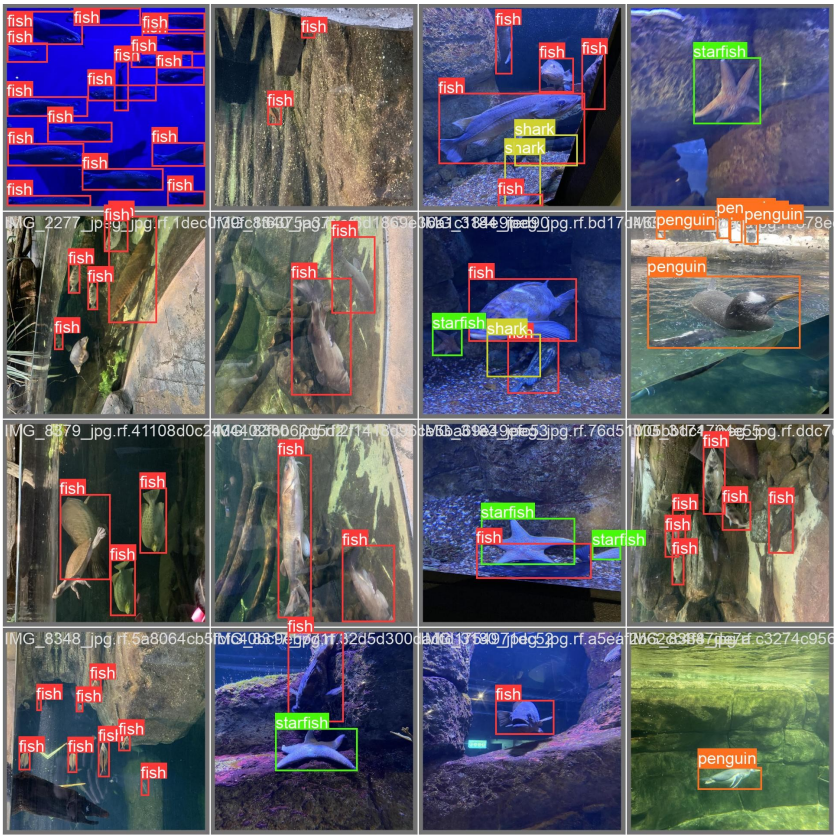
precision-confidence curve



Recall-confidence curve



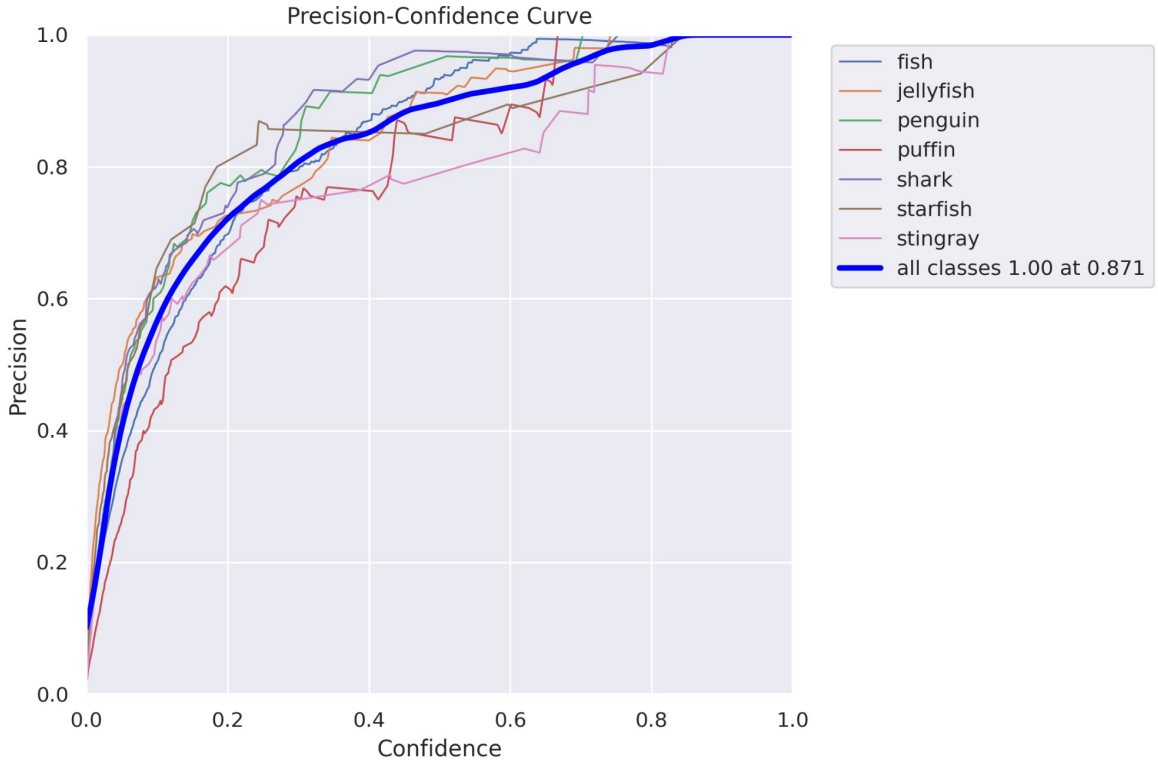
Confusion matrix



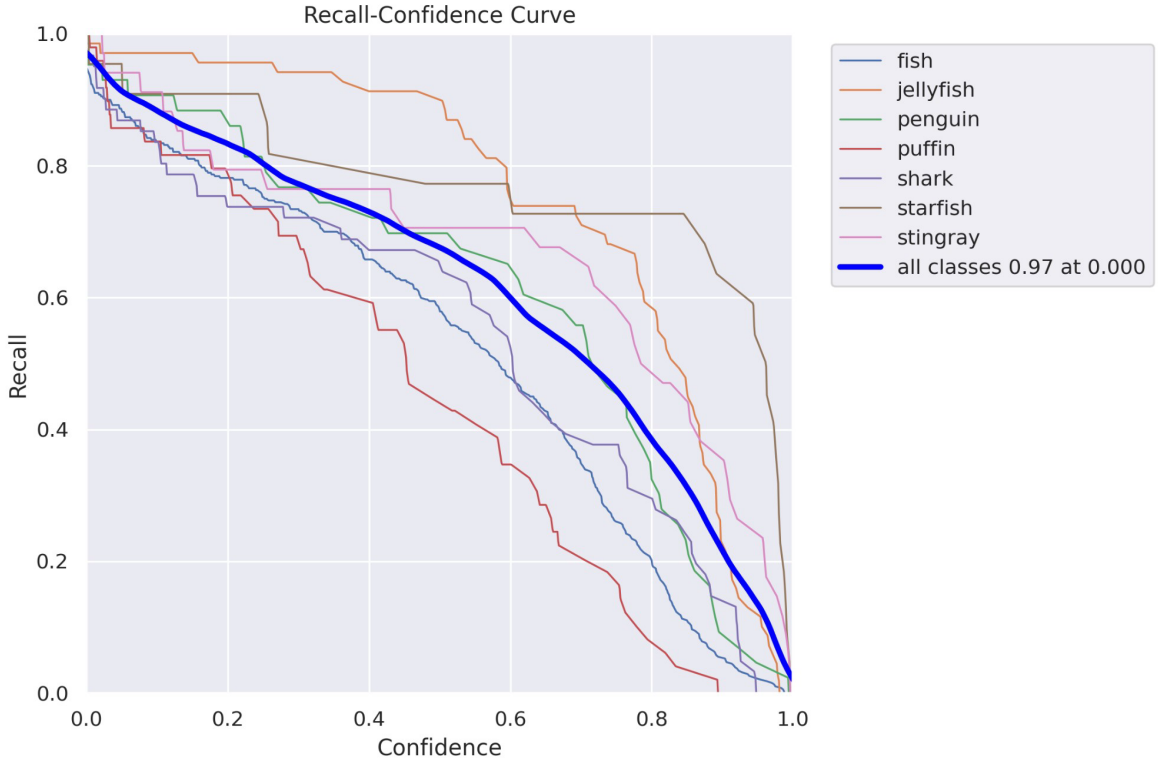
Detection results from YOLOv8

YOLOs-transformer model

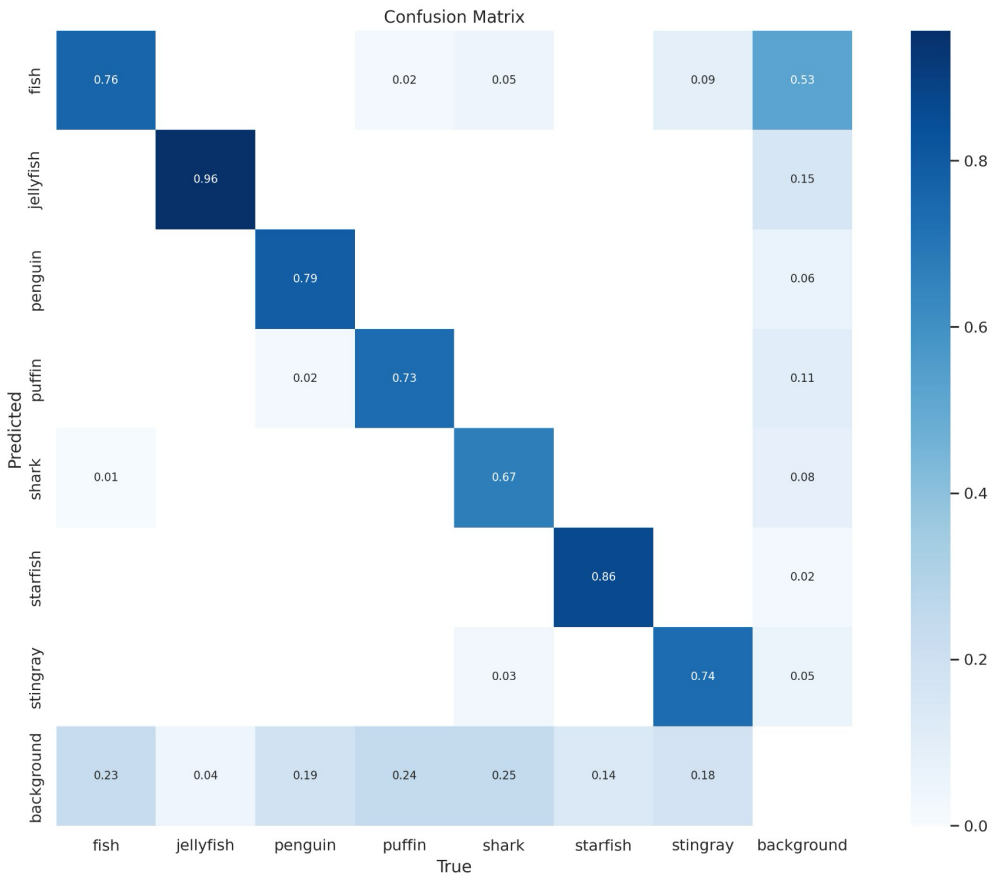
The YOLOS model was proposed in You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection by Yuxin Fang, Bencheng Liao, Xinggang Wang, Jiemin Fang, Jiyang Qi, Rui Wu, Jianwei Niu, Wenyu Liu. YOLOS proposes to just leverage the plain Vision Transformer for object detection, inspired by DETR. It turns out that a base-sized encoder-only Transformer can also achieve 42 AP on COCO, similar to DETR and much more complex frameworks such as Faster R-CNN. Followed their instruction on hugging face to train and evaluate the model.



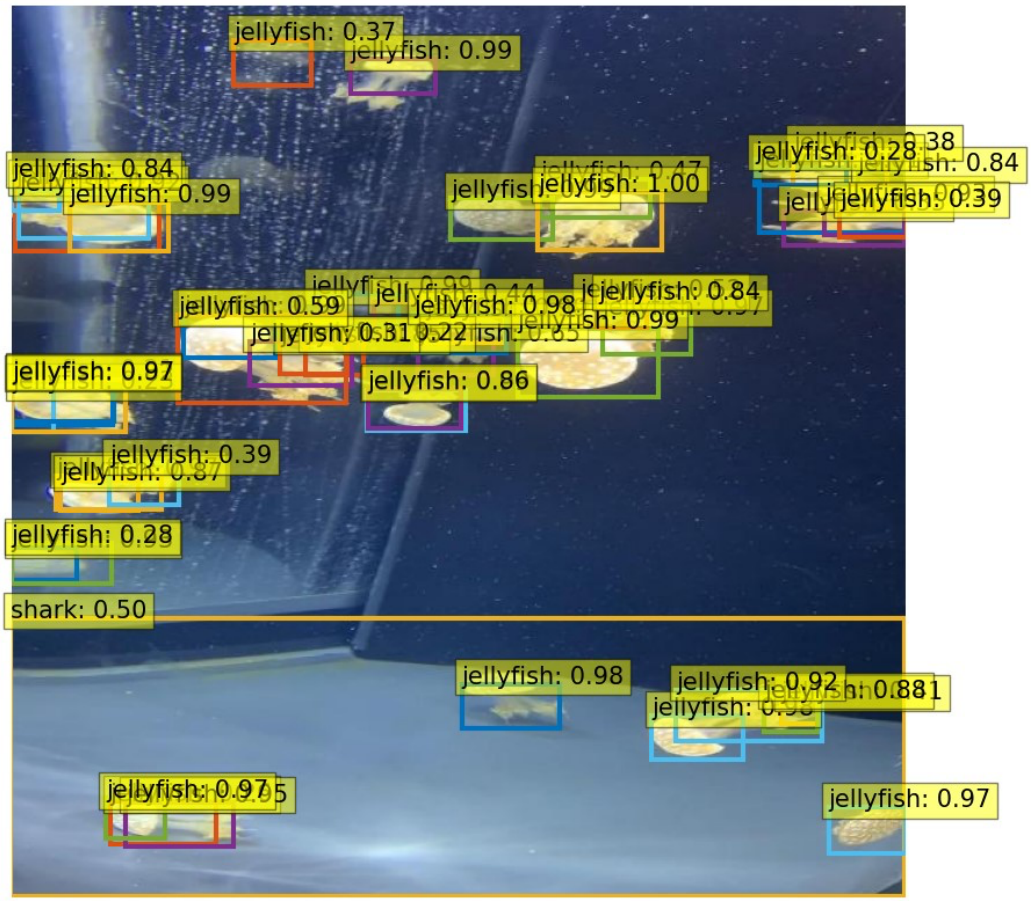
precision-confidence curve



recall-confidence curve



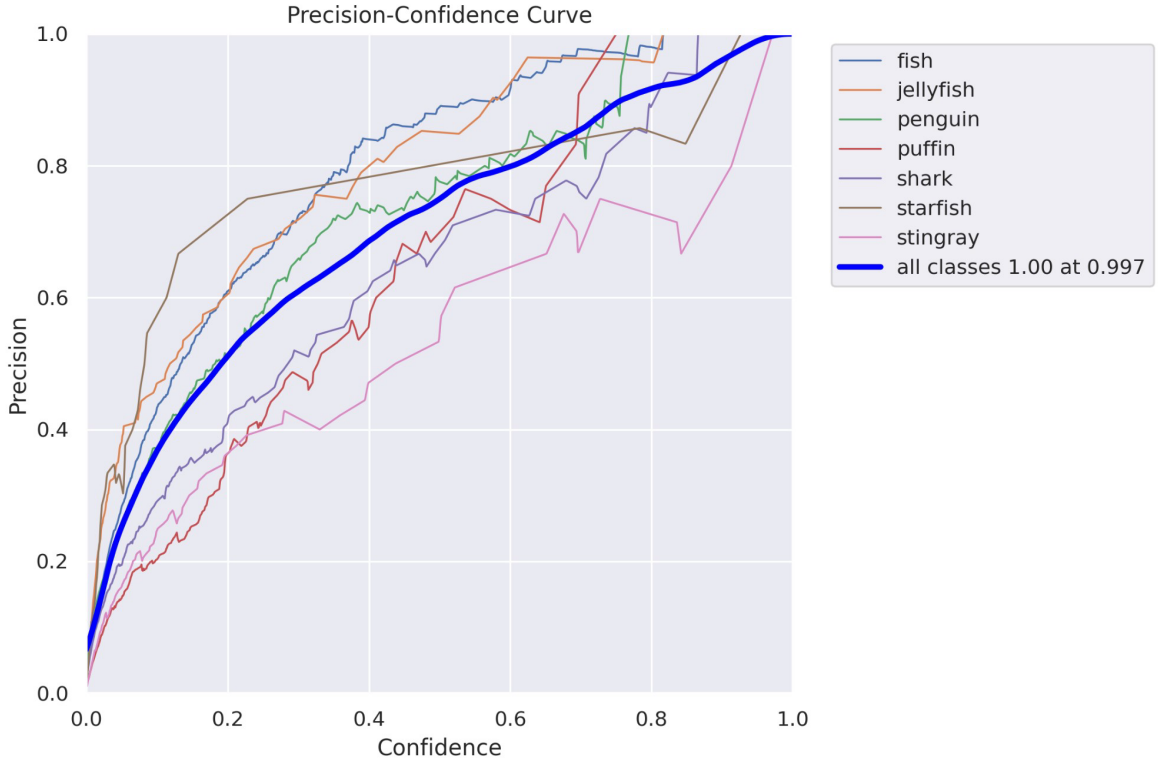
Confusion matrix



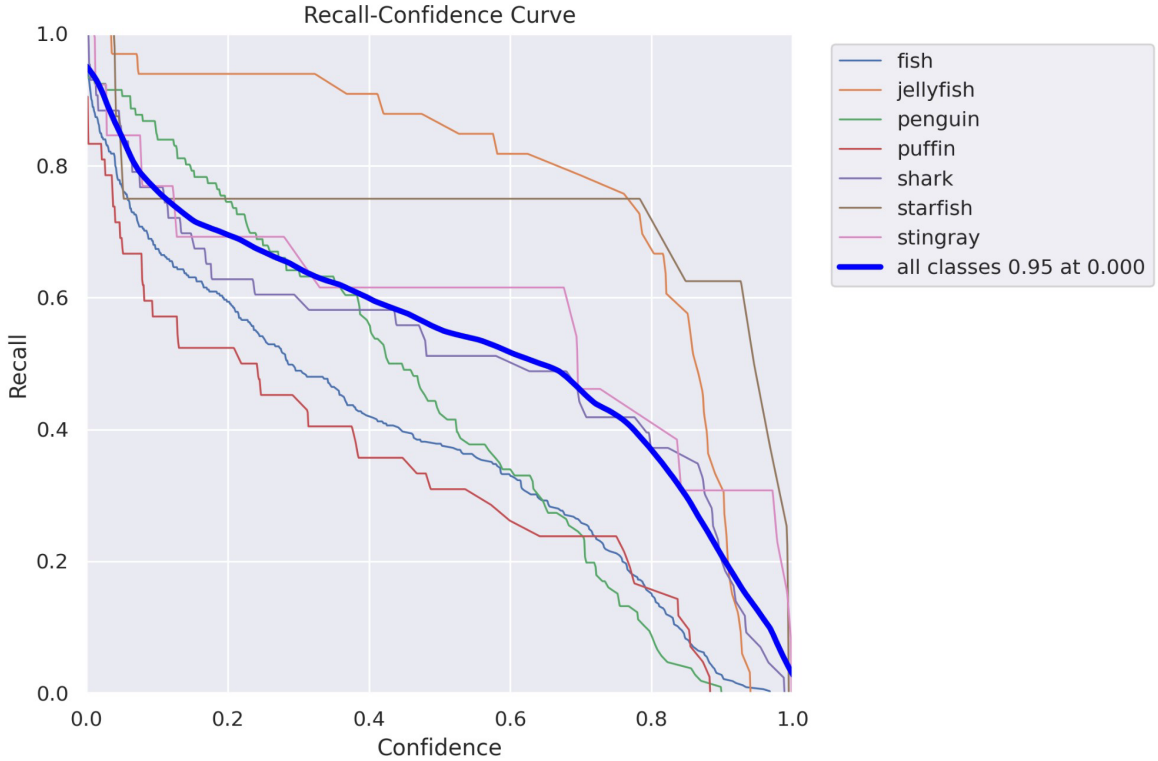
Detection on YOLOs-transformer model

<u>Detection Transformer (DETR) moel:</u>

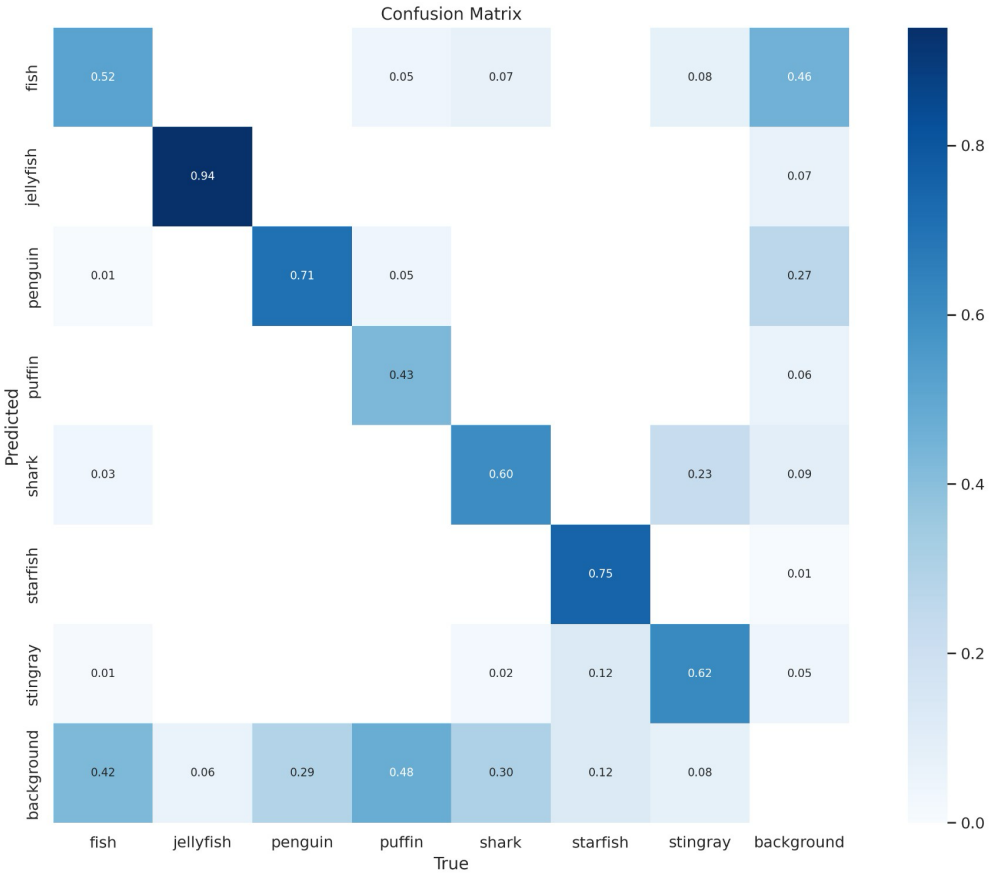
The DETR model was proposed in <u>End-to-End Object Detection with Transformers</u> by Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov and Sergey Zagoruyko. DETR consists of a convolutional backbone followed by an encoder-decoder Transformer which can be trained end-to-end for object detection. It greatly simplifies a lot of the complexity of models like Faster-R-CNN and Mask-R-CNN, which use things like region proposals, non-maximum suppression procedure and anchor generation. Moreover, DETR can also be naturally extended to perform panoptic segmentation, by simply adding a mask head on top of the decoder outputs.



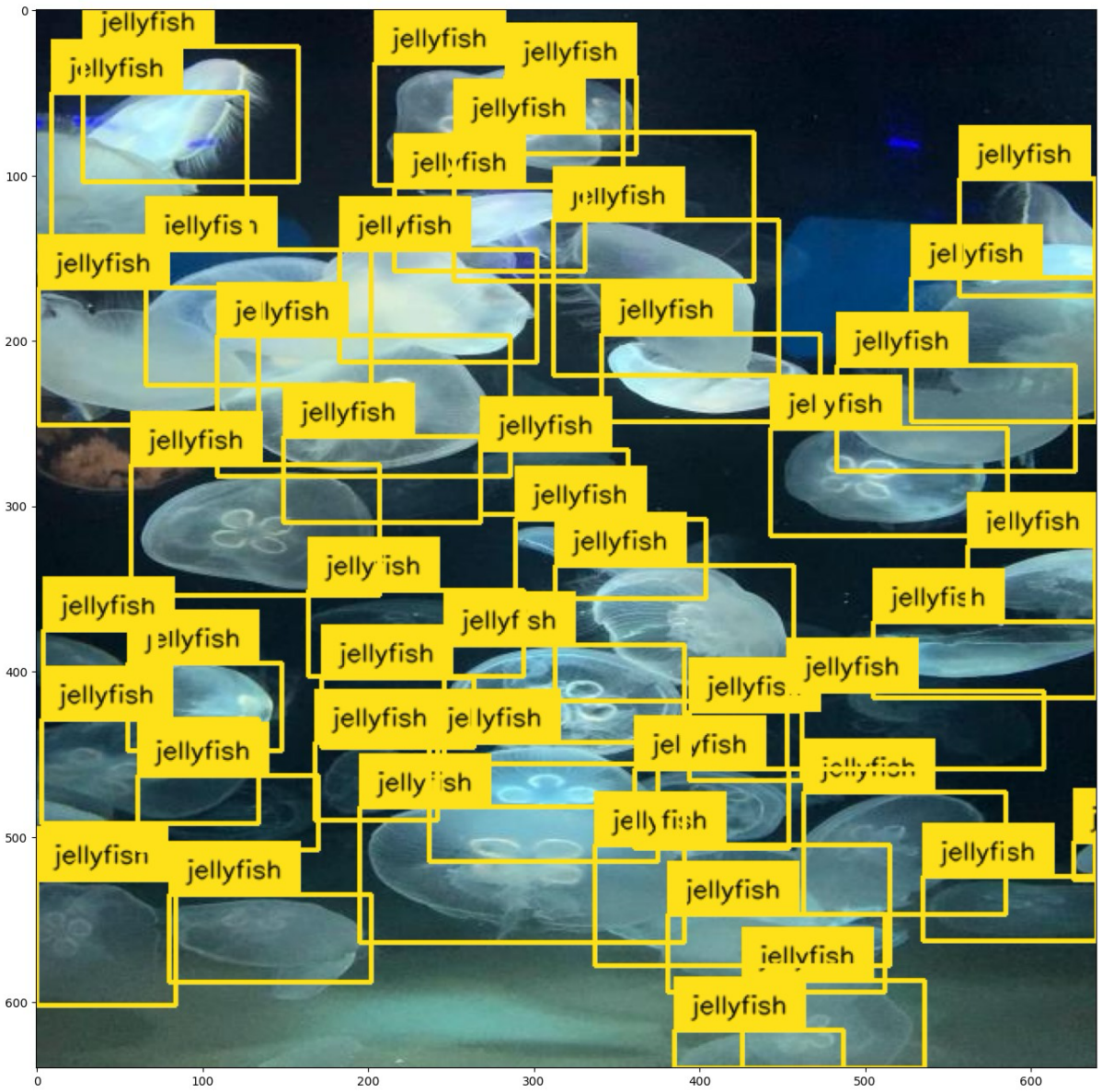
precision-confidence curve



recall-confidence curve



Confusion matrix



Detection on DETR model

Model Evaluation

Trained every model on 40 epocs with same train-validation set. And evaluated every model on same test set. Here's are the results comparisons below.

Evaluated each of the model in terms of precision, recall and f1-score:

```
Validating finetuned/train/weights/best.pt...
Ultralytics YOLOv8.0.20 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11128293 parameters, 0 gradients, 28.5 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%
                   all        127        909      0.816      0.733      0.778      0.487
                  fish        127        459      0.858      0.728      0.824      0.488
             jellyfish        127        155       0.91      0.912      0.952      0.546
               penguin        127        104      0.701      0.721       0.71      0.333
                puffin        127         74      0.687      0.527      0.562      0.303
                 shark        127         57      0.784      0.638      0.707       0.47
              starfish        127         27      0.952      0.815      0.869      0.689
              stingray        127         33      0.819      0.788      0.825      0.577
Speed: 0.4ms pre-process, 5.4ms inference, 0.0ms loss, 2.9ms post-process per image
```

Performance of YOLOv8 on test set

```
 IoU metric: bbox
  Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.210
  Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.451
  Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.163
  Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.050
  Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.174
  Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.360
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.139
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.307
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.401
  Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.139
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.339
  Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.594
```

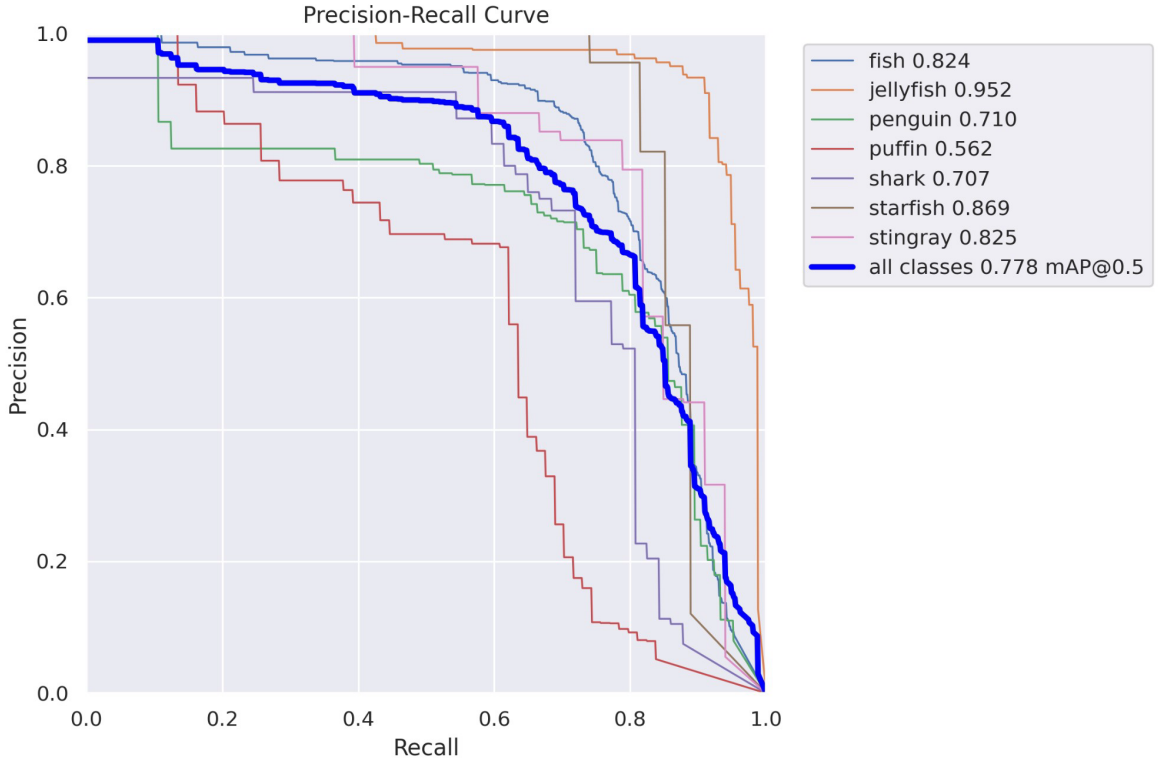
Performance of YOLOs-transformer on test set

```
 IoU metric: bbox
  Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.224
  Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.459
  Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.214
  Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.085
  Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.171
  Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.380
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.103
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.298
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.338
  Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.164
  Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.278
  Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.491
```

Performance of DETR on test set

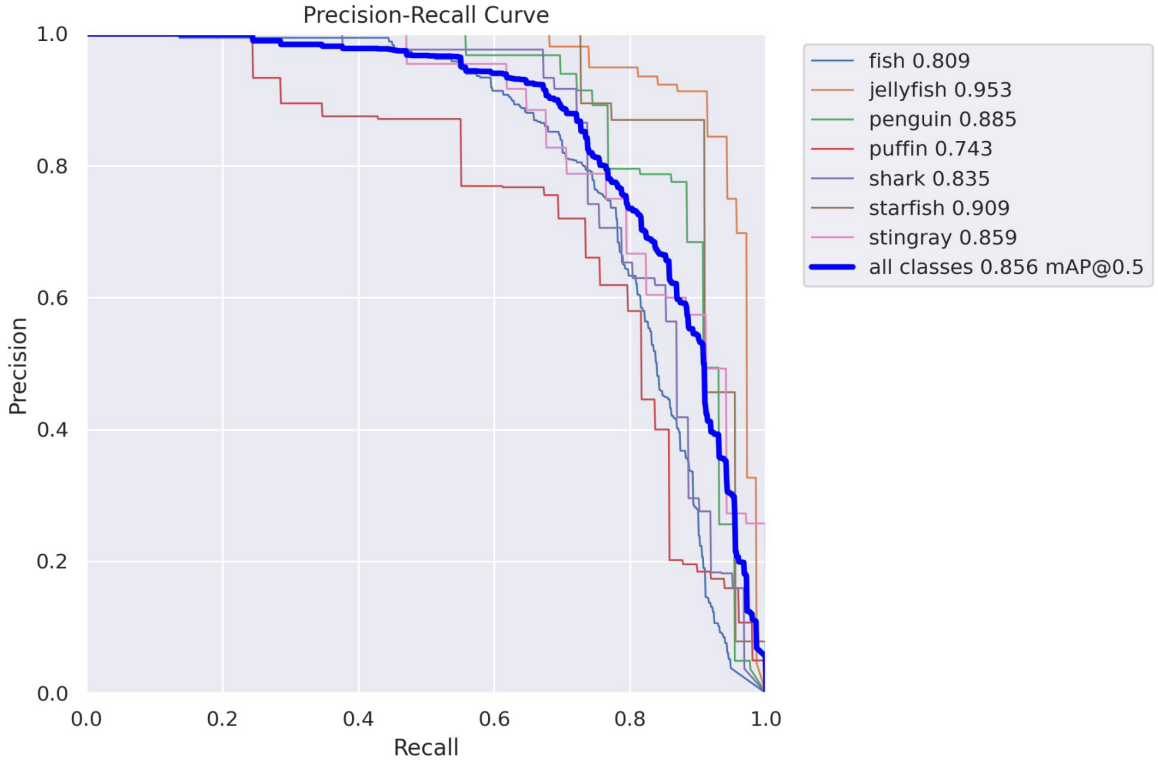
YOLOv8 model is performing better at every aspect than the other two models. For all classes, YOLOv8 achieved 81.6% precision on average for all classes, and 73.3% recall for all classes. While in 50-95% IoU, YOLOs and DETR achieved 21.0 % and 22.4% precision, the YOLOv8 achieved 48.7%. And at a lower confidence rate, YOLOv8 achieved
77.8% which is lot higher compared to YOLOs's 45.1% or DETR's 45.9%.

Precision-Recall curve for each model is given below:

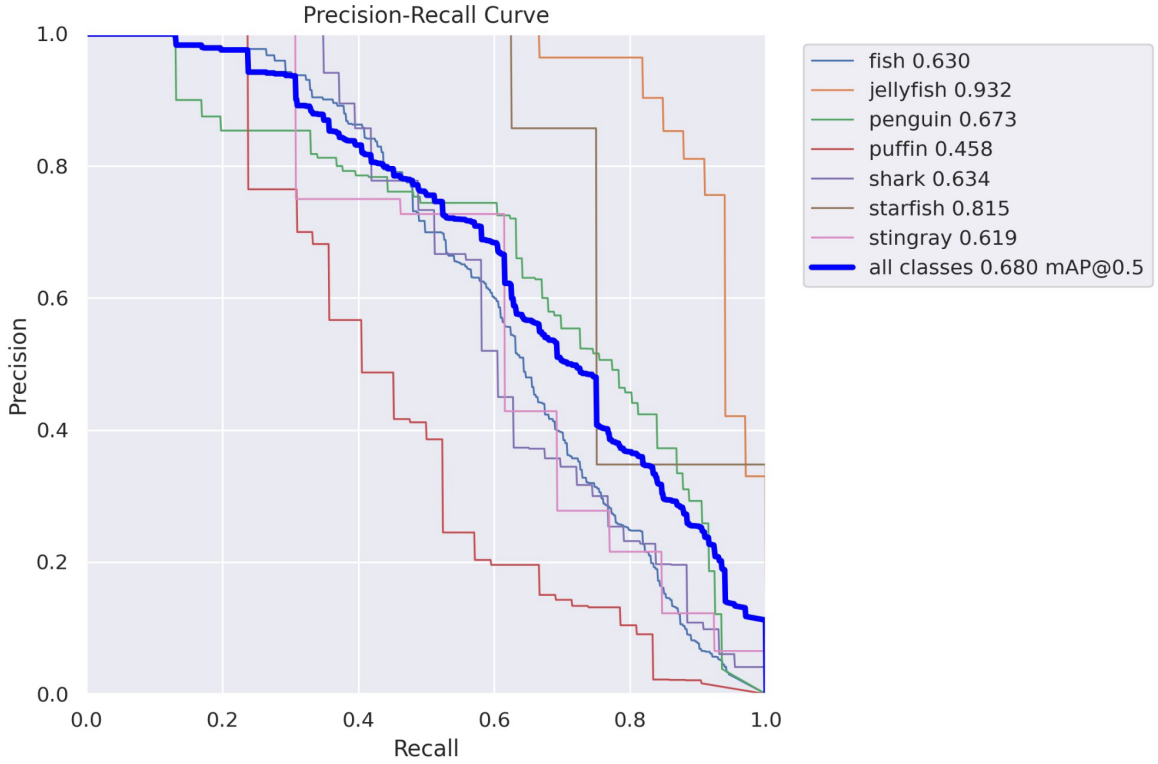


YOLOv8

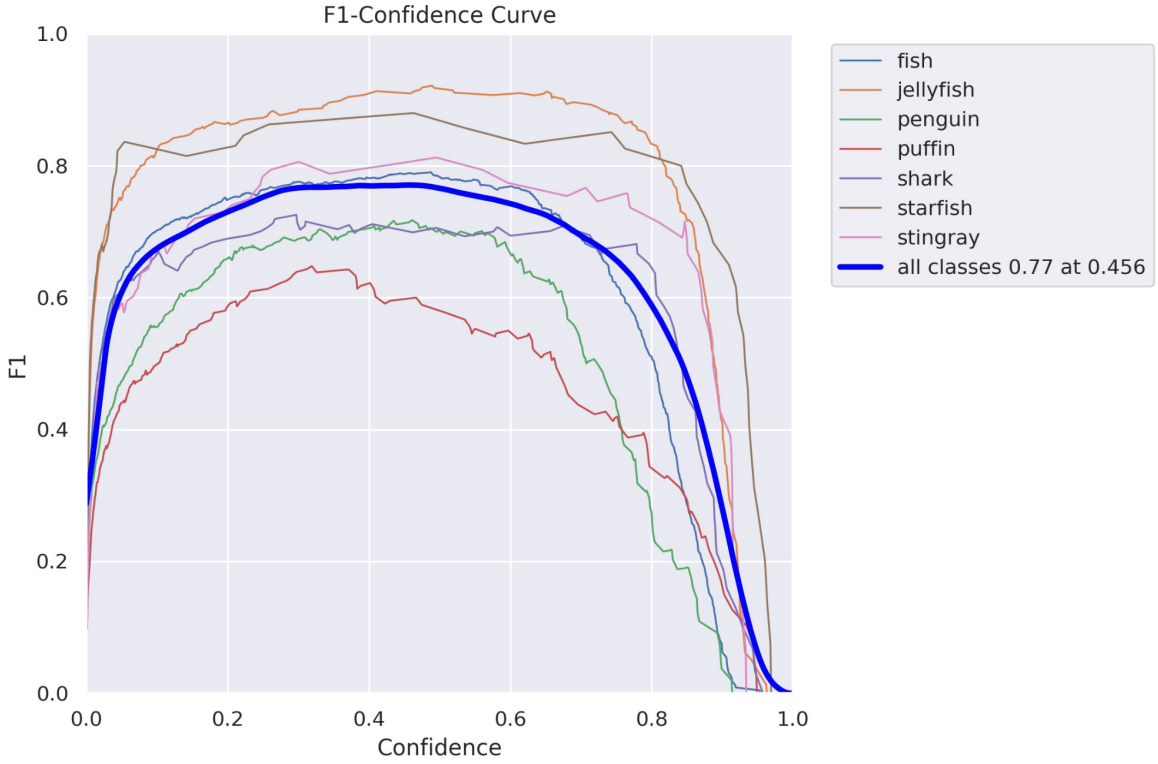
From, the precision-recall comparison the YOLOs-transformer models are performing better in than the others. It achieved 85.6% AUC for all classes for mAP@0.5 while YOLOv8 got 77.8% and DETR got 68.9%. So, from the higher AUC in precision-recall curve it can be said that in overall performance the YOLOs-transformer is performing better.



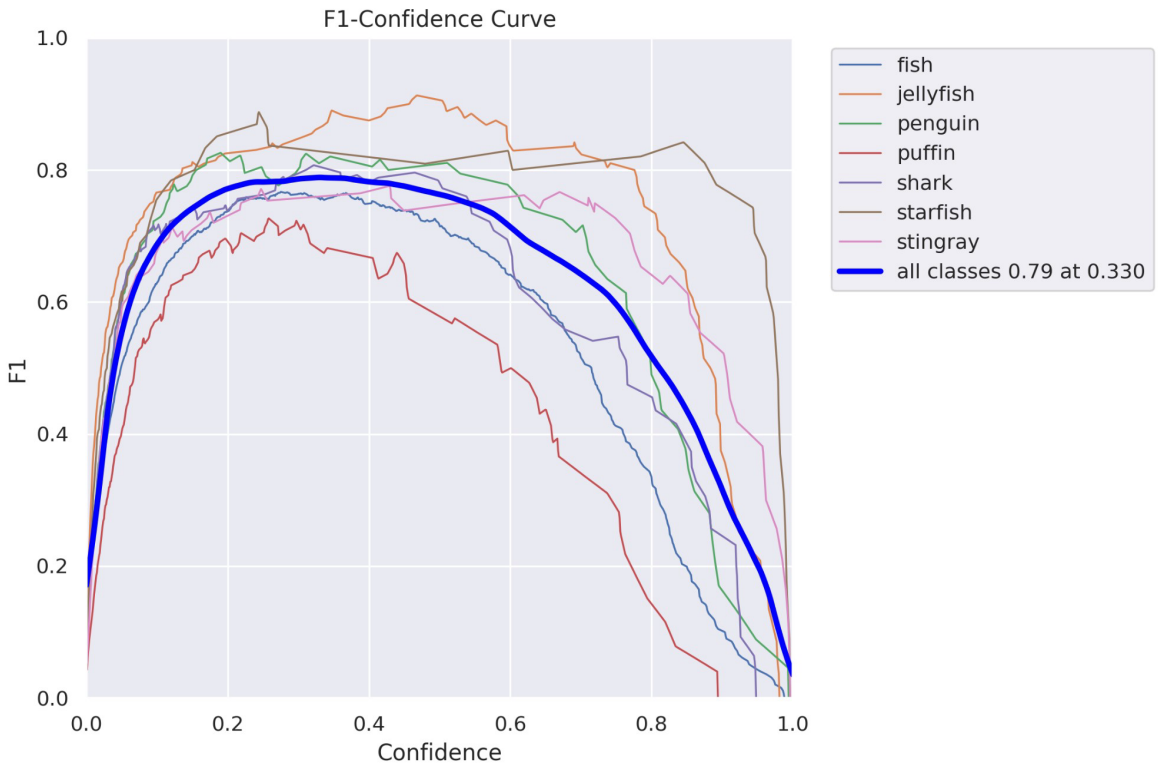
YOLOs-transformer



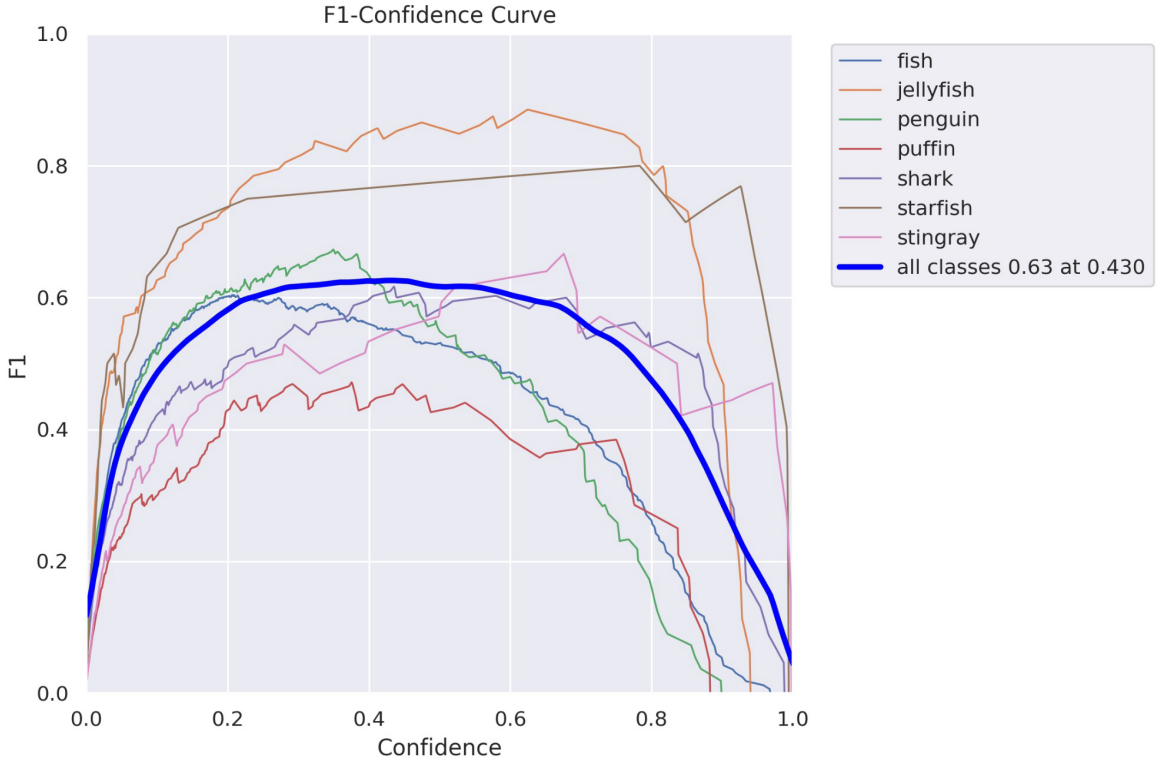
Detection Transformer (DETR) **F1-confidence curve for each model is given below:**



YOLOv8



YOLOs-transformer



Detection Transformer (DETR)

F1=(2×Precision+Recall)/(Precision x *Recall)*

<u>Insights and Observations</u>

The F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It's a useful metric when you want to balance precision and recall and need a single metric to evaluate the model's performance. From here, the YOLOs model is performing better than the other ones in a higher threshold of confidence level. Although, the YOLOv8 model is not far behind too.

At 40 epochs of training, while the YOLOv8 is getting more performance for all classes on average, so it can be said from the precision-recall and f1 score that the YOLOs model is much better optimized and can achieve much better performance than YOLOv8 with more training and optimization.