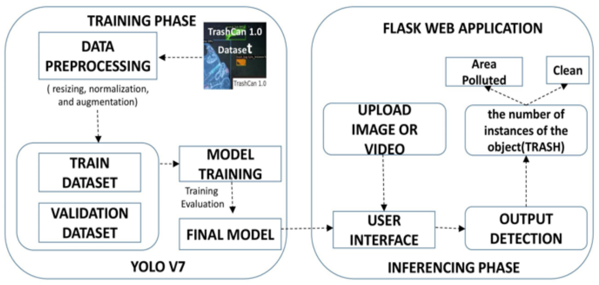
Global NEST Journal, Vol 27, No 4, 06077

Deep learning models for trash detection in underwater through image processing

Sairamesh L.[1]*, Selvakumar K.[2] and Sindhu T.[3]
[1,3]Department of CSE, St. Joseph's Institute of Technology, Chennai, India
[2]Department of Computer Application, NIT Trichy, India
Received: 18/04/2024, Accepted: 22/02/2025, Available online: 07/03/2025
*to whom all correspondence should be addressed: e-mail: sairamesh.ist@gmail.com
https://doi.org/10.30955/gnj.06077

Graphical abstract



Abstract

The increasing problem of underwater trash and its detrimental impact on marine ecosystems necessitates effective detection and mitigation strategies. This work presents an approach for underwater trash detection by integrating the YOLOv7 deep learning model with a Flask web application. The proposed system enables users to upload images or videos through the web application's user interface for real-time detection of underwater trash objects. To train the YOLOv7 model, a comprehensive dataset of annotated underwater trash images is curated, encompassing diverse types of marine debris commonly encountered in aquatic environments. The model is fine-tuned using this dataset to accurately recognize and localize underwater trash objects in real-time. The Flask web application serves as a user-friendly platform, allowing individuals to easily upload images or videos from their devices for analysis. Once uploaded, the application processes the media content using the trained YOLOv7 model. It enables the monitoring of marine pollution, empowers users to identify underwater trash hotspots, facilitates cleanup initiatives, and promotes awareness about the significance of preserving marine ecosystems. The user-friendly nature of the web application encourages active user participation and engagement in combating underwater trash. The system has the potential to aid in the preservation of marine environments by facilitating proactive efforts to mitigate the impact of underwater trash.

Keywords: Trash detection, YOLO V7, image processing, marine pollution, web application

1. Introduction

Image processing is a field that encompasses a wide range of techniques and algorithms aimed at analyzing and manipulating digital images. It plays a vital role in various domains. By utilizing sophisticated algorithms, image processing enables tasks such as image enhancement, restoration, segmentation, feature extraction, and object recognition. These techniques allow us to extract valuable information, improve image quality, and gain deeper insights into visual data. From noise reduction and image restoration to complex tasks like pattern recognition and image understanding, image processing provides powerful tools to analyze, interpret, and manipulate images. With the advancements in computational power and algorithmic techniques, image processing continues to evolve, contributing to numerous real-world applications and pushing the boundaries of visual understanding and image-based decision-making.

The problem addressed in this work is the need for an integrated solution that combines the YOLOv7 deep learning model with a Flask web application for underwater trash detection. With marine pollution becoming an increasing concern, there is a demand for an efficient and user-friendly system that allows users to upload images and videos for analysis, accurately detects and localizes underwater trash objects, and overcomes challenges specific to underwater imagery, such as varying lighting conditions and object occlusion.

By integrating YOLOv7 with a Flask web application, this work aims to empower marine biologists, researchers, and users to actively contribute to marine pollution mitigation efforts by detecting and reporting underwater trash. Ultimately, the proposed work seeks to raise awareness about the importance of preserving marine ecosystems and drive positive environmental change.

The aim of this work is to develop an integrated system that combines the YOLOv7 deep learning model with a Flask web application to enable efficient detection and localization of underwater trash in images and videos uploaded by users. This will empower marine biologists, researchers, and users to actively contribute to marine pollution mitigation efforts and raise awareness about the importance of preserving marine ecosystems through accurate trash detection and reporting.

Object detection refers to the task of identifying and locating objects within an image or video. It involves recognizing and localizing specific objects of interest in a given scene. Object detection algorithms leverage computer vision and machine learning techniques to analyze visual data, identify objects based on their characteristics or features, and generate bounding box coordinates to indicate the object's location in the image or video frame. Object detection is a fundamental technology in various applications, including autonomous driving, surveillance, robotics, and image understanding. It plays a crucial role in enabling machines to perceive and interact with their environment by detecting and recognizing objects of interest.

1.1. YOLOV7

YOLOv7 is an advanced object detection algorithm that stands for. You Only Look Once version 7. It is a deep learning model that achieves real-time object detection by simultaneously predicting object classes and their corresponding bounding box coordinates. YOLOv7 builds upon its predecessors and incorporates improvements in network architecture, feature extraction, and training techniques. It is known for its speed, accuracy, and versatility in detecting objects across various categories and in complex scenes. YOLOv7 is widely used in computer vision applications, including autonomous vehicles, surveillance systems, and robotics, where real-time object detection is crucial. Its efficiency and effectiveness make it a popular choice for tasks that require accurate object localization and recognition.

1.2. Flask

Flask is a popular web framework for building web applications in Python. It provides a simple and flexible development environment with a wide range of features to streamline web application development. With Flask, developers can efficiently create web applications by leveraging its user-friendly interface and powerful tools. Flask offers a lightweight and modular design, allowing developers to choose the components they need for their specific requirements. It provides a built-in development server, which makes it easy to test and debug applications locally. Flask also supports the use of templates, enabling developers to create dynamic and interactive web pages.

This article was organized under six sections. Section 2 describes the detail about literature survey and applications related to this work. Section 3 consists System design of the proposed work with its overall architecture diagram and modules. Section 4 discusses about the algorithms applicable for object detection and trash classification. Section 5 describes implementation of the proposed system with its performance analysis. Section 6 contains the conclusion of this work and also refers for future work.

2. Literature survey

The work done by J. C. Hipolito *et al*. (2021) was to better understand the current state of the field's knowledge and methods for detecting underwater marine plastic debris. The survey sought to identify the gaps in the existing research as well as the demand for a deep transfer learning technique and an upgraded low sample size dataset. The survey included a thorough analysis of pertinent academic papers, research articles, conference proceedings, and other works in the fields of machine vision systems and the identification of marine plastic garbage. Techniques for processing images, object detection algorithms, deep learning paradigms, transfer learning, and data augmentation strategies were some of the important topics investigated.

The work done by Bing *et al*. (2021) aimed to explore the existing research and methodologies related to deep-sea debris detection using deep neural networks. The survey was designed to better understand the difficulties in detecting trash in deep-sea habitats and the efficiency of deep learning methods in overcoming these difficulties. A thorough analysis of pertinent research papers, journal articles, conference proceedings, and related works in the fields of deep-sea debris identification and deep neural networks was done for the survey. Techniques for processing images, object detection algorithms, deep learning architectures, and the availability of labelled deep-sea trash datasets were some of the major topics investigated. Convolutional neural networks (CNNs) and recurrent neural networks (RNNs), among other deep learning architectures, were also recognised in the investigation as having been successfully used to detect debris in various underwater environments. In evaluating these architectures' benefits and drawbacks, accuracy, speed, and computing complexity were taken into account.

The investigation also looked at the accessibility of datasets of labelled deep-sea trash, which are crucial for developing and testing deep neural networks. The lack of such datasets was cited as a problem that prevented the development and benchmarking of deep-sea debris identification techniques. The survey also covered mitigation options for this problem, like data augmentation methods and cooperative data collection initiatives.

Chia-Chin (2019) aimed to explore the existing research and advancements in object detection using deep learning techniques specifically tailored for underwater environments. The survey was designed to better understand the difficulties posed by the distinctive features of the undersea environment and the efficiency of deep learning approaches in resolving these difficulties.

Techniques for processing images, object detection algorithms, deep learning architectures, and the availability of labelled underwater datasets were some of the major topics covered. According to the literature review, underwater environments provide a number of difficulties for object detection, including dim lighting, colour distortion, limited visibility, and complicated background conditions. In these difficult underwater settings, conventional computer vision techniques frequently struggle to deliver satisfactory results. Convolutional neural networks (CNNs) in particular have great promise for enhancing the precision and resilience of item detection in aquatic environments, according to the survey. The survey also found a number of deep learning architectures and algorithms, including YOLO, SSD, and Faster R-CNN, that have been successfully used for underwater object detection.

Wang *et al* (2021) aimed to explore the existing research and advancements in underwater object detection using the YOLOv4 architecture. The survey was designed to better understand the difficulties in underwater object recognition and the performance of the YOLOv4 model in those conditions.

A thorough analysis of pertinent research papers, journal articles, conference proceedings, and related works in the fields of underwater object detection and the YOLOv4 architecture was done as part of the survey. Deep learning architectures, underwater application-specific modifications to the YOLOv4 model, underwater imaging conditions, object detection algorithms, and other key areas were all thoroughly investigated. According to the literature review, difficulties with underwater object identification include dim lighting, low contrast, light dispersion, and complicated background conditions. The effectiveness of conventional object detection techniques can be considerably impacted by these variables. The survey did, however, draw attention to the potential of deep learning methods, particularly the YOLOv4 model, to enhance the precision and effectiveness of underwater object recognition.

The survey also revealed particular ways in which the YOLOv4 model had been enhanced for underwater applications. The network design, training methods, loss functions, or pre-processing procedures might all be altered as part of these advances. The survey evaluated how well these updates addressed the difficulties associated with underwater object detection and improved the YOLOv4 model's performance there.

Akshita *et al* (2019) aimed to explore the existing research and advancements in object detection in underwater images using edge detection techniques with adaptive thresholding. The survey's main goals were to comprehend the difficulties underwater imaging environments present and the efficiency of adaptive thresholding techniques in identifying object edges.

The difficulties of underwater imaging, edge detection algorithms, adaptive thresholding techniques, and their use for object detection in underwater photos were some of the key topics investigated. According to the literature review, underwater imaging settings can be difficult because of things like low visibility, colour distortion, light attenuation, and noise. The effectiveness of conventional object detection techniques that rely on colour or texture cues may be hampered by these issues. The investigation did draw attention to the ability of edge detection methods to capture item boundaries and improve the detection precision in underwater photos.

The search also uncovered different adaptive thresholding techniques and edge detection algorithms that have been effectively used to detect underwater objects. These strategies use adaptive thresholding to find significant edges in order to segregate objects from the background. The survey evaluated how well various techniques handled the difficulties of underwater object recognition and increased the precision of item location.

Edge detection was the important processing in image analysis. In underwater image anlayis also it plays a major role. Sivanesh *et al* (2023) proposed a system for detecting E-coli bacteria in water images. Sairamesh *et al*. (2021) used the contour based segmentation method for classifying the species of fish. Vatchala *et al* (2022) proposed a household object detection system using CNN model. This system identifies the object through image analysis using CNN. Soundarajan *et al* (2022) proposed a method for detecting abnormal activities in human through thermal images. Anusha *et al* (2018) proposed a system to recognize the food items and evaluate the calorie value of the recognized food item. This will be helpful for the diabetic patients to easily identify the dietary food for them. All these methodologies using deep learning models for image analysis and provide more efficiency.

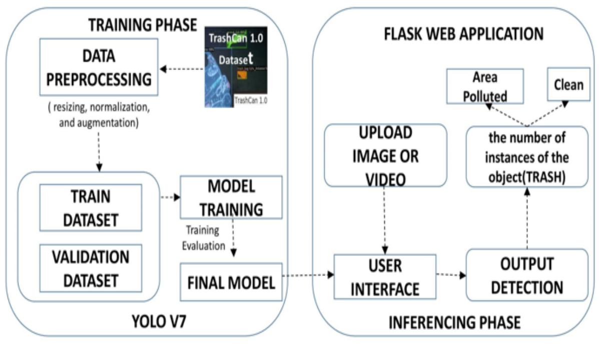


Figure 1. Proposed Architecture for Trash detection using YOLOV7

3. System design

The overall system architecture of the proposed work is shown in **Figure 1**.

3.1. Dataset

Trashcan 1.0 was the dataset taken for research purpose. It contains more than 7k annotated images of under water ocean images. The image contains the flora, fauna, different species of fish and different types of trashes. The imagery in TrashCan is sourced from the J-EDI (JAMSTEC E-Library of Deep-sea Images) dataset, curated by the Japan Agency of Marine Earth Science and Technology (JAMSTEC). The eventual goal is to develop efficient and accurate trash detection methods suitable for onboard robot deployment.

3.2. Data preprocessing

Data preprocessing plays a crucial role in training YOLO (You Only Look Once) models effectively. The process involves several steps to prepare the data in a format suitable for YOLO's requirements. Firstly, the object annotations need to be converted into YOLO format, which includes the object class and normalized bounding box coordinates. Next, all the images in the dataset should be resized to a consistent size to ensure compatibility during training. It's essential to split the dataset into training and validation sets to assess the model's performance accurately. Class label encoding is performed to assign unique numerical labels to each object class present in the dataset. Additionally, anchor boxes, which determine default bounding box sizes, can be generated using clustering algorithms. Normalizing the pixel values of the images to a common scale, typically ranging from 0 to 1, is crucial for stable training. Finally, text files are created, containing the file paths to the preprocessed images and their corresponding annotations, to serve as inputs during training. These preprocessing steps ensure that the data is properly formatted and ready to be used for training a YOLO model.

3.2.1. Model training

The training phase of the YOLOv7 (You Only Look Once) model is a critical step in building an accurate object detection system. This phase involves several key steps to train the model on a specific dataset. The first step is to set up the YOLOv7 configuration. This includes defining the network architecture, selecting appropriate hyperparameters, specifying the input image size, determining the number of object classes, and setting the anchor box sizes. The configuration should be adjusted based on the requirements.

Once the configuration is established, the training data needs to be prepared. This involves ensuring that the dataset is properly preprocessed. The object annotations should be formatted in the YOLO format, including the class label and normalized bounding box coordinates. The images should also be resized to a consistent size for compatibility during training. Additionally, text files need to be created to list the file paths of the training images and their corresponding annotations. The next step is to initialize the YOLOv7 model. This can be done by either using randomly initialized weights or utilizing pre-trained weights from a similar dataset, such as COCO. Pre-trained weights provide a starting point for the model and can help speed up the convergence process.

With the model initialized, the training process begins. During training, the model is fed with the training images and their corresponding annotations. The model's parameters are optimized to minimize the loss function, typically using techniques like stochastic gradient descent (SGD) or Adam optimization. YOLOv7 often incorporates a multi-scale approach, where the model is trained on different image scales to improve object detection accuracy. Throughout the training process, the model iteratively adjusts its parameters, learning to detect objects more accurately in the given dataset. The training phase typically involves multiple epochs, with each epoch consisting of forward and backward passes through the network.

3.2.2. Training evaluation

When evaluating the performance of the YOLOv7 model, two commonly used metrics are mean average precision (mAP) and accuracy. Mean Average Precision (mAP): mAP is a widely used metric for evaluating object detection models. It measures the precision-recall trade-off and provides an overall assessment of the model's performance across different object classes and detection thresholds. The mAP score is calculated by considering the precision and recall values at various IoU (Intersection over Union) thresholds and averaging them. A higher mAP indicates better object detection performance. **Accuracy:** Accuracy is a metric that measures the overall correctness of the model's predictions. In the context of YOLOv7, accuracy refers to how well the model correctly detects and classifies objects in the test dataset. It is calculated as the ratio of the number of correctly predicted objects (both bounding box and class) to the total number of objects in the dataset. Accuracy provides a single numerical value to gauge the model's overall performance, but it does not provide insights into class-wise performance or the precision-recall trade-off.

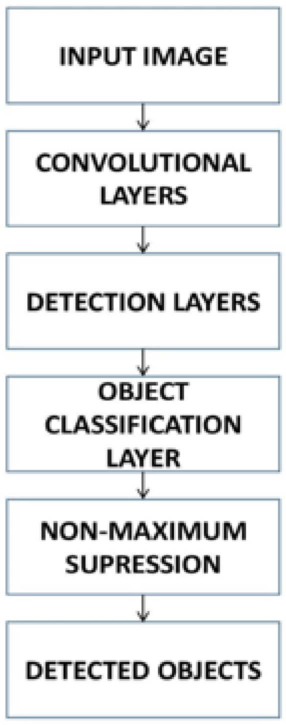


Figure 2. Object Detection flow

3.2.3. Model output

The final output of YOLOv7 (You Only Look Once) is a set of bounding boxes, along with their corresponding class labels and confidence scores, representing the detected objects in an input image as shown in the **Figure 2**. Each bounding box consists of four coordinates (x, y, width, height) that define the position and size of the detected object within the image. The class label indicates the category or class of the object, such as" trash", animal, or plant. The confidence score reflects the model's confidence in the accuracy of the detection, with higher scores indicating higher confidence.

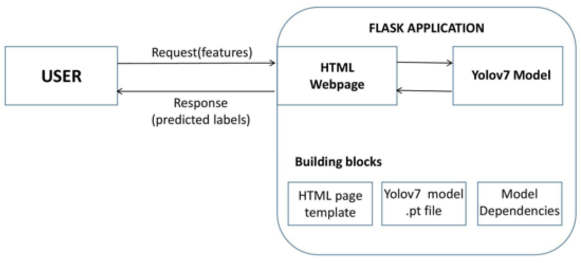


Figure 3. Flask Architecture

Flask application

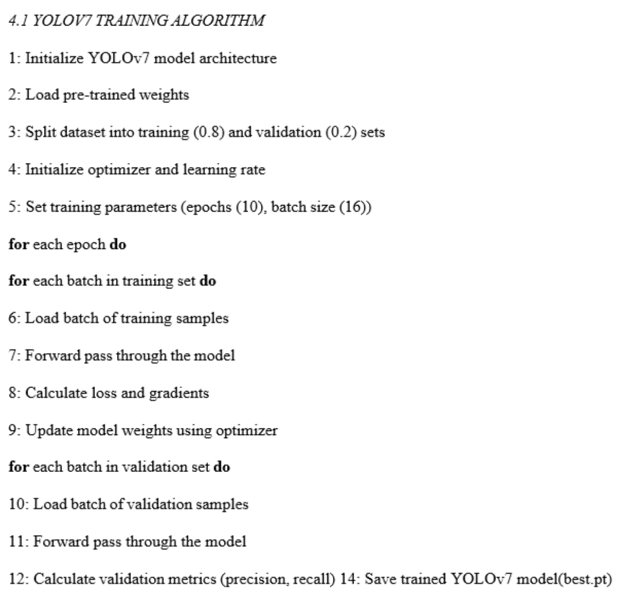
In this Flask application shown in **Figure 3**, the necessary modules, including Flask and the request module for handling file uploads are imported initially. Once the Flask app was initialized the YOLOv7 model was configured for underwater trash detection. The route corresponds to the home page, which is rendered using the index.html template. This template typically contains an HTML form that allows users to select and upload an image or video file.

The upload route is triggered when the user submits the form with a file. The uploaded file is retrieved from the request using request files ['file']. The file is saved to a temporary location in the uploads folder. Next, the saved file is passed to the YOLOv7 model for underwater trash detection. The specifics of this step depend on the implementation of the YOLOv7 model and may involve loading the model, preprocessing the image or video, and performing object detection.

Once the results are detected and processed, it will be displayed to the user. The results.html template can be customized to show the detected objects, their bounding boxes, and any relevant information or visualizations. Finally, run the Flask app with app.run(debug=True) to start the server.

4. Algorithms

YOLOv7, or You Only Look Once version 7, is a state-of-the-art deep learning algorithm for object detection. It builds upon the success of its predecessors, YOLOv6 and YOLOv5, and aims to improve the accuracy and efficiency of object detection tasks. YOLOv7 follows the one-stage object detection approach, where it predicts bounding boxes and class probabilities directly from input images in a single pass. It utilizes a deep convolutional neural network architecture with feature extraction and detection layers to enable real-time and accurate object detection. YOLOv7 introduces various improvements, including advanced backbone networks, feature pyramid networks, and anchor-free bounding box prediction techniques. It utilizes advanced techniques for precise object localization. It employs regression to predict accurate bounding box coordinates, allowing it to precisely locate objects within an image. It is designed to be flexible and adaptable to different object detection scenarios. It can be fine-tuned on specific datasets or domains to achieve better performance on specific object classes or environmental conditions. It can be deployed on a wide range of platforms, including desktop computers, embedded systems, and even mobile devices. Its efficiency and accuracy make it suitable for real-time applications with limited computational resources.



4.1 YOLOV7 TRAINING ALGORITHM

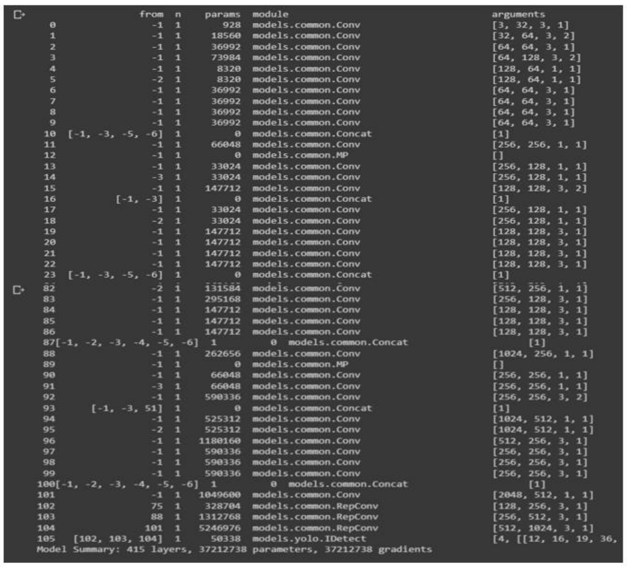
```
1: Initialize YOLOv7 model architecture
2: Load pre-trained weights
3: Split dataset into training (0.8) and validation (0.2) sets
4: Initialize optimizer and learning rate
5: Set training parameters (epochs (10), batch size (16))
for each epoch do
    for each batch in training set do
        6: Load batch of training samples
        7: Forward pass through the model
        8: Calculate loss and gradients
        9: Update model weights using optimizer
    for each batch in validation set do
        10: Load batch of validation samples
        11: Forward pass through the model
        12: Calculate validation metrics (precision, recall) 14: Save trained YOLOv7 model(best.pt)
```

YOLOv7 training algorithm which consists of several steps to train a YOLOv7 model for object detection. Firstly, the YOLOv7 model architecture is initialized, specifying the layers and filters. Pre-trained weights can be loaded to leverage prior knowledge. The dataset is then split into training and validation sets, with an 80:20 ratio. An optimizer and learning rate are initialized for weight updates during training. Training parameters like the number of epochs (set to 10) and batch size (set to 16) are defined. During each epoch, the algorithm iterates over batches of training samples. For each batch, the samples are loaded and passed through the model, followed by calculating the loss and gradients for weight updates. The model's weights are then updated using the optimizer. Similarly, batches of validation samples are loaded and passed through the model to calculate validation metrics, such as precision and recall. Optionally, the learning rate can be adjusted during training. Finally, the trained YOLOv7 model, represented by the weights file best.pt, is saved for future use. This algorithm provides a systematic approach to train the YOLOv7 model and improve its object detection capabilities.

YOLOv7 Object Detection Algorithm for detecting objects in images. Firstly, a pre-trained YOLOv7 model is loaded, which has been trained on a large dataset and learned to recognize various objects. Then, an input image is loaded, and the algorithm preprocesses it by resizing it to a specific size, typically 640x640 pixels. Next, the preprocessed image is passed through the YOLOv7 model, which applies deep learning techniques to analyze the image and make predictions. The algorithm retrieves the predicted bounding boxes and class labels, which represent the objects detected in the image. To filter out overlapping detections, non-maximum suppression is applied, ensuring that only the most relevant and accurate detections remain. The algorithm then iterates over each detected object, retrieving its coordinates and class label. It draws a bounding box around the object and labels it accordingly on the image. Finally, the image with the annotated bounding boxes and labels is displayed, providing a visual representation of the detected objects.

4.2 YOLOV7 OBJECT DETECTION ALGORITHM

```
1: Load pre-trained YOLOv7 model(best.pt)
2: Load input image
3: Preprocess the image (resize img 640x640)
4: Pass the image through the YOLOv7 model
5: Retrieve predicted bounding boxes and class labels(animal,plant,rov,trash)
6: Apply non-maximum suppression to filter out overlapping detections
for each detected object do
7: Retrieve object coordinates and class label
8: Draw bounding box and label on the image
9: Display the image with detected objects
```

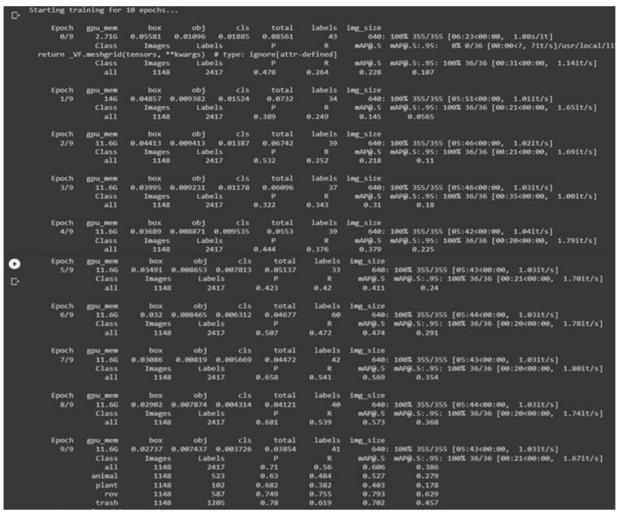


```
                 from  n    params  module                                  arguments
  0                -1  1       928  models.common.Conv                      [3, 32, 3, 1]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1      8320  models.common.Conv                      [128, 64, 1, 1]
  5                -2  1      8320  models.common.Conv                      [128, 64, 1, 1]
  6                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  7                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  8                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  9                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
 10  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
 11                -1  1     66048  models.common.Conv                      [256, 256, 1, 1]
 12                -1  1         0  models.common.MP                        []
 13                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 14                -3  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 16          [-1, -3]  1         0  models.common.Concat                    [1]
 17                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 18                -2  1     33024  models.common.Conv                      [256, 128, 1, 1]
 19                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 20                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 21                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 22                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 23  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
 82                -2  1    131584  models.common.Conv                      [512, 256, 1, 1]
 83                -1  1    295168  models.common.Conv                      [256, 128, 3, 1]
 84                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 85                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 86                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 87[-1, -2, -3, -4, -5, -6]  1         0  models.common.Concat                [1]
 88                -1  1    262656  models.common.Conv                      [1024, 256, 1, 1]
 89                -1  1         0  models.common.MP                        []
 90                -1  1     66048  models.common.Conv                      [256, 256, 1, 1]
 91                -3  1     66048  models.common.Conv                      [256, 256, 1, 1]
 92                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 93       [-1, -3, 51]  1         0  models.common.Concat                    [1]
 94                -1  1    525312  models.common.Conv                      [1024, 512, 1, 1]
 95                -2  1    525312  models.common.Conv                      [1024, 512, 1, 1]
 96                -1  1   1180160  models.common.Conv                      [512, 256, 3, 1]
 97                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 98                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 99                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
100[-1, -2, -3, -4, -5, -6]  1         0  models.common.Concat                [1]
101                -1  1   1049600  models.common.Conv                      [2048, 512, 1, 1]
102                75  1    328704  models.common.RepConv                   [128, 256, 3, 1]
103                88  1   1312768  models.common.RepConv                   [256, 512, 3, 1]
104               101  1   5246976  models.common.RepConv                   [512, 1024, 3, 1]
105   [102, 103, 104]  1     50338  models.yolo.IDetect                     [4, [[12, 16, 19, 36,
Model Summary: 415 layers, 37212738 parameters, 37212738 gradients
```

Figure 4. Model Summary

5. Results and analysis

The **Figure 4** indicates that the YOLOv7 model consists of 415 layers in total. This includes various convolutional layers, pooling layers, and fully connected layers. The model has a total of 37,212,738 parameters, which are the learnable weights and biases in the model. These parameters are updated during the training process to optimize the model's performance on the given task. The number of gradients is also mentioned as 37,212,738. Gradients represent the derivatives of the loss function with respect to each parameter in the model. These gradients are computed during the backward pass of the training process and used to update the model's parameters through gradient descent or a similar optimization algorithm. The complexity and size of the YOLOv7 model, as well as the number of parameters and gradients that play a crucial role in the training and optimization process.



```
Starting training for 10 epochs...

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       0/9     2.71G   0.05581   0.01096   0.01885   0.08561        43       640: 100% 355/355 [06:23<00:00,  1.08s/it]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95:   0% 0/36 [00:00<?, ?it/s]/usr/local/li
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:31<00:00,  1.14it/s]
                 all      1148      2417     0.478     0.264     0.228     0.107

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       1/9       14G   0.04857  0.009382   0.01524    0.0732        34       640: 100% 355/355 [05:51<00:00,  1.01it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:21<00:00,  1.65it/s]
                 all      1148      2417     0.389     0.249     0.145    0.0565

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       2/9     11.6G   0.04413  0.009413   0.01387   0.06742        39       640: 100% 355/355 [05:46<00:00,  1.02it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:21<00:00,  1.69it/s]
                 all      1148      2417     0.532     0.252     0.218      0.11

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       3/9     11.6G   0.03995  0.009231   0.01178   0.06096        37       640: 100% 355/355 [05:46<00:00,  1.03it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:35<00:00,  1.00it/s]
                 all      1148      2417     0.322     0.343      0.31      0.18

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       4/9     11.6G   0.03689  0.008871  0.009535    0.0553        39       640: 100% 355/355 [05:42<00:00,  1.04it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:20<00:00,  1.79it/s]
                 all      1148      2417     0.444     0.376     0.379     0.225

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       5/9     11.6G   0.03491  0.008653  0.007813   0.05137        33       640: 100% 355/355 [05:43<00:00,  1.03it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:21<00:00,  1.70it/s]
                 all      1148      2417     0.423      0.42     0.411      0.24

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       6/9     11.6G     0.032  0.008465  0.006312   0.04677        60       640: 100% 355/355 [05:44<00:00,  1.03it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:20<00:00,  1.78it/s]
                 all      1148      2417     0.507     0.472     0.474     0.291

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       7/9     11.6G   0.03086   0.00819  0.005669   0.04472        42       640: 100% 355/355 [05:43<00:00,  1.03it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:20<00:00,  1.80it/s]
                 all      1148      2417     0.658     0.541     0.569     0.354

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       8/9     11.6G   0.02902  0.007874  0.004314   0.04121        40       640: 100% 355/355 [05:44<00:00,  1.03it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:20<00:00,  1.74it/s]
                 all      1148      2417     0.681     0.539     0.573     0.368

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
       9/9     11.6G   0.02737  0.007437  0.003726   0.03854        41       640: 100% 355/355 [05:43<00:00,  1.03it/s]
               Class    Images    Labels         P         R    mAP@.5  mAP@.5:.95: 100% 36/36 [00:21<00:00,  1.67it/s]
                 all      1148      2417      0.71      0.56     0.606     0.386
              animal      1148       523      0.63     0.484     0.527     0.279
               plant      1148       102     0.682     0.382     0.403     0.178
                 rov      1148       587     0.749     0.755     0.793     0.629
               trash      1148      1205      0.78     0.619     0.702     0.457
```

Figure 5. Model Training

The **Figure 5** indicates the training progress of the YOLOv7 model over the course of 10 epochs. Each epoch represents a complete pass through the entire training dataset. For each epoch, the output displays the GPU memory usage, average losses for bounding box regression (box), object prediction (obj), and class prediction (cls), as well as the overall total loss. These values indicate the training progress and the optimization of the model's parameters. After each epoch, the model's performance on the validation dataset is evaluated. The output provides evaluation metrics such as precision (P), recall (R), mean average precision (mAP) at different intersection over union (IoU) thresholds (mAP@0.5, mAP@0.5:0.95). These metrics reflect the model's ability to detect objects accurately and generalize well to unseen data. Furthermore, the output includes metrics for each specific class (e.g., animal, plant, rov, trash), including precision, recall, and mAP.These class-specific metrics give insights into the model's performance on individual classes and can help identify areas that require improvement. The training process takes approximately 1.047 hours to complete all 10 epochs. By monitoring the losses and evaluation metrics throughout the epochs, one can assess the model's progress and make adjustments if necessary, such as modifying the learning rate or applying regularization techniques, to further enhance the model's performance.

The **Figure 6** represents a visual representation of the performance of a classification model. The rows represent the actual (true) values of the classes, while the columns represent the predicted values of the classes. The values within the matrix represent the proportions or percentages of instances that were classified into each class. The confusion matrix provides a comprehensive overview of the model's performance in predicting different classes. By examining the values in each cell, we can assess the accuracy, precision, recall, and other evaluation metrics for each class. This information helps in understanding how well the model is performing for each specific class and can guide further improvements in the model or data collection process.

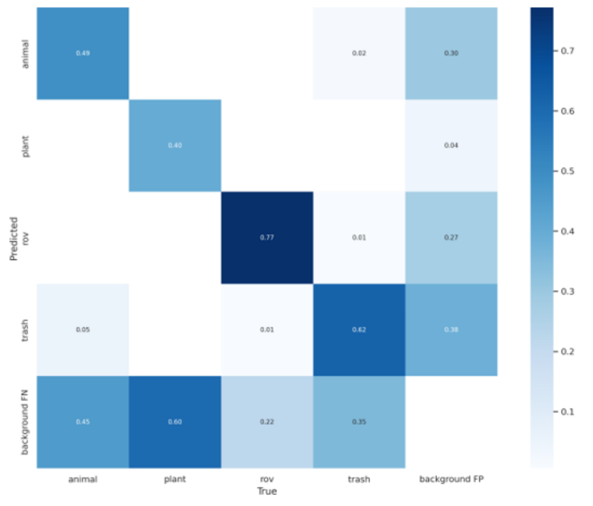


Figure 6. Confusion Matrix

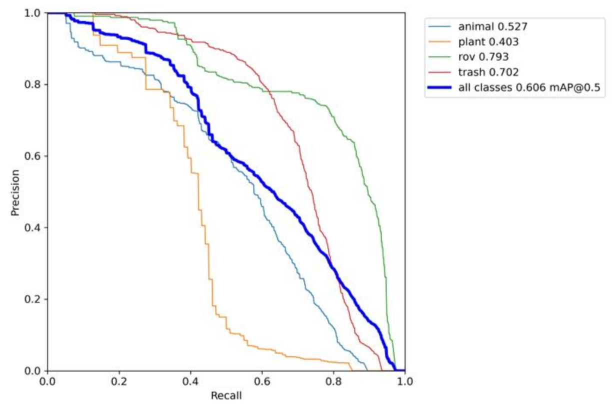


Figure 7. PR CURVE

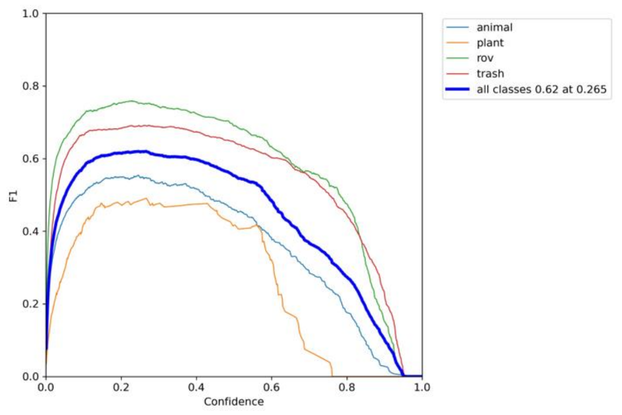


Figure 8. F1 CURVE

The **Figure 7** represents a visual representation of the trade-off between precision and recall, offering insights into the overall model performance across all classes. The precision-recall curve analysis reveals the performance of an object detection model for different classes. For the animal class, the model achieved a precision of 0.5, indicating that when it predicted an object as animal, it was correct in 50 of the cases. Similarly, for the plant class, the precision was 0.4, indicating a 40 accuracy in correctly identifying objects as plant. The rov class showed a higher precision of 0.79, suggesting a relatively stronger ability to accurately predict objects as rov. The trash class had a precision of 0.70, indicating a 70 accuracy in correctly classifying objects as trash.

The **Figure 8** provides a balanced assessment of the model's ability to correctly identify positive samples while minimizing false positives and false negatives. A value of 0.62 suggests a moderate level of performance in terms of precision and recall trade-off. It indicates that the model achieves a reasonable balance between accurately classifying positive samples and minimizing incorrect predictions across all classes.

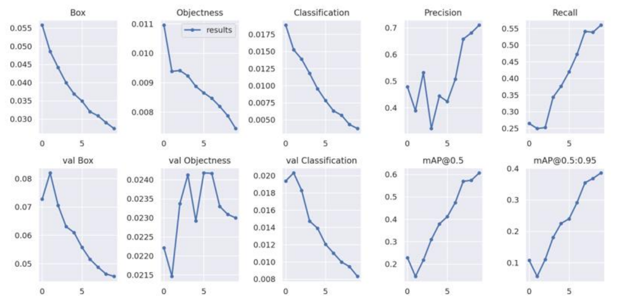


Figure 9. Evaluation Results

The **Figure 9** displays various important components that assess the performance of an object detection model. The Box component visually represents the predicted bounding boxes around detected objects in the image, indicating their estimated locations. The Objectness component indicates the confidence or probability assigned to each bounding box, serving as a measure of the model's certainty in identifying objects versus background regions. The Classification component assigns predicted class labels to the objects within the bounding boxes, allowing for categorization based on the model's training. The Precision metric quantifies the accuracy of the model in correctly identifying objects by measuring the proportion of correctly predicted positive samples among all predicted positives. The Recall metric gauges the model's ability to detect all instances of objects by calculating the proportion of correctly predicted positives among all actual positives. Lastly, the mAP (Mean Average Precision) assesses the overall performance of the model by averaging the precision values at various objectless thresholds.

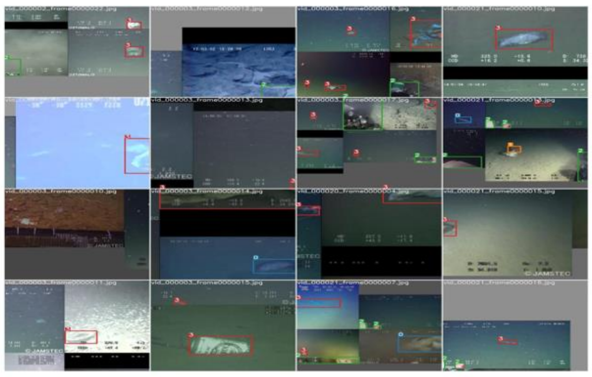


Figure 10. Training Results

The **Figure 10** shows the training results of the yolov7 model and **Figure 11** shows the predicted results with value.

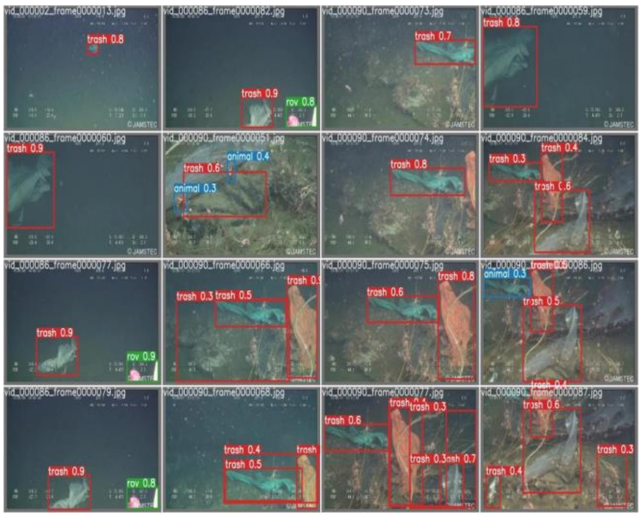


Figure 11. Testing Results

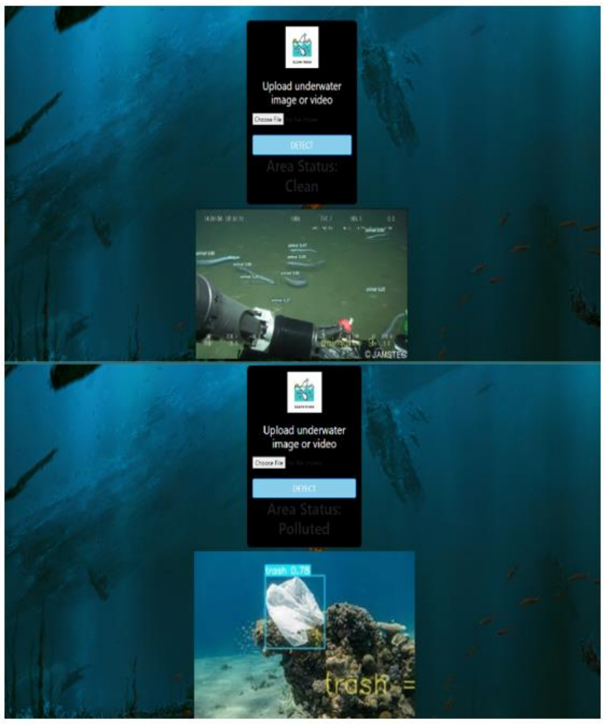


Figure 12. Trash detection using Flash application

5.1. Flask application

The **Figure 12** shows the detection results of the image that is uploaded by the user and the area status. The area status will be shown based upon the detection of trash, if no trash is detected the area status will be clean.

6. Conclusion

This proposed system presents a comprehensive solution for underwater trash detection by integrating the YOLOv7 deep learning model with a Flask web application. The system allows users to upload images or videos through the user-friendly web interface, enabling real-time detection of underwater trash objects. The YOLOv7 model is trained using a curated dataset of annotated underwater trash images, ensuring accurate recognition and localization of marine debris. The integration of YOLOv7 with the Flask web application offers several benefits, including the monitoring of marine pollution, identification of trash hotspots, and facilitation of cleanup initiatives. By promoting awareness about the importance of preserving marine ecosystems, the system encourages active user participation in combating underwater trash. Overall, this work provides a practical and effective solution for real-time underwater trash detection, contributing to the preservation of marine environments and the mitigation of the detrimental impact of underwater trash.

This work can expand the object categories beyond underwater trash detection. This would involve incorporating the capability to detect and classify other marine objects, organisms, or specific types of pollution. The overall accuracy achieved by the proposed system was 0.91 whereas the existing works was not more than 0.87. But still the accuracy need to improve to 0.99. The system accuracy was reduced without the preprocessing steps and it also require additional times if we are executing with preprocessing. Improving the accuracy of object detection without preprocessing by using advanced deep learning models in challenging underwater environments is also a key area of focus in future. Exploring techniques such as data augmentation, advanced network architectures, and domain-specific knowledge can contribute to enhancing the model's performance. By addressing these areas, the system can contribute to more effective and comprehensive underwater trash detection and play a vital role in marine ecosystem preservation efforts in future.

References

Anusha B., Sabena S. and Sairamesh L. (2018). Optimized Food Recognition System for Diabetic Patients, In *Smart and Innovative Trends in Next Generation Computing Technologies: Third International Conference, NGCT,* Springer Singapore, 504–525.

Bing X., Huang B., Wei W., Chen G., Li H., Zhao N. and Zhang H. (2021), An efficient deep-sea debris detection method using deep neural networks. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, **14**, 12348–12360.

Hao W. and Xiao N. (2021), Research on underwater object detection based on improved YOLOv4. In *2021 8th International Conference on Information, Cybernetics, and Computational Social Systems (ICCSS)*, 166–171, IEEE.

Hipolito J.C., Alon A.S., Amorado R.V., Fernando M.G.Z. and De Chavez P.I.C. (2021). Detection of Underwater Marine Plastic Debris Using an Augmented Low Sample Size Dataset for Machine Vision System: A Deep Transfer Learning Approach. In *2021 IEEE 19th Student Conference on Research and Development (SCOReD)*, IEEE, 82–86.

Sai Ramesh L., Rangapriya C. N., Archana M., and Sabena S. (2021), Multi-Scale Fish Segmentation Refinement Using Contour Based Segmentation, *Advances in Parallel Computing Technologies and Applications*, ***40***, 358–369.

Saini A. and Biswas M. (2019), Object detection in underwater image by detecting edges using adaptive thresholding. In

2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 628–632). IEEE.

Sivanesh S., Glory Sangeetha R., Mani G. and Sairamesh L. (2023), Detection and Classification ff E. Coli Bacteria in Water Using Bacterial Foraging Algorithm, *Journal of Environmental Protection and Ecology*, **24**,1519–1527.

Soundararajan K., Soundararajan A. and SaiRamesh L. (2022), Abnormality Detection in Human Action Using Thermal Videos, *Advances in Parallel Computing Algorithms, Tools and Paradigms*, ***41***, 449.

Vatchala S., Sasidevi S., Dhanlakshmi R. and SaiRamesh L. (2022), Smart Household Object Detection Using CNN, *Advances in Parallel Computing Algorithms, Tools and Paradigms*, ***41***, 464.

Wang C.C., Samani H. and Yang C.Y. (2019), Object detection with deep learning for underwater environment, In *2019 4th International Conference on Information Technology Research (ICITR)*, 1–6, IEEE.