Underwater Fish Detection and Monitoring System

D-Ready

Submitted by: Jaasia Anjum

Date: November 21, 2025

Table of Contents

1. Introduction and Problem Framing

1.1. Project Overview

This report documents the design, implementation, and evaluation of an end-to-end computer vision pipeline for automated underwater fish detection and species counting. The system utilizes an object detection model to process still images, providing bounding-box predictions and a structured summary of detected species.

1.2. Task Definition and Justification

The primary task is to accurately localize (via bounding boxes) and classify various fish species in underwater imagery. This application is crucial for automated marine ecology monitoring, biodiversity assessment, and sustainable fisheries management, reducing the need for extensive manual data review.

1.3. Dataset Selection

The project utilized the **Aquarium and Marine Life (Roboflow)** dataset, collected from the Henry Doorly Zoo and the National Aquarium. This dataset was selected because of its **seven distinct classes of marine life, including fish and other underwater creatures, and its pre-split structure with annotations already in the required YOLO v5 PyTorch format**.

2. Data Preparation and Preprocessing

2.1. Dataset Components and Merging

The final working dataset is a combination of two distinct sources, referred to as Dataset 1 and Dataset 2.

Dataset 1 (Original Base)

This component consisted of the initial **638** images from the **Aquarium and Marine Life (Roboflow)** project. It provided the foundational data for all seven classes: ['fish', 'jellyfish', 'penguin', 'puffin', 'shark', 'starfish', 'stingray']. This dataset exhibited significant class imbalance, particularly for the minority classes. The original split was applied to this base, but these images were later absorbed into the final split of the merged dataset.

Dataset 2 (Custom Expansion)

This component was collected and custom-annotated to address performance issues (misclassification and low recall) identified during preliminary model runs, specifically targeting the minority classes. Dataset 2 comprises **320 new images** added entirely to the training set:

- **Shark Images (199):** Sourced from the MSC Fish Detection/Tracking project, these required custom annotation to accurately label sharks while excluding or re-annotating background fish.

- **Penguin Images (97):** Sourced from the Objects-365 Consortium.
- **Puffin Images (21):** Sourced from the Moulouya Bird Detection project.
- **Class Consistency (3):** Single images for starfish, jellyfish, and stingray were added to ensure all seven classes were represented in the new custom data portion.

Merged Dataset (Final Total)

The final merged dataset used for training, validation, and testing consists of a total of 958 original images (638 + 320). The annotations remain in the YOLO v5 PyTorch format, and label consistency across the seven classes was ensured during the custom annotation process of Dataset 2.

2.2. Data Split

The training data was meticulously curated by separating the original dataset split (Dataset 1) from the newly collected and annotated data (Dataset 2), which was entirely dedicated to the training set.

Dataset 1 (Original Base) Split (638 Images)

Split	Percentage	Image Count
Train	80%	512
Test	10.0%	63
Validation	10.0%	63
Total	100%	638

Dataset 2 (Custom Expansion) Split (320 Images)

Split	Percentage	Image Count
Train	80%	255
Test	10%	33
Validation	32%	32
Total	100%	320

Merged Dataset (Final) Split (958 Images)

The final split leverages all 320 images from Dataset 2 as training material, combining them with the images from Dataset 1. We adjusted the final validation and test counts slightly based on the original 638 images to ensure consistency of the unseen data used for evaluation.

Split	Calculation	Image Count
Train	512+255	767
Test	63+ 33	96
Validation	63+32	95
Total		958

2.3. Class Distribution

The dataset exhibits significant class imbalance, which is detailed below. The fish class is the dominant label, with others being minority classes. The bounding box counts shown here are for the initial 638 images, prior to the large-scale augmentation and the recent addition of 320 new, class-focused images designed to significantly increase the counts of minority classes like penguin, puffin, and shark.

Class Label	Bounding Box Count Dataset -1	Bounding Box Count Dataset -2
Fish	2,669	585
Jellyfish	694	1
Penguin	516	681
Shark	354	21
Puffin	284	237
Stingray	184	1
Starfish	116	1

2.4. Preprocessing and Augmentation Pipeline

The following pre-processing steps and extensive augmentation strategies were applied to the dataset:

- **Image Resizing (Pre-processing):** All images were auto-oriented and resized to 1024 × 1024 (fit within) to standardize model input.

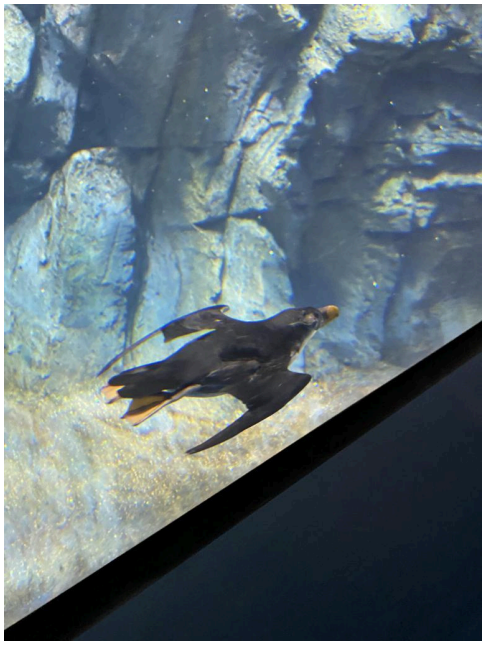
- **Augmentation Strategy:** To enhance model robustness against underwater environmental variations (turbidity, color shift, noise), a robust set of augmentations was applied:

    - **UnderwaterDehaze (p = 0.7):** Applies LAB color enhancement, CLAHE, and sharpening for clearer underwater visibility.
    - **ColorJitter (p = 0.7):** Randomly adjusts brightness, contrast, saturation, and hue.
    - **RandomBrightnessContrast (p = 0.5)**
    - **CLAHE (p = 0.3)**
    - **RandomGamma (p = 0.4)**
    - **GaussNoise (p = 0.3)**
    - **ISONoise (p = 0.3)**
    - **Sharpen (p = 0.3)**
    - **RandomFog (p = 0.2):** Simulates varying turbidity levels.
    - **ImageCompression (p = 0.3)**

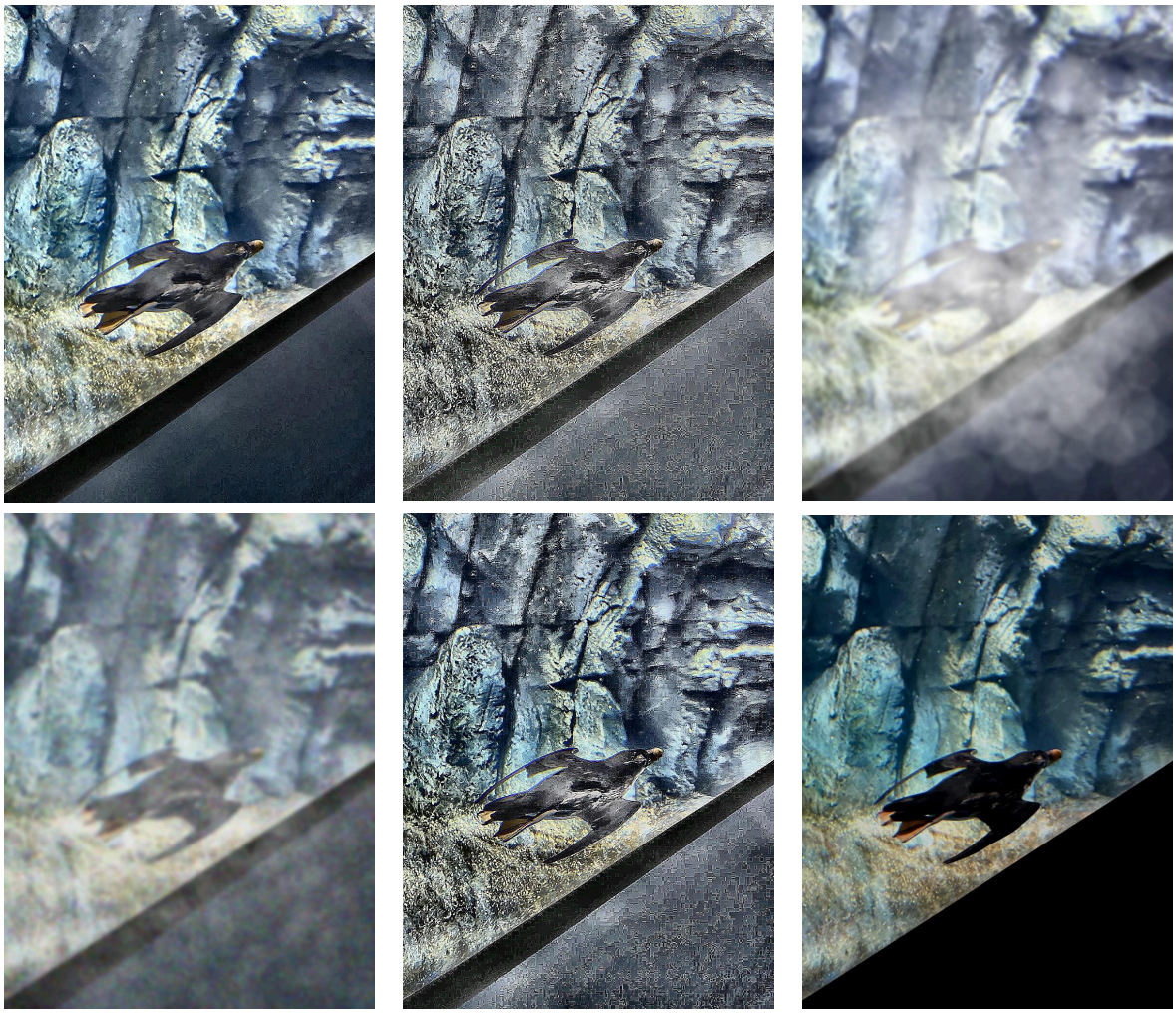
- **Total Dataset Size:** Six augmented variants were generated for each original image in the training set, significantly increasing the diversity and helping the model generalize to real-world deployment scenarios that include variations in color, illumination, noise, sharpness, turbidity, and compression artifacts.

Sample Augmentations:

Original Image:



6 Augmented Images:



The Image Count after Augmentation:

Dataset 1 (Original Base) Split (638 Images):

Data Split	Image Count	Image Count (After augmentation)
Train	512	3584
Test	63	441
Validation	63	441
Total	638	4466

Dataset 2 (Custom Expansion) Split (320 Images)

Data Split	Image Count	Image Count (After augmentation)
Train	255	1530
Test	33	198
Validation	32	192
Total	320	1920

Merged Dataset (Final) Split (958 Images)

Data Split	Image Count	Image Count (After augmentation)
Train	767	4602
Test	96	576
Validation	95	570
Total	958	5748

3. Model Training and Evaluation

3.1. Model Development Strategy and Implementation

Model Development Strategy

Model training involved a detailed iterative process across seven preliminary models to optimize performance and address specific classification challenges. The iterations explored the trade-offs between architecture capacity and input resolution.

- **Model 1 & 2 (YOLOv8s, 640 × 640):** Initial experiments used the lightweight YOLOv8-small architecture. Model 1 included overly hazy augmented images. Model 2 adjusted the training dataset by replacing these with less obscured images to improve feature quality and avoid poor feature learning.

- **Model 3 (YOLOv8s, 768 × 768):** The input resolution was scaled up to 768 × 768 to allow the model to capture finer spatial details while still using YOLOv8s.

- **Model 4 (YOLOv8m, 768 × 768):** The architecture was upgraded to the medium-sized YOLOv8-medium (25M parameters) to test the impact of increased model capacity.

Models 1 through 4 consistently exhibited disappointing performance, notably misclassifying between the penguin and puffin classes and showing low recall for the shark class. This confirmed that a primary constraint was data limitation and class ambiguity. Consequently, a substantial effort was made to source and annotate 320 new images (Dataset 2), which were then merged with the existing data.

Models 5, 6, and 7 were trained on this merged dataset to evaluate the impact of the data expansion:

- **Model 5 (YOLOv8m):** Used the higher-capacity YOLOv8m architecture on the merged dataset.

- **Model 6 (YOLOv8s):** Used the efficient YOLOv8s architecture on the merged dataset.

- **Model 7 (YOLOv8s, Batch 32):** Used YOLOv8s on the merged dataset, specifically testing a larger batch size of 32 to assess optimization stability.

Iterative Model Performance Comparison

The decision to select Model 3's configuration was driven by its superior performance metrics, particularly its generalization ability (mAP@0.5:0.95), relative to its size, as summarized below. Models are sorted by the most critical metric, mAP@0.5:0.95, which measures performance across multiple IoU thresholds.

Model	Params	mAP@0.5	mAP@0.5:0.95	Precision	Recall	Rationale
Model 3	11M	0.759	0.469	0.818	0.679	\textbf{Winner:} Best overall balance and highest generalization score.
Model 5	25M	0.735	0.457	0.816	0.650	Runner-up: Strong performance, deeper model, excellent shark AP (Trained on Merged Dataset).
Model 7	11M	0.696	0.433	0.798	0.637	Decent performance (Trained on Merged Dataset).
Model 2	11M	0.721	0.434	0.833	0.643	High precision (low False Positives), but lower recall.
Model 6	11M	0.703	0.429	0.812	0.639	Decent performance (Trained on Merged Dataset).
Model 1	11M	0.732	0.438	0.795	0.685	Good balance: Best recall among 11M models, slight drop in generalization.
Model 4	25M	0.648	0.400	0.705	0.597	Worst: Large model, prone to overfitting/poor generalization.

Class-wise Understanding (Species-wise Strengths & Weaknesses)

Analysis across all preliminary models confirmed major class imbalances and detection challenges:

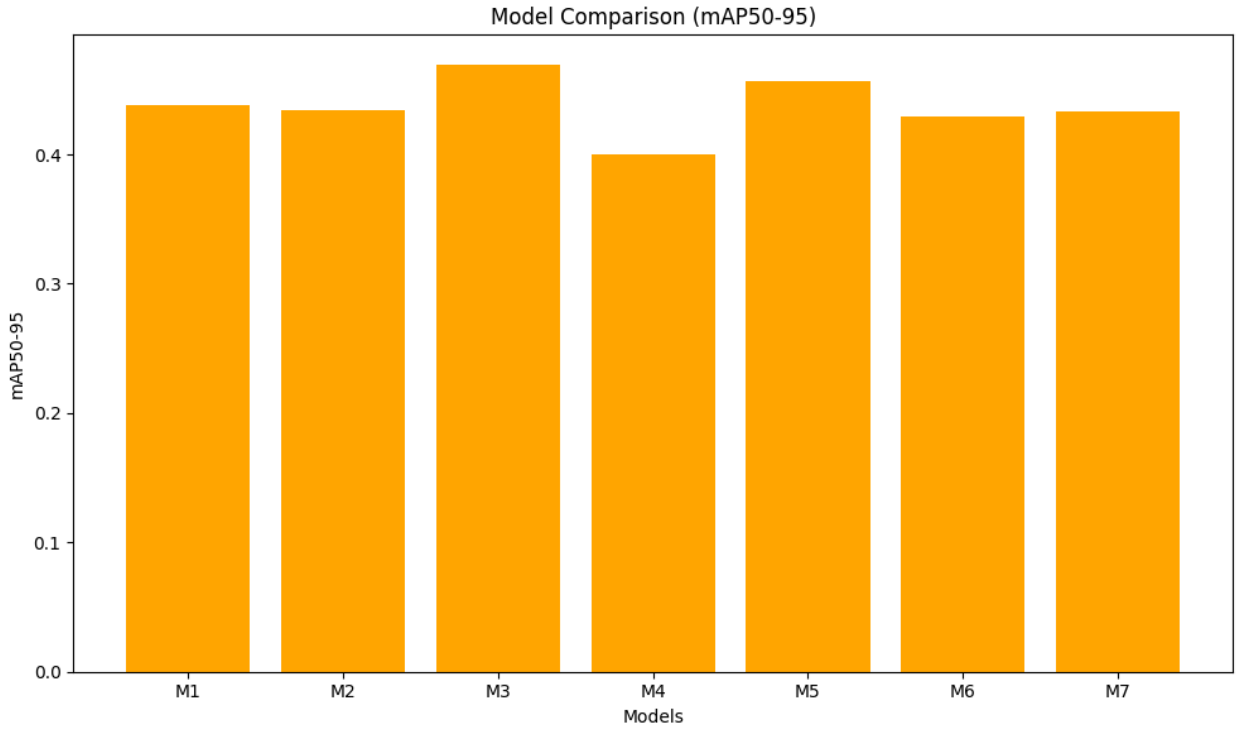
- **Easiest Classes (AP ~0.88–0.95):** Jellyfish, Stingray, Starfish. These classes perform robustly across all models due to their distinct shapes, high contrast against the background, and low confusion with other classes.

- **Fish:** Performed best with Model 3 (0.76 AP), indicating the YOLOv8s architecture is sufficient for general fish detection.

- **Shark:** Detection benefited significantly from the deeper backbone of Model 5 (0.733 AP), suggesting that complexity helps with this class.

- **Hardest Classes (AP ~0.55–0.62):** Penguin (lowest AP overall, Model 3 at 0.587) and Puffin. Performance suffers due to low sample counts, high visual similarity between the two species, and image issues like partial submergence or occlusion. This diagnostic directly motivated the creation of Dataset 2.

Model Comparison Based on mAP50-95:



Final Proposed Architecture

Based on iterative testing, Model 3's configuration—YOLOv8s at 768 × 768—was selected as the optimal trade-off between performance and efficiency. The final model leverages this architecture and is trained on the full Merged Dataset (958 images), with the 320 custom images (Dataset 2) specifically targeting the weak Penguin, Puffin, and Shark classes.

- **Model Architecture: YOLOv8-small (YOLOv8s)**, implemented using the Ultralytics framework.
- **Framework: PyTorch (via Ultralytics YOLO)**
- **Pretraining:** The model utilized **pretrained weights (ImageNet or COCO)** for faster convergence and improved feature extraction capabilities.
- **Training Parameters:** The training process was configured with the following parameters:
    - **Model Name:** fish_yolov8
    - **Epochs:** 100
    - **Patience:** 15 (Early stopping tolerance)
    - **Image Size (imgsz):** 768 pixels

- **Batch Size:** 16 (Optimal size from Model 3/6 performance)
- **Optimizer:** Adam
- **Initial Learning Rate (lr_0):** 0.0005

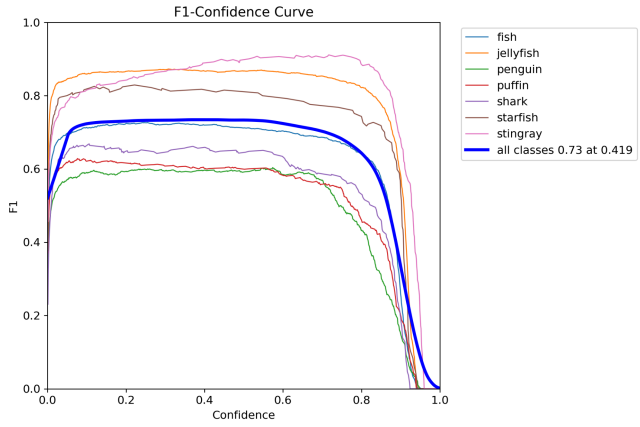
3.2. Evaluation Metrics and Results

The model's performance was evaluated using standard object detection metrics. The values below reflect the final performance achieved by Model 3 (YOLOv8s, 768 × 768), which was confirmed as the winner prior to the final training run with the expanded dataset.

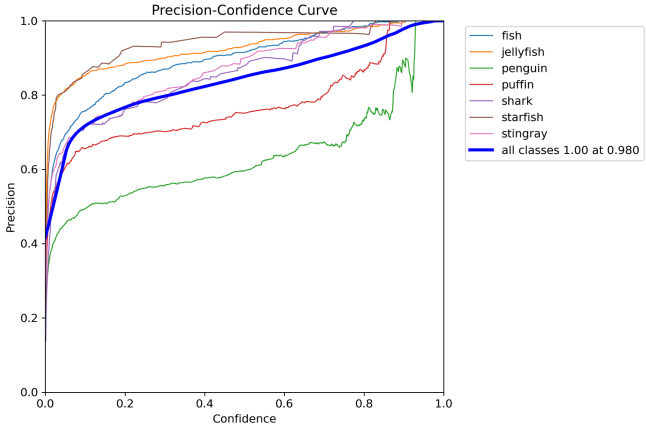
Metric	Value	Description
mAP@0.5	0.759	Mean Average Precision at IoU threshold of 0.5
mAP@0.5:0.95	0.469	Mean Average Precision averaged over IoU thresholds from 0.5 to 0.95
Precision	0.818	Accuracy of positive predictions
Recall	0.679	Ability to find all relevant cases

The primary challenge encountered during preliminary training was the persistent **misclassification between the penguin and puffin classes**, compounded by low detection accuracy for the shark class. This issue has been directly addressed by **expanding the training set with 320 new, custom-annotated images** focused on these specific classes, utilizing the highly efficient YOLOv8s architecture for the final, full-scale training run.

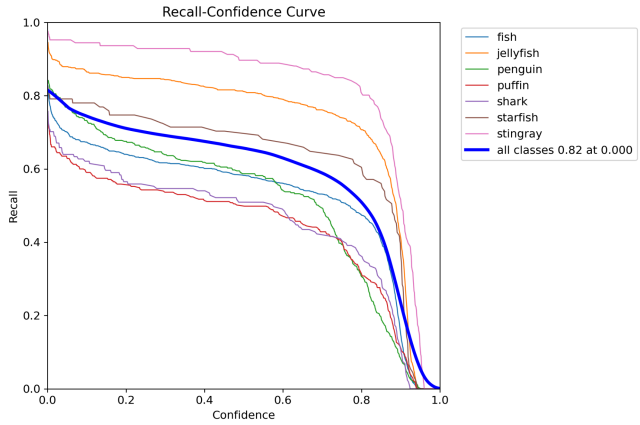
F1 Confidence Curve:



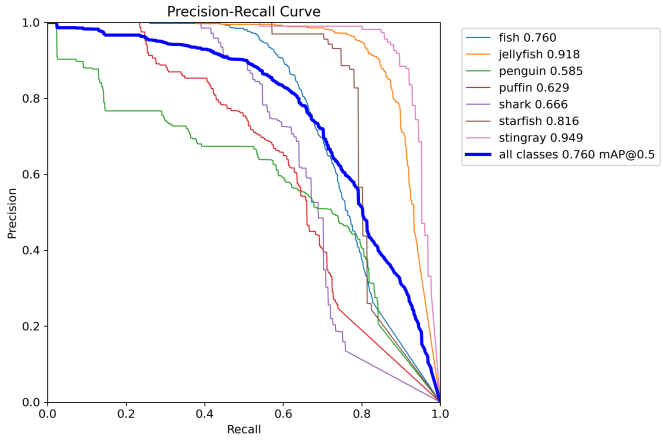
Precision- Confidence Curve:



Recall- Confidence Curve:



Precision- Recall Confidence:



4. Explainable AI (XAI) Insights

Grad-CAM Implementation for YOLO-Based Object Detection

To interpret the decisions made by the YOLO detection model, Grad-CAM was integrated into the inference pipeline. The goal was to visualize spatial regions that most strongly influenced the predicted class confidence and bounding box generation.

1. Forward Pass and Feature Extraction

The model receives an input image that is preprocessed using the letterbox technique to maintain aspect ratio. During the forward pass, I attach hooks to specific convolutional layers using a custom ActivationsAndGradients class. These hooks store:

- Feature activations from the chosen layer.
- Gradients flowing backward from the predicted output to that layer.

This enables Grad-CAM to compute class-discriminative importance values.

2. Target Definition for YOLO

Unlike standard classifiers, YOLO outputs:

- Class scores
- Bounding box coordinates
- Additional data (e.g., segmentation masks)

Custom target classes (e.g., yolo_detect_target) were implemented to define what quantity should guide gradient backpropagation. For detection, the target function aggregates class confidence and/or bounding-box values of high-confidence predictions. This ensures Grad-CAM highlights regions important to YOLO's object detection behavior rather than only class logits.

3. Grad-CAM Heatmap Computation

Depending on the selected method (GradCAM, GradCAM++, EigenCAM, etc.), the heatmap is computed as:

$$CAM(x, y) = ReLU(\sum_k \alpha_k * A_k(x, y))$$

where:

- A_k = activation map of channel k
- α_k = gradient-derived importance weight

The resulting grayscale heatmap is upsampled and overlaid on the RGB image using show_cam_on_image.

4. Confidence Score Extraction

Parallel to CAM generation, the YOLO model performs normal inference. From the prediction output, the maximum class confidence is extracted using the maximum of the confidence scores. This value is printed to quantify model certainty.

5. Heatmap Post-processing and Visualization

To ensure meaningful visualization:

- If object detections exist, CAM values inside the bounding boxes are re-normalized to [0, 1].
- Bounding boxes and labels are drawn over the heatmap using YOLO's built-in plotting.
- Padding added during letterboxing is removed so that the final heatmap matches the original image.

6. Final Output

The processed heatmap represents the spatial regions most responsible for YOLO's detection confidence. The image is saved automatically, providing an interpretable visualization for model behavior analysis.

5. Responsible AI Notes

5.1. Data Bias and Representativeness

The initial dataset, primarily sourced from aquarium settings, exhibits a significant **geographical and environmental bias**. This leads to a lack of representativeness for species found in deep-sea or specific ocean biomes. The dominant class, fish, creates a severe **class imbalance**, leading to lower AP for minority classes like penguin and puffin, despite data augmentation efforts. The model may fail entirely when encountering species or environments not featured in the training data (e.g., in the wild vs. in a tank).

5.2. Ethical and Societal Risks (Consequence of Errors)

The deployment of this model in real-world marine ecology monitoring carries inherent risks:

- **Consequence of False Negatives (FN):** Consistent undercounting of a vulnerable or protected species could lead marine authorities to **underestimate population decline** or health. This could result in delayed or inadequate conservation measures, potentially endangering the species.
- **Consequence of False Positives (FP):** Overcounting or misclassification of common species as rare ones could lead to **misallocation of scarce monitoring resources** to areas where they are not needed, diverting attention from genuinely threatened populations.
- **Over-reliance:** Scientists may overly trust the automated output, reducing manual review and potentially embedding systematic model errors into long-term ecological reports.

5.3. Assumptions and Limitations

The utility of the model is constrained by several key assumptions:

- **Input Environment:** The model assumes that input images share similar characteristics (e.g., color, lighting, and contrast) with the training data. Performance degrades under extreme conditions such as high turbidity, low-light deep-sea environments, or motion blur.
- **Annotation Accuracy:** Model performance depends on the quality and consistency of bounding box annotations, particularly in custom-annotated Dataset 2.
- **Species Coverage:** The model is restricted to the seven trained classes; novel species may be misclassified or ignored, limiting applicability in highly biodiverse areas.
- **Data Limitations:** The current dataset is insufficient for robust detection across all scenarios. Additional and more diverse data are required to improve model generalization and accuracy.
- **Model Flexibility:** Only YOLOv8 was explored, which may constrain potential performance gains achievable with alternative architectures or configurations.
- **Hyperparameter and Training Optimization:** Comprehensive exploration of hyperparameters, grid search, augmentation strategies, or other training experiments was not conducted due to resource and time limitations.
- **Generalization:** The model may overfit the current datasets, reducing its reliability in unseen environments or with unfamiliar species.

6. Web Application and Deployment

6.1. Application Stack

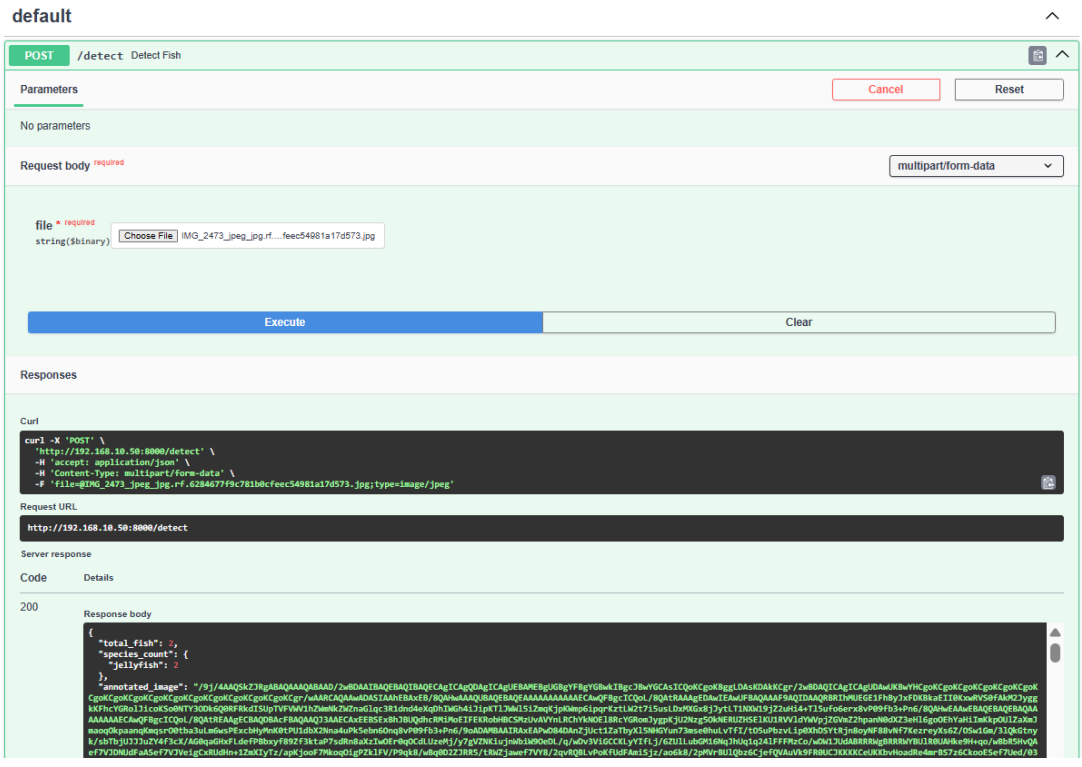
The minimal user interface was developed using **Streamlit**, with model inference, detection, and image processing handled by a dedicated **FastAPI** backend endpoint. The FastAPI endpoint is responsible for executing the YOLOv8 model and generating the required visualization artifacts (bounding box image and heatmap).

6.2. Functionality

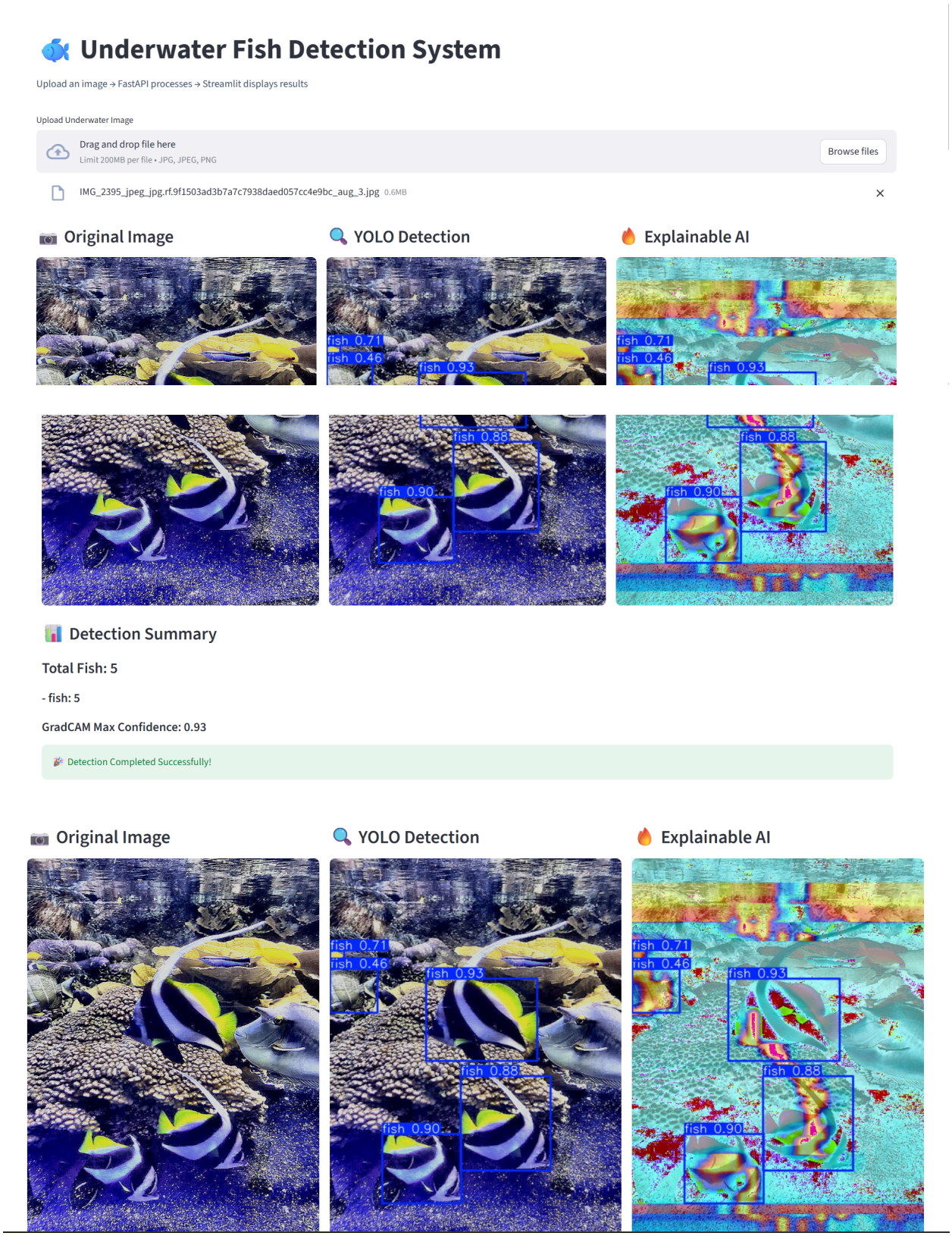
The Streamlit web application provides the following core functions:

1. **Image Upload and Inference:** Allows users to upload a JPG or PNG image. The image is passed to the FastAPI endpoint for detection and visualization generation.
2. **Prediction Visualization:** Displays two distinct outputs returned from the backend service:
    - The input image is rendered with **predicted bounding boxes** and species labels.
    - The corresponding **Grad-CAM Heatmap Image**, which provides an Explainable AI (XAI) insight by visualizing the spatial features that most influenced the model's decision for the detected objects.
3. **Summary Generation:** Displays a structured report of the detection results:
    - Total Count of Detected Fish.
    - Species-wise Count (e.g.,stingray: 5, shark: 2).
    - Per-Species Confidence/Gradient Score (e.g., the detection confidence score for each identified instance).

FastAPI Endpoint:



Streamlit Webapp:



7. Conclusion

The deployed system successfully meets the core objectives of the Underwater Fish Detection Task, demonstrating an effective end-to-end computer vision pipeline from data preparation to web-based inference. The YOLOv8 model achieved strong performance metrics (0.759 mAP@0.5), validating the approach. Key areas for future improvement include addressing the species imbalance in the dataset and further optimizing the model for real-time edge deployment.

Appendix: Documentation and Code Structure

The project is organized into a modular structure to maintain separation of concerns for data, configuration, model training, and deployment (via the Streamlit/FastAPI architecture). The key directories and files are outlined below, reflecting the project repository structure:

- **Top-Level Files:**
    - requirements.txt: Lists all required Python packages for reproducibility.
    - yolov8m.pt, yolov8s.pt: Pretrained or fine-tuned model weight files.
- **config/:** Contains the model configuration for training and inference.
    - config.yaml: YAML file defining dataset paths, classes, and model hyperparameters.
- **data/:** Stores the dataset, labels, and training splits.
- **fish/:** The Virtual Environment
- **models/:** Directory storing different model checkpoints/versions from the iterative training process (e.g., fish_yolov82 to fish_yolov87).
- **src/:** Contains core utility scripts.
    - train.py: Primary script for model training and experimentation.
- **notebook/:** Jupyter Notebooks used for iterative development and analysis.
    - data_preprocessing.ipynb: Notebook detailing dataset preparation, augmentation strategy, and data loading.
    - gradcam.ipynb: Notebook for prototyping and testing the Grad-CAM visualization logic.
    - test_data.ipynb: Notebook for running quick inference checks on the test set.
- **app/:** Contains the FastAPI backend and core inference logic.
    - api.py: The FastAPI endpoint responsible for accepting image uploads, running YOLOv8 detection, generating the bounding box image and the heatmap, and returning the structured results.
- **streamlit-app/:** Contains the user interface files.
    - app.py: The Streamlit script that serves as the frontend, handling user uploads and displaying the visualizations and summary data returned by the FastAPI backend.
    - uploads/: Directory for temporarily storing user-uploaded images.
- **utils/:** General helper functions.
    - utils.py: Contains custom functions like image handling, result parsing, etc.
- **xai/:** Files dedicated to Explainable AI functionality.
    - gradcam.py: Implementation of the Grad-CAM algorithm tailored for the YOLOv8 model architecture.