DEEP LEARNING FOR AUTOMATIC RECOGNITION OF BOATS AND SHIPS TO AVOID COLLISIONS IN MARINE TRANSPORT

Abstract

The objective of this work is to explore the use of deep learning for the automatic recognition of ships and boats. The study uses a deep convolution neural network (DCNN) architecture to analyze a large data set of ship and boat images. The data set is prepared and labelled for supervised learning, and CNN is fine-tuned using the YOLO series group of versions 5, 6 and 8 to improve recognition accuracy. The proposed method involves training a deep convolution neural network on a large data set of ship and boat images and using the trained model to classify new images. The performance of the model is evaluated on a separate test set and compared to other state-of-the-art methods. The results of this study show that the deep learning model is effective in automatically recognizing ships and boats with an accuracy of mAP (Mean Average Precision). The model's performance is also compared to traditional machine learning algorithms, and CNN outperforms these methods. In this work we present the architecture, design and implementation of an object detection model deployed on an IMX8M Plus hardware board, to be used on the gathered image data model to recognize and label the ships and boats at the edge. We conduct transfer learning on the state-of-the-art trained YOLO model by introducing a labelled BS(boat-ship) image data set. We use the trained model to do predictions on a test image set to evaluate the model's performance. The result of the model can predict labels with an accuracy of 72.1 mAP of YOLOv8 and inference FPS time that detect it to do so in real-time with the board. Results show that the proposed deep learning approach outperforms existing methods, achieving high accuracy, and demonstrating the

Authors

Abhishek Barandooru Janavejirao
Master's in Data Science
Department of Physics
Polytechnic Science School
University of Naples Federico II,
Italy

Longo Giuseppe
Professor & Head of Data Science
Department of Physics
Polytechnic Science School
University of Naples Federico II,
Italy

potential of deep learning for the automatic recognition of boats and ships in marine environment system.

Keywords: Computer Vision, Deep Learning, Image Processing, Boats and Ships, Detection, Classification

I. INTRODUCTION

The ability of computers to perceive and analyze visual input from their surroundings is referred to as computer vision. Computer vision can be useful in navigation, object recognition, and situational awareness in the setting of marine vehicles. Here are some of the most important aspects of computer vision in maritime vehicles:

Navigation: Computer vision techniques can be used to assist marine vehicles in navigating the sea. Cameras and other sensors, for example, can be used to detect and track things like buoys, landmarks, and other vessels, allowing the vehicle to remain on course and prevent collisions. Object Detection: A computer vision technique can be used to detect and classify items in and out of the water, such as marine vehicles, marine life, debris, and other objects. hazards. This may help the vehicle in avoiding crashes and navigating through the water safely. Situational Awareness: Computer vision can provide real-time information about its surroundings, assisting the vehicle in understanding and responding to changing conditions. Cameras and sensors, for example, can be utilized to detect vehicles, weather conditions, sea state, and other environmental aspects that may affect vehicle performance [1].

Autonomous Operations: Computer vision can be applied to enable autonomous operations of marine vehicles such as unmanned surface vessels and underwater items. These cars can operate with minimum human interaction by employing cameras and sensors to navigate and sense the environment. Overall, computer vision in object recognition has the potential to dramatically improve the capabilities of maritime vehicles, boosting their safety, efficiency, and effectiveness in a wide range of applications. Boats and ships play important roles in a variety of activities such as transportation, trading, fishing, and military operations. The capacity to recognize boats and ships automatically from photos and videos can be beneficial in a variety of applications, including safety and security, maritime surveillance, search and rescue operations, and oceanography. However, manually recognizing boats and ships takes time and might be mistake prone. As a result, automatic boat and ship recognition has become a hot study topic in recent years. Deep learning, a branch of machine learning, has produced encouraging results in a variety of computer vision applications, including picture categorization. Deep convolutional neural networks (DCNNs) have been found to be effective at object recognition from photos. We investigate the use of deep learning approaches for automatic recognition of boats and ships from photographs in this work. We present a deep learning technique for classifying and detecting new photos that involves training a CNN object recognition algorithm on a large dataset of ship and boat photographs and then utilizing the learned model to classify and detect new images. The remainder of this work is structured as follows: Section 2explains the suggested deep learning strategy for recognizing boats and ships. Section 3 presents experimental results and compares the suggested method to existing cutting-edge methods. Finally, in Section last, we wrap up the paper and outline future directions in this field [12].

Boat and ship recognition is an essential field of study and development for marine vehicles since it can help with navigation, collision avoidance, and other duties. The most advanced object recognition technology for boats and ships employs computer vision techniques and machine learning algorithms to detect and classify items on or near the water. Deep learning for autonomous object detection of boats and ships at the cutting edge involves the use of convolutional neural networks (CNNs) and other advanced machine learning methods to recognize and classify items on or near the water. These techniques are highly effective in recognizing different types of boats and ships in images and video. One key development in this field is the use of deep learning algorithms, such as convolutional neural networks (CNNs), to recognize objects in images or videos. These approaches are quite good in identifying various types of boats and ships in photos and video. The use of deep learning techniques, such as convolutional neural networks (CNNs), to recognize objects in photos or videos is a significant advancement in this subject. These algorithms were trained on a huge set of photos of boats and ships, as well as other objects on the sea, such as buoys, markers, and other vessels. CNNs can learn to recognize different types of boats and ships with a high degree of accuracy by analyzing the features and patterns in these photos. Another important innovation is the use of transfer learning, which involves changing pre-trained deep learning models to recognize boats and ships. Developers and researchers can obtain high accuracy with fewer training examples by finetuning these algorithms on new datasets of marine photos[2].

The use of sensors and cameras for object detection in aquatic environments is a novel discovery. These sensors and cameras are capable of detecting things on or near the water, even in low-light settings, and can offer crucial data to navigation and collision avoidance systems. Deep learning for autonomous object detection of boats and ships, in general, entails training CNN to recognize various types of boats and ships based on patterns and features in photos and video. These strategies have the potential to drastically enhance the safety and efficiency of marine vehicles with future research and development. Methods for object detection developed by researchers over the last 20 years improved in accuracy and many other qualities from year to year [3].

1. **Motivation:** In today’s world, there is more improvement in the automobile industry in all the transport ways of “Air, Road and Sea” ways. Now it is evolving from manual to automation techniques. And due to faster advanced technological developments and safety purposes reasons. This project is related to seaways marine environment for the recognition of ships and boats objects. Object recognition is to maintain the safety order to avoid a collision. Detect objects for betterment in safety concerns to save the people and boats in a marine environment. This project contributes to the field of the marine and automobile industry. Potential increases in theoretical, practical implementation, policies, and safety. The outcome of the project algorithm is to detect objects in the real world using a camera. Detection using algorithms gives importance to marine transport to maintain safe, and security and to avoid collisions. The above figure is the prototype design of ship and boat detection in real-world time in a marine environment.

II. SYSTEM DESIGN& METHODOLOGY

2. **Network Architecture:** Object detection models are classified into two types: two-stage object detectors and single-stage object detectors. A single-stage detector is a computer vision object identification system that can find things in photos or videos by evaluating a single, complete image in one shot. Convolutional neural networks (CNNs) are capable of learning sophisticated feature representations from images and are used in the most common single-stage detectors[4]. Deep CNNs are commonly used in these detectors, and they are trained on a large dataset of annotated photos to learn the visual properties of objects. A single-stage detector works by dividing the image into a number of rectangular sections called anchors or priors that are centered at different points and have different

sizes and aspect ratios. CNN is then applied to each of these anchors to predict whether or not it contains an object and, if so, what class and location the object belongs to [5] [13].

The YOLO (You Only Look Once) detector is a popular single-stage detector. YOLO is quick and accurate, detecting objects in real time on a single GPU. SSD (Single Shot Multibox Detector) is another common single-stage detector that takes a similar method but has a different network architecture. To create dense predictions, single-stage object detector architectures (such as YOLO) are made up of three components: a backbone, Neck, and Head. Because they only require a single forward pass of the network, single-stage detectors are quick and efficient. They are, however, less precise than two-stage detectors, which generate a set of region proposals before identifying objects within those regions. A single-stage detector's architecture typically consists of three major components: the backbone network, the feature pyramid network (FPN), and the detection head. A high-level overview of each component follows:

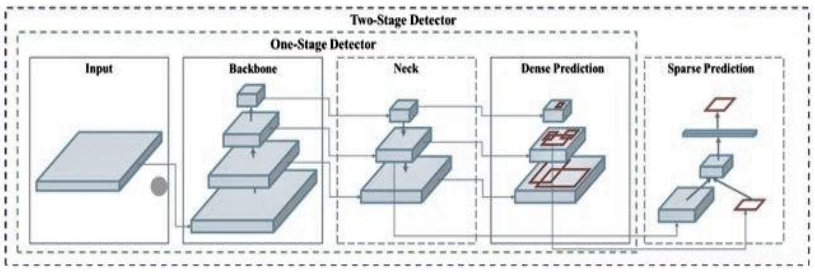


Figure 2.1: Single Stage Detector

- **Backbone network:** Because of their effectiveness in extracting information from input images, convolutional neural networks (CNNs) are widely used as the backbone network. Large-scale picture classification datasets, such as ImageNet, are commonly used to pretrain the backbone network, allowing it to learn a set of general properties that may be used to recognize objects. Backbone networks used in single-stage detectors include Res-Net, Mobile-Net, and Efficient-Net.

- **Feature pyramid network (FPN):**The multi-scale FPN architecture generates a set of feature maps with varying resolutions. By integrating low-level characteristics with high-level semantic data, the FPN generates a set of feature maps from the input image that capture objects of varied sizes and scales. The FPN was used to build a list of possible locations for object detection.

- **Detection head:** Every object in the image that the detecting head recognized was assigned a bounding box suggestion and a related class likelihood. The detection head is frequently composed of convolutional and fully connected layers that use anchor boxes to anticipate item bounding boxes. Furthermore, the detection head predicts the class probabilities for each bounding box proposal, indicating the possibility that the proposal contains an object from a certain class. Single-stage detectors employ the output of the detecting head to generate a set of final predictions for each object in the image. Using non-maximum suppression, the final predictions are frequently post-processed to minimize duplicate detections and low-confidence detections. SSD and YOLO are two well-known single-stage detectors.

3. **YOLOv5:** YOLOv5 there are different model size in each yolo series depending on parameters and neural network designed: In our project we choose YOLOv5s which is small having 14MB size with high inference compared to others and it required less computational power. YOLOv5 models are composed of the same 3 components: CSP-Darknet53 as a backbone, SPP and PANet in the model neck and the head used in YOLOv4.

   - **Backbone:** The backbone is a trained network that extracts rich feature representations from images. This aids in lowering the image's spatial resolution while improving its feature (channel) resolution. The core of YOLOv5 is CSP-Darknet53. CSPDarknet53 is the Darknet53 convolution network to which the authors used the Cross Stage Partial (CSP) network technology in its most basic form. This network was the foundation for YOLOv3. The CSPNet (Cross Stage Partial Network) backbone architecture is employed in YOLOv5.The CSPNet backbone in YOLOv5 is designed to improve object detection accuracy while maintaining the fast inference time of previous YOLO versions. The convolutional layers that make up the backbone are divided into several levels. Each stage has a cross-stage partial connect that allows communication between network levels. Other optimization strategies, such as the usage of SPP (Spatial Pyramid Pooling) and PAN (Path Aggregation Network) modules to improve network feature representation, are also incorporated in the CSPNet backbone [7].

   - These modules let the network gain multi-scale properties, which are necessary for detecting objects of varying sizes. Other advancements to the YOLOv5 backbone include the use of Group Normalization (GN) and the Mish activation function. The popular Batch Normalization (BN) layer is replaced with the Group Normalization strategy, which is intended to provide more stable training in small batch sizes. The novel activation function Mish has been shown to improve the functionality of neural networks. Overall, the CSPNet backbone in YOLOv5 is a major advance over prior YOLO backbones. It improves accuracy while keeping inference speed constant.

   - **Neck:** The feature pyramids are extracted using the model neck. This helps with the model's generalization to objects of various sizes and scales. In the context of the YOLOv5 object detection architecture, the "neck" refers to the collection of intermediary layers located between the backbone network and the detection head. The neck's job is to aggregate and refine the feature maps provided by the backbone network to build more accurate object detection representations. The SPP (Spatial Pyramid Pooling) module and the PAN (Path Aggregation Network) module make up the neck in YOLOv5. The SPP module collects features at many scales using multi-scale pooling approaches, allowing the model to recognize objects of varying sizes. To generate a multi-scale representation, the SPP module performs pooling operations at many scales and then concatenates the resulting feature maps.

     The PAN module, which consists of a sequence of intermediary convolutional layers, performs feature fusion and spatial aggregation. The module employs top-down and bottom-up paths to transmit features from higher resolution layers to lower resolution layers and from lower resolution layers to higher resolution layers, respectively. As a result, the model may generate feature maps that comprise both high-level and low-level information. The neck in YOLOv5 plays an important role in providing high-quality feature maps for object detection by fusing and augmenting the representations provided by the backbone network.

- **Head**: The last stage activities are carried out by the model head. It uses anchor boxes on feature maps to generate the final output, which includes classes, objectness scores, and bounding boxes. In the YOLOv5 object detection architecture, the "head" is the set of output layers that deliver the final object detection findings. The feature maps formed by the neck are processed by the head to provide bounding boxes and relevant class probabilities for things recognized in the input image. Convolutional layers are head constituents in YOLOv5, which reduce the spatial dimensions of feature maps while increasing the representation's depth. Following that is a set of detection layers, each of which forecasts the class probabilities and bounding box coordinates for each object found in the image. Anchor boxes, which are pre-defined boxes with different sizes and aspect ratios, are used by the detection layers in YOLOv5 to anticipate bounding boxes. The model generates the final bounding box coordinates for each detected item using offsets from these anchor boxes. The detection layers estimate the class probability of each bounding box, which reflects the possibility that it contains an object of a given class. Utilizing a single-stage object identification algorithm, the head of YOLOv5 predicts the bounding boxes and class probabilities without utilizing an intermediary proposal generating phase. The architecture was chosen for its high level of efficiency and appropriateness for real-time object detection applications.

4. YOLOv6

- **Backbone:** The YOLOv6 model makes advantage of the CSP (Cross Stage Partial Connections) backbone network. It combines a deep neural network (DNN) and a convolutional neural network (CNN) to improve object identification accuracy while minimizing computation time. The CSP backbone's aim is to extract features from the input image and process those features through numerous stages, each of which focuses on a different component of the image. The backbone of the YOLOv6 model is made up of several stages and is built on CSP (Cross Stage Partial connections). Each stage contains numerous levels. YOLOv6's backbone can have a range of levels depending on the implementation. The backbone of the YOLOv6 model is expected to include roughly 70 layers in total [8].

- **Neck:** The YOLOv6 model employs an SPP-neck (Spatial Pyramid Pooling), which is located halfway between the model's head and backbone. The model's head can recognize objects by merging data from various levels of the backbone network using the fixed-size feature map provided by the SPP-neck. To modify the features to the appropriate output size, the SPP-neck layers convolutional, pooling, and up sampling operations. The SPP-neck of the YOLOv6 model is expected to have roughly 20 layers, though this can vary depending on the individual implementation.

- **Head:** The head of the YOLOv6 model makes the final prediction of object classes and bounding boxes from the feature maps provided by the SPP-neck. After several layers of convolutional, batch normalizations, and activation layers, a final output layer that yields the predictions is usually obtained. The final output layer is frequently a fully connected layer that generates a set of bounding box regression offsets and score mAP’s, one for each object class. Although the exact number of layers in the YOLOv6 model's head may vary depending on the individual implementation, it is predicted to be around 20 layers.

- **YOLOv8:** In our work we used YOLOv8s and unity for all YOLOv5 and YOLOv6 and now we followed the same pattern for v8 also. In YOLOv8 architecture is modified and updated version of YOLOv5. There is lot of improvements in v8 to increase the performance and inference in YOLOv8. The architecture as follows in Appendix section in detailed ways.

- **Backbone:** In the Backbone component of YOLOv8, the Cross Stage Partial (CSP) concept is employed to partition the feature map into two pieces. Part 1 employs convolution processes, while Part 2 concatenates the results of Part 1's convolution operations. CSP design improves Convolutional Neural Network (CNN) learning capabilities while decreasing computing carry on the network model. YOLOv8 employs the C2f module, as opposed to the C3 module employed by YOLOv5. While the C2f module is made up of two ConvModules and one Bottleneck connected by Split and Concat, the C3 module is made up of three ConvModules and one bottleneck. It keeps the algorithm model light while allowing YOLOv8 to collect additional gradient flow information. Furthermore, the YOLOv8 technique reduces the number of blocks in each step from 3,6,9,3 in YOLOv5 to 3,6,6,3 to drastically reduce the model's computing overhead. The SPPF module used in YOLOv5 is succeeded by YOLOv8 in Stage 4. An SPPF replaces the SPP to boost the model's inference speed [9].

- **Neck:** Deeper networks, in general, produce more detailed feature information and better object prediction outputs. Nonetheless, deeper networks diminish the information about the position of things. To reduce information loss for microscopic objects, multi-scale feature fusion using FPN and PAN architectures is necessary. The Neck section of the architecture uses multi-scale feature fusion of images, in which the top features gain more information from more network layers while the lower features lose less locational information from fewer convolution layers. The FPN structure is used by YOLOv5 to up-sample the bottom feature map from top to bottom in order to increase the amount of feature information contained, and the PAN structure is used to down-sample the top feature map from bottom to top in order to increase the amount of position information contained. These two feature outputs are finally blended to ensure trustworthy prediction for photos of varying sizes. YOLOv8 reduces convolution procedures during the up-sampling stage while maintaining FPN and PAN structures.

- **Head:** In contrast to YOLOv5, which uses the linked head, YOLOv8 employs the decoupled head, which separates the classification and detection heads. YOLOv8 retains only the classification and regression branches, removing the objectness branch. Anchor Base first creates many anchors in the image before calculating the four offsets of the regression item with respect to the anchors to correct the accurate object location. Anchor-Free replaces Anchor-Base in YOLOv8, which locates the object by its center and then estimates the distance from the center to the bounding box.

5. **Data Label:** Data annotation can be done manually by each image using programming, or automatically using programmer like makesense.ai, Roboflow, LabelMe, and others. Human annotation is typically thought to be more accurate, but it can be time-consuming and expensive. Automated annotation can be faster and less expensive, but it may be less accurate, especially for more complicated activities. I labelled the images using makesense.ai, a free online platform application for identifying images. In our approach, the YOLO label format is used for each image. The YOLO labelling format data is saved in a normalized.txt file.

Bounding box is represented in 4 values along with the class name. YOLO format is:

[class name, X-center, Y-center, width, height]

X-center and Y-center are the normalized coordinates of the bounding box's center. To normalize coordinates, we take the pixel values 'X' and 'Y' of the image that mark the center of the bounding box on the X and Y axes. Then we divide the value of X by the image's width and the value of Y by the image's height. It represents the bounding box's width and height.

$$\text{Xcenter} = \text{Xmin} + \text{Width}/2 \text{ ....1}$$
$$\text{Ycenter} = \text{Ymin} + \text{Height}/2 \text{ ....2}$$

To find co-ordinates of bounding box center is by using the equation 1 and 2.

In YOLO, bounding boxes can have one of four values (x-center, y-center, width, and height). The x- and y-centers of the bounding box serve as its centers in this case. To normalize the coordinates, we must first find the pixel values for both x and y, which indicate the middle of the bounding box on the x and y axes. The x and y values are then divided by the width and height of the image, respectively. The width and height of the bounding box are represented by the variable's width and height. They have also become normalized.

Mathematical steps are followed for the above ship bounding box normalized.

H = height of Image, W = width of Image

$$\frac{\frac{\text{Xmin} + \text{Xmax}}{2}}{\text{W}}, \quad \frac{\frac{\text{Ymin} + \text{Ymax}}{2}}{\text{H}}, \quad \frac{\text{Width of the Bounding Box}}{\text{W}}, \quad \frac{\text{Height of the Bounding Box}}{\text{H}}$$

6. Data Analysis

In this project we used a data set from the Images Computer visionwebsite. It is open source; in this datait contains 1462 images of 9 types of classes [6].

The 9 types of classes are:

Class Name	Total Images	% of Images
Buoy	68	4.6%
Cruise Ship	239	16.3%
Freight(cargo)	29	1.9%
Ferry	81	5.5%
Gondola	242	16.5%
Inflatable Boat	21	1.4%
Kayak	254	17.3%
Paper Boat	40	2.7%
Sailboat	488	33.3%

Table 2.3.1: Each class Dataset Information

Total Class = 9, Total Images = 1462

Once I finished the Data labelling, I explored the categorical data analysisfor cross-verifying with its image name and label name. I encountered 4 duplication images present in different class folders. From this below table, we can justifythat the total number of images is 1462 and the unique is 1458 which contains 4duplicates with the same name. I removed those 4 images from the folder to notconfuse the model for object classification and detection concerning it.

Here the total Bounding Box was 2942 which contains of 1458 images are:

7. Data-set Split Information:

Data-set Split Information: Dataset is splitted into (80:20) % ratio 80:10: 10 :: Training : Validation : Testing

The data set is divided 80:20 percent for each of9classes separately. Our data set is unbalanced for each class information. Each class distribution of train, val and test data is in below table:

Table 2.3.2: Train Val Inference class Dataset Distribution

Class Name	Train	/alidation	Inference
Buoy	54	6	8
Cruise Ship	190	23	25
Freight(cargo)	23	2	4
Ferry	64	8	9
Gondola	192	24	25

Inflatable Boat	16	2	3
Kayak	203	25	26
Paper Boat	32	4	4
Sailboat	389	48	49
Total	**1163**	**142**	**153**

The maximum bounding box is Sailboat of 954 annotation, almost 32% of bounding box and lowest is Freight of 61 annotation which is 2% of total bounding box.

8. **Model Loss Functions:** There are different type of loss function, but for object detection we uses three different loss functions to train its object detection model.

- **Classification Loss:** This loss function targets the model for correctly classifying an object when it is wrongly identified. It is determined using binary cross-entropy loss between the projected and actual class probabilities.
- **Bounding Box Loss:** This loss function has an effect on the model if it predicts the incorrect bounding box coordinates for an object. It is computed using the smooth L1 loss between the expected and actual ground truth bounding box coordinates.
- **IOU Loss:** When evaluating the effectiveness of object detection, intersection over union (IoU) is utilized to compare the predictedbounding box to the ground truth bounding box.
- **Object Loss:** This loss function targets the model when it predicts the wrong objectness score, which is the probability that an object will be present in a certain grid cell. The difference between the projected objectness score and the actual objectness score is computed using binary cross-entropy loss.

Loss Function is calculated by.

$$\mathrm{Loss} = L_{box} + L_{obj} + L_{cls}$$

$$L_{box} = \lambda_{coord} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . 1_{ij}^{0} . bj \left[(x_i - x_i^{\wedge})^2 + (y - y_i^{\wedge})^2\right] + \lambda_{coord} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . 1_{ij}^{0} . bj \left[(w_i - w_i^{\wedge})^2 + (h_i - h_i^{\wedge})^2\right]$$

$$L_{obj} = \lambda_{coord} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . 1_{ij}^{0} . bj \left[(c_i - c_i^{\wedge})^2 + \lambda_{nobj} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . 1_{ij}^{n} . obj \left[(C_i - C_i^{\wedge})^2\right]\right.$$

$$L_{cls} = \lambda_{cls} . 1_{i}^{0} . bj \sum_{cclasses} . (\rho(C) - \rho(C))^2]$$

Where,

$L_{box} = L_{coord}$ is bounding box regression loss

L_{box} is classification loss

L_{obj} is confidence loss

i = current cell number

j = current anchor number

B = number of anchor boxes

x,y,w,h = Bounding Box (Grounding Truth)

$C_{i,j}$ = Confidence

$$x^{\Lambda}, y^{\Lambda}, w^{\Lambda}, h^{\Lambda} = \text{Predicted Box}$$

$$C^{\Lambda}_{ij} = \text{Predicted}$$

$$1^{0}_{ij}\ bj = 1 \text{if} C_{i,i} = 1{,}0(\text{Probablility of object exist})$$

$$1^{0}_{ij}\ bj = 1 \text{ if } C_{i,j} = 1,\ 0(\text{Probability of object exist})$$

This is the general loss function of YOLO is in this form. Later they modified with different techniques of anchor small, medium, and large bounding boxes with different scales of object. The enhanced YOLO loss function is in below [10].

$$\text{Loss}_{i,j} = \text{Loss}^{xywh}_{i,j} + \text{Loss}^{p}_{i,j} + \text{Loss}^{c}_{i,j}$$

$$\text{Loss}^{xywh}_{i,j} = \frac{\Upsilon_{coord}}{N\,L^{obj}} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . L^{0bj}_{ij} [(x_{i,j} - x^{\Lambda}_{i,j})^2 + (y_{i,j} - y^{\Lambda}_{i,j})^2 + (\sqrt{w_{i,j}} - \sqrt{w^{\Lambda}_{i,j}})^2 + (\sqrt{h_{i,j}} - \sqrt{h^{\Lambda}_{i,j}})^2]$$

$$\text{Loss}^{p}_{i,j} = - \frac{\Upsilon_{coord}}{N\,L^{obj}} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . L^{0bj}_{ij} \sum_{c\varepsilon classes} . p^{c}_{i,j}\ log\ (p^{c}_{i,j})$$

$$Loss^{c}_{i,j} = \frac{\Upsilon_{coord}}{N\,L^{obj}} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . L^{0bj}_{ij} (IOU^{GroundTrut\ h_{i,j}}_{Predictions\ \ i,j} - C^{\Lambda}_{i,j})^2 + \frac{\Upsilon_{coord}}{N\,L^{obj}} \sum_{i=0}^{s^2} . \sum_{j=0}^{B} . L^{n0bj}_{ij} (0 - C^{\Lambda}_{i,j})$$

$$\frac{\Upsilon_{coord}}{N\,L^{obj}} = Normalization$$

$$Loss^{obj}_{i,j} = \text{Object exists}$$

$$x_{i,j}, y_{i,j}, w^{\Lambda}_{i,j}, h^{\Lambda}_{i,j} = \text{square distances of width and height}$$

For object class predictions we use cross entropy loss function

$$\text{Cross Entropy Loss} = - (y_i \log (y^{\Lambda}_{i}) + (1 - y_i) \log(1 - y_i)$$

9. Industrial Developments

- **Training Pattern**

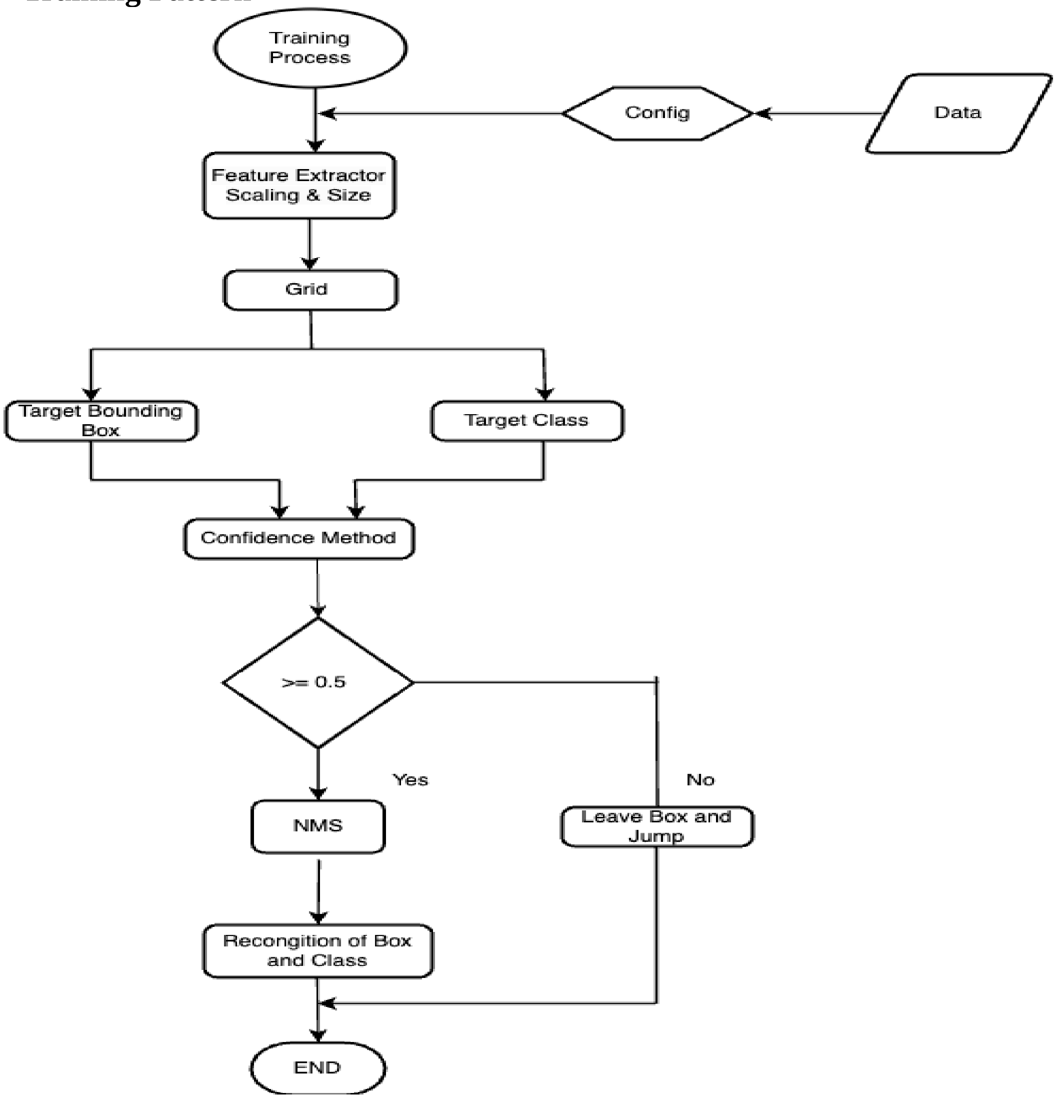


Figure 2.5.1: Model Training Flow Chart

This model training plan structure is followed for version5, 6 and version 8 thereis no change in model designing.

- **Reparametrizing Optimizer:** There are numerous approaches for tuning the parameters, but for our model, we chose grid cross validation. Instead of selecting all of the values in the tables, we employed a strategy that is closest to the prior value. Following that, we select neighboring data values for the best optimization value in the table depending on the preceding value. Different values for each performance are obtained through hyper parameter adjustment. Here are some of the values that were tested in order to get the ideal parameter value.

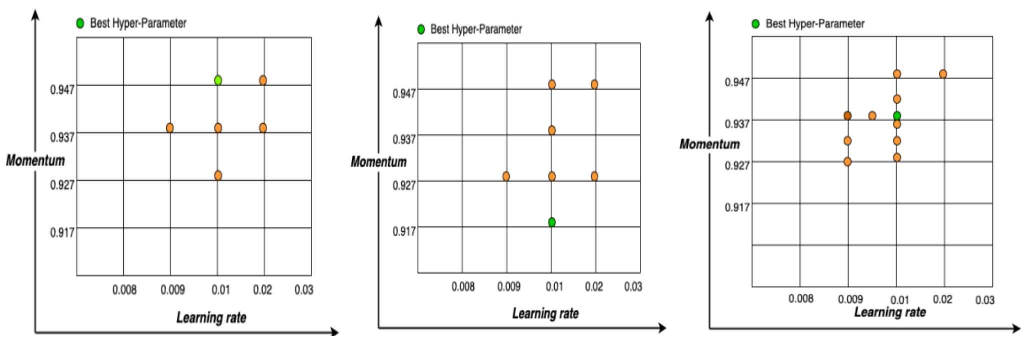


Figure 2.6.1: Data Points Graph Grid Search for Cross validation (V5, V6 and V8 from left to right)

Table 2.6.1: Grid Search Cross Validation for YOLOv5.

Lr(Learning rate)	m (Momentum)	Val mAP(0.5)	Train mAP(0.5)
0.01	0.937	66.20%	65.57%
0.01	0.927	69.30%	65.64%
0.01	0.917	72.10%	69.25%
0.01	0.947	67.70%	63.30%
0.02	0.947	64.80%	61.44%
0.02	0.927	66.80%	65.30%
0.009	0.927	58.70%	59.64%

According to this table, the best performance in learning rate and momentum is 71.1 at 0.01 and 0.947, as well as 0.009 and 0.937, respectively. In this case, both parameters performed well during the training and testing phases. However, it takes less time than other methods.

In hyper parameter tuning except momentum(m) and learning rate(lr). We adopted the same data augmentation and other parameters in uniform way. So, in this optimization phase using grid search table we find the best value for (Lr and momentum). This technique we applied for all 3 models in each phase.

Table 2.6.2: Grid Search Cross Validation for YOLOv6.

Lr(Learning rate)	m (Momentum)	Val mAP(0.5)	Train mAP(0.5)
0.01	0.937	66.20%	65.57%
0.01	0.927	69.30%	65.64%
0.01	0.917	**72.10%**	69.25%
0.01	0.947	67.70%	63.30%
0.02	0.947	64.80%	61.44%
0.02	0.927	66.80%	65.30%
0.009	0.927	58.70%	59.64%

Based to this table, the best performance in learning rate and momentum of mAP is 72.1 at 0.01 and 0.917, respectively. When compared to the training phase, this parameter fared well in the testing phase.

In YOLOv8 the learning rate of 0.009 and momentum 0.937 with mAP(0.5) of 72.1 well performance of all other parameters in grid search comparison during best hyper parameter finding.

Table 2.6.3: Grid Search Cross Validation for YOLOv8

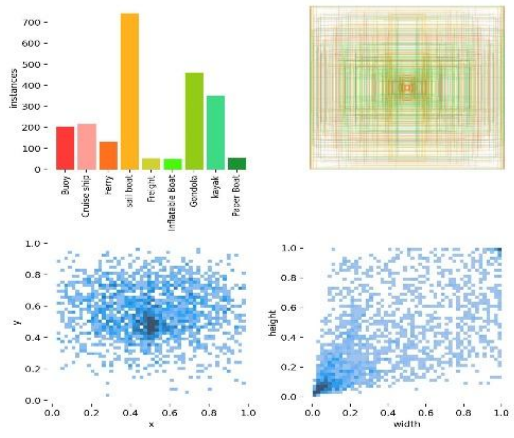
Lr(Learning rate)	m (Momentum)	Val mAP(0.5)	Train mAP(0.5)
0.01	0.937	72.10%	71.20%
0.01	0.927	69.90%	70.00%
0.01	0.947	68.30%	69.4%
0.01	0.931	66.90%	66.6%
0.01	0.938	70.90%	70.50%
0.01	0.936	66.60%	65.60%
0.02	0.947	59.80%	61.80%
0.009	0.927	71.60%	71.8%
0.01	0.947	71.10%	71.1%
0.009	0.937	72.10%	70.50%
0.0095	0.937	69.10%	69.30%

.

In YOLOv8 performed well in each parameter except 1 learning rate and momentum remaining all near to + - 70%. 2 parameter perform outstanding well with 72.1% mAP(0.5) but in training phase learning rate(lr) = 0.01 and momentum(m) = 0.937. Achieved best in testing and training.

III. EXPERIMENTS

1. Implementation Plan



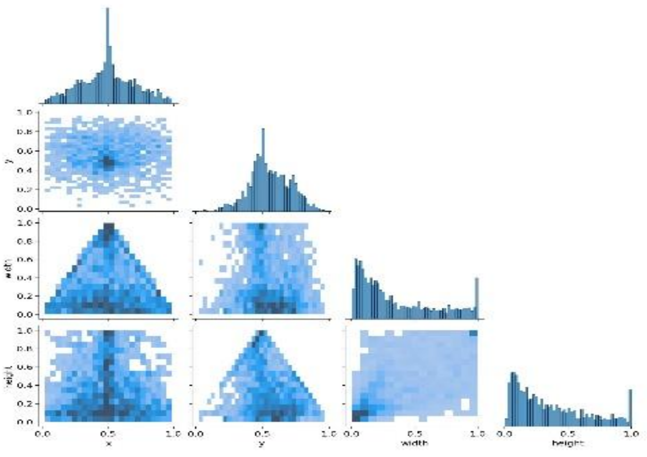


Figure 3.1.1: Class Bounding box Width Height Spread Figure 3.1.2: Cologram for Bounding Box Width Height

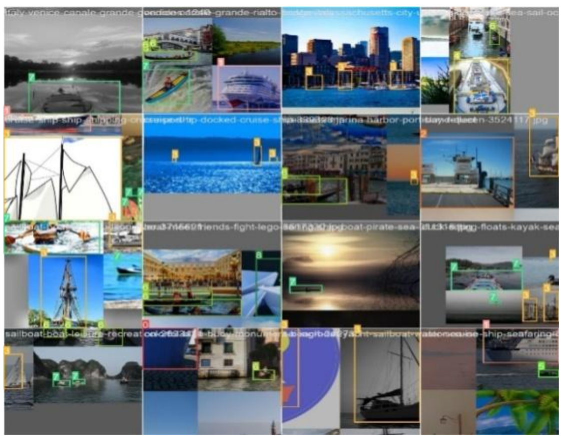
- **Data Augmentation:** Data augmentation is a technique used in machine learning and computer vision to improve the size and diversity of a dataset by creating new instances from existing data through various modifications. The goal is to improve machine learning model performance by avoiding over-fitting and boosting generalization. Data augmentation is the process of creating new instances from existing data by modifying it in various ways, such as rotation, scaling, flipping, cropping, and adding noise. For example, data augmentation approaches for image classification jobs may include flipping, cropping, and adjusting an image's brightness or contrast. Data augmentation can serve to improve the size and diversity of the dataset, correct class imbalance, and prevent over-fitting. The machine learning model gets exposed to a greater variety of instances when utilizing data augmentation, which might help it generalize and perform better on fresh, unknown data. In this project, data augmentation is performed on image space and color space using an augmentation library in an online data loader with random image generation (original picture + three random images) for each parameter.

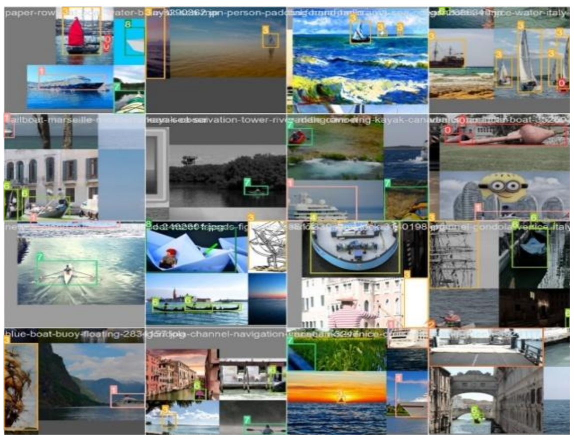
  YOLO feeds training data into a data loader, which augments data in real time. Colour space changes, scaling, and mosaic data augmentation are the three major forms of augmentations used by the data loader. The following are the values of the hyper-parameters utilised for data augmentation:

Table 3.1: Data Augmentation Parameter Values

Parameter	Value	Parameter	Value	Parameter	Value
perspective	0.0001	degrees	0.2	Hsv_s:	0.7
scale	0.9	Fliplr	0.5	Hsv_v:	0.4
shear	0.2	Flipud	0.5	mixup	0.1
translate	0.1	Hsv_h:	0.015	mossaic	1.0

In all 3 models they adopted strong data augmentation parameters to train the data. In this Augmented data can be either slightly modified copies of existing data. There are different techniques used in this Augmentation process like HSV, Degree, Translation, Scaling, Shear, flip mosaic and mix-up. These parameters are set same for all 3 model as default without any changing values from 1 model to another in order to avoid confusion and maintain unity. We can change based on our requirements from (0 to 1) which is normalized. Here YOLOv8, YOLOv6 andYOLOv5 are the same parameter and value followed.





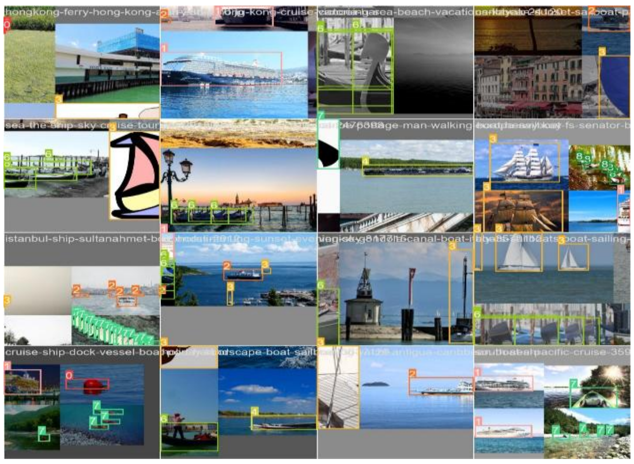


Figure 3.1.3: Data Augmentation Results

2. **Modelling**

In our project we used Deep learning models that are primarily based on the structure and operation of the human brain, allowing them to process data and make predictions in the same way that humans do.

- **Training :** This is the first phase of modelling after the data pre-processing steps deep learning algorithm is trained at this point by being fed to data sets. This is the phase where learning occurs. The prediction rate of the DL model can be considerably increased with consistent training of data. The model's weights must be initialized earlier. The algorithm will learn to modify the weights appropriately inthis way. To get good results, we want to adjust basic parameters and hyper-parameters. In the next section explained in detailthe required parameters in our work.
- **Model V5:** The pre-design model utilized for the paper is YOLOv5s, which is the smallest (14.12 MB) and fastest network model among the available versions, with around 7.2M parameters when weights and biases of other models are considered.

The model was re-configured with our own customized parameter below.

Table 3.2.1: V5 Architecture Model Training Parameter Values

Parameter	Value	Parameter	Value
Model	5s	Solver	SDG
Backbone	CSPDarknet-53	Data augmentation	HSV
Epochs	100	Image size dimension	640x640
Batch size	16	Device	GPU
Learning rate	0.01	Anchor boxes:	9
Momentum	0.947	Weight decay	0.0005

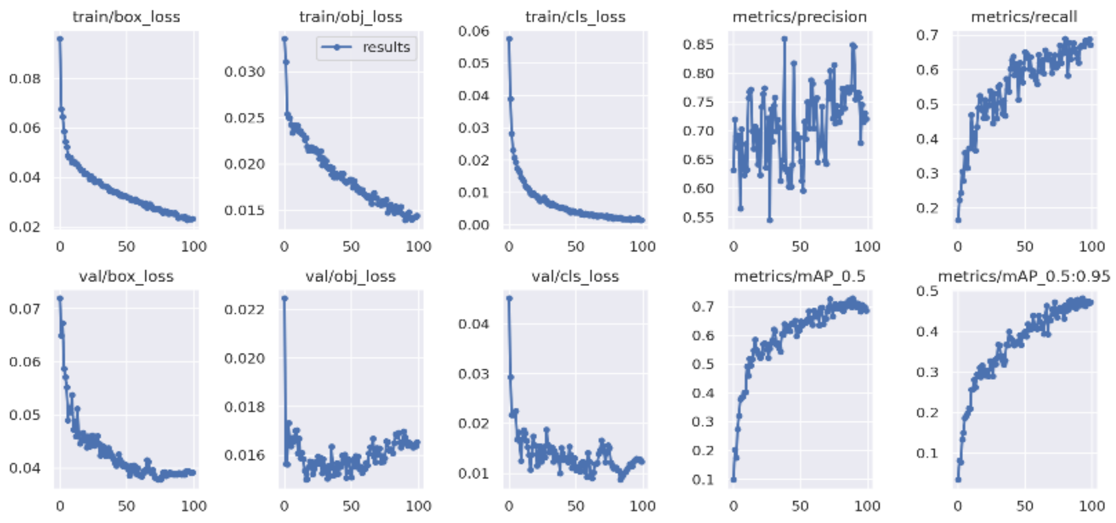


Figure 3.2.1: Training Graph Results YOLOv5

The results of YOLOv5 with 100 epochs training results are shown here in above training graph of all the metrics of loss (Bounding Box loss, object confidence loss, class loss) and mAP @ (0.5 and 0.5:0.95). In loss since it is 0 at initial stage of epochs and later at few epochs it is decreasing the loss (box loss and cls loss it takes some iteration(epochs) to decrease it.

The training result of YOLOv5 mAP @0.5 is 71.0% and mAP @0.5:0.95 48.1% with 2.58hr in Google Colab with 12GB Ram and 1 GPU.

- **Model V6:** To retain unity, we used the identical Training Parameters and Data Augmentation parameters as in YOLOv5, except for the network architecture, and we tuned the parameters using grid search (learning rate and momentum). We select the best parameter value with the highest mAP from the grid search (see section 4.4 Model tune YOLOv6 hyper parameter tweaking). The difference in score can be seen here. YOLOV6(36.3MB) has 1.57 times the weight of YOLOV5(14MB).

  We followed the same pattern except model architecture designed in neural network for this model.

  Backbone: EfficientRep (RepVGG and CSPREPStack).
  Anchor boxes: nine are used: ((10, 13), (19, 19), (33, 23), (30, 61), (59,59), (59, 199), (116, 90), (185, 185), (373, 326))
  Momentum: 0.917

Table 3.2.2: YOLOv6 Training Results.

Model	Epochs	mAP @0.5	mAP @0.5:0.95	Hours
YOLOv6s	100	69.25%	47.65%	2.77hr

YOLOv6 training data show that mAP @0.5 is 69.25% and mAP @0.5:0.95 is 47.65 after 2.77 training hours. Due to higher FLOPS and a larger number of parameters, YOLOv6 has a 1.75% drop in comparison to YOLOv5 (71.0%) and 11.2 training hours in comparison to YOLOv5.

- **Model V8:** Here the Anchor is a free based model based on regression technique and Momentum: 0.937. rest all followed the same.Anchor boxes are used to discover object classes with the suitable scale and aspect ratio. A predefined collection of boxes with fixed heights and widths is used. During detection, they are tiled across the image and selected based on the size of objects in the training dataset.

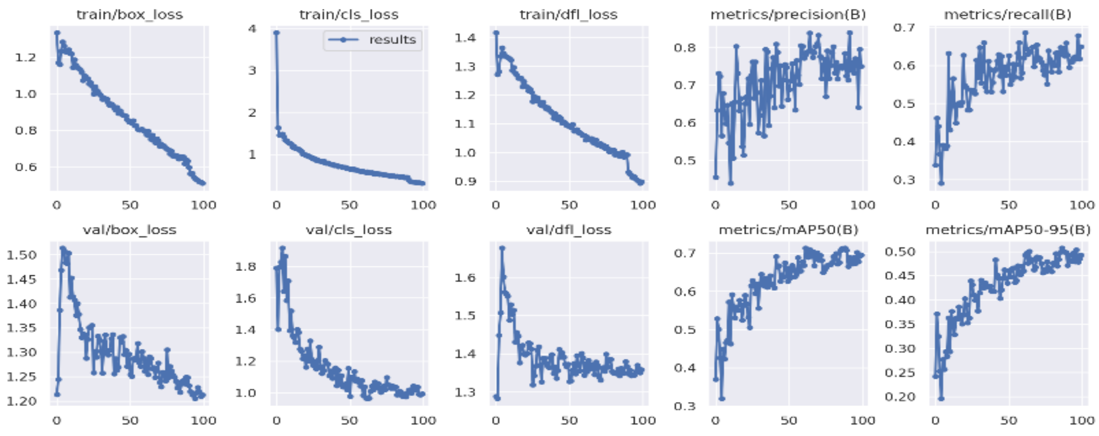


Figure 3.2.2: Training Graph Results YOLOv8

Box loss and confidence loss declines in YOLOv8 began during the early stages of epochs. It is linearly connected to loss and epoch until the latter epochs, and class loss is greater. In comparison to box and confidence loss, classification loss is greater. It reduced the loss but did not reduce the other two losses.

YOLOv8 Model Training results are in Fig 4.15 with trained 100 Epochs with mAP@0.5 72.1% and mAP@51.2% and time duration are 2.74hr less compared to previous Version 6.

- **Validation:** Model evaluation is the process where the performance of a fully trained model is evaluated on a testing data set.

- **Model V5:** These are the configuration we provided during the validating the data:

Weights:	The best trained PyTorch weights in training phase of model
Batch size:	16
Image Size:	640
Image Data:	142
Confidence Threshold:	0.25
IoU Threshold:	0.50
Maximum Detection:	300

It gives the misclassified object in the test dataset in the confusion matrix. The following confusion matrix is for YOLOv5. The confusion matrix, together with accuracy and the PR curve, can be used to estimate recall and precision, which are measures used to evaluate model performance.

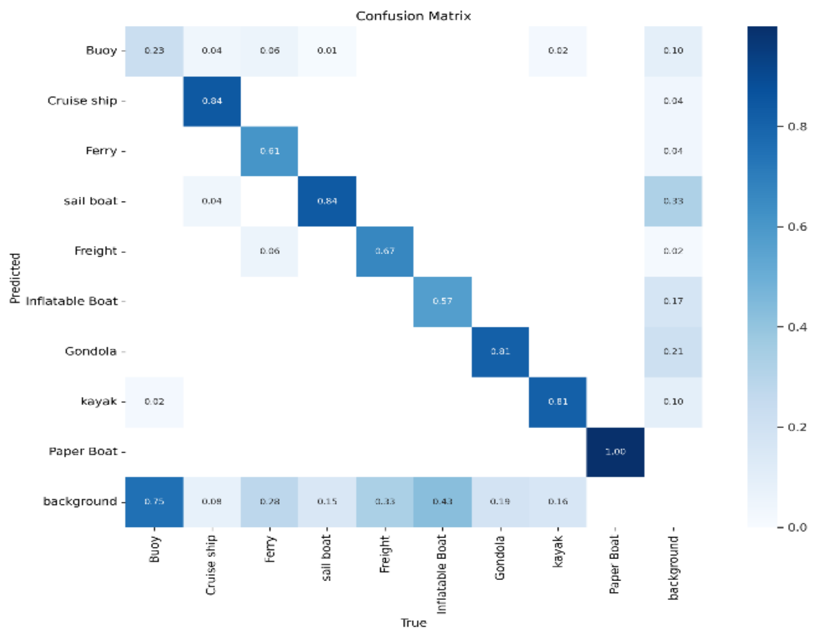


Figure 3.2.3: Confusion Matrix Validation Yolov5

In confusion matrix it gives the mis-classified object in test dataset. The belowconfusion matrix gives for YOLOv5. Recall and Precision can be measured usingthe confusion matrix, along with accuracy and the PR curve, which are the metricsused for evaluating the performance of models.

The 76.6% Precision, 66.8 % Recall, 0.7 F1 score and mAP@0.5 is 71.1% along with step 0.05 mAP@0.5:0.95 is 48.3% in YOLOv5. Information of Precision Recall and F1 score calculation is in Metric section of Training and Evaluation.

Table 3.2.3: YOLOv5 Image Computing Time Results.

Speed	**Time(ms)**
Pre-process time	0.3ms
Inference time	10.2ms
NMS time	4.8ms

The amount of pre-process time was taken 0.3ms, Inference 10.ms and NMS is 4.8ms. Thecomputing time is for each image out of 142 images in model evaluations.

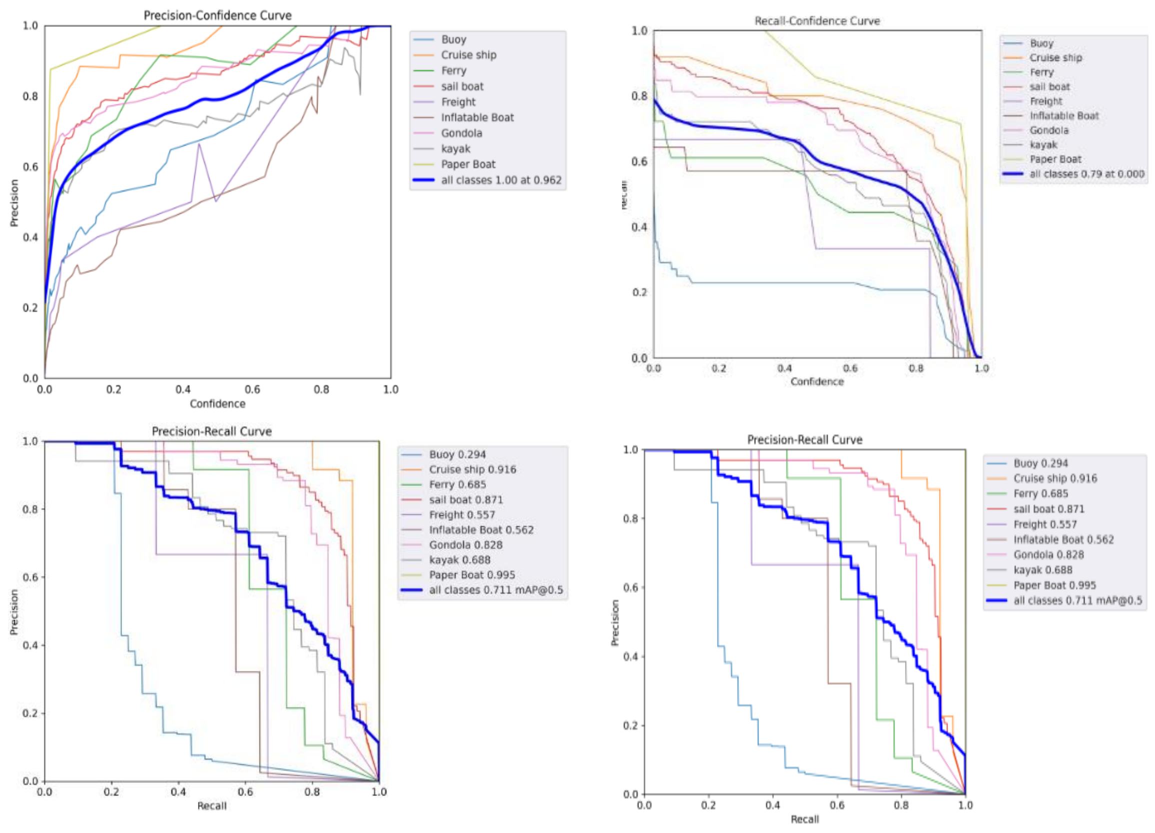


Figure 3.2.4: P, R, PR Curve F1 score Val YOLOv5

The evaluation Results of YOLOv5 Precision Confidence, Recall Confidence and PR curve is in graph with mAP 0.5 is 71.1. F1 Curve Validation Yolov5 .70 at .43

The evaluation of the performance of an object detection model YOLOv5 of F1 score value is .70 at confidence 0.435. In graph classes are distributed each other paper boating class is over fitting and Buoy class is under-fitted. Cruise ship is good fitted in order to consider as best among all class.

- **Model V6:** As previously explained in 3.3.2 section and same followed for YOLOv5 version. In YOLOv6 in the confusion matrix majority of TP are more and predicted good apart from few class less errors.

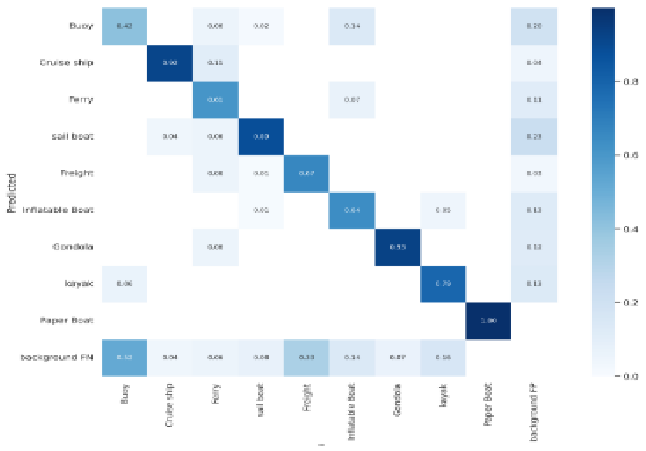


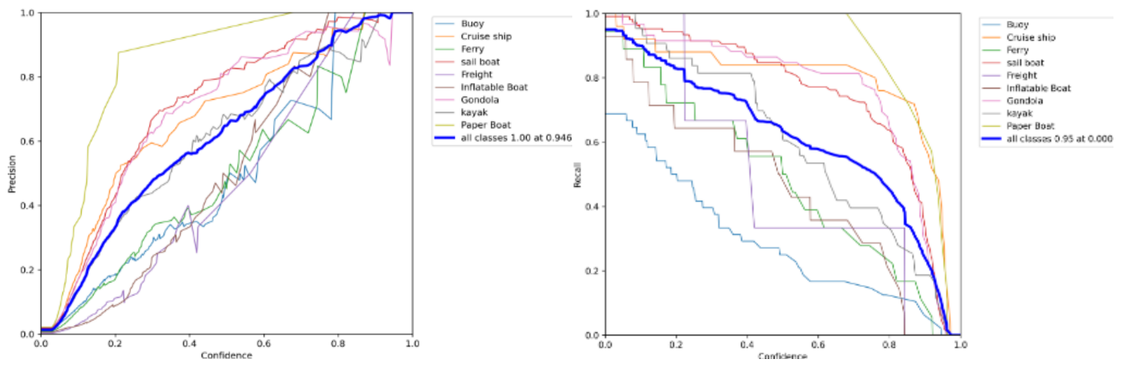
Figure3.2.5: Confusion Matrix Validation YOLOv6

YOLOv6 mAP @0.5 of testing accuracy is 72.1%, in testing we got good and more mAP accuracy but in training we got 69.2%.

Table 3.2.4: YOLOv6 Image Computing Time Results.

Speed	**Time(ms)**
Pre-process time	0.22ms
Inference time	9.52ms
NMS time	2.63ms

The testing speed results performance of each image average results here of 142 images. Pre-processed time is 0.22ms, Inference time 9.52ms and NMS(post process time) is 2.63ms of each image results respectively in Table 4.9. Compared to Version v5(table 4.8) all 3 speed of pre-process time, NMS and inference time is faster in YOLOv6.



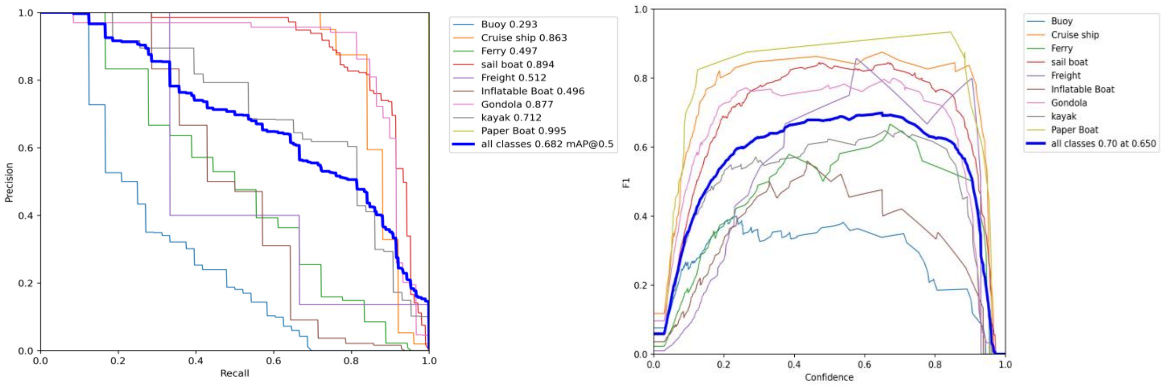


Figure 3.2.6: P, R, PR Curve, F1 score YOLOv6 Validation

YOLOv6 Precision is 0.8 number of positives are more in this model as of v5 compare and Recall is .18 less towards ground truth.
F1 Curve Validation Yolov6 .7 at 0.65
The P curve value is 80.4, Recall 65, P-R curve value 72.10 @mAP(0.5), F1 value is 0.65 accuracy of model and mAP@(0.5:0.95) is 48.3 as shown in above graph of each metrics values. In F1 value model of each class performed good except blue line (0.65) average of other class, except 4 class rate.

- **Model V8:** In YOLOv8 in the confusion matrix majority of TP are more and predicted good apart from few classes with few errors. The majority of 7 classes out of 9 class are more than 50% TP.

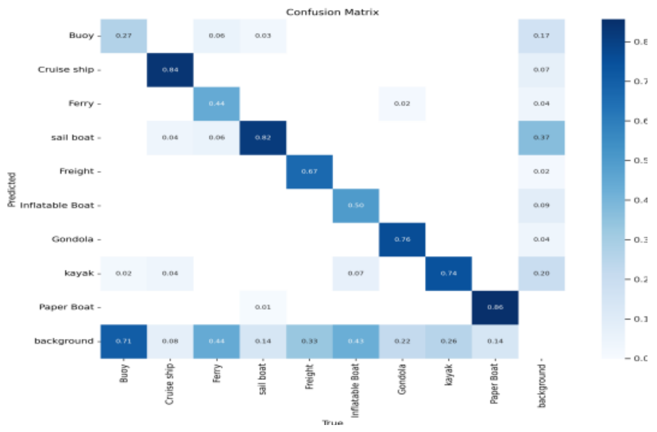


Figure 3.2.7: Confusion Matrix Validation Yolov8

In YOLOv8 achieved good results both in training and testing performance of mAP@0.5 is 72.1% and 51.2% respectively in above table.

Table 3.2.5: YOLOv8 Image Computing Time Results.

Speed	Time(ms)
Pre-process time	0.21ms
Inference time	9.4ms
NMS time	2.54ms

The image process time of each image with average of all 142 images is less in YOLOv8 with pre-process time 0.21ms, inference time 9.4ms and NMS time 2.54ms. In all other model is more than comparable in terms of computational time. This is one of the advantages in inference for next steps.

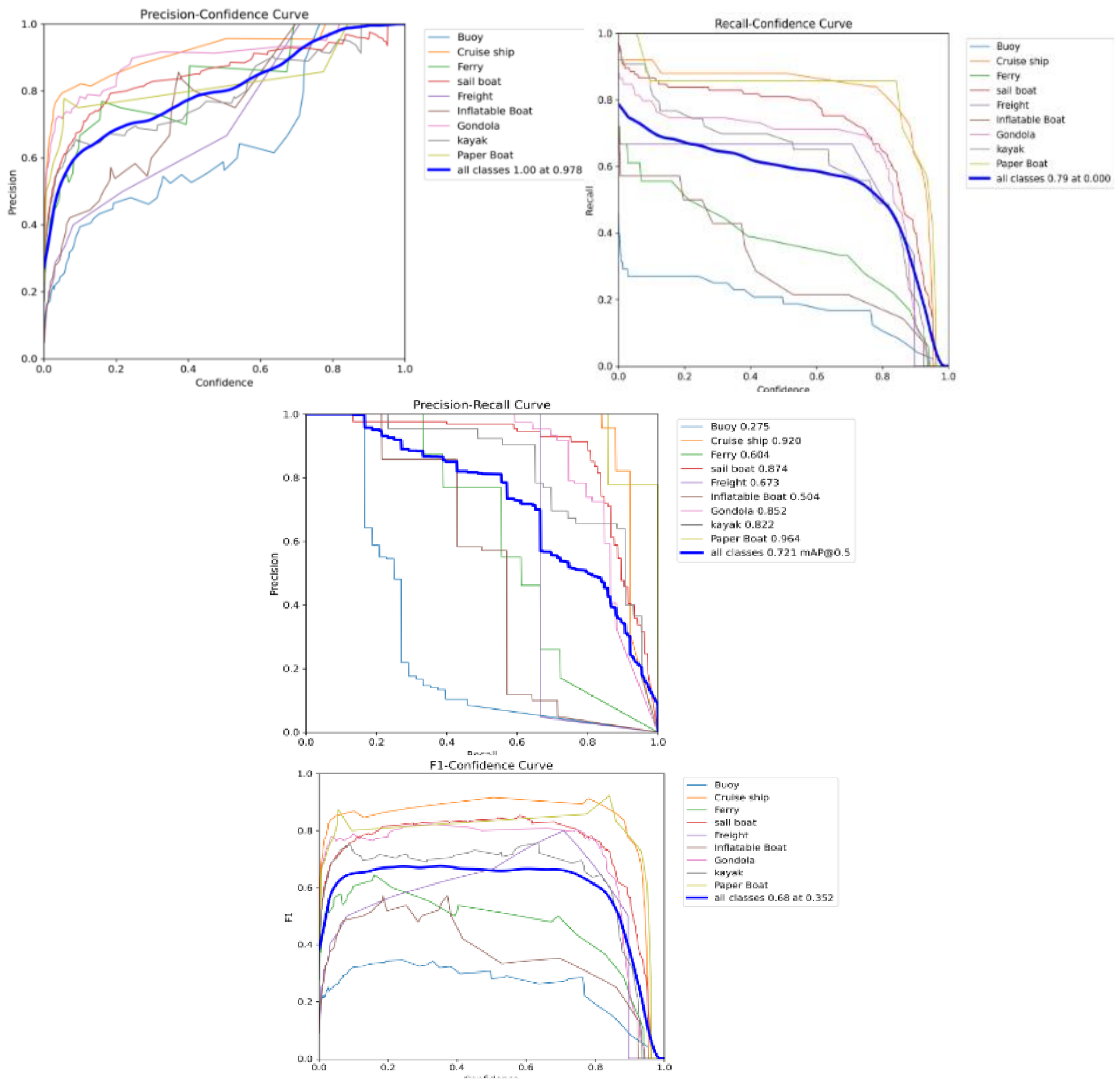


Figure 3.2.8: P, R, PR Curve Validation YOLOv8

YOLOv8 Precision is 0.73 number of positives are in this model and Recall is .64 is in above graph of each class performance in PR curve.

F1 Curve Validation Yolov8 0.68 at .35

The F1 Score of YOLOv8 is .68 at 0.35 confidence with mAP @0.5 as in above graph. many class are performance is difference due to several reasons of data imbalance and others. As here all 3 models are having different values for F1 value and curves is in each evaluation phase of model results.

- **YOLOv8 TensorFlow Lite Evaluation:** To evaluate the TFLite model, which cannot be used directly, we must first convert from PyTorch to ONNX format, then TensorFlow, and finally TFLite. The robust and open Open Neural Network Exchange (ONNX) format was created for communicating machine learning models. It overcomes the issue of easily for edge devices by providing a uniform intermediary model format. The detailed explanation of model conversion may be found in the final deployment part, step by step (for conversion model detailed steps in deployment section).

  In order to save weight, the YOLOv8 model is transformed to TensorFlow lite for edge device testing. So I adapted the TFLite model in PyTorch. I calculated the same data for the TFLite model. Even if it loses some percentage in mAP@0.5 and mAP0.5:0.95, the model performance is good. The table below provides thorough information on the performance of the float16 and float32 bit models.

Table 3.2.6: TFLite YOLOv8 Validation Results.

Model	Precision	Recall	PR	F1	mAP@0.5	mAP@0.5:0.95
float16	79	56.1	72.10(@0.5)	0.68 (@ 0.5)	67.5	47.5
float32	78.6	56.1	72.10(@ 0.5)	0.68(@0.5)	67.3	47.6

The float16 and float32 perform quite well in mAP@0.5, with only a 0.02 loss percentage in float32 because to its strong convergence, and with +0.1 higher accuracy in mAP(0.5:0.95).

Table 3.2.7: YOLOv8 Image Computing Time Results.

Speed	float16	foat32	int8
Pre-process time	0.5ms	0.3ms	0.3ms
Inference time	875.9ms	864.7	1055.9ms
NMS time	1.5ms	1.6ms	1.7ms

Each TFlite model performs on float and int quantization results.

- **Model Inference:** Trained deep neural networks (DNN) draw inferences or make predictions when provided with new or novel data that the model has never seen before. We decided to convert model edge for YOLOv8 since model performance is good in terms of mAP@0.5 and mAP@0.5:0.95 in terms of both training and testing of metric consideration. Here are YOLOv5 and YOLOv6 for model discussion. The table below contains information on the computing outcomes of each model type.

Speed	Time(ms)
Pre-process Time	0.6ms
Inference Time	21.3ms
NMS Time	1.7ms

YOLOv8 PyTorch Inference Speed Results

Speed	**Time(ms)**
Pre-process Time	0.6ms
Inference Time	21.3ms
NMS Time	1.7ms

YOLOv8 ONNX Inference Speed Results

Table3.2.8:YOLOv8 TFLite Inference Speed Results

Speed	float16	float32	int8
Pre-process time	20.3ms	17.9ms	18.7ms
Inference time	875.9ms	833.0ms	976.1ms
NMS time	1.9ms	1.6ms	1.9ms

The YOLOv8 performance of each model in inference stage from PyTorch model to tflite for 153 images in pre-process, inference, and NMS time.

10. Comparisons with Models

Each Class Evaluated AP@0.5

Table 3.3.1: Average Precision (@0.5) Obtained for each class evaluation.

Class	**YOLOv5**	**YOLOv6**	**YOLOv8**
Buoy	0.294	0.349	0.275
Cruise Ship	0.916	0.901	0.920
Ferry	0.685	0.585	0.604
Sailboat	0.871	0.887	0.889
Freight (Cargoship)	0.557	0.912	0.673
Inflatable Boat	0.512	0.473	0.540
Gondola	0.828	0.788	0.852
Kayak	0.688	0.607	0.822
Paper Boat	0.995	0.981	0.9064
mAP	0.711	0.721	0.721

Evaluation of each model with each class performance in testing phase. There are 5 class have more accuracy and some of classes are dominates near to score.

Table 3.3.1: mAP of Training and Testing results.

Model	**Training %**	**Testing %**
YOLOv8s	71.2 %	72.1 %

YOLOv8 model performance in training and testing while 0.9% is more in evaluating the model with mAP@0.5.

Table 3.3.2: mAP of tflite.

Model	**Testing %**
YOLOv8s.tflite	67.5 %

TFLite Model performance in testing phase with mAP@0.5 is 67.5% for next stage of deployment phase. But PyTorch to TFLite .056% is reduced due to converting from one stage to another stage.

Results of YOLOv5, YOLOv6, YOLOv8 Comparisons

Model Results YOLOv5 YOLOv6 YOLOv8

Table 3.3.3: Model Results.

Method	Size	Epoch	APv(50)	APv(50:95)	Par(M)	FLOPS(G)	APtr(50)	APtr(50:95)	Hrs
YOLO5s	640	100	71.1	48.3	7.03	15.8	71.0	48.1	2.58
YOLO6s	640	100	72.1	48.3	18.51	45.18	69.25	47.65	2.77
YOLO8s	640	100	72.1	51.2	11.13	28.7	71.3	50.7	2.74

From this table YOLOv8 performance is good when compare to YOLOv5 and YOLOv6 in terms of mAP(0.5) and mAP(0.5:0.95) both testing and training. So we choose YOLOv8 and achieved with best new stage of art.

IV. DEPLOYMENT

Several steps are required to convert a PyTorch file to TensorFlow Lite (TFLite) format for edge deployment. Here's a rundown of the procedure:

- **Save the PyTorch model as an ONNX file:** To convert a PyTorch model to TensorFlow Lite format, first export the model to the ONNX format. ONNX is an open deep learning model representation format that may be utilised by a variety of frameworks, including TensorFlow. The torch.onnx.export() function can be used to convert a PyTorch model to ONNX format. The PyTorch model, an example input tensor, and the output route where the ONNX model will be saved are all passed to this function. The PyTorch model is loaded from a file called yolo.pt in this code excerpt. Finally, we use the onnx.export() function to convert the PyTorch model to ONNX format and save it to a file called yolo.onnx.

- **Converting the ONNX model to TensorFlow format:** Now that we have the PyTorch model in ONNX format, we can convert it to TensorFlow format. We can accomplish this by utilising the tf2onnx package, which includes the onnx-tf module, which can convert an ONNX model to TensorFlow format. The input ONNX model file (yolo.onnx) and the output TensorFlow model file (yolo.pb) are specified in this. The ONNX model will be converted to TensorFlow format and saved to the given output file by the onnx-tf library.

- **To convert the TensorFlow model to TFLite format, follow these steps:** The TensorFlow model must now be converted to TensorFlow Lite format. This is possible thanks to the TensorFlow Lite Converter, a Python library that includes a Converter class that can convert a TensorFlow model to TFLite format.

First, use the load function to load the TensorFlow model from the yolo.pb file. We then use the tf.lite.TFLiteConverter() function to generate a Converter object from the supplied model. Finally, we use the converter.convert() function to convert the TensorFlow model to TFLite format and save the resulting TFLite model.

The model tested in hardware board and results are in below table

Table 3.3.4: YOLOv8s .tflite Model performance on IMX8MP Board

IMX8MP Hardware with YOLOv8s Model [11]			
Processor	Model type	Img Size(pixels)	FPS
CPU	Float16	640x640	0.28FPS
CPU	Float32	640x640	0.29FPS
CPU	INT8	640x640	0.41FPS

Original Model vs Our Reconfigure Model

Table 3.3.5: Original Model

Method	**APv(50)**	**APv(50:95)**	**APt(50)**	**APt(50:95)**
YOLO5s	69.4	44.3	69.0	44.9
YOLO6s	65.64	45.19	69.2	47.1
YOLO8s	71.1	50.1	69.90	48.5

Table 3.3:6: Our Model

Method	**APv(50)**	**APv(50:95)**	**APt(50)**	**APt(50:95)**
YOLO5s	71.1	48.3	71.0	48.1
YOLO6s	72.1	48.3	69.25	47.65
YOLO8s	72.1	51.2	71.3	50.7

V. RESULTS

Our model is not 100% so somewhere is error or miss classified or detect in results. Here detection results as follows. In the figure it is detected correctly as sailboat but in image there are still objects couldn’t able to recognize other images. Here in this point of stage model performance is down, and if we observe carefully in confusion matric there, we can find TN and FP results classified.

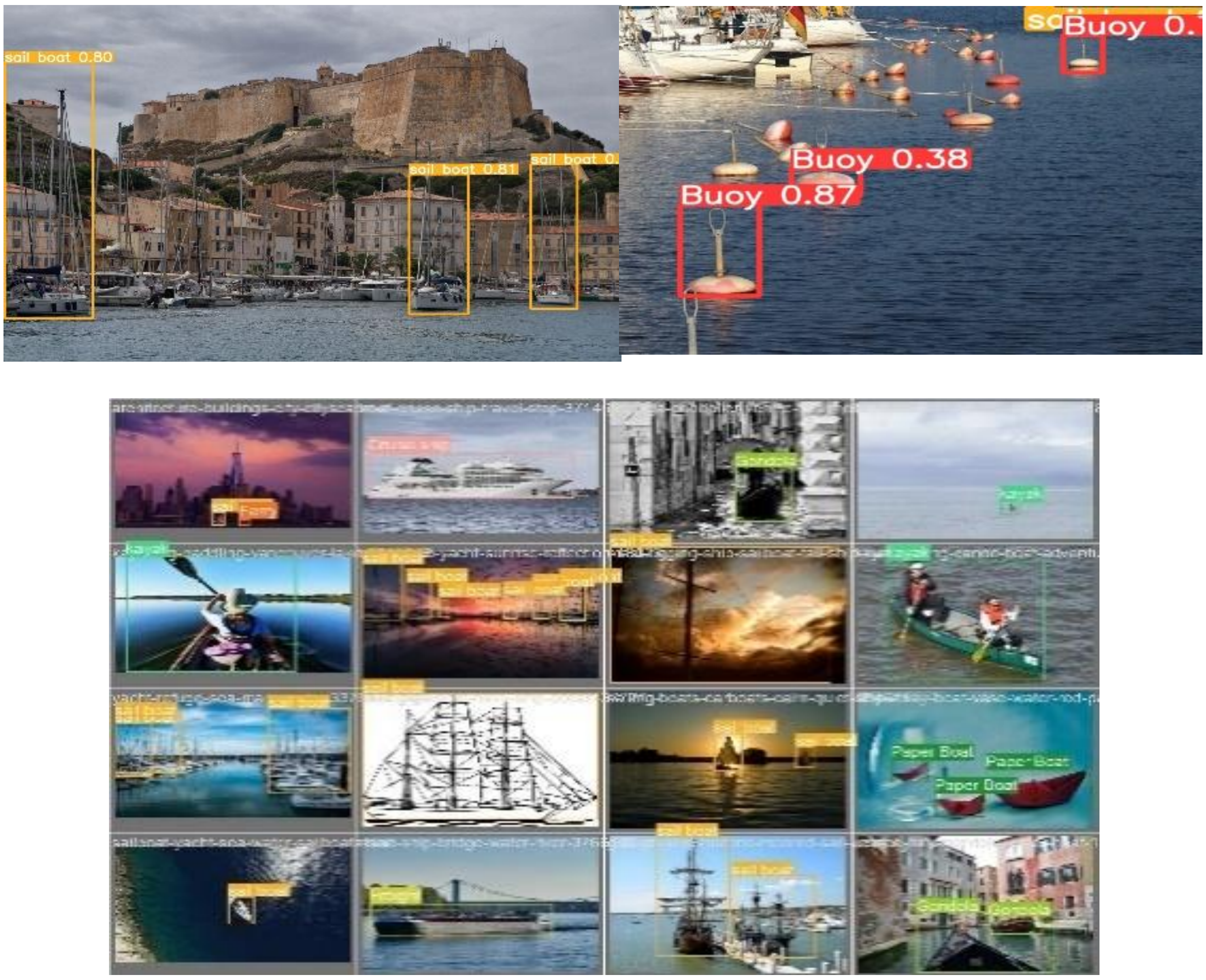


Figure 3.3.1: Detection Results

These are the results from models shown in all possible classes with large image, small image, cropped image, top view image, with low and high resoluted image with 640X640 pixels.

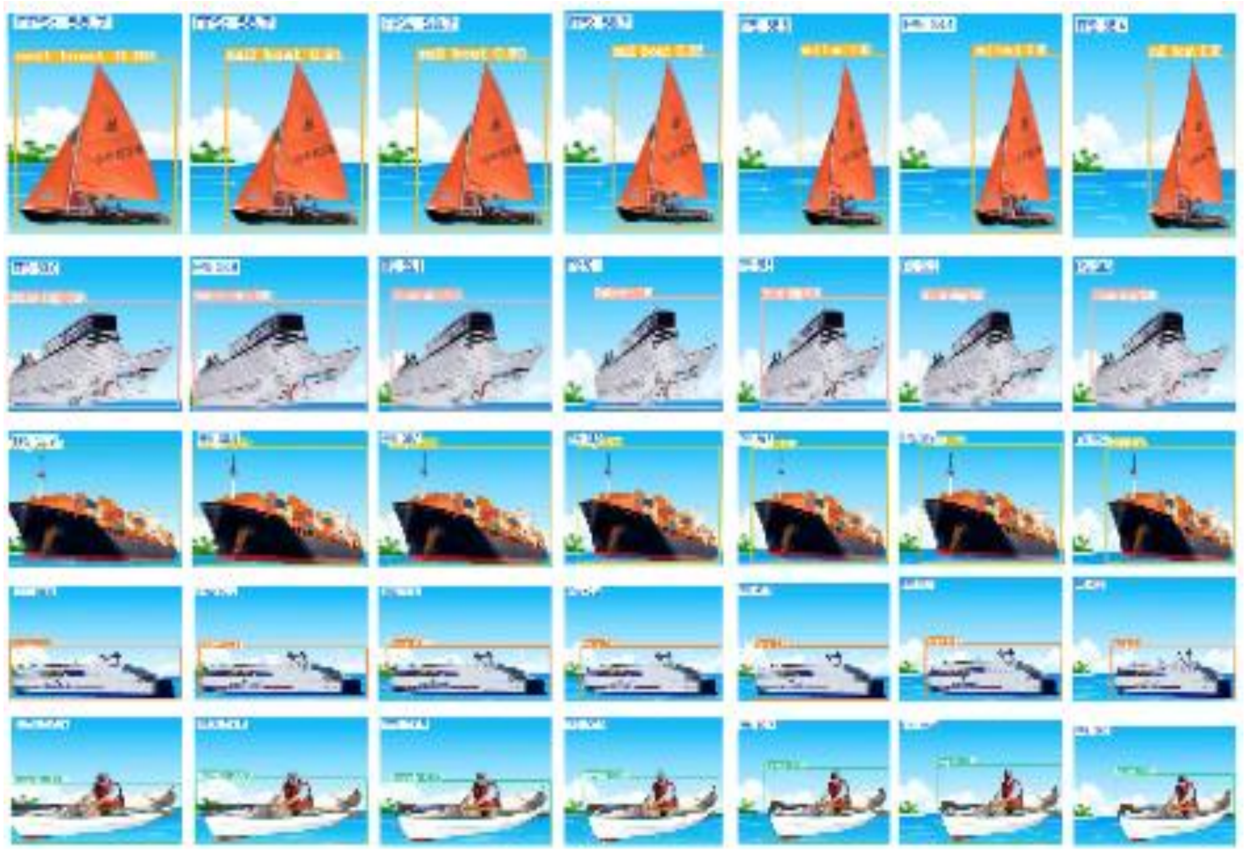


Figure 3.3.2: Video Processed Image

VI. CONCLUSION

In this work is a proposed for the real time implementation of Automatic Recognition of Boats and Ship method for detecting ship and boats in marine environment. Based on our experiment and the result we obtained, we can see how YOLO models deep learning can be implemented by using object detection algorithm for developing a object recognition system. The CPU and hardware tools used to conduct this Experiment work are state-of-the-art platforms regarding deep neural network and object recognition model. Our main contribution to the work is to apply the state-of-the-art YOLO model to do the detection of ship and boat detection in marine environment. Even with a small dataset, the model performance is good compared to version 5 and version 6. The presented comparison of three deep convolutional neural network YOLOv5, YOLOv6 and YOLOv8. shows that YOLOv8 reconfigured has the highest accuracy, reaching approximately 72.1% mAP(0.5) in testing and training 71.1% mAP(0.5) accuracy. And to testhardware, we converted PyTorch model to TensorFlow Lite model for edge device IMX8MP hardware with different model type with float16, float32 and int8 with 640 x 640-pixel size image. Model with int8 quantization performed good compared to float16 and float32 with .41 FPS with camera in real time testing. The primary conclusion from our analysis of the results is that neural networks can learn and identify things, as shown in the results section and in the comparison of our three models.

- **Discussion &Future Work:** This work is an experimentation of real time object recognition in the real world for marine ships and boats. To work in the realworld, it should me more robust and productive for the next production stage. There are several improvements to be made to our model. In terms of data-set size of each class should increase at least 100 which is having below 100 images in dataset. This could be done by more labelling of the images of having less data. Since time required for the computation is more, we were not able to run more epochs means more than 100. We followed maximum 100 epochs for all 3 models even in hyper-parameter tuning to see how it works in the long run due to limited resources and time. For addressing these parameters defining and more parameters in GPU environment on cloud for large data in future step which is having less mAP(0.5) accuracy in testing results. So, by addressing these changes in setup the model and system is more robust and efficient. Based on the characteristics of object recognition algorithm it can be used in different domain technologies to detection and classification for real time application such as

Computer Vision Applications for Transportation

1. Object recognition in ariel or satellite view.
2. Automatic vehicle number plate recognition in toll gate
3. Vehicle parking using smart allocated detection area
4. Vehicle counting system

Computer Vision Applications for Agriculture

1. Vegetables or Fruits detect and classify in food industries
2. Poultry or domestic animal in monitoring animals

Intrusion Detection or trespassing areas

- People counting in shopping malls
- Airport facial recognition and security purposes
- Healthcare and Medical imagining applications

So object recognition systems have a many tremendous potential application indifferent domain areas in engineering research.

VII. ACKNOWLEDGEMENT

I take this opportunity to express thankfulness to Professor Giuseppe Longo, the chair and coordinator of the Data Science course. The Kineton S.r.l (Innovation Lab) Naples, Italy for the traineeship opportunity and support provided to me during this experience. Specialists and manager guided me during these months in the development of the research work.

VIII. CONFLICT OF INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] Guo, Z. Z. K. C. Z. S. Y. & Ye, J. Object Detection in 20 Years: A Survey. Proceedings ofthe IEEE (2023).
[2] Navaneeth Bodla Bharat Singh, R. C. & Davis, L. S. Improving Object Detection WithOne Line of Code. IEEE International Conference on Computer Vision (ICCV) (2017).
[3] Joseph Redmon Santosh Divvala, R. G. & Farhadi, A. You Only Look Once: Unified, RealTime Object Detection. IEEE Conference on Computer Vision and Pattern Recognition(CVPR) (2016).
[4] Redmon, J. & Farhadi, A. YOLOv3: An Incremental Improvement. IEEE Conference onComputer Vision and Pattern Recognition (CVPR)(2018).
[5] Alexey Bochkovskiy, C.-Y. W. & Liao, H.-Y. M. YOLOv4: Optimal Speed and Accuracy ofObject Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)(2020).
[6] Computer vision image. Data set computer vision images for academic and researchers. BellSyst. Tech. J. https://images.cv/category/Boat#google_vignette (2022).
[7] Jocher, G. YOLOv5: By Ultralytics. Ultralytics Yolov5 (2020).
[8] Vision Development, M. I. A. YOLOv6: A Single-Stage Object Detection Framework forIndustrial Applications. IEEE Conference on Computer Vision and Pattern Recognition(CVPR)2022 (2022).
[9] Jocher, G. YOLOv5: By Ultralytics. Ultralytics Yolov8 (2023).
[10] Rafael Padilla, S. N. & da Silva, E. A Survey on Performance Metrics for Object-DetectionAlgorithms. International Conference on Systems, Signals and Image Processing (IWSSIP)(2020).
[11] IMX8MP. https://www.variscite.it/product/system-on-module-som/cortex-a53-krait/var-som-mx8m-plus-nxp-i-mx-8m-plus/.
[12] Chien-Yao Wang, A. B. & Liao, H.-Y. M. YOLOv7: Trainable bag-of-freebies sets newstate-of-the-art for real-time object detectors. IEEE Conference on Computer Vision andPattern Recognition (CVPR)2022 (2022).
[13] Robert Ross Marcin Elvis, G. A. & Thomas. Computer vision and Pattern matching. Paperswith code of Computer Vision.

Appendix A

Acronym: The following abbreviations are used in this manuscript:

1. AI: Artificial Intelligence
2. ML: Machine Learning
3. DL: Deep Learning
4. NN: Neural Network
5. DNN: DeepNeuralNetwork
6. CNN:ConvolutionNeuralNetwork
7. YOLO:YouOnlyLookOnce
8. mAP:MeanAverage Precison
9. CV:Computer Vision
10. CV: Cross Validation
11. IOU: IntersectionOverUnion
12. FP:FalsePositve
13. TN: TrueNegative
14. TP:TruePositive
15. AP: AveragePrecision
16. Lr:LearningRate
17. TF:Tensorflow
18. SB DatasetShip – BoatDataset
19. FPS:FramePerSecond
20. Lr:LearningRate
21. SVM:SupportVectorMachine
22. NMS:Non-MaxSupression

23. FPN: Feature Pyramid Networks
24. CSP: Cross Stage Parital Network
25. SPP: Spatial Pyramid Pooling
26. PAN: Path Aggregation Networks
27. CSP: Cross Stage Partial networks
28. AP: Average Precision
29. ONNX: Open Neural Networks Exchange

Appendix B

Workflow of Project:

The project's flow is shown in the below diagram, with each step leading from one stage to the next being denoted by a number and an arrow. Data was initially obtained from the image.cv website's database and then placed onto the local system disk. Once the data has been gathered, it becomes challenging to analyze the data for the image. I either need to use Excel or another format to extract the data in much more detail. Data extraction is done in parallel, utilizing the Makesense.AI cloud tool for csv files in step 2, in order to perform data analysis. Here, I externally divide the data for the train, val, and test, and I begin labeling the classes in step 3.

Step 4 involves loading the local disk with the transformed csv data and annotation files. Once more, we used the EDA procedure to cross-check the data with the csv file. Removed duplicate data from one class to another and crosschecked it with the annotation and image files in the train-val-test folder using a csv file. The model train-val-test process is then performed in step 5 in conjunction with a quicker approach using one GPU. For good accuracy, we performed hyper-parameter adjustment here. We converted the model from Pytorch-ONNXTensorflow-TFLite for edge devices in step 6 in order to deploy the model after model training. The changed model is then loaded once more into the local system for inference check. This is the last phase of the project's step 9,10 real-time check deployment. To check the object with FPS, a TFLite Model is deployed onto an IMX8MP MEK hardware board with a camera .

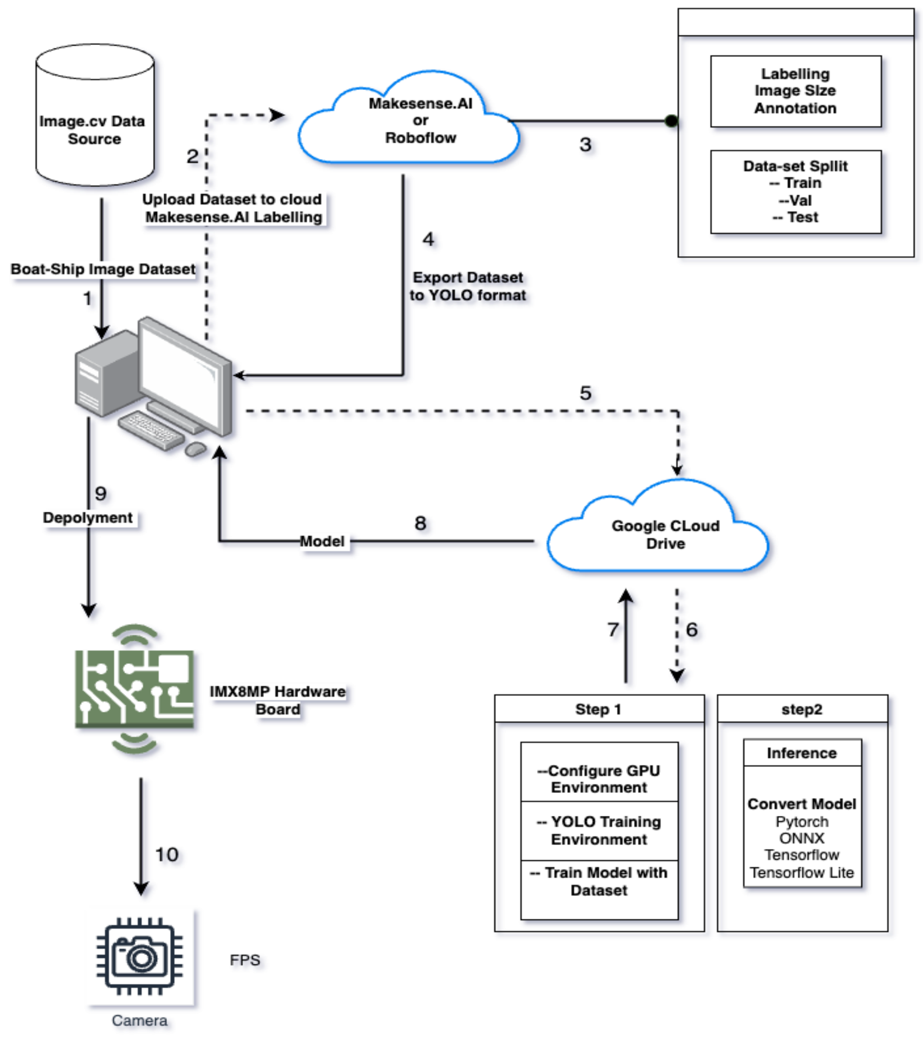


Image.cv Data Source
2
Upload Dataset to cloud Makesense.AI Labelling
Makesense.AI or Roboflow
3
Labelling Image SIze Annotation
Data-set Spllit -- Train --Val -- Test
Boat-Ship Image Dataset
1
4
Export Dataset to YOLO format
5
9
Depolyment
8
Model
Google CLoud Drive
7
6
IMX8MP Hardware Board
Step 1
--Configure GPU Environment
-- YOLO Training Environment
-- Train Model with Dataset
step2
Inference
Convert Model Pytorch ONNX Tensorflow Tensorflow Lite
10
FPS
Camera

YOLOv5

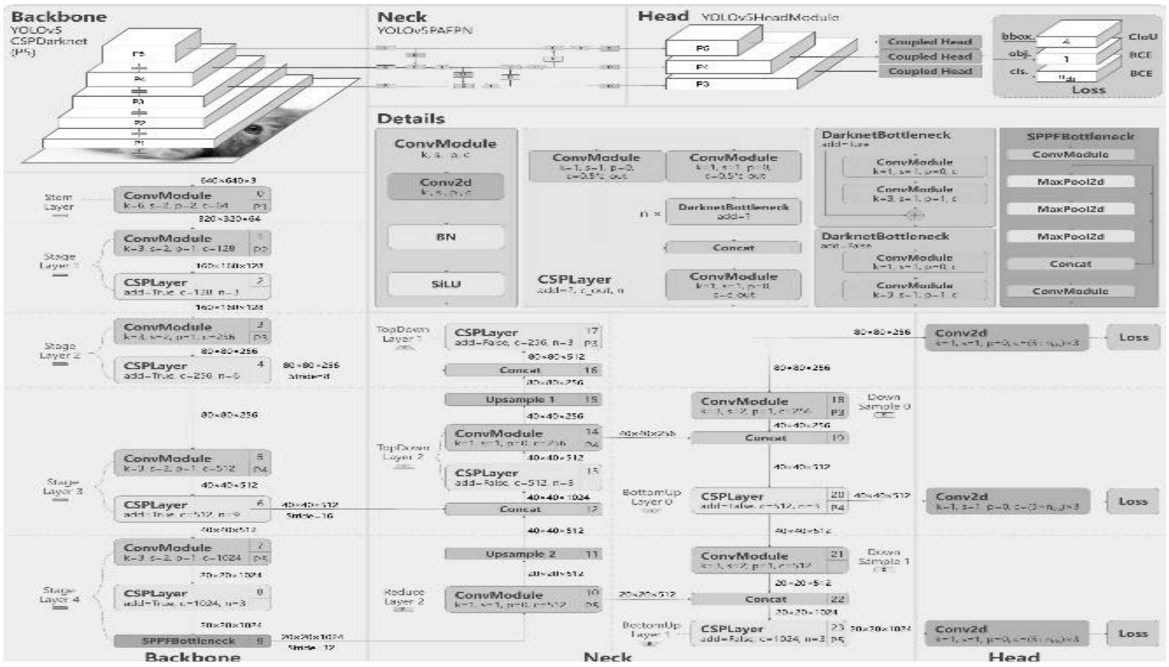


Figure B1: Yolov5 Deep Architecture

YOLOv6

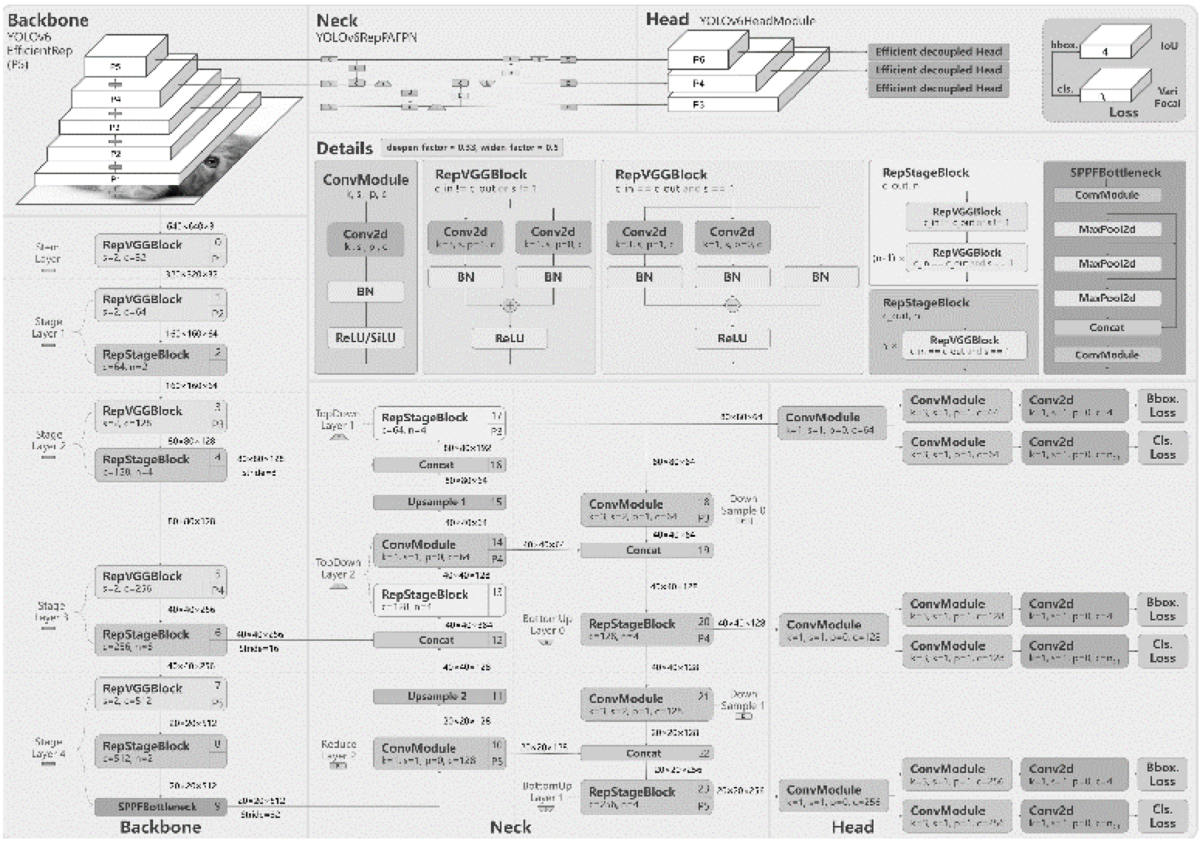


Figure B.2: Yolov6 Deep Architecture

YOLOv8

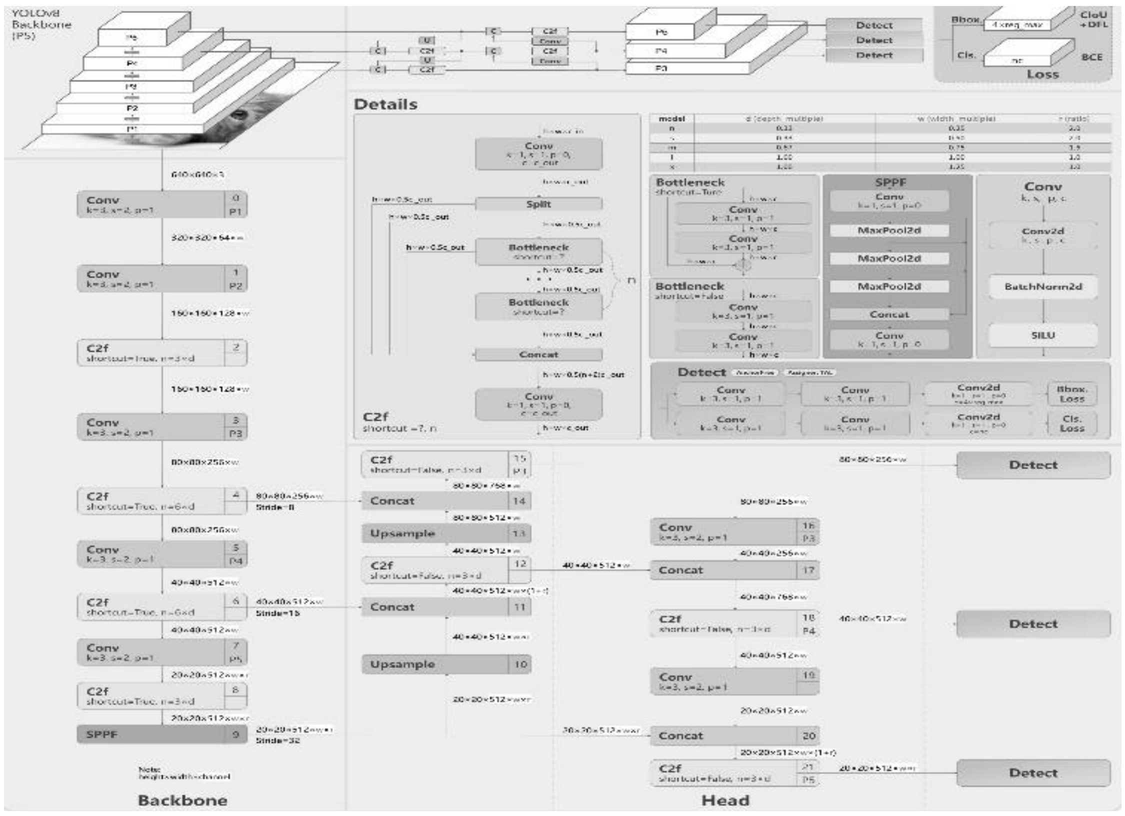


Figure B.3: Yolov8 Deep Architecture