King Abdulaziz University

Faculty of Computing and Information Technology

Computer Science Department

CPCS – 432

Fish Species Classification,

Segmentation and Detection

Group Members:

Name	ID	Section
Talah Rabea Faloudah	2206666	CAR
Samar Mishal Alamri	2206831	EAR
Jumanah Jamal Banabilah	2206898	EAR
Rana Ahmed AlZahrani	2105077	EAR

Instructor: Dr. Arwa Basbrain

Submission date: 15-12-2025

Work Distribution:

Name	Used Dataset	Models
Talah Rabea Faloudah	A Large-Scale Fish Dataset	LBP(ANN+KNN+SVM) HuMoments (ANN+KNN+SVM) CNN Models1 Transfer Learning (InceptionV3) Object Detection (YOLOV8n, RT-DETR, YOLOV8n Model 1)
Samar Mishal Alamri		GLCM (ANN+KNN+SVM) HOG (ANN+KNN+SVM) CNN Model 2 Transfer Learning (MobileNetV2) Instance Segmentation (U-net , DeepLabV, FCN)
Jumanah Jamal Banabilah	Fish Recognition Ground-Truth Dataset + Aquarium Dataset	HOG(ANN+KNN+SVM) ORB(ANN+KNN+SVM) CNN Model3 Transfer Learning (ResNet18) Object Detection(YOLOv5, YOLOv11, YOLOV11update1)
Rana Ahmed AlZahrani	Fish Recognition Ground-Truth Dataset	LBP(ANN+KNN+SVM) GLCM(ANN+KNN+SVM) CNN Model4 Transfer Learning (EfficientNet-B0) Object segmentation (YOLOv8, Mask R-CNN, Custom SegNet)

Table of Contents

1 Introduction

The proposed project aims at the design and implementation of a comprehensive computer vision approach that can automatically perform three basic tasks concerning aquatic analysis, namely Classification, Detection, and Segmentation. By performing these tasks, the proposed approach will work towards achieving accurate identification of fish species, accurate localization of fish within larger images, and pixel-wise segmentation of fish shapes.

This project is of particular relevance to the computer vision industry, given that it deals with the issue of fish image processing. This is because the current monitoring and inventorying of species that take place on the sea or markets is often carried out manually, which is time-consuming and often rendered inaccurate. This project, on the other hand, aims to replace this process with a fast and accurate AI system that will allow the monitoring, population estimation, and measurement of sizes of species for both marine biologists and the private industry.

For these purposes, the project proposes a methodology that works together to implement and evaluate a wide range of models, from traditional approaches to deep learning-based approaches. Overall, the aim is to explore the best model combination for the development of a reliable automated analysis system for the marine environment.

1.1 Contribution to the project

Key technical contributions of this project include:

1. Extract Bounding Boxes from Masks (GT) For A Large-Scale Fish Dataset
2. The segmentation task involved adjustments to the U-Net layers for optimal binary mask creation, along with modifications to the SegNet architecture that leverage encoder–decoder layers and pooling indices for efficient spatial reconstruction.
3. The object detection task involved adjusting the depth of the YOLO backbone layers to enhance feature extraction.

2 Dataset Analysis

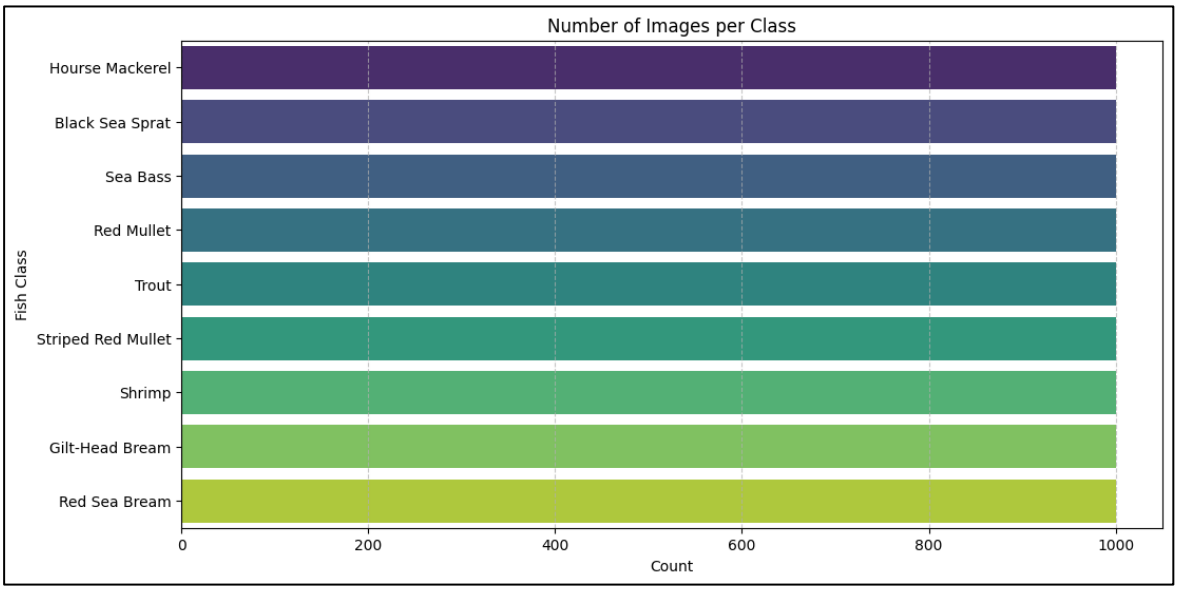
2.1 Used Datasets

Dataset	A Large-Scale Fish Dataset	Fish Recognition Ground-Truth Dataset	Aquarium Dataset
Author	O. Ulucan, D. Karakaya, M. Turkan (Izmir University of Economics)	B. J. Boom, P. X. Huang, et al.	Brad Dwyer
Source	Kaggle, originally published in 2020 ASYU Conference	Kaggle, originally from Fish4Knowledge Project	Roboflow Universe
Total Samples	9k fish images	27,370 fish images	638 fish images
Data Format	PNG, RGB	PNG, RGB	JPG, RGB
Number of Classes	9	23	7
Class Balance	Balanced (1k per class)	Highly Imbalanced	Moderate Imbalanced
Collection Methodology	Collected from a supermarket in Izmir, Turkey using two different cameras	Extracted from live underwater video surveillance	Collected from two US aquariums (Henry Doorly Zoo, National Aquarium)
Preprocess Methods	Resized to 590x445 and augmented via flipping and rotating	Clustered based on visual features and manually verified by marine biologists	Auto-Orientation and Resizing
License	CC BY 4.0	MIT License	CC BY 4.0

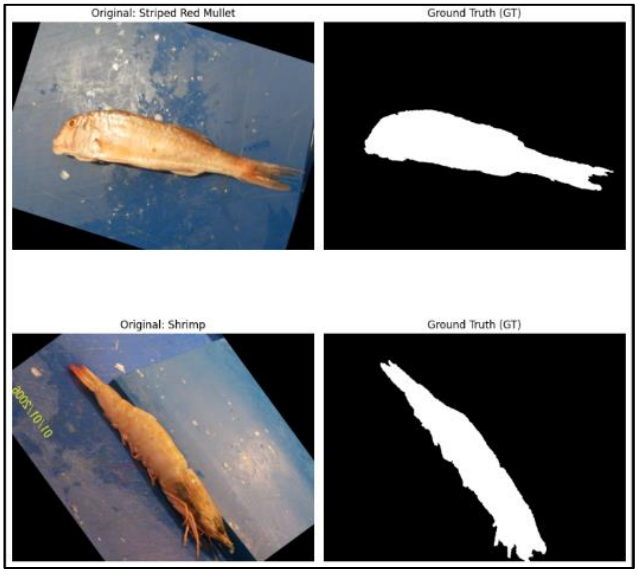
2.1.1 A Large-Scale Fish Dataset

Visualizations:

The figure presents a bar chart illustrating all 9 bars with identical height. This confirms that the dataset is perfectly balanced, which ensures the model will not be biased toward any particular class during training.



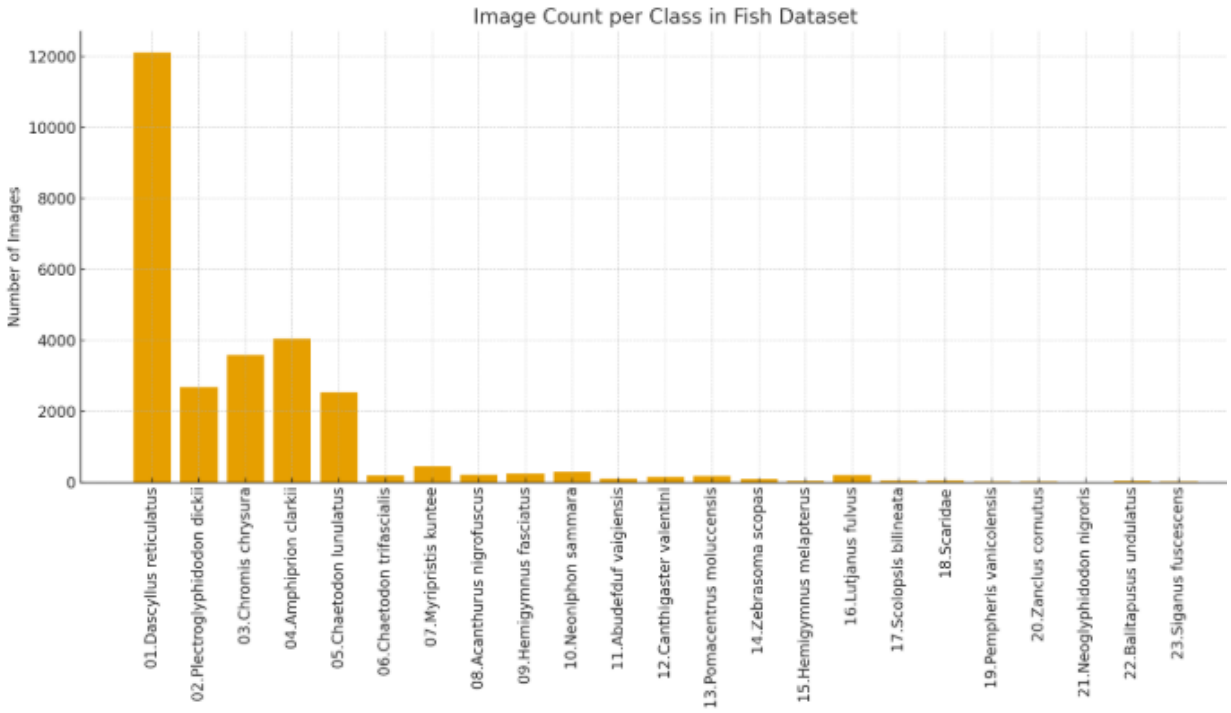
The second figure represent samples from the dataset. The original images with the labels for classification and GT masks for binary segmentation.



2.1.2 Fish Recognition Ground-Truth Dataset

Visualizations:

The figure below shows the dataset distribution, which is highly imbalanced. In the second figure we can see some mask\images samples from the dataset.



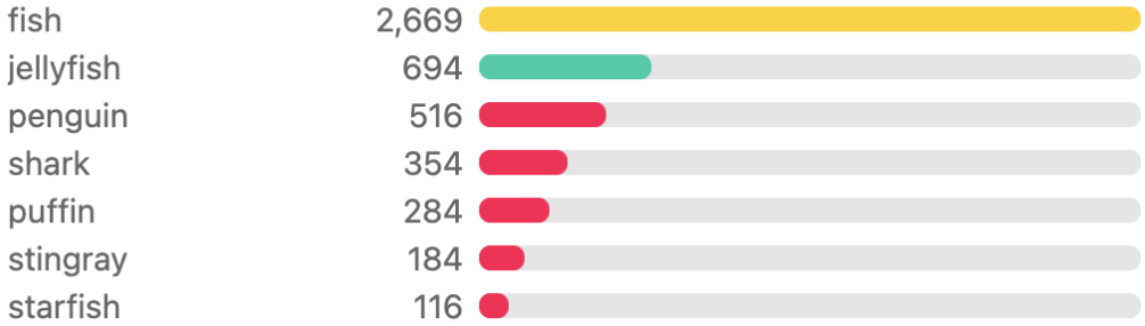
fish_03

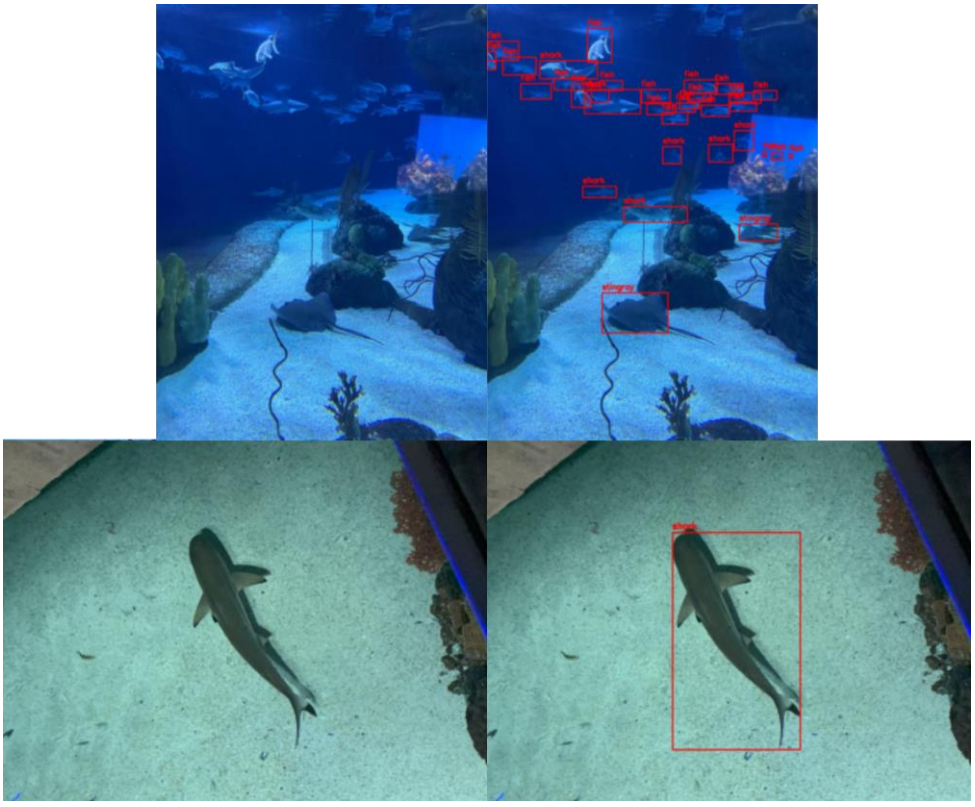
fish_04

2.1.3 Aquarium Dataset

Visualizations:

The dataset is moderately imbalanced as shown in the figure below. The class sizes remain reasonable, ranging from 2669 images (fish) to 116 images (starfish). Because the dataset already includes sufficient natural variation, augmentation is unnecessary and could distort the data or even lead to overfitting. The second figure shows some samples of the dataset with bounding boxes.





3 Image Preprocessing and Augmentation

3.1 Techniques Applied

3.1.1 A Large-Scale Fish Dataset

3.1.1.1 Classification

1.Image Resizing: All images were resized to 128×128 to ensure consistent input dimensions and reduce computational cost.

2. Normalization: Extracted feature vectors were standardized using StandardScaler, while images used in CNN models were normalized by scaling pixel values to the range [0, 1] for more stable and efficient training.

3.1.1.2 Instance Segmentation

1. Image Resizing: Both the input images and the ground truth masks were resized to a fixed dimension of 128x128 pixels for consistent input dimensions and reduce computational cost.

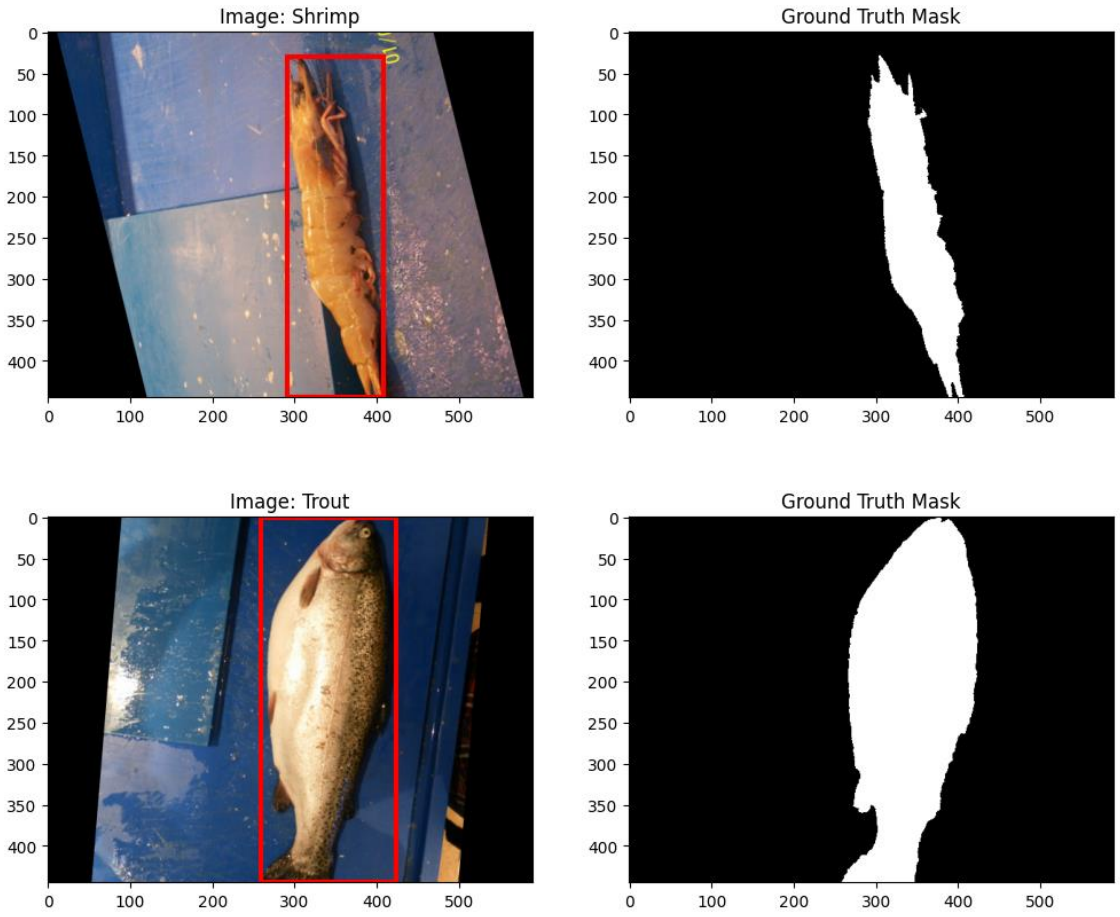
2. Normalization: The image pixel values were scaled to **[0, 1]** by dividing by 255.0 within the dataset class.

3.1.1.3 Object Detection

1.Extract Bounding Boxes from Masks (GT)

Since my dataset was mainly designed for classification and segmentation, it did not contain any bounding-box annotations. To generate these annotations, I implemented a mask-based bounding-box extraction approach. For each image, I briefly read its corresponding segmentation mask and identified all non-zero pixels (mask > 0). Using the minimum and maximum x/y coordinates of these pixels, I computed the bounding box that tightly encloses the object. I then saved the image path, mask path, and extracted bounding-box coordinates into a DataFrame.

This method allowed me to automatically convert segmentation masks into usable bounding-box metadata for object detection tasks.



2.Image Resizing: All images were resized to 640×640 to match the required input resolution and improve training efficiency.

3.Normalization:

Pixel values were normalized internally by the model pipeline to stabilize training and speed up convergence.

3.1.2 Fish Recognition Ground-Truth Dataset

1. Class Selection and Dataset Rebalancing: The Fish Recognition Ground-Truth dataset was found to be highly imbalanced, with some classes containing a very large number of images (e.g., Dascyllus reticulatus with 12,112 images), while others had extremely few samples (e.g., Neoglyphidodon nigroris with only 16 images). Such severe imbalance can negatively impact model training and bias predictions toward majority classes. To address this issue, a subset of six classes (fish_01, fish_02, fish_03, fish_04, fish_05, fish_07) were selected, each with a controlled number of samples (2,500 images per class), ensuring a more balanced and reliable training set.

2. Data Augmentation for Minority Classes: After class selection, data augmentation techniques were applied to increase the number of samples in smaller classes, raising them to 1,000 images. This step enhanced dataset balance and diversity while reducing overfitting and improving the generalization capability of the trained models.

3.1.2.1 Classification

1. Image Resizing: All images were resized to 128×128 to ensure a consistent input shape, allowing the models to process the images uniformly and preventing dimensional mismatches during training.

2. Normalization: Most models operate directly on raw pixel values (0–255). In some models like HOG performs internal normalization to reduce illumination effects. ResNet-18 model uses rescale=1./255, ensuring that all images are scaled to the 0–1 range.

3.1.2.2 Instance Segmentation

1. Image Resizing: All input images and ground-truth masks were resized to a fixed resolution to ensure a consistent input shape for all segmentation models and to reduce computational load during training.

2. Image Normalization: Pixel values were normalized using standard mean and standard deviation values to stabilize network training, improve convergence speed, and enhance feature learning.

3. Data Augmentation: Random horizontal flips and geometric transformations were applied to increase dataset variability, minimize overfitting, and improve model generalization to unseen images.

3.1.3 Aquarium Dataset

3.1.3.1 Object Detection

1. Image Resizing: YOLO models internally resize all images to a fixed resolution (**640×640**). This resizing ensures consistent input dimensions and reduces computational load.

2. Normalization: Normalization is handled automatically inside YOLO, where pixel intensities are rescaled from **0–255** to **0–1**. To prevents unstable gradients and keeps training stable.

4 Experimental Setup

4.1 Environment setup

Samar's Environment setup:

Hardware:

- 12th Gen Intel(R) Core(TM) i5-12400F
- NVIDIA GeForce RTX 3060 Ti
- 16.0 GB RAM
- 1 TB SSD storage

Software:

- Visual Studio
- Jupyter Notebook
- Python 3.11

Libraries:

- TensorFlow / Keras
- PyTorch
- Torchvision
- OpenCV (cv2)
- Scikit-Image (skimage)
- Scikit-Learn (sklearn)
- NumPy
- Matplotlib
- Seaborn
- tqdm
- os / glob

Talah's Environment setup:

Hardware:

- ASUS Zenbook UX3402ZA laptop
- Intel Core i7-1260P (12th Gen)
- Intel Iris Xe Graphics
- 16 GB RAM
- 1 TB SSD storage

Software:

- Google Colab
- Python 3
- Google Drive for storing Google Colab notebooks

Libraries:

- NumPy
- Pandas
- os
- Matplotlib
- Seaborn
- scikit-learn
- cv2
- scikit-image
- TensorFlow
- Keras

Jumanah's Environment setup:

Hardware:

- hp laptop
- Intel Xeon
- GPU
- CPU: 11th Gen Intel(R) Core(TM) i7
- Total Disk Space: 475.62 GB
- RAM: 15.76 GB

Software:

- Google collab
- Visual studio code
- Python 3.10.9
- Google drive

Libraries:

- TensorFlow\ keras
- PyTorch
- NumPy
- Ultralytics YOLO
- os/glob
- Matplotlib
- Seaborn
- scikit-learn
- shutil
- Pandas
- Cv2

Rana's Environment setup:

Hardware:

- Lenovo IdeaPad Slim 3 15IRH8 laptop
- 13th Gen Intel Core i7-13620H
- Intel UHD Graphics
- 16 GB RAM
- 1 TB SSD storage

Software:

- Google Colab
- Python 3
- Google Drive

Libraries:

- OS
- random
- numpy
- pandas
- cv2
- PIL
- Matplotlib
- Torch
- Torchvision
- Timm
- Sklearn
- Ultralytics
- detectron2
- seaborn
- tensorflow
- skimage

4.2 Model Architectures

4.2.1 A Large-Scale Fish Dataset

Model	Architecture Modification	Justification
ANN Model 1	Consists of two fully connected hidden layers (128 and 64 neurons) preceded by Batch Normalization and ReLU activation, followed by a Softmax output layer for multi-class classification.	Batch Normalization improves training stability and convergence. The two fully connected layers provide sufficient non-linear modeling capacity while limiting overfitting.
ANN Model 2	Consists of three fully connected hidden layers (256, 128, 64 neurons) arranged in a funnel architecture, followed by a Softmax output layer for classification.	progressively compress high-dimensional feature vectors. This balances depth for non-linear learning with a narrowing width to force feature abstraction and reduce overfitting.
CNN Model 1	Consists of two convolutional layers (32 and 64 filters) with ReLU activation, each followed by max-pooling, then flattening and a fully connected layer (128 neurons) with dropout, and a Softmax output layer for classification.	The convolutional layers extract hierarchical spatial features from images, while pooling reduces dimensionality. Dropout helps prevent overfitting, resulting in improved generalization and stable training.
CNN Model 2	Consists of two convolutional blocks followed by a fully connected classifier.	This architecture was chosen to establish a performance benchmark for raw spatial feature extraction without the regularization complexity, allowing us to measure the pure fitting capacity of the network.
Unet Model 1	Model was lightweighted by reducing the initial filter count to 32 Utilizing "same" padding throughout Eliminating the need for cropping and ensuring output mask dimensions (128 × 128)	This design significantly lowers computational cost compared to the standard U-Net while preserving exact input-to-output spatial resolution, which is critical for precise mask alignment.
YOLOV8n Model 1	Uses a lightweight YOLOv8n backbone with increased depth in the final C2f block, while keeping the original detection head structure and configuring the Detect layer for the target number of classes.	Enhancing backbone depth improves high-level feature representation for object detection, while preserving the original head maintains efficiency and real-time performance.

4.2.2 Fish Recognition Ground-Truth Dataset

Model	Architecture Modification	Justification
ANN Model 3	two-hidden-layer architecture consisting of 256 and 128 neurons. using the ReLU activation function. Used the Adam optimizer. the training process was limited to 40 iterations.	The use of multiple hidden layers increased the network's capacity for handling complex and high-dimensional feature representations. The decreasing number of neurons from 256 to 128 allowed for progressive feature compression, improving generalization while avoiding model complexity.
ANN Model 4	The ANN architecture was modified by introducing two hidden dense layers with ReLU activation instead of a single-layer perceptron, enabling the model to capture more complex non-linear patterns from the extracted LBP and GLCM feature vectors.	Aiming at dealing with complex non-linear relationships embodied in handcrafted feature vectors such as LBP and GLCM, a funnel architecture with multiple fully connected layers in the ANN model can boost model capacity. As a result, class separability will be enhanced with lower computation complexity, and multi-class decision can be achieved via Softmax output.
CNN Model 3	Two Convolutional layers (64,128). Flatten Layer converts 2D into a 1D vector. Fully Connected (Dense) Layer. Addition of a Dropout(0.5) layer. Remove the initial 32-filter Conv2D layer and the rescaling operation.	Two convolutional layers provides sufficient capacity to extract relevant features while keeping the network lightweight. Flatten + Dense layer transforms features into a 1D vector for effective classification. Dropout(0.5) reduces overfitting.
CNN Model 4	A custom-built convolutional neural network consisting of two convolutional blocks with Batch Normalization and MaxPooling, followed by a fully connected classifier. A second variant includes Dropout layers to reduce overfitting. This model was implemented manually and not taken from pre-built architecture.	The CNN model design consisted of two convolutional blocks to address both learning capability and efficiency in a way that is amenable to learning with lower-resolution images. Batch normalization and max pooling layers were incorporated to help with model stability during learning and to remove redundancy in space, respectively. Dropout layers were added to counter observed overfitting in learning during initial experiments.
SegNet	A custom encoder–decoder design where downsampling is done through convolution and pooling, and upsampling is achieved using transposed convolutions. The	SegNet architecture was used for carrying out per-pixel segmentation and maintaining this information with an Encoder-Decoder network. Downsample layers were incorporated for creating

	architecture ensures output masks remain the same size as the input without requiring cropping.	high-level semantic information, and transposed convolution layers were used for upsampling or maintaining spatial information. The end 1x1 convolution with Sigmoid activation function was used for creating a segmentation mask in a format of annotations in a dataset.

4.2.3 Architecture of Stacking

The chosen strategy to implement fusion is Stacked Generalization (Stacking), a technique of decision-level fusion that organizes the classification task into a two-tier to maximize predictive accuracy. Level 0 “base Models” consists of multiple diverse models that independently analyse the raw image data to generate initial predictions. Level 1 “Meta Model” is a higher-level model that does not see the raw images; instead, it trains on the probability scores output by the Level 0 models, learning the optimal strategy to combine their specific strengths for a final decision.

4.3 Transfer Learning: Pre-trained Models and Fine-Tuning Strategy

4.3.1 MobileNetV2

Fine-Tuning Strategy

- **Transfer Learning Initialization:** The model was initialized with ImageNet pretrained weights to leverage learned visual features.
- **Layer Freezing Strategy:** The pretrained backbone was kept frozen during training, allowing only the newly added layers to be trained.
- **Training Scope:** Fine-tuning was limited to the custom classification head, reducing training time and preventing overfitting on the target dataset.

Architecture Modifications

- **Custom Head:**
  The custom head consists of a fully connected layer with ReLU activation to learn discriminative features, followed by dropout for regularization, and a final classification layer to generate class probability outputs**.**
- **Pooling Strategy**
  Global Average Pooling was used instead of the original fully connected pooling mechanism.
- **Regularization Layers**
  Batch normalization and dropout layers were added to improve training stability and generalization.

4.3.2 InceptionV3

Fine-Tuning Strategy

- **Two-phase training** was used to stabilize transfer learning and avoid damaging pretrained ImageNet features early.
- **Phase 1 (feature extractor):** all InceptionV3 layers were frozen and only the new classification head was trained using a higher learning rate (Adam, 1e-3) to quickly adapt to the target dataset.
- **Phase 2 (fine-tuning):** the **last 20 layers** of the backbone were unfrozen to allow high-level features to specialize for the new classes, while keeping earlier layers frozen to preserve general visual representations. A much smaller learning rate (Adam, 1e-5) was used to perform small, stable weight updates and reduce the risk of catastrophic forgetting.

Architecture Modifications

- **Backbone:** InceptionV3 pretrained on ImageNet with (include_top=False) removes the original ImageNet classifier.
- **Pooling:** GlobalAveragePooling2D was added to aggregate feature maps into a compact feature vector, significantly reducing the number of parameters and overfitting compared to using a Flatten layer.
- **Custom head:** The custom head uses a Dense (256, ReLU) layer to learn discriminative features, followed by Dropout (0.5) to reduce overfitting, and a final Softmax layer to output multi-class probabilities.

4.3.3 ResNet 18

Fine-Tuning Strategy:

- All layers of the ResNet-18 network were trained rather than freezing early layers. This allowed the model to fully adapt low-level and high-level features.
- Training was performed using the Adam optimizer with a low learning rate of 1e-4, which enabled gradual weight updates, preserved stable convergence, and prevented the destruction of previously learned representations.

Architecture Modifications

- Global Average Pooling was applied to reduce spatial dimensions and parameter count.
- Custom classifier head was added on top of the feature extractor.
- A fully connected layer with 256 neurons and ReLU activation was added to improve class separability.
- A Dropout layer with a rate of 0.3 to reduce overfitting.
- The output layer used a softmax activation to support multi-class classification.

4.3.4 EfficientNet-B0

Fine-Tuning Strategy

The EfficientNet-B0 model was fine-tuned using a transfer learning technique in which the weights learned from the ImageNet dataset were used without altering, and the early layers were set to freeze in order to lock in the common visual information. The first 200 parameters were set to freeze, and the remaining layers and a classifier head added to classify images in the fish dataset were learned.

Architecture Modifications

The original EfficientNet-B0 model architecture was altered in such a way that a custom lightweight classifier with a Dropout layer followed by a fully connected Dense layer with a size equal to the number of fish classes with a Softmax activation function replaced the default classifier head in the original model. Furthermore, modifications were made to input resolution to 128x128 pixels to achieve a balance between model efficiency and representation.

4.4 Models' Hyperparameters

4.4.1 A Large-Scale Fish Dataset

4.4.1.1 Classification

Model	Hyperparameters
LBP + ANN Model 1	Learning Rate: 1e-3 to 1e-5 Batch Size: 64 Epochs: 200
LBP + KNN	N neighbors: 10 Metric: euclidean Weights: distance
LBP + SVM	Kernel: rbf C: 100 Gamma: 0.1 decision_function_shape: ovr
HuMoments + ANN Model 1	Learning Rate: 1e-3 Batch Size: 64 Epochs: 300
HuMoments + KNN	N neighbors: 3 Metric: euclidean Weights: distance
HuMoments + SVM	Kernel: rbf C: 100 Gamma: 0.1 decision_function_shape: default

CNN Model1	Learning Rate: 1e-3 Batch Size: 32 Epochs: 10
Transfer Learning (InceptionV3)	Learning Rate: 1e-3 Batch Size: 32 Epochs: 30
GLCM + ANN Model 2	Learning Rate: Adam Batch Size: 64 Epochs: 20
HOG + ANN Model 2	
GLCM + KNN	N neighbors: 3 Metric: uniform Weights: minkowski
HOG + KNN	
GLCM + SVM	Kernel: rbf C: 1.0 Gamma: scale decision_function_shape: ovr
HOG + SVM	
CNN Model2	Learning Rate: Adam Batch Size: 64 Epochs: 10
Transfer Learning (MobileNetV2)	Learning Rate: Adam(1e-5) Batch Size: 64 Epochs: 10

4.4.1.2 Instance Segmentation

Model	Hyperparameters
Unet Model 1	Learning Rate: 1e-3 Batch Size: 8 Epochs: 10
DeepLabV	
FCN	

4.4.1.3 Object Detection

Model	Hyperparameters
YOLOV8n	Learning Rate: Default Batch Size: 16 Epochs: 20
RT-DETR	
YOLOV8n Model1	

4.4.2 Fish Recognition Ground-Truth Dataset

4.4.2.1 Classification

Model	Hyperparameters
ORB + ANN Model 3	Learning Rate: default 0.001 Batch Size: 200 Epochs: max 40
HOG + ANN Model 3	
ORB + KNN	N neighbors: 5 Metric: minkowski Weights: uniform
HOG + KNN	
ORB + SVM	Kernel: rbf C: 1.0 Gamma: scale decision_function_shape: ovr
HOG + SVM	
CNN Model3	Learning Rate: 0.001 Batch Size: 32 Epochs: 10
Transfer Learning (ResNet 18)	Learning Rate: 1e-4 Batch Size: 32 Epochs: 10
LBP + ANN Model 4	Learning Rate: 0.001 Batch Size: 32 Epochs: 10
GLCM + ANN Model 4	
LBP + KNN	N neighbors: 5 Metric: euclidean Weights: uniform
GLCM+ KNN	N neighbors: 5

	Metric: Manhattan Weights: uniform
LBP + SVM	Kernel: rbf C: 1.0 Gamma: scale decision_function_shape: ovo
GLCM + SVM	
CNN Model4	Learning Rate: 0.001 Batch Size: 32 Epochs: 10
Transfer Learning (EfficientNet-B0)	Learning Rate: 1e-4 Batch Size: 32 Epochs: 20

4.4.2.2 Instance Segmentation

Model	Hyperparameters
SegNet	Learning Rate: 0.001 Batch Size: 8 Epochs:10
YOLOv8	Learning Rate: 0.01 Batch Size: 16 Epochs: 10
Mask R-CNN	Learning Rate: 0.00025 Batch Size: 2 images per GPU Epochs/iteration: 4000

4.4.3 Aquarium Dataset

4.4.3.1 Object Detection

Model	Hyperparameters
YOLOv5	Learning Rate: 0.01 Batch Size: 16 Epochs: 10
YOLOv11	Learning Rate: 0.001 Batch Size: 16 Epochs: 10
YOLOv11 update1	

4.5 Training Process

4.5.1 A Large-Scale Fish Dataset

4.5.1.1 Classification

Training-Testing split: 80% Training and 20% Testing.

Evaluation Metrics:

- Accuracy
- Precision
- Recall
- F1-Score

Cross-validation: A 5-Fold Stratified Cross-Validation was performed on all models to ensure more reliable and stable performance estimation.

Optimization Methods: Feature scaling was applied using StandardScaler ensuring that all numerical features were normalized before model training.

Regularization Techniques:

- Dropout layers were used in deep learning models to reduce overfitting.
- Early Stopping was used to prevent the model from training beyond the point where validation loss stopped improving.

4.5.1.2 Instance Segmentation

Training-Testing split: 50% Training / 50% Validation.

Evaluation Metrics:

- Pixel Accuracy
- Dice Score
- Mean IoU
- Avg Inf. Time

Cross-validation: 2-Folds Cross Validation were applied to all models.

Optimization Methods: Adam

4.5.1.3 Object Detection

Training-Validation split: 80% Training and 20% Validation.

Evaluation Metrics:

- mAP@50
- mAP@50–95
- Precision
- Recall
- Inference Time

Cross-validation: It was not performed for YOLO or RT-DETR because each training session requires substantial GPU resources, and with limited Colab GPU availability and slow CPU fallback, running multiple folds was not computationally feasible. Therefore, a standard 80% training / 20% validation split was used instead, relying on each model's built-in validation during training.

4.5.2 Fish Recognition Ground-Truth Dataset

4.5.2.1 Classification

Training-Testing split: 80% Training and 20% Testing.

Evaluation Metrics:

- Accuracy
- Precision
- Recall
- F1-Score

Cross-validation: cross validation was applied to all models using 5 folds with 10 epochs in each.

Optimization Methods: Adam optimizer.

Regularization Techniques: Dropout(0.5) layer was added to reduce overfitting.

4.5.2.2 Instance Segmentation

Training-Testing split: 80% training and **20% validation**

Evaluation Metrics:

- Dice accuracy
- Pixel accuracy
- Precision & recall
- Confusion matrix
- (YOLO-only) mAP50&mAP50-95

Cross-validation: Not applied for segmentation models

Optimization Methods:

- For YOLOv8 & Mask R-CNN :SGD
- For SegNet: Adam

4.5.3 Aquarium Dataset

4.5.3.1 Object Detection

Training-Testing split: 70% training, 20% validation, 10% testing

Evaluation Metrics:

- precision
- Recall
- mAP@50
- mAP@50–95
- Inference Time

Cross-validation: Cross-validation was applied on YOLOv5 and YOLOv11 using 2 folds, with 10 epochs per fold.

Optimization Methods: YOLOv5 optimizer is SGD, YOLOv11 and YOLOv11 update1 is Adam.

5 Model Comparison and Performance Analysis

5.1 A Large-Scale Fish Dataset

5.1.1 Classification

Models Result:

Model	Accuracy	Precision	Recall	F1-score
LBP + ANN	0.6069	0.6121	0.6065	0.6047
HuMoments +ANN	0.6264	0.6291	0.6273	0.6253
HOG + ANN	**0.9333**	**0.9344**	**0.9333**	**0.9334**
GLCM +ANN	0.8394	0.8424	0.8394	0.8395
LBP + KNN	0.8562	0.8579	0.8563	0.8563
HOG + KNN	**0.9178**	**0.9205**	**0.9178**	**0.9183**
GLCM + KNN	0.8306	0.8342	0.8306	0.8308
HuMoments +KNN	0.7764	0.7780	0.7770	0.7765
LBP + SVM	0.7458	0.7527	0.7455	0.7470
HuMoments +SVM	0.5569	0.5615	0.5578	0.5519
HOG + SVM	**0.9233**	**0.9257**	**0.9233**	**0.9236**
GLCM +SVM	0.7028	0.7050	0.7028	0.7003
CNN Model1	**0.9847**	**0.9847**	**0.9846**	**0.9846**
CNN Model2	0.9844	0.9845	0.9844	0.9844
InceptionV3	**0.9785**	**0.9785**	**0.9785**	**0.9785**
MobileNetV2	0.9672	0.9679	0.9673	0.9671

Result Analysis:

The results indicate that HOG-based models consistently achieved the strongest performance among handcrafted feature approaches, with HOG + ANN and HOG + SVM reaching accuracy levels above 0.92, outperforming LBP, GLCM, and Hu Moments due to their superior edge and shape representation. Hu Moments-based models showed the weakest performance across classifiers, reflecting their limited ability to capture discriminative visual patterns. Overall, deep learning models substantially outperformed traditional methods, highlighting the advantage of automatic feature learning. CNN Model 1 achieved the highest overall accuracy (0.9847), benefiting from a deeper architecture and dropout regularization, while CNN Model 2 demonstrated comparable but slightly lower performance. Transfer learning models also performed strongly, with InceptionV3 achieving high accuracy using pretrained features and MobileNetV2 providing a favorable balance between accuracy and computational efficiency.

Models' strength and weakness:

Model	Strength	Weakness
LBP + ANN	Captures local texture patterns effectively and performs well on texture-rich images.	Sensitive to noise and illumination variations.
HuMoments + ANN	Provides rotation and scale-invariant shape representation.	Limited ability to describe complex textures and fine details.
HOG + ANN	Effectively captures edge and gradient information for object structure.	Performance degrades when object orientation varies significantly.
GLCM + ANN	Extracts strong statistical texture features based on spatial relationships.	Computationally expensive and sensitive to parameter selection.
LBP + KNN	Simple and effective for texture-based classification with small datasets.	Sensitive to noise and increases in computation with larger datasets.

HuMoments + KNN	Robust to rotation and scale changes in object shapes.	Distance-based classification struggles with high-dimensional features.
HOG + KNN	Preserves local shape and edge information well.	High memory usage and slow inference for large datasets.
GLCM + KNN	Effective in distinguishing textures using spatial gray-level statistics.	Classification speed decreases significantly as dataset size grows.
LBP + SVM	Achieves good generalization with limited training samples.	Sensitive to kernel and hyperparameter selection.
HuMoments + SVM	Performs well for shape-based classification with clear boundaries.	Limited performance when discriminative texture information is required.
HOG + SVM	Strong performance in capturing structural and edge-based features.	Computationally expensive for large-scale datasets.
GLCM + SVM	Provides robust classification for texture-dominant datasets.	Feature extraction and model tuning are time-consuming.
CNN Model1	Improved feature extraction through deeper layers.	Higher risk of overfitting on small datasets.
CNN Model2		
InceptionV3	Efficient multi-scale feature extraction with high classification accuracy.	High computational and memory requirements.
MobileNetV2	Lightweight architecture optimized for fast inference and low resource usage.	Slightly lower accuracy compared to heavier deep CNN models.

5.1.2 Instance Segmentation

Models Result:

Model/Metrics	Dice Score	Pixel Accuracy	Mean IoU	Avg Inf. Time
DeepLabV3	0.9637	0.9880	0.9300	4.84 ms/batch
FCN_ResNet50	0.9427	0.9804	0.8916	4.30 ms/batch
Unet Model 1	0.9500	0.9834	0.9047	1.76 ms/batch

Result Analysis:

The instance segmentation results highlight clear performance differences among the models. DeepLabV3 achieved the highest Dice score and Mean IoU, indicating the most accurate segmentation overall, though with the highest inference time. U-Net offered a strong trade-off between accuracy and efficiency, delivering competitive segmentation performance with significantly faster inference. FCN_ResNet50 showed consistent, but lower Mean IoU compared to the other models. Overall, the results emphasize the balance between segmentation accuracy and computational efficiency across architectures.

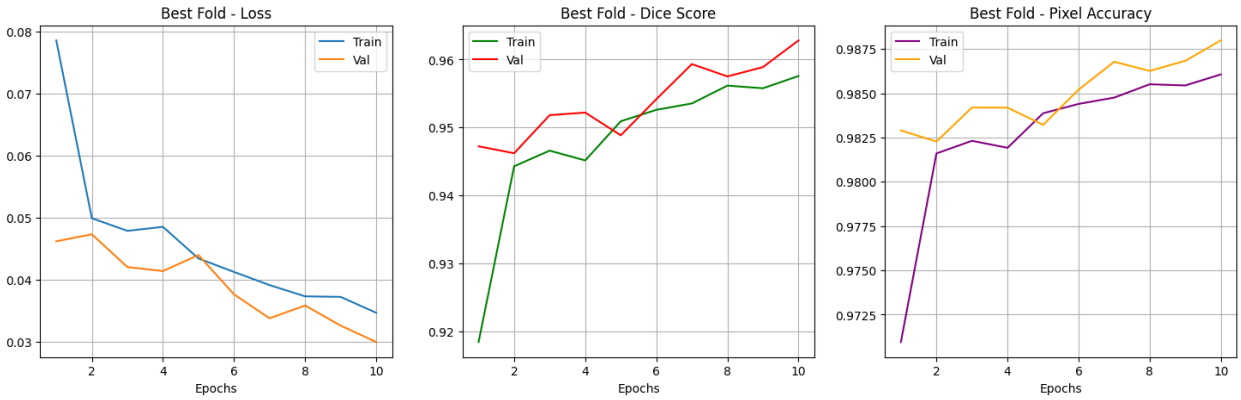
Models' strength and weakness:

Model	Strength	Weakness
DeepLabV3	Achieves the highest segmentation accuracy, effectively capturing fine object boundaries.	Higher computational cost and slower inference time compared to lighter models.
FCN_ResNet50	Provides stable and consistent segmentation performance with good balance between accuracy and complexity.	Lower Mean IoU than DeepLabV3 and limited ability to refine fine object boundaries.

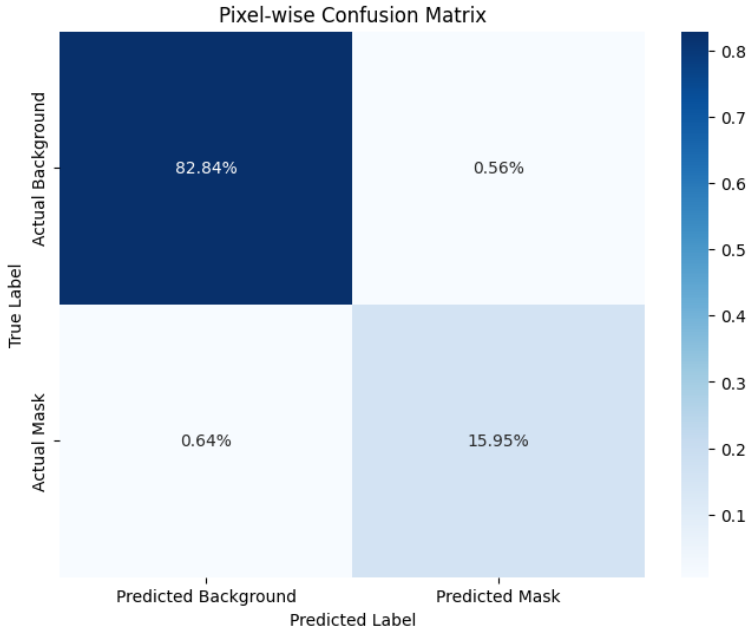
Unet Model 1	Fastest inference time with strong Dice and IoU scores, making it suitable for real-time segmentation.	Slightly less accurate than DeepLabV3 on complex boundary details.

Best Performance DeepLabV3

Training and Validation Performance Curves:



Confusion Matrix:



5.1.3 Object Detection

Models Result:

Model/Metrics	Precision	Recall	mAP@50	mAP@50-95	Inference Time
YOLOV8n	99.5%	99.6%	99.3%	92.4%	4.014 ms
RT-DETR	99.6%	99.6%	99.2%	94.3%	33.798 ms
YOLOV8n Model1	99.1%	99.4%	99.0%	92.3%	4.140 ms

Result Analysis:

The object detection results show that all models achieved very high precision and recall values. RT-DETR obtained the highest mAP@50–95, indicating superior performance in detecting objects across multiple IoU thresholds. YOLOv8n achieved comparable accuracy with significantly faster inference time, making it suitable for real-time applications. The customized YOLOv8n Model1 maintained high detection accuracy while preserving fast inference speed. Overall, the results highlight a trade-off between accuracy and computational efficiency among the evaluated models.

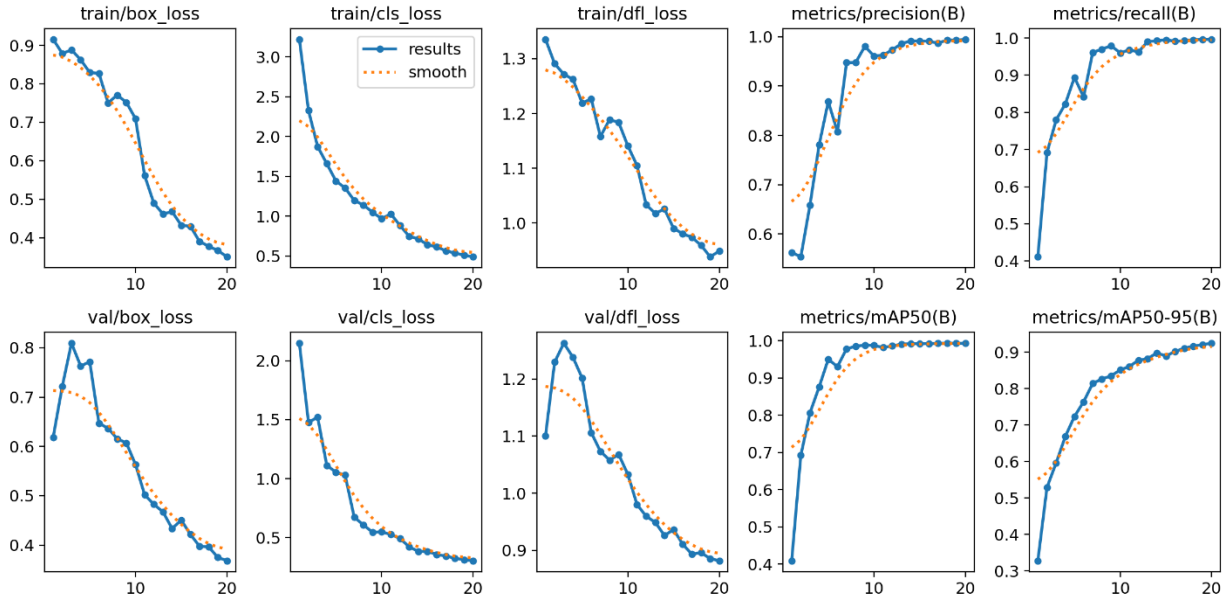
Models' strength and weakness:

Model	Strength	Weakness
YOLOV8n	Efficient one-stage detector that delivers high detection accuracy with very fast inference, making it well suited for real-time applications	Limited global context modelling compared to transformer-based detectors
RT-DETR	Transformer-based architecture provides strong global feature reasoning, leading to robust multi-scale object detection	High computational complexity results in slower inference time

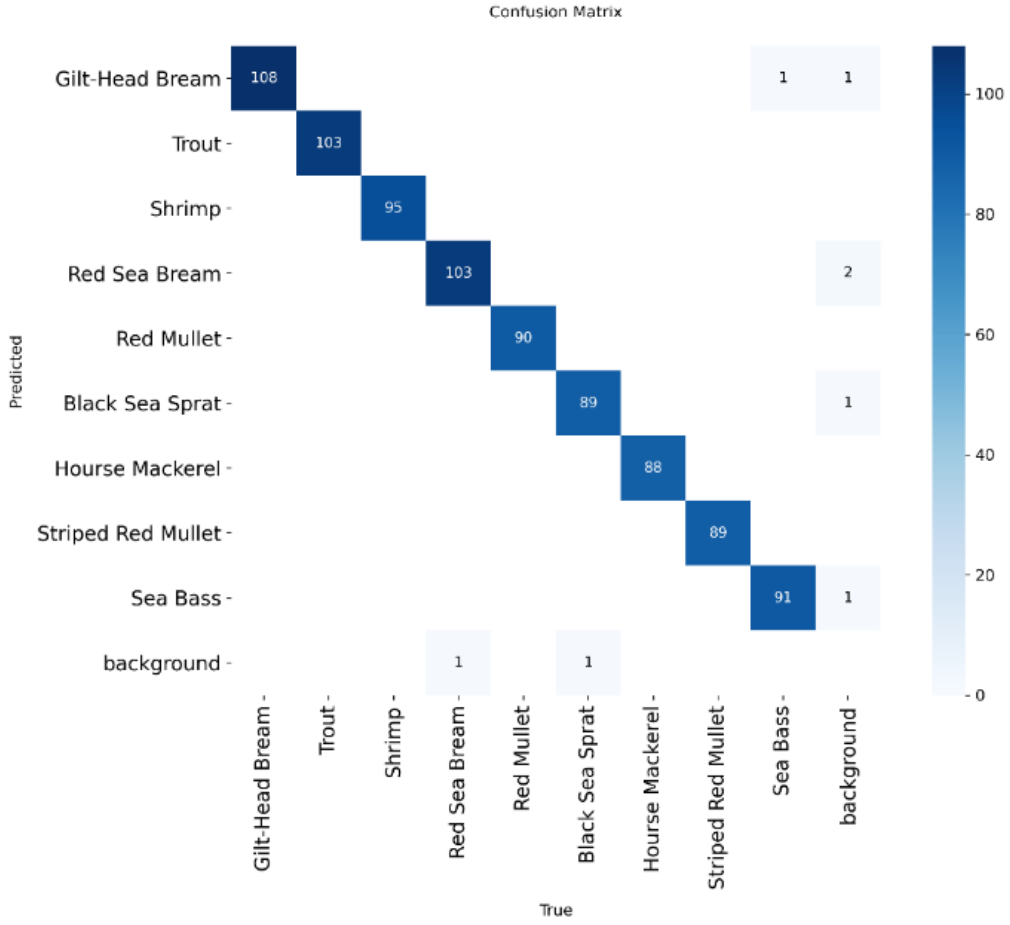
YOLOV8n **Model1**	Maintains YOLOv8 efficiency while enhancing feature representation, achieving a balanced trade-off between accuracy and speed	Slight increase in model complexity may reduce deployment efficiency on low-resource devices

Best Performance YOLOV8n

Training and Validation Performance Curves:



Confusion Matrix:



5.2 Fish Recognition Ground-Truth Dataset

5.2.1 Classification

Models Result:

Model	Accuracy	Precision	Recall	F1-score
HOG + ANN	**0.97**	**0.97**	**0.97**	**0.97**
ORB+ANN	0.78	0.77	0.78	0.77
LBP + ANN	0.64	0.65	0.64	0.64
GLCM +ANN	0.73	0.74	0.73	0.73
HOG+ KNN	**0.91**	**0.91**	**0.91**	**0.91**
ORB+ KNN	0.69	0.70	0.69	0.68
LBP + KNN	0.59	0.60	0.59	0.59
GLCM +KNN	0.73	0.73	0.73	0.73
HOG + SVM	**0.96**	**0.96**	**0.96**	**0.96**
ORB +SVM	0.76	0.76	0.76	0.76
LBP +SVM	0.64	0.65	0.64	0.64
GLCM +SVM	0.73	0.73	0.73	0.73
CNN Model3	0.89	0.83	0.89	0.85
CNN Model4	**0.97**	**0.97**	**0.97**	**0.97**
ResNet 18	**0.99**	**0.99**	**0.99**	**0.99**
EFFICIENTNET-B0	0.98	0.98	0.98	0.98

Result Analysis:

It is evident from the results that there is a noticeable difference in performance between classical models that work with features, on one hand, and deep learning models on the other hand. In classical models, the best performance was shown by HOG-based models with HOG + ANN & HOG + SVM with an accuracy of 0.97 & 0.96 respectively, whereas LBP-based models showed the worst performance. GLCM & ORB features showed some reasonable performance with various models. On comparing the performance of classical models with deep learning models, it is evident that deep learning models outperformed classical models with CNN Model4 showing the best performance among them with an accuracy of 0.97. Additionally, transfer learning models ResNet-18 & EfficientNet-B0 showed the best performance with accuracy of 0.99 & 0.98 respectively. This superior performance is due to the use of pretrained feature extractors learned from large-scale datasets, which enable the models to capture rich and discriminative visual features such as edges, textures, and object structures.

Models' strength and weakness:

Model	Strength	Weakness
HOG+ANN	Can learn complex, non-linear patterns from HOG features. Has flexible architecture.	Needs more data and training time. With low epochs, it may underfit easily.
ORB+ANN	Flexible architecture that can learn richer patterns than SVM/KNN.	ORB descriptors are shallow features, limiting ANN performance.
LBP + ANN	Were improved by a two-layer neural network, which captured non-linear patterns in LBP histograms across folds.	Overfitting on some folds because of the low dimensionality of LBP features and because there was not that much real texture variation.
GLCM +ANN	The fully connected neural network was trained for multiple epochs to learn complex relationships between several GLCM properties.	Demand great care in normalization and still showed limited improvement due to the handcrafted nature of the input features.
HOG+ KNN	Very simple it needs no training time. Works fine with well-separated HOG features.	Sensitive to feature scaling and noise.

ORB+ KNN	ORB descriptors are distance-based making KNN naturally compatible.	Slow prediction because it compares with all samples.
LBP + KNN	Demonstrated reasonable performance because it used uniform LBP histograms extracted from resized 64×64 grayscale images with standardized scaling.	It cannot be that accurate because LBP lost the fine details of texture in resizing and became sensitive to changes in illumination even after augmentation.
GLCM +KNN	They used several computed GLCM properties such as contrast, homogeneity, energy, and correlation on resized images that allowed statistical texture discrimination.	Performance varied greatly across folds, since the GLCM features were highly sensitive to the resizing and grayscale quantization of images.
HOG + SVM	Works very well with high-dimensional features like HOG.	Training becomes slow when dataset grows large.
ORB +SVM	Good at capturing non-linear class boundaries. Usually gives stable results even.	Sensitive to feature scaling and kernel parameters. Training becomes slower on larger datasets.
LBP +SVM	It has performed much better than KNN because it learned clearer class boundaries using scaled LBP feature vectors with cross-validation.	The classification errors increased for visually similar fish classes, as the model depended only on handcrafted texture features.
GLCM +SVM	Managed to attain more stable accuracy by incorporating normalization of GLCM features with kernel-based learning and cross-validation.	Increased training time and some misclassifications remained for classes with overlapping texture statistics.
CNN Model3	Very lightweight architecture it trains quickly. Dropout (0.5) reduces overfitting and improves generalization.	Only two Conv layers cannot learn complex patterns.
CNN Model4	The dropout variant yielded better generalization and smoothness in the validation curves.	The shallow architecture and small input resolution limit the model from capturing fine-grained details of fish, especially without dropout.

ResNet 18	Strong feature extraction, it uses residual connections, helping it learn deep features	Only 10 epochs per fold may cause underfitting.
EFFICIENTNET-B0	Achieved high accuracy by leveraging a pre-trained backbone with a custom classification head and ImageNet normalization on 128×128 inputs.	This required careful fine-tuning, as freezing many layers restricted adaptation to subtle inter-class differences in the fish dataset.

5.2.2 Instance Segmentation

Models Result:

Model/Metrics	Dice Score	Pixel Accuracy	Mean IoU	Avg Inf. Time
YOLOv8	0.89	**0.99**	**0.82**	2.1 ms/image
Mask R-CNN	**0.908**	**0.9729**	0.83	120-180 ms/image
SegNet	0.771	0.930	0.65	20-30 ms/image

Result Analysis:

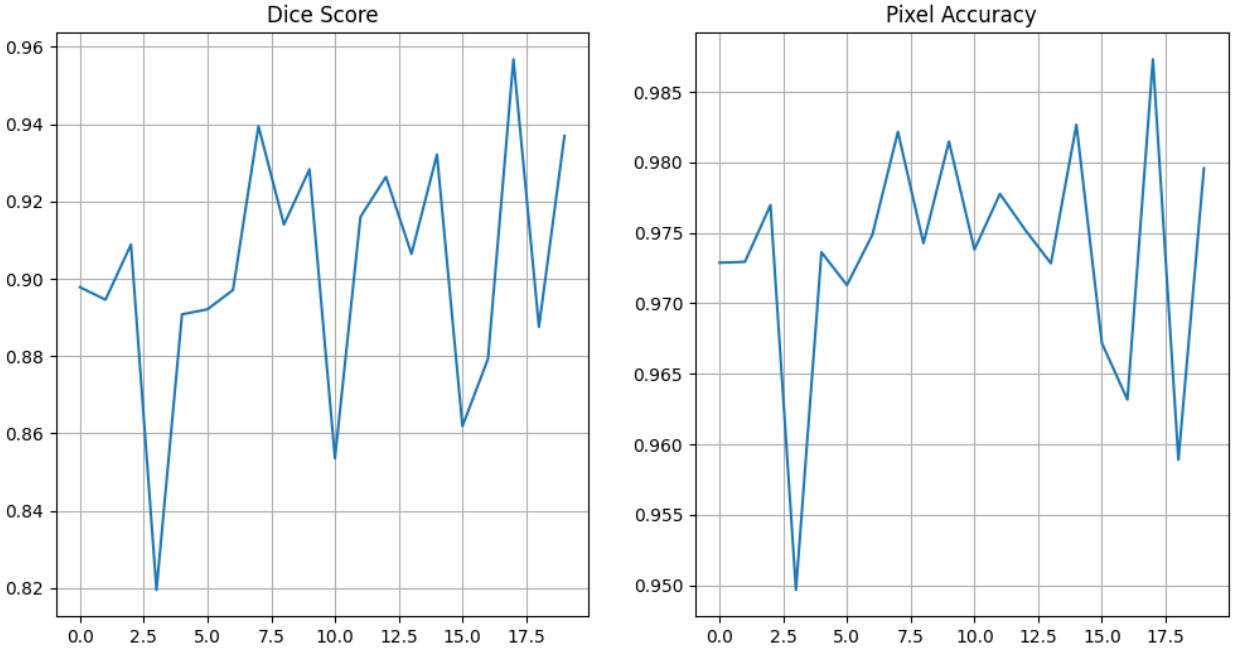
The results show that YOLOv8 strikes an excellent balance between segmentation accuracy and inference speed since it yields a very high Dice score of 0.89, Mean IoU of 0.82, with the fastest inference time, at approximately 2.1 ms per image, making it highly suitable for real-world applications in real time. This was mainly due to its single-stage segmentation architecture that jointly uses object localization and mask prediction from shared feature representation, which enables the model to grasp both global context and fine-grained details with efficiency. Mask R-CNN slightly outperformed YOLOv8 on Dice score, with 0.908, and Mean IoU, with 0.83. This indicates it could yield better mask boundaries, but at great computational time investments with much slower inference times, 120–180 ms per image. On the other hand, SegNet yielded acceptable performance, with a pixel accuracy of 0.930; however, its lower Dice score, 0.771, and Mean IoU, 0.65, revealed a boundary accuracy limitation because of its simpler encoder–decoder design.

Models' strength and weakness:

Model	Strength	Weakness
YOLOv8	Offers high segmentation accuracy with very rapid inference speeds, making it useful for real-time multi-class segmentation applications.	A bit less accurate in object boundaries relative to Mask R-CNN, especially for small or overlapping objects.
Mask R-CNN	Reaches the best Dice Score and Mean IoU, with more accurate and finer segmentation masks.	Inference time is very slow, which is a hindrance for applications requiring real-time processing. Moreover, it consumes high computational resource.
SegNet	Light-weight architecture with good pixel accuracy and computational needs lower than Mask R-CNN.	Lower Dice Score and Mean IoU, implying poor boundary segmentation capability and performance in complex scenes.

Best Performance YOLOv8 results:

Model's curves:



5.3 Aquarium Dataset

5.3.1 Object Detection

Models Result:

Model/Metrics	Precision	Recall	mAP@50	mAP@50-95	Inference Time
YOLOv5	**74.2%**	**63.9%**	**70.7%**	**43.9%**	**150.7ms**
YOLOv11	62.3%	55.5%	59.1%	36.3%	74.8ms
YOLOv11 update1	76.3%	57.3%	65.5%	39.5%	21.0ms

Result Analysis:

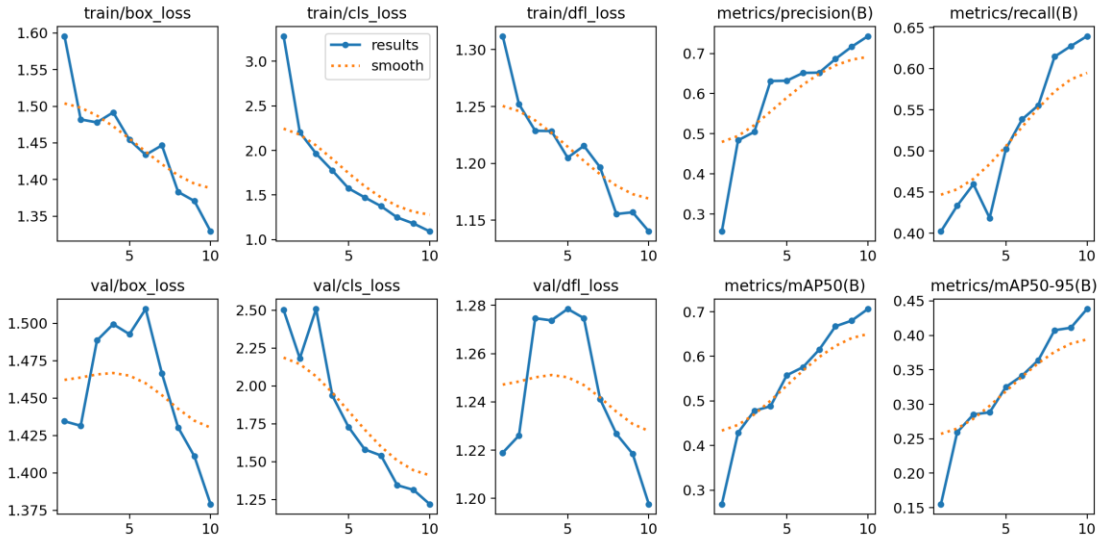
The performance comparison between YOLOv5, YOLOv11, and YOLOv11 update1 reveals a clear trade-off between accuracy and speed. YOLOv5 achieves the highest recall (63.9%) and mAP values (mAP@50 of 70.7% and mAP@50-95 of 43.9%), indicating that it detects the largest number of true positives. Its precision is also high at 74.2%, but it suffers from the slowest inference time of 150.7 ms. In contrast, the baseline YOLOv11 exhibits lower performance, with the lowest precision (62.3%), recall (55.5%), and mAP scores (mAP@50 of 59.1% and mAP@50-95 of 36.3%), although its inference time of 74.8 ms is faster than YOLOv5. The YOLOv11 update1 model demonstrates a balanced improvement, achieving the highest precision at 76.3% and faster inference at 21.0 ms, making it the most suitable for real-time detection scenarios. While its recall (57.3%) and mAP scores (mAP@50 of 65.5% and mAP@50-95 of 39.5%) are slightly lower than YOLOv5, it effectively reduces false positives and provides a practical compromise between detection accuracy and speed, making it ideal for applications that require fast and reliable object detection.

Models' strength and weakness:

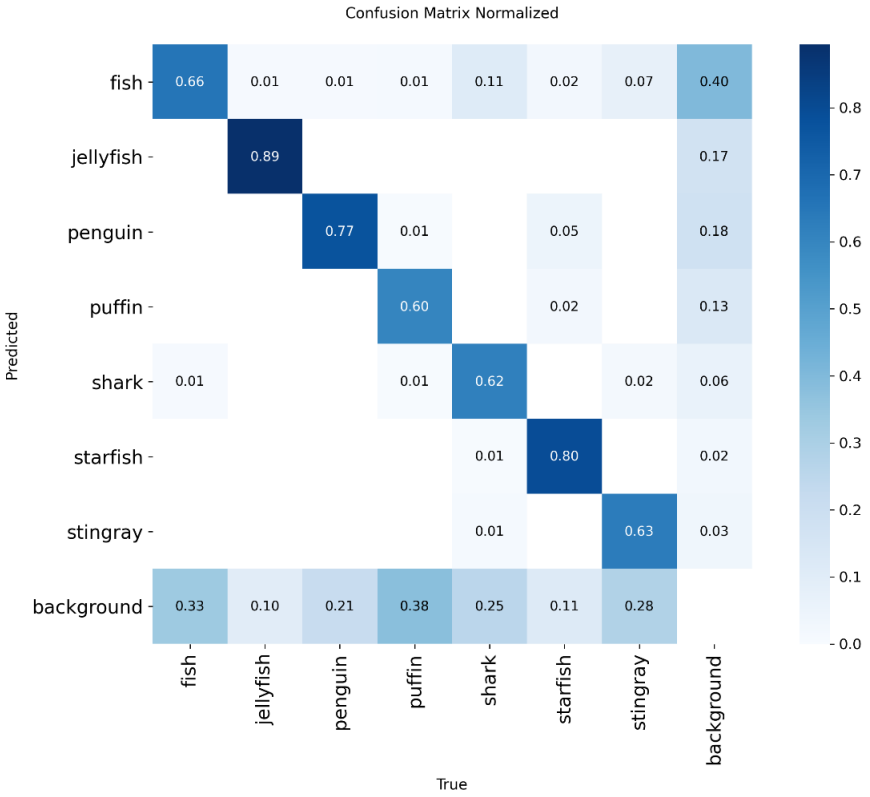
Model	Strength	Weakness
YOLOv5	Fast and lightweight. Works well even when data is limited compared to larger YOLO models.	has limited capacity for very small or overlapping objects. The version has fewer layers.
YOLOv11	Improved feature extraction and detection head compared to YOLOv5. Extremely fast inference.	Lower accuracy than larger YOLOv11 variants. With small or imbalanced datasets, performance may drop noticeably.
YOLOv11 update1	Lightweight and memory efficient. Stable learning setup of 0.001.	Lower accuracy than larger YOLO models. Short training duration (10 epochs).

Best Performance YOLOv5 results:

Model's curves:



Model's confusion matrix:



6 Discussion

Model Performance Across Tasks

Model performance differed across tasks due to how well each model's design aligned with the problem requirements. In classification, deep learning models performed best because CNNs learn hierarchical representations that jointly capture shape, texture, and contextual information. Among handcrafted approaches, HOG-based methods achieved higher accuracy as they effectively represent edge orientation and global shape cues that are important for fish discrimination. For instance segmentation, DeepLabV3 outperformed other models by integrating multi-scale contextual information while preserving spatial details, leading to more accurate boundary segmentation. In object detection, YOLOv8 achieved the best balance between accuracy and inference speed due to its single-stage architecture and efficient feature fusion, enabling high detection performance with real-time efficiency.

Effect of Data Preprocessing and Augmentation

The impact of data preprocessing and augmentation varied across the evaluated models. Not all models benefited equally from data augmentation, as CNN Model 3 showed degraded performance after augmentation due to the introduction of augmented samples for a specific class, which the model failed to learn effectively. This suggests that the augmented data may have introduced variations that increased intra-class complexity, leading to confusion during training and reduced generalization. In contrast, other classification models, including HOG-based classification models for example, demonstrated improved performance after dataset augmentation. The additional synthetic samples helped these models better capture class boundaries, reduce bias toward majority classes, and enhance overall robustness. These findings indicate that while data augmentation can improve performance, its effectiveness is highly dependent on the model architecture and the nature of the augmented data.

Computational Limitations and Motivation for Fusion

Despite the overall strong performance, some limitations were observed. Cross-validation was not feasible for some object detection and segmentation models due to high computational requirements, which may limit robustness assessment. Considering these computational

constraints, a decision-level fusion strategy was adopted specifically for the classification task, where models are computationally lighter and faster to evaluate. Classification models are therefore more suitable for ensemble learning under limited computational resources. The following describes the structure of the proposed decision-level fusion framework, including the selection of base models, the design of the meta-model, and the resulting classification performance.

7 Decision-Level Fusion

1. Base Models (Level 0)

Three heterogeneous models were selected to ensure a diversity of feature representation:

- **ANN with HOG** acting as a geometric expert that captures edge orientation and structural patterns.
- **CNN Model 1** for domain-specific and hierarchical visual features patterns.
- **MobileNetV2** utilizing transfer learning for extracting generalized and high-level semantic representations.

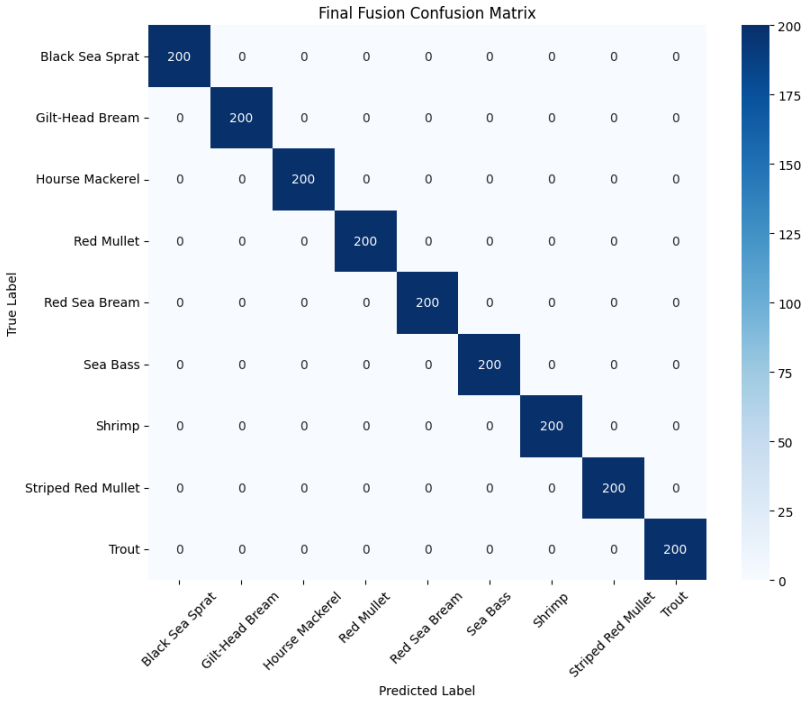
These were chosen for their high individual accuracy (>90%) and fundamentally different mathematical approaches. By combining rigid shape-based analysis with deep semantic learning, the ensemble reduces correlated errors, allowing misclassifications from one model to be effectively corrected by others, thereby improving overall robustness.

2. Meta-Model (Level 1)

As for the Meta-Model “Logistic Regression” was selected because the input data at this level is low-dimensional and highly structured. A simple, linear model effectively learns to assign importance weights to each expert without the risk of overfitting.

3. Experimental Results

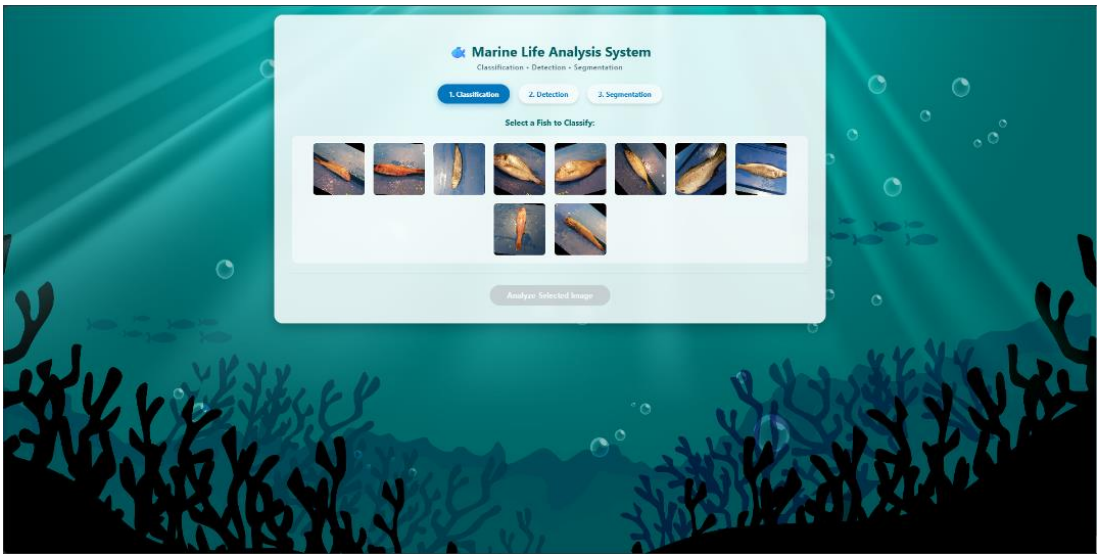
The proposed decision-level fusion framework achieved a classification accuracy of 100.00% on the validation set, outperforming all individual base models. As illustrated by the confusion matrix, all predictions fall on the main diagonal with no off-diagonal entries, indicating zero misclassifications and perfect precision and recall across all nine fish classes. This result confirms the effectiveness of the fusion strategy in producing a stable and highly accurate final prediction.

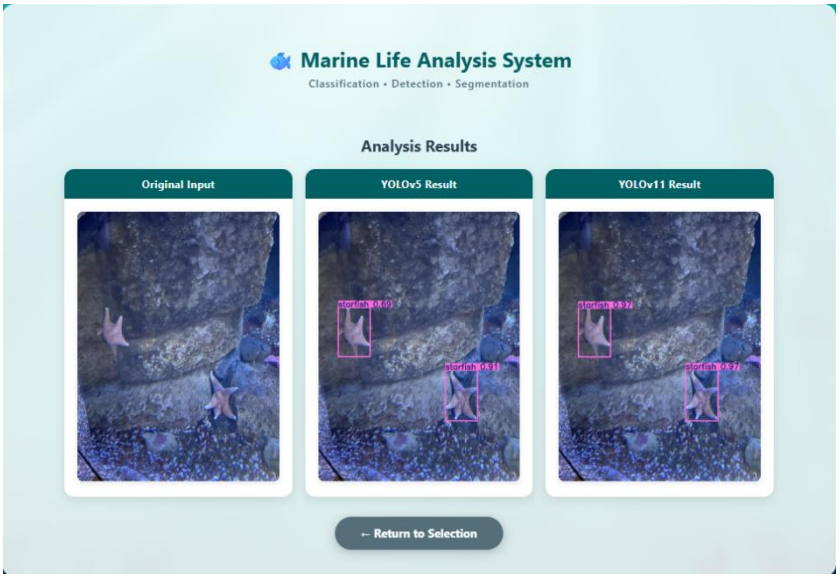

Final Fusion Confusion Matrix
Black Sea Sprat
Gilt-Head Bream
Hourse Mackerel
Red Mullet
Red Sea Bream
Sea Bass
Shrimp
Striped Red Mullet
Trout
True Label
Predicted Label
200
175
150
125
100
75
50
25
0

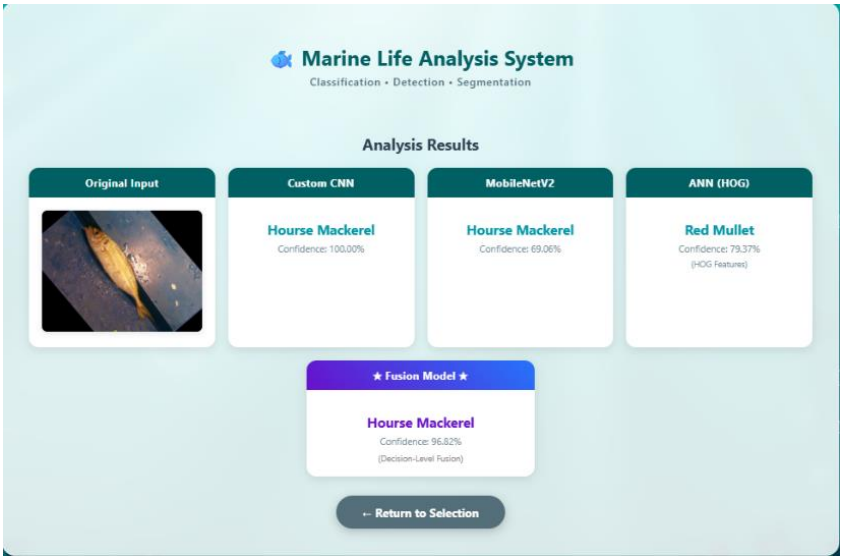
8 Program Implementation

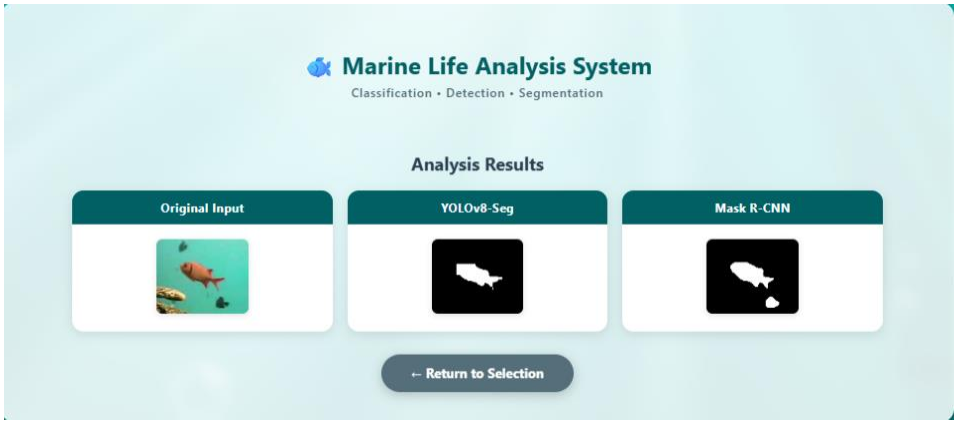
To enable the efficient implementation and usage of the suggested computer vision models, a web-based application has been created with the help of the Flask framework. It uses a client-server model and an MVC pattern. The frontend is created with HTML, CSS, and JavaScript with dynamic templates created with the help of Jinja2. The backend handles HTTP requests, image preprocessing, loading models, and model execution. A central route named `/predict` handles inputs and routes them to appropriate task pipelines.

The system includes three main processing streams. For classification, images are pre-processed with a size resolution of 128×128, normalized, and processed concurrently with a Custom CNN, a MobileNetV2, and an ANN trained on HOG features, with a decision-level fusion meta-learner combining the outputs to produce a prediction. Detection is done using YOLOv5 and YOLOv11, and finally, segmentation is done using YOLOv8-Seg and Mask R-CNN. All models are loaded at initialization for better performance, and image-processing operations are done with OpenCV and scikit-image. The web app presents a unified, efficient, and convenient platform for demonstrating the applicability of the developed hybrid framework. Below is Screenshots of the website Interface:




Marine Life Analysis System
Classification • Detection • Segmentation
Analysis Results
Original Input
YOLOv5 Result
YOLOv11 Result
← Return to Selection


Marine Life Analysis System
Classification • Detection • Segmentation
Analysis Results
Original Input
Custom CNN
Hourse Mackerel
Confidence: 100.00%
MobileNetV2
Hourse Mackerel
Confidence: 69.06%
ANN (HOG)
Red Mullet
Confidence: 79.37%
(HOG Features)
★ Fusion Model ★
Hourse Mackerel
Confidence: 96.82%
(Decision-Level Fusion)
← Return to Selection


Marine Life Analysis System
Classification • Detection • Segmentation
Analysis Results
Original Input
YOLOv8-Seg
Mask R-CNN
← Return to Selection

9 Conclusion

9.1 Summary

This project proposes a comprehensive computer vision approach for fish species classification, instance segmentation, and object detection using A Large-Scale Fish Dataset and the Fish Recognition Ground-Truth Dataset. Additionally, the Aquarium Dataset was used by the second team to perform object detection, as the Fish Recognition Ground-Truth Dataset lacks the required annotations for this task. A wide range of techniques was explored, spanning handcrafted feature-based methods combined with ANN, KNN, and SVM, as well as custom convolutional neural networks and widely adopted transfer learning approaches. In addition, state-of-the-art architectures for object detection and instance segmentation were evaluated.

For classification, results across both datasets showed that HOG-based methods performed best among handcrafted approaches, while deep learning models consistently achieved higher accuracy. On the A Large-Scale Fish Dataset, CNN Model 1 achieved the highest accuracy (0.9847), with MobileNetV2 and InceptionV3 offering efficient accuracy–complexity trade-offs. On the Fish Recognition Ground-Truth Dataset, transfer learning models performed best, with ResNet-18 and EfficientNet-B0 achieving accuracies of 0.99 and 0.98, respectively, and decision-level fusion further improving performance to 100% validation accuracy.

For segmentation, DeepLabV3 achieved the most accurate results on the A Large-Scale Fish Dataset, while YOLOv8 provided the best balance between segmentation accuracy and inference speed on the Fish Recognition Ground-Truth Dataset, making it suitable for real-time use.

For object detection, a clear accuracy–efficiency trade-off was observed. RT-DETR achieved the highest localization accuracy, whereas YOLOv8n offered comparable accuracy with much faster inference on the A Large-Scale Fish Dataset. On the Aquarium Dataset, YOLOv5 achieved the highest detection accuracy, while YOLOv11 (update 1) delivered the fastest inference with high precision.

9.2 Future Work

Despite the excellent performance that has been obtained, there are some areas that could be further explored for improvement in the future. Firstly, by performing more intensive fine-tuning of the models that utilize transfer learning, for example, by step unfreezing more layers of both ResNet-18 and EfficientNet-B0 models, as well as incorporating learning rate scheduling, the performance of the models could be optimized for classification tasks. Secondly, the models could be tested on bigger data sets that would include pictures taken in other underwater settings.

Concerning the segmentation and detection tasks, for example, future research could encompass cross-validation or multiple experiments using various random seeds, if computational infrastructure is feasible, for more robust result outputs. Moreover, the analysis could extend to more agile segmentation models or attempt to prune or quantize current models for better deployment efficiency on low-resource platforms. Finally, integrating video analysis or real-time monitoring of the underwater setting would contribute significantly to real-world applications of the developed system.

10 References

[1] Roboflow, "Aquarium Dataset," Roboflow, 2020. [Online]. Available: https://public.roboflow.com/object-detection/aquarium. [Accessed November 2025].

[2] B. Dwyer, "How to Build Computer Vision Models with Apple CreateML," Roboflow, 18 Sep 2020. [Online]. Available: https://blog.roboflow.com/createml/?utm_source=chatgpt.com. [Accessed December 2025].

[3] B. B. B. a. R. B. Phoenix X. Huang, “Fish Recognition Ground-Truth data,” kaggle, 2012. [Online]. Available: https://www.kaggle.com/datasets/madhushreesannigrahi/fish-recognition-ground-truth-data/data. [Accessed November 2025].

[4] O. a. K. D. a. T. M. Ulucan, "A Large Scale Fish Dataset," kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/crowww/a-large-scale-fish-dataset/data. [Accessed November 2025].