POZNAŃ UNIVERSITY
OF ECONOMICS
AND BUSINESS

Michał Skowronek

Vessel type detection for shore cameras using YOLOv8

Detekcja rodzajów statków dla kamer brzegowych z wykorzystaniem YOLOv8

Bachelor's thesis

Supervisor: dr hab. Krzysztof Węcel, prof. UEP

Programme: Informatics and Econometrics

Specialization: Information Systems for Business and Administration

Poznań 2023

Contents

Chapter 1

Introduction

1.1 Background

In recent years, significant progress has been made in the field of computer vision, specifically in the area of real-time object detection. Notably, the You Only Look Once (YOLO) models have emerged as prominent and successful contributors to this field. One of the latest advancements in the YOLO series is YOLOv8 (Jocher, Chaurasia & Qiu, 2023), which became publicly available in January 2023. Since its release, YOLOv8 has swiftly gained widespread recognition and acclaim for its exceptional performance in real-time object detection tasks.

Moreover, there is an increasing interest in the application of real-time object detection methods to maritime environments. Rather than solely detecting the presence of ships, researchers have started to create specialized datasets and deploy models that are specifically designed to detect different types of ships. This growing emphasis on maritime applications highlights the versatility and expanding potential of real-time object detection in addressing specific challenges within the ship detection domain. For instance, one real-world problem in this context is the issue of illegal trespassing of maritime borders, such as the South China Sea, by fishing vessels and other types of ships (Ablan & Garces, 2005; Yoon, 2021). This demonstrates the practical relevance and significance of accurate ship detection in tackling complex and crucial scenarios.

1.2 Motivation

The primary motivation behind conducting this research was to gain a comprehensive understanding of state-of-the-art (SOTA) object detection models and their underlying principles. Additionally, an important driving factor was the desire to develop a practical tool that could effectively address real-world challenges and contribute to problem-solving efforts.

1.3 Problem Formulation

We aim to delve into the evolution of YOLO model to offer a comprehensive understanding of its development over time. Currently, there exists a single research paper that describes the entire lineage of YOLO algorithms, starting from YOLOv1 and culminating in YOLOv8 (Terven & Cordova-Esparza, 2023). However, it is worth noting that the final model, YOLOv8, is only briefly discussed in half a page of the paper.

Additionally, our investigation will examine the practical application of the latest model, YOLOv8, in real-time ship recognition using existing datasets. We aim to determine the model's effectiveness and level of satisfaction achieved in accurately identifying various types of ships.

Furthermore, we will assess the feasibility and ease of enhancing the performance of pre-trained YOLOv8 model through fine-tuning. This analysis will shed light on the potential for optimizing the model's capabilities without making substantial changes to its architecture.

In summary, our research will focus on three key questions:

1. What are the theoretical foundations and historical advancements of the YOLO series?
2. Can YOLOv8 successfully and satisfactorily recognize various types of ships in real-time using established datasets?
3. How feasible and straightforward is it to enhance the performance of pre-trained YOLOv8 model through fine-tuning?

1.4 Research Plan

Business Understanding:

- Understand the issue of ship detection, identify the key problems that need to be addressed in this field, and explore the technologies that can be utilized to effectively overcome these challenges.
- Gain domain knowledge of Deep Neural Network, Convolutional Neural Networks (CNN) and object detection techniques.
- Understand the importance and applications of object detection in various domains.
- Understand the YOLO algorithm as the primary object detection approach for investigation.
- Study and analyze the architecture, working principles, and advancements of YOLOv8.

Data Understanding:

- Identify and acquire a suitable dataset specifically related to ship detection.
- Evaluate the dataset's quality, size, and diversity to ensure it meets the research objectives.

Data Preparation:

- Perform data preprocessing for the ship dataset.
- Analyze the ship dataset to gain insights into the distribution of ship classes, image characteristics, and any potential challenges or biases.
- Explore techniques to enhance the ship dataset, such as data augmentation or incorporating additional external data sources.

Model Training and Experimentation:

- Train the YOLOv8 model using the enhanced ship dataset.
- Experiment with different training parameters and training strategies to optimize the model's performance.
- Monitor and evaluate the model's training progress, including metrics such as loss and accuracy.

Results Analysis:

- Evaluate the performance of the trained YOLOv8 model using appropriate evaluation metrics (e.g., accuracy, losses, F1 score, confusion matrices).

- Analyse and interpret the results, identifying the model's strengths, weaknesses, and limitations.

Deployment:

- Use the trained YOLOv8 for deployment in some exemplary practical situations.
- Discuss potential use cases and scenarios where the deployed model can provide value.

Conclusion:

- Summarize the research findings answering on previously formulated problems.
- Highlight the contributions, limitations, and potential future directions for improving the model and extending the research.

Chapter 2

Theoretical Basis of Object Detection

2.1 Artificial Neural Networks

The name and action of Artificial Neural Networks (ANNs) comes from the fact that they were created based on the action of neurons that are found in the human brain (Jain, Mao & Mohiuddin, 1996). For this reason, it allows us to perform complex tasks involving analyzing and processing complicated data quickly. Their advantage over traditional regression is that ANN is able to model complex nonlinear relationships, at the same time having a high fault tolerance and highly scalable when it comes to parallel processing (J. Zou, Han & So, 2008).

The basis of ANN is a single artificial neuron, which is a simple mathematical function. The basic principle of the artificial neuron can be reduced to three actions: multiplication, summation and lastly activation (Krenker, Bester & Kos, 2011) (Figure 2.1).

The process starts by multiplying the input values with their corresponding weights. The products are then summed together with the bias term. Finally, the resulting sum is passed through an activation function, also known as a transfer function, that determine the property of the neuron. Among the popular transfer functions are: Sigmoid function, Rectified Linear Unit (ReLU) and Softmax function (Krenker, Bester & Kos, 2011).

More information on transfer function can be found at (Dubey, Singh & Chaudhuri, 2021). However, in order to solve complex problems, we combine artificial neurons in artificial neural network, and the way this is done is called either topology or architecture. We distinguish feed-forward topology and recurrent topology. In case of feed-forward ANN information flows in one direction, without any back-loops, where recurrent ARR doesn't have any restrictions regarding back-loops (Figure 2.2).

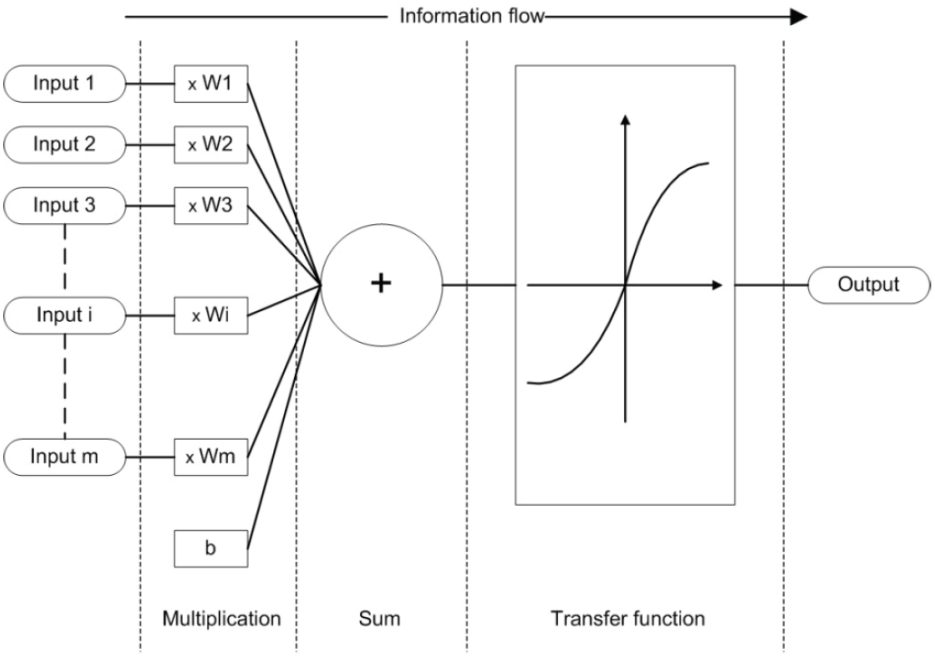

Figure 2.1. Working principle of an artificial neuron.

Source: (Krenker, Bester & Kos, 2011)

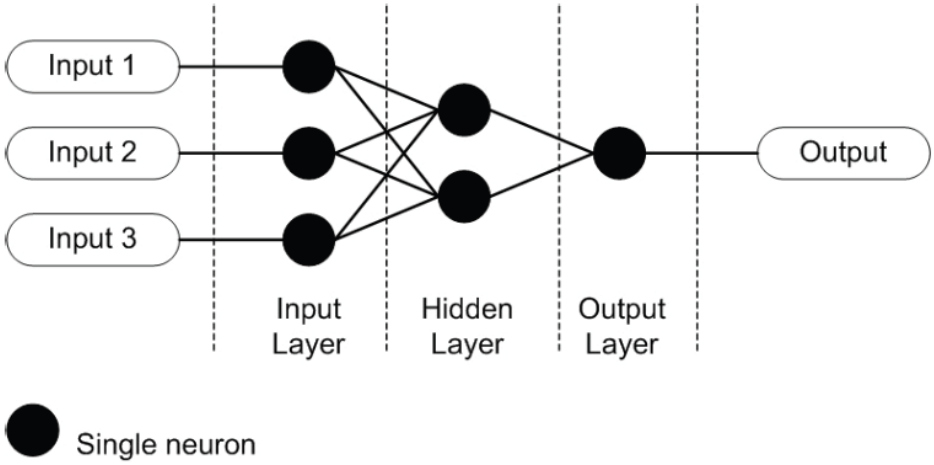

Figure 2.2. Feed-forward topology of an ANN.

Source: (Krenker, Bester & Kos, 2011)

When considering the learning approaches of ANNs, there are three distinct types:

- supervised learning – We utilize a training set consisting of input-output pairs to adjust the parameters of the Artificial Neural Network (ANN). This iterative process allows the algorithm to learn and make accurate predictions on unseen inputs (Krenker, Bester & Kos, 2011).
- unsupervised learning – The algorithm does not receive labeled data, which means it has to autonomously discover patterns and underlying structures without explicit guidance (Jain, Mao & Mohiuddin, 1996).
- reinforcement learning – The algorithm learns to make decisions by interacting with an environment and receiving rewards or penalties, so by trial-and-error (Krenker, Bester & Kos, 2011).

2.2 Convolutional Neural Networks

A crucial concept in object detection is the Convolutional Neural Network (CNN), a specialized neural network for computer vision. To grasp its functioning, it is essential to understand the components of a CNN (Figure 2.3).

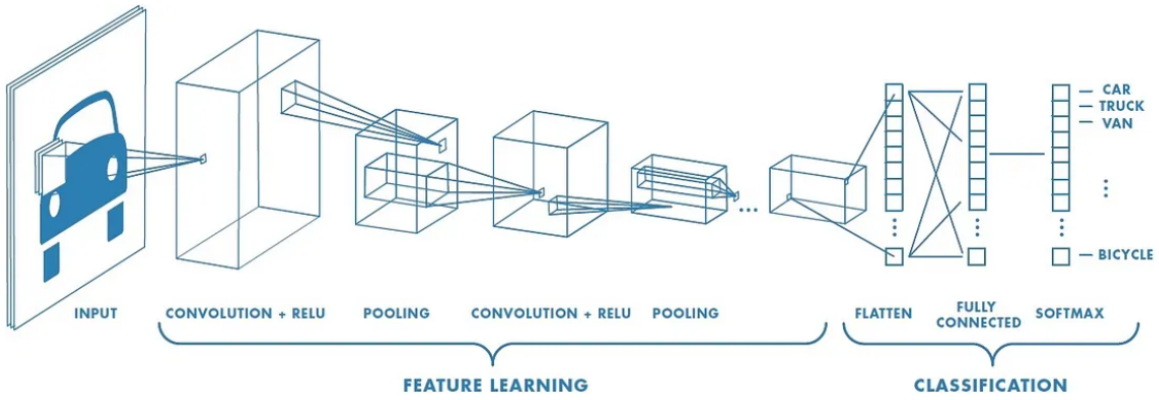

Figure 2.3. Neural network with many convolutional layers.

Source: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

2.2.1 Input

Currently the most popular color format used in computer vision is RGB, it means that we have always 3 channels, where each of them has H rows and W columns, so in the end we get a tensor

H x W x 3, it means an order 3 tensor, but higher order tensors are also allowed. Usually such tensor is an input for CNN and it's built of pixels, which build a similar to matrix arrangement (Wu, 2017). The value of each pixel pinpoints to its hue and brightness.

2.2.2 Convolution Layer

To exemplify how convolution works let's suppose we have image with dimensions 6 x 6 and a convolution kernel of size 3 x 3 (figure 2.4).

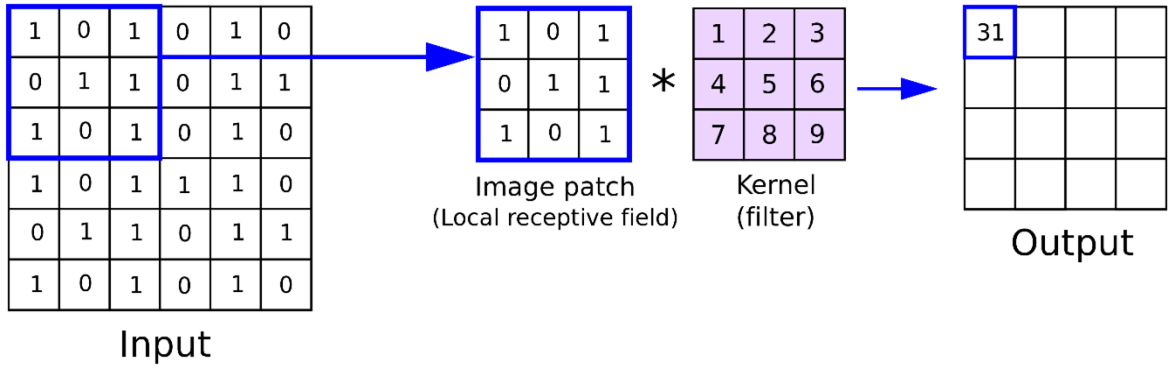

Figure 2.4. Convolution between a kernel and an image.

Source: https://anhreynolds.com/blogs/cnn.html

To convolute our input image we overlap it with kernel and compute the result by multiplying numbers of the same position in both matrixes and then summing them together, what gives us a single number. Then we overlap next part by moving one pixel further. We do so as long as we will reach the bottom right corner. The Figure shows us this operation for order 2 tensor, but for order 3 tensor it looks similar way. If our order 3 tensor has size $H^l \times W^l \times D^l$ then kernel has size $H \times W \times D^l$. We overlap the input tensor and compute results in all Dl channels and then sum the HWDl products (Figure 2.5).

There are two caveats worth remembering. One is the concept of stride. Stride means how much we move with every change of spatial location during convolution process. In both cases here we used stride = 1, but we could use also any stride > 1. Another concept is the fact, that our output in both cases is smaller than input size, in second example the size changed from 7*x*7 to 5*x*5. To maintain the same size of images during convolution, padding can be used. It is done by adding extra rows and columns to the input image before convolution. The amount of padding needed depends on the size of the kernel. We insert an equal number of rows at

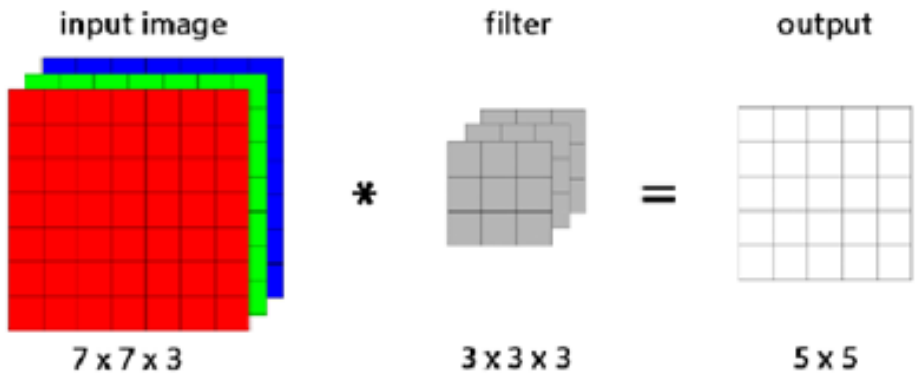

Figure 2.5. A convolution on a RGB image. The three 2D channels result in a 2D matrix.

Source: https://www.researchgate.net/publication/335875913_Filters_in_Convolutional_Neural_Networks_as_Independent_Detectors_of_Visual_Concepts

the top and bottom of the image and an equal number of columns on the left and right (Wu, 2017). After this operation we will achieve same size of input and output image.

The most important reason we use convolution is that it helps us to extract features from the input image, such as edges, textures, corners and others, creating feature maps. Moreover, thanks to sharing parameters convolutional layers learn features that can recognize the same pattern in different sections of the image, the direct gain is that we minimize the number of parameters and it helps to generalize the model (Wu, 2017).

2.2.3 Pooling Layer

Usually in CNN after a convolutional layer we add a pooling layer to decrease the dimensionality of the feature maps given by convolutional layers (Bhatt et al., 2021). We can distinguish 3 main benefits of pooling layers:

- As it reduces the dimensionality of the feature maps then ipso facto it reduces the number of parameters and computational power needed to process the network (Gholamalinezhad & Khosravi, 2020).
- Thanks to it even if we will introduce some small variations in the input, it will not seriously change the output. Let us assume that our input image is of a ship, but a bit shifted to the left, if we would not use pooling, it could end up in not recognizing the ship as a ship, but thanks to the pooling the most important features of the ship are still in down-sampled feature map, so CNN will still recognize this boat, even with some changes.

- It allows to extract the most important features from the feature maps by highlighting the strongest activations, this helps reduce the risk of overfitting, which occurs when a model learns too many parameters from a limited number of examples (Krogh, 2008).

The 2 most popular types of pooling layers are: average pooling and maximum pooling (Bhatt et al., 2021).

Average pooling

While doing average pooling, a kernel of a fixed size slides over the input image and calculate the average value, then place it in the output matrix. Just like in the case of convolution, we do the same for all channels. This way we reduce the spatial size of the input image while retaining important information about the features (Gholamalinezhad & Khosravi, 2020) (Figure 2.6).

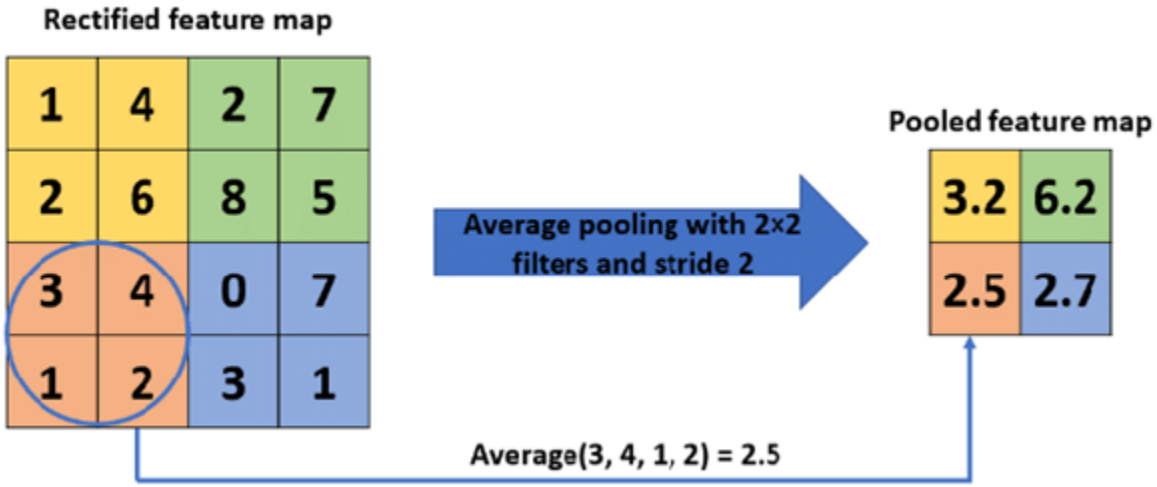

Figure 2.6. Example of average pooling.

Source: (Gholamalinezhad & Khosravi, 2020)

Maximum pooling

Just like in the case of average pooling we use a kernel to slide over the input to downsample it, but instead of calculating the average value in each window we just take the maximum value and then place it in the output (Gholamalinezhad & Khosravi, 2020) (Figure 2.7).

More details and different methods of pooling can be found at (Gholamalinezhad & Khosravi, 2020).

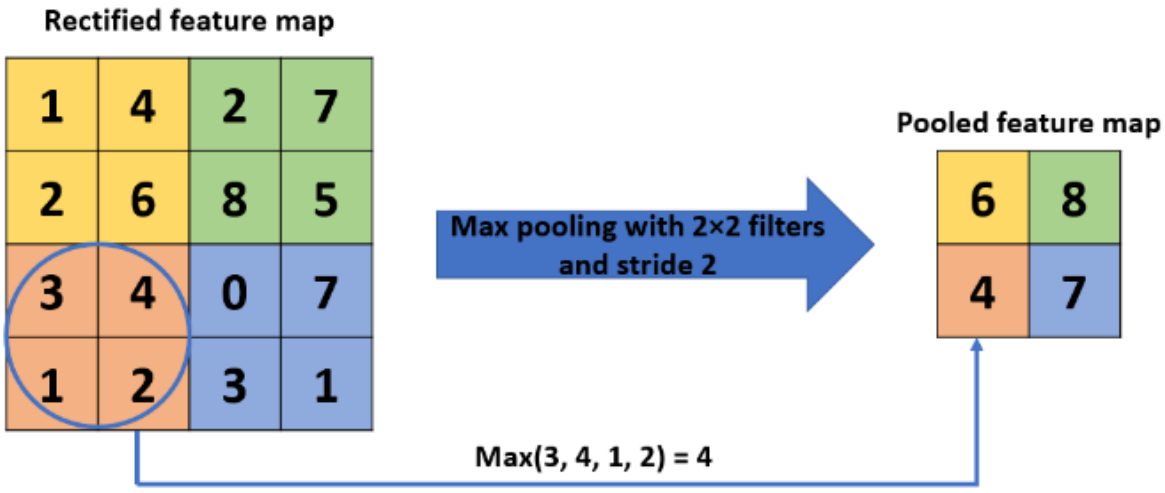

Figure 2.7. Example of max pooling.

Source: (Gholamalinezhad & Khosravi, 2020)

2.2.4 Activation Function

An activation function is a non-linear function applied to the weighted sum of a neuron or layer of neurons, along with its bias. It determines whether the neuron should be activated and generates the corresponding output (Alzubaidi et al., 2021). Here are some popular activation functions in CNN:

- Sigmoid – this function maps numbers to a value between 0 and 1 (Alzubaidi et al., 2021).
- ReLU (Rectified Linear Unit) – this very efficient non-linear function sets all negative values to zero leaving positive ones unchanged.
- Tanh – maps real-valued numbers to a value between -1 and 1 (Alzubaidi et al., 2021).
- Softmax – used often in the output layer for multiclass classification problems. It normalizes real-valued scores into a probability distribution over the classes (Alzubaidi et al., 2021).

2.2.5 Fully Connected Layer

In this type of layer each neuron is connected to every neuron in the previous layer. Usually it can be found at the end of the CNN after the final pooling layer or convolutional layer (Alzubaidi et al., 2021).

This layer performs a non-linear transformation on the input, which is in the form of a vector, and the output of the FC layer is commonly the final output of the CNN.

2.3 Object Detection

According to (Liu et al., 2018) object detection "seeks to locate object instances from a large number of predefined categories in natural images" and due to the complexity of this task it is one of the most challenging problems of the whole research in the area of computer vision. Object detection itself is the foundation of even more interest areas such as autonomous driving, robotics, surveillance, medical imaging, augmented reality and video analysis. More modern problem is to create good models that can be used for generic object detection, it means than instead of recognizing, like in the past, only one object (e.g. face of the pedestrian), a model can recognize objects from multiple predefined categories and then return their specifications. Main problems for any solutions in the area of generic object detection is to create an algorithm with high accuracy and high efficiency (figure 2.8).

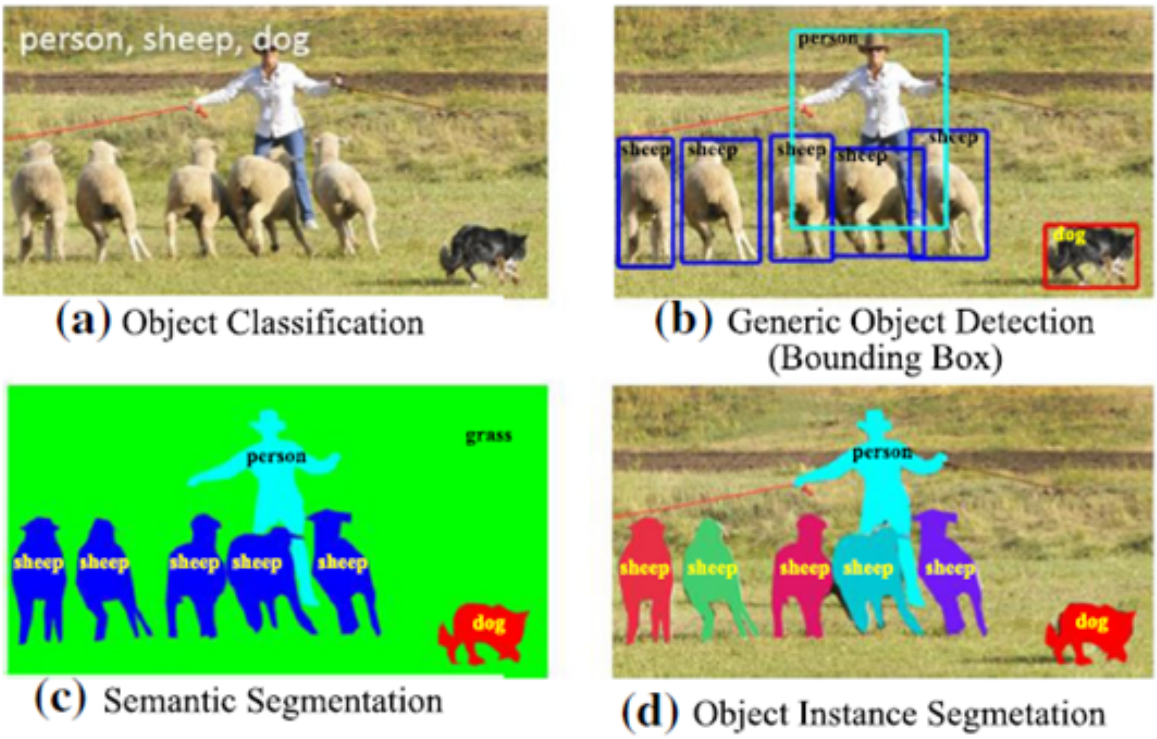

Figure 2.8. Recognition problems related to generic object detection.

Source: (Liu et al., 2018)

There are typically two types of generic object detection models: two-stage detectors and one-stage detectors (Zhao et al., 2019).

2.3.1 Region Proposal-Based Framework

Two-stage detectors at first generate region proposals and afterwards match proposals with various object categories (Zhao et al., 2019). In just few recent years numerously region proposal-based methods were created, so just few most important will be described.

Faster R-CNN

Faster R-CNN was introduced by the Microsoft Research team led by Shaoqing Ren in 2015 (Ren et al., 2015). It was the first near-realtime deep learning detector (Z. Zou et al., 2019). In previous models of the R-CNN family, a slow selective search algorithm was used to generate region proposals, which took around 2 seconds to process an image. However, Faster R-CNN introduced the region proposal network (RPN), which significantly improved the speed, requiring only 10ms for the same task (Figure 2.9) (Ren et al., 2015).

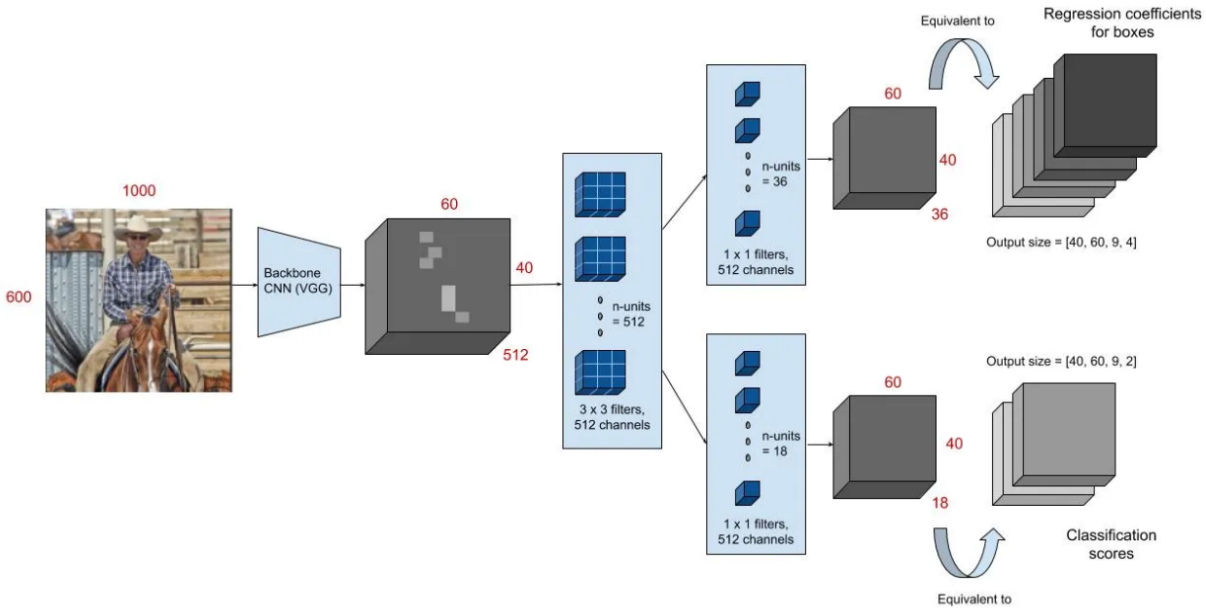

Figure 2.9. The architecture of the region proposal network.

Source: https://towardsdatascience.com/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46

The region proposal network operates by resizing the input and generating a smaller feature map using a network stride. Each point in the feature map represents a potential object location (Ren et al., 2015). Anchors, which are predefined boxes of different sizes and shapes, are placed at regular intervals over the image (Liang et al., 2022). This process is repeated for all locations in the final output of the RPN's backbone network.

To determine if an anchor contains an object, the RPN applies convolution to the final feature map. The decision of whether an anchor is labeled as "positive" depends on specific cri-

teria, such as overlap with ground truth objects (Ren et al., 2015). The rest of the anchors are just ignored when it comes to the training.

What makes Faster R-CNN is not only RPN but also Fast R-CNN as a detector network. More about Fast R-CNN can be found at (Girshick, 2015).

Feature Pyramid Networks (FPN)

Proposed by Lin et al. in 2017 Feature Pyramid Network is an architecture that address the problem of previous feature pyramids, mainly the problems of compute and memory intensity (Lin et al., 2016). One of the first attitudes was to build feature pyramids built upon image pyramids (a), which although let to produce multi-scale feature representation with semantically strong levels, the inference time is considered too high. Later it was opted to use single feature maps (b), but it didn't give satisfying results when it comes to accuracy. The solution applied in for example Single Shot Detector (SSD) applied feature hierarchy (c), but it gives maps with low-level features which incapacitates object recognition (Figure 2.10).

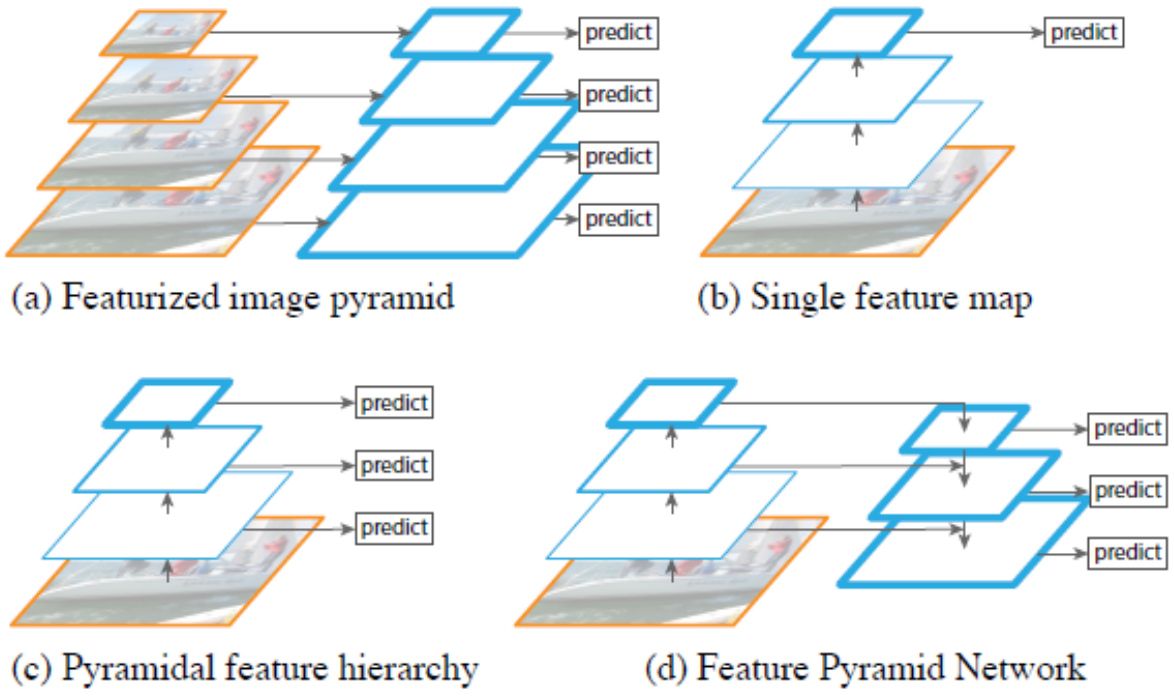

Figure 2.10. Feature maps are indicated by blue outlines and thicker outlines denote semantically stronger features.

Source: (Lin et al., 2016)

Feature Pyramid Network solved problems of accuracy and speed quite effectively. It is a type of top-down architecture, meaning that it starts with a high-level representation of the

image and progressively refines it to produce more detailed representations at lower scales, as we can see at (d).

This architecture is widely used as a backbone network in state-of-the-art object detection models like Mask R-CNN or aforementioned Faster R-CNN.

2.3.2 One-Stage Detectors – YOLO Series

As in two-stage detectors object detection was performed by at first features extraction and then classification and localization, one-stage detectors deploy much simpler architecture, because they consider all region the region proposals in one stage, so they give the bounding boxes and class probabilities in one process (Diwan, Anirudh & Tembhurne, 2023). In this sector we will explore deeply the evolution of YOLO algorithm and its innovations, because it is one of the most popular and crucial algorithms in the area of one-stage detectors, that is used widely in various industries, especially when there is a need of efficient real-time detection, such as: agriculture (S. Zhang et al., 2023) (Yao et al., 2021) (Tian et al., 2019) or traffic surveillance (Laroca et al., 2018) (Y. Zhang et al., 2022) (Han, Chang & Wang, 2021).

2.3.2.1 YOLOv1

In 2015 Redmond et al. introduced one-stage detector that quickly gathered appreciation in the area of object detection – YOLO (Redmon et al., 2015). The main revolution was to solve the problem of localization as a regression problem instead classification problem using deep neural networks.

The name of the algorithm YOLO, comes from You Only Look Once is so because YOLO trains on full images in a single evaluation (Redmon et al., 2015). At first a model divide input into a grid of cells (S x S), then for each of those grids a fixed number of bounding boxes is predicted, confidence score for those bounding boxes and probability for every class (Terven & Cordova-Esparza, 2023). The confidence score represents the model's level of confidence that a bounding box contains an object. Each bounding box has five attributes: the (x, y) coordinates of its center, width, height, and the confidence score. The final score is obtained by multiplying the confidence score with the class probabilities, resulting in the overall score for each bounding box. (Figure 2.11) (Redmon et al., 2015).

The network architecture of YOLO is also very simple, as it contains simply 24 convolutional layers and 2 fully connected layers in the end. It takes an input image of 448x448 and then

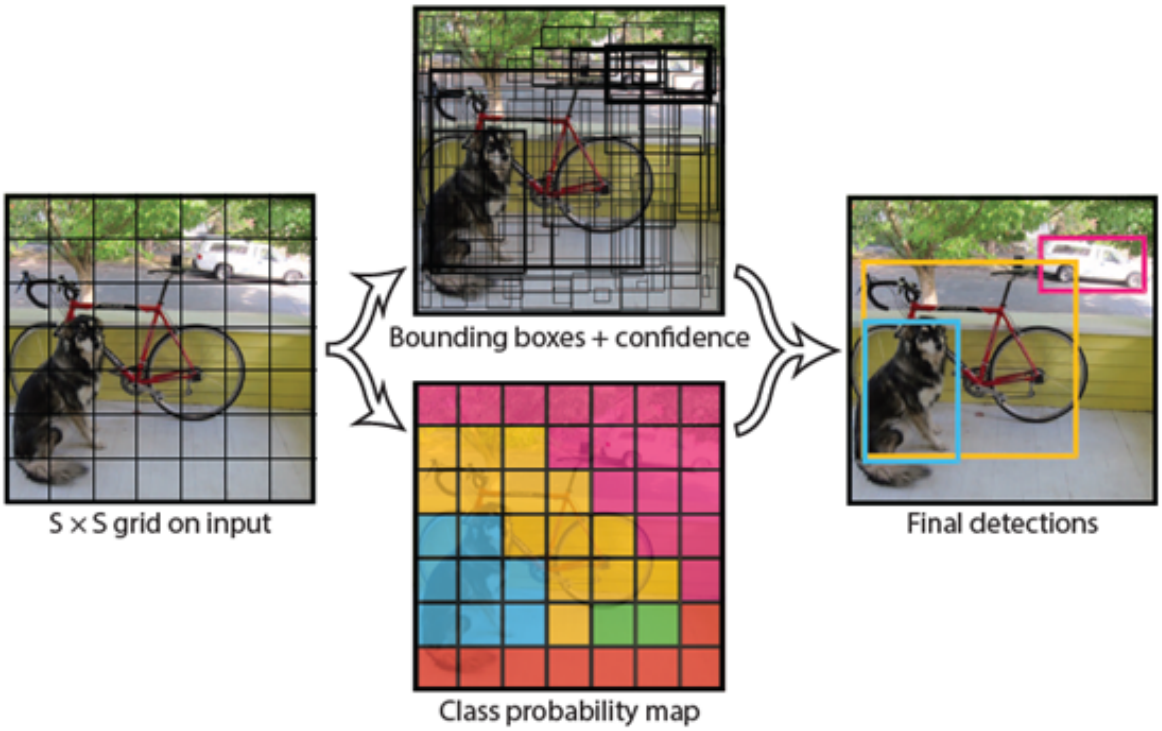

Figure 2.11. The model of YOLO.

Source: (Redmon et al., 2015)

proceed to extract features using aforementioned convolutional layers (Figure 2.12) (Redmon et al., 2015).

2.3.2.2 YOLOv2

Shared in (Redmon & Farhadi, 2016) YOLOv2 is an endeavor to improve the drawbacks of the YOLOv1: noticeable number of localization error and low recall. A number of solutions were applied to address those problems:

Batch normalization. It aids in the convergence of the model and reduces the reliance on additional regularization techniques. It also contributes to the regularization of the model itself, resulting in improved performance and stability during training (Redmon & Farhadi, 2016).

High Resolution Classifier. Earlier object detection models typically operated on low-resolution input images, such as 224x224 in the first version of YOLO. However, with the advent of High Resolution Classifiers, like YOLOv2, the input images are of higher

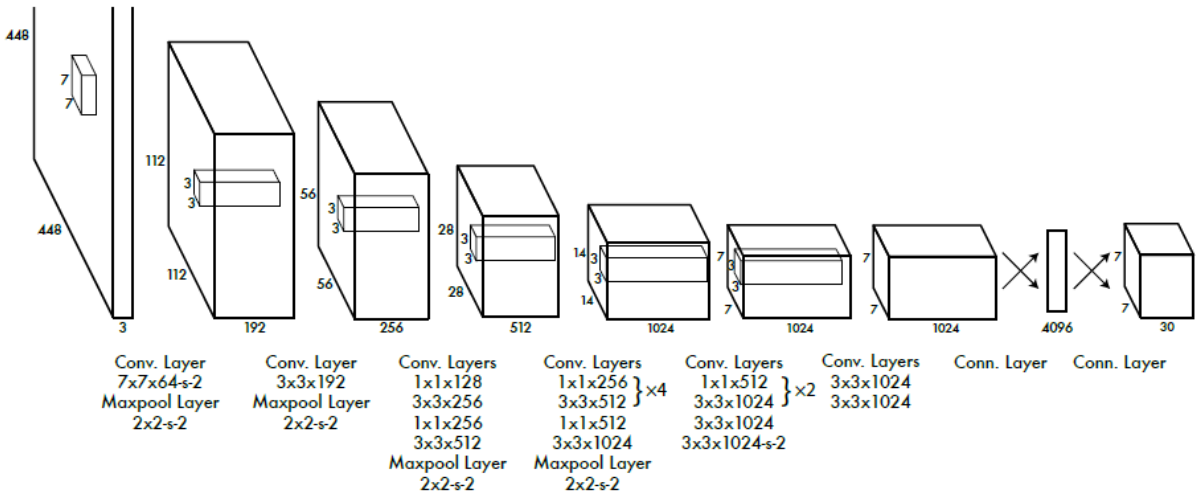

Figure 2.12. The architecture of YOLO.

Source: (Redmon et al., 2015)

resolution, such as 448x448. This increase in resolution allows for improved accuracy without compromising real-time detection speeds (Redmon & Farhadi, 2016).

Convolutional With Anchor Boxes. YOLOv2 also uses this technique, very similar to the one used by Faster R-CNN, in which anchor boxes are used to predict bounding boxes and then the class and objectness (probability assigned to an object being present in a given image) are predicted for each of the anchor boxes (Redmon & Farhadi, 2016).

Dimension Clusters. Rather than manually selecting anchor boxes, they are determined using k-means clustering on the training set. This approach aims to identify the optimal anchor boxes that will enable the model to achieve high Intersection over Union scores, thereby improving overall performance (Redmon & Farhadi, 2016).

Direct location prediction. Previously predicting the coordinates of the bounding boxes was done using random initialization, but this way cause model instability and loss of mAP. Instead bounding box coordinates are predicted relative to the grid-cell locations (Redmon & Farhadi, 2016).

Fine-grained features. Using feature map of 13 x 13 make it easy to predict large objects, but in order to predict effectively smaller objects it is needed to fine-grain features. In YOLOv2 this problem is solved by taking example from skip connections in ResNet, it means by adding a pass through layer that reshape feature map to 26 x 26 resolution and then such features are concatenated with low resolution ones, creating finally a 13

x 13 x 3072 tensor, for which detector can be expanded and thanks to that it can take us of those fine grained features (Redmon & Farhadi, 2016).

Multi-Scale Training. Multi-scale training introduces variability in image scales during training stage. Unlike the previous approaches that relied on fixed image sizes, it embraces the diversity of object sizes that are met in real-world scenarios. The scales are randomly selected (but always are a multiply of 32) and applied to the training images, so that the model is able to learn to detect objects at diverse scales, both small and large objects (Redmon & Farhadi, 2016).

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓
anchor boxes?				✓	✓			
new network?					✓	✓	✓	✓
dimension priors?						✓	✓	✓
location prediction?						✓	✓	✓
passthrough?							✓	✓
multi-scale?								✓
hi-res detector?								✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 2.13. The path from YOLO to YOLOv2.

Source: (Redmon & Farhadi, 2016)

2.3.2.3 YOLOv3

The research article ”YOLOv3: An Incremental Improvement” by Joseph Redmon and Ali Farhadi, released in 2018, introduced YOLOv3 (Redmon & Farhadi, 2018).

In comparison to the previous YOLOv2 the new model was enhanced using features such as:

Detection at multiple scales. It is the ability to identify objects of varying sizes by analyzing information at various scales across a picture. In order to do this, YOLOv3 divides the input picture into a grid and applies 3 anchor boxes of varying sizes to each grid cell. These anchor boxes are in charge of identifying items with particular dimensions and aspect ratios. Thus the model can recognize items of all sizes by taking into account

several scales and anchor boxes. This improves its capacity to handle objects at various distances, especially in case of small objects, what was a problem in previous versions.

Bounding box. In addition to assigning 4 coordinates YOLOv3 also adds an objectness score for each bounding box applying logistic regression. One point is awarded to the anchor box that has the largest overlap with the ground truth object, while zero points are awarded to the others. Only classification loss is experienced by YOLOv3 when an item is not assigned to any anchor boxes (Terven & Cordova-Esparza, 2023).

Darknet-53 backbone. There are 53 convolutional layers total, some of which include residual connections to address the vanishing gradient issue. To capture hierarchical features, the backbone employs a network of convolutional layers with various filter sizes, batch normalization, and leaky ReLU activation functions. The spatial dimensions gradually get smaller while the channel depth gets deeper as the backbone grows. The network may gather both local and global context data using this method.

Class prediction. YOLOv3 utilizes a multi-label classification approach, where each bounding box can be associated with multiple object classes simultaneously, like in case of classes ”Animal” and ”Dog”.

2.3.2.4 YOLOv4

This version of YOLO was the first one to be not published by original authors but other researchers, namely Alexey Bochkovskiy, Chien-Yao Wang and Hong-Yuan Mark Liao (Bochkovskiy, Wang & Liao, 2020).

YOLOv4 pushes the boundaries of object detection by introducing several key enhancements. Firstly, the backbone architecture of YOLOv4 undergoes a significant upgrade with the introduction of CSPDarknet53, which leverages cross-stage partial connections to improve information flow and facilitate better feature extraction.

Other advanced features includes Mish activation function, which introduces a smooth non-linearity, replaces the traditional ReLU activation, leading to better gradients and enhanced model capacity. Additionally, PANet (Path Aggregation Network) is integrated into the architecture to refine feature maps at different scales, enabling better detection of objects across various sizes.

YOLOv4 introduced also an optimal configuration strategy, utilizing larger input sizes during training and testing. This enhanced the model's ability to capture finer details and improves detection accuracy. Moreover, the network employs various training strategies, including mosaic data augmentation, CIOU (Complete Intersection over Union) loss, and enhanced data augmentation techniques, to further improve model robustness and generalization.

2.3.2.5 YOLOv5

YOLOv5 was released by Ultralytics, an AI research company. It introduced several technical improvements and optimizations over previous versions and one of the most significant differences was to implement those enhancements in Pytorch, not Darknet (Jocher et al., 2022).

Biggest changes in comparison to previous versions:

AutoAnchor technique. The AutoAnchor technique in YOLOv5 utilizes clustering algorithms, such as k-means, to analyze the ground truth bounding box dimensions in the dataset. It groups together similar-sized ground truth boxes and calculates the optimal sizes for anchor boxes based on these clusters. This approach automatically configures the anchor boxes to closely match the object sizes present in the dataset. By dynamically adapting the anchor box sizes, YOLOv5 achieves improved accuracy and efficiency in object detection, as the anchor boxes are better aligned with the objects of interest (Terven & Cordova-Esparza, 2023).

Mosaic Data Augmentation. It combines multiple images into a single training sample. By merging different images, the model gains exposure to diverse visual contexts, enabling it to better understand complex scenes.

CutMix and MixUp. CutMix randomly selects patches from different images and combines them to create new training samples, while MixUp blends images and their labels. These techniques promote better generalization of the model to unseen data, enhancing its capacity to handle occlusions and overlapping objects. Its intention is to facilitate preventing overfitting (Pahde et al., 2018).

Self-Adversarial Training. This technique introduces variations, such as random image transformations, noise, or occlusions, to simulate challenging real-world conditions. By training on these adversarial examples, the model learns to adapt and become more resilient,

improving its ability to handle diverse and complex scenarios in object detection tasks (H. Zhang et al., 2017).

2.3.2.6 YOLOX

YOLOX was introduced by the research team at Megvii Inc., a leading AI company based in China (Ge et al., 2021). The most important features of this model are:

Decoupled Head. The regression head and the classification head are two distinct heads in YOLOX. This pair of heads is referred to as a ”decoupled head” since it is not connected to the backbone network and runs on its own. Due to the decoupling, each head may be optimized independently, improving the flexibility, modularity, and model compression effectiveness.

Anchor-free approach. With anchor-free object recognition, bounding box coordinates and objectness scores are directly predicted by the model without the requirement for predefined anchor boxes. The model directly learns to regress the bounding box coordinates relative to a cell's position rather than guessing offsets and scales for anchor boxes. By streamlining the detection procedure, it becomes more adaptable and effective. When working with objects of different sizes and aspect ratios, anchor-free approaches function better since they are not dependent on anchor boxes.

simOTA. The simOTA method in YOLOX tackles the issue of assigning accurate labels to objects when their bounding boxes overlap. It achieves this by optimizing the assignment of labels to bounding boxes, aiming to minimize the transportation cost between them. By simplifying the Optimal Transport (OT) algorithm and formulating the label assignment as an OT problem, YOLOX enhances the precision of object detection. This approach helps to improve the accuracy of object labeling, particularly in scenarios where multiple objects are closely located and their bounding boxes intersect.

2.3.2.7 YOLOv7

YOLOv7 was introduced in July 2022 and extraordinarily quickly gained wide recognition for being the fastest real-time object detection model up to that time (Wang, Bochkovskiy & Liao, 2022).

Also in this model we have 3 different versions that depends in this case on our GPU:

YOLOv7. The default version that is dedicated especially for normal GPU.

YOLOv7-tiny. Designed for usage on edge GPU, this model may be used, for instance, on mobile devices, which is something that is increasingly used in today's commercial solutions that give users real-time object recognition. The usage of ReLU activation function, as opposed to SiLu utilized by the other two versions of the model, is another significant distinction. The former is computationally efficient, which is expected given the limited processing capability of edge devices.

YOLOv7-W6. A specialized model designed for cloud-based GPU computation, specifically created to handle massive datasets, leverage substantial computational resources, and address a wide range of application domains.

In this model that is based on YOLOv4, such the most important improvements made are:

Model Scaling. The YOLOv7 model leverages a concurrent scaling technique that simultaneously adjusts the depth and width of the network. By incorporating layer concatenation, this approach maintains the model's optimal architecture while effectively adapting to various scales.

Re-parameterized convolution. An analysis was conducted on the combination of re-parameterized convolution (RepConv) with various network architectures. The results revealed that the identity connection in RepConv had a detrimental effect on the residual in ResNet and the concatenation in DenseNet. To overcome this issue, a modified version of re-parameterized convolution, called planned re-parameterized convolution, was introduced without the identity connection. This design choice was based on the notion that when replacing a convolutional layer with residual or concatenation, the identity connection should be eliminated. The effectiveness of planned re-parameterized convolution was validated through an ablation study, showcasing its benefits in both residual-based and concatenation-based models (Wang, Bochkovskiy & Liao, 2022).

Extended efficient layer aggregation network (E-ELAN). In YOLOv7, the E-ELAN (Extended Efficient Layer Aggregation Network) strategy was introduced to boost the learning and convergence efficiency of deep models. By merging and shuffling cardinality, E-ELAN enables the combination of features from different groups while maintaining the integrity

of the original gradient path. This approach effectively addresses the challenge of controlling the shortest longest gradient path. It is worth noting that E-ELAN is specifically designed to work seamlessly with models that feature an unlimited number of stacked computational blocks (Terven & Cordova-Esparza, 2023).

2.3.2.8 YOLOv8

Yolov8 was released just in January 2023 by the authors of YOLOv5 - Ultralytics. Despite being the fastest real-time model available at the time of writing, still there is no research paper created by authors that would explain in detail what improvements and what motivations are staying behind this state-of-the-art object detection model, that achieves both excellent accuracy and performance at the same time. That is why in order to describe YOLOv8 one must rely on the github where it was released (Jocher, Chaurasia & Qiu, 2023) and multiple research papers that either implemented YOLOv8 or modified it (Aboah et al., 2023; Ahmed et al., 2023; Imran et al., 2023; Kim, Kim & Won, 2023; Y. Li et al., 2023; Munin et al., 2023).

This model, like the majority of new ones, was offered in a limited number of configurations in an effort to meet the market's widely diverse demands. Figure 2.1 contains a comparison table of the performance characteristics of several YOLOv8 object detection model versions as measured by the benchmark dataset COCO val2017, where:

- size - Resolution of the input images used during inference.
- mAPval - Mean Average Precision (mAP) value achieved on the validation set.
- Speed CPU ONNX (ms) - Inference speed in milliseconds using the CPU with ONNX runtime.
- Speed A100 TensorRT (ms) - Inference speed in milliseconds using the A100 GPU with TensorRT.
- Params (M) - Number of model parameters in millions.
- FLOPs (B) - Number of floating-point operations (FLOPs) in billions. FLOPs (B) is a computational metric utilized to assess the complexity of a model. It quantifies the number of arithmetic operations, including additions and multiplications, executed in a neural network during the forward pass for a specific input.

All versions of YOLOv8 achieve better mAP50-95 when compared to other YOLO models in terms of parameters and latency, as can be seen in Figure 2.14, even though it occasionally

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Table 2.1. Performance metrics of different variants of the YOLOv8

Source: (Jocher, Chaurasia & Qiu, 2023)

seems like a small increase of about 1-2%, compared to contemporary object detection models it is actually a significant difference.

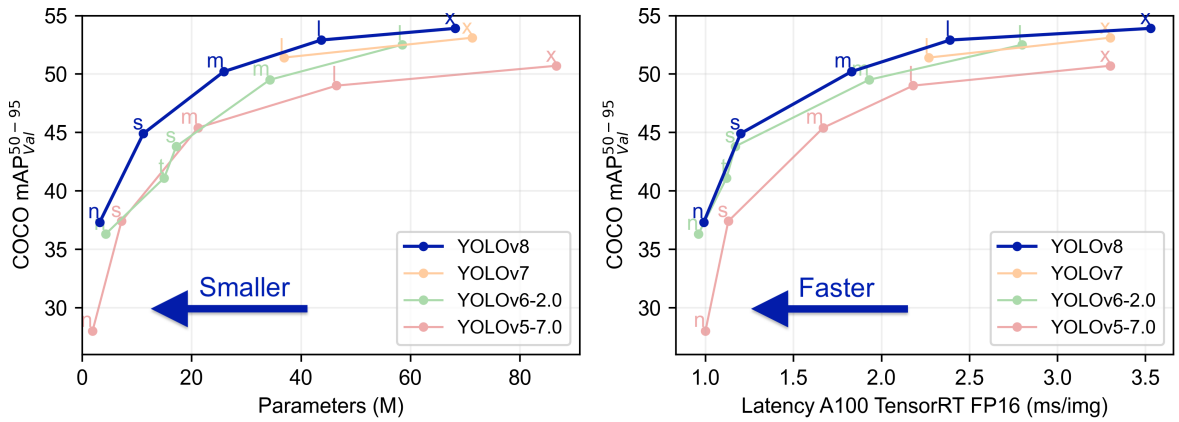

Figure 2.14. Comparison between YOLOv8, YOLOv7, YOLOv6 and YOLOv5.

Source: (Jocher, Chaurasia & Qiu, 2023)

On the Figure 2.15 an architecture of YOLOv8 is visualized. Main differences made in backbone in comparison to the predecessor YOLOv5 are:

- Replacing the C3 module with C2f that integrates context of the image with high-level features to enhance overall accuracy (Terven & Cordova-Esparza, 2023).
- The first 6x6 convolutional layer was replaced with a 3x3 layer to capture more local details in the input image. This change maintains the stride of 2 for downsampling and adds only one pixel of padding to preserve edge information.

- Removing two convolutional layers in the neck without adding anything in their place and without changing other layers.
- By replacing the initial 1x1 convolutional kernel in the Bottleneck with a 3x3 convolutional kernel, it is now easier to capture connections between nearby features and more intricate patterns in the input.
- An anchor-free model with with a decoupled head architecture is utilized, and as part of this modification, the objectness branch is removed. This design change allows for a more streamlined and efficient model architecture (Terven & Cordova-Esparza, 2023).

2.4 Evaluation Indicators

In order to effectively evaluate and compare the new model, we employ five key indicators that offer insights into different aspects of the model's accuracy, robustness, and efficiency:

- precision
- recall
- mean average precision (mAP)
- F1 score
- frames per second (FPS).

To understand those indicators it is needed firstly to elucidate what binary confusion matrix is. The binary confusion matrix is a valuable tool used to assess the performance of a classification model in a classification problem. It organizes the model's predictions into four categories: true positives, true negatives, false positives, and false negatives. True positives represent the instances correctly identified as positive by the model, while true negatives represent instances correctly identified as negative. False positives indicate instances incorrectly labeled as positive, and false negatives represent instances incorrectly labeled as negative (H. Li et al., 2021).

Precision. It measures the proportion of correctly predicted positive instances out of all instances predicted as positive by the model. Mathematically, precision is calculated as the ratio of true positives (TP) to the sum of true positives and false positives (FP):

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

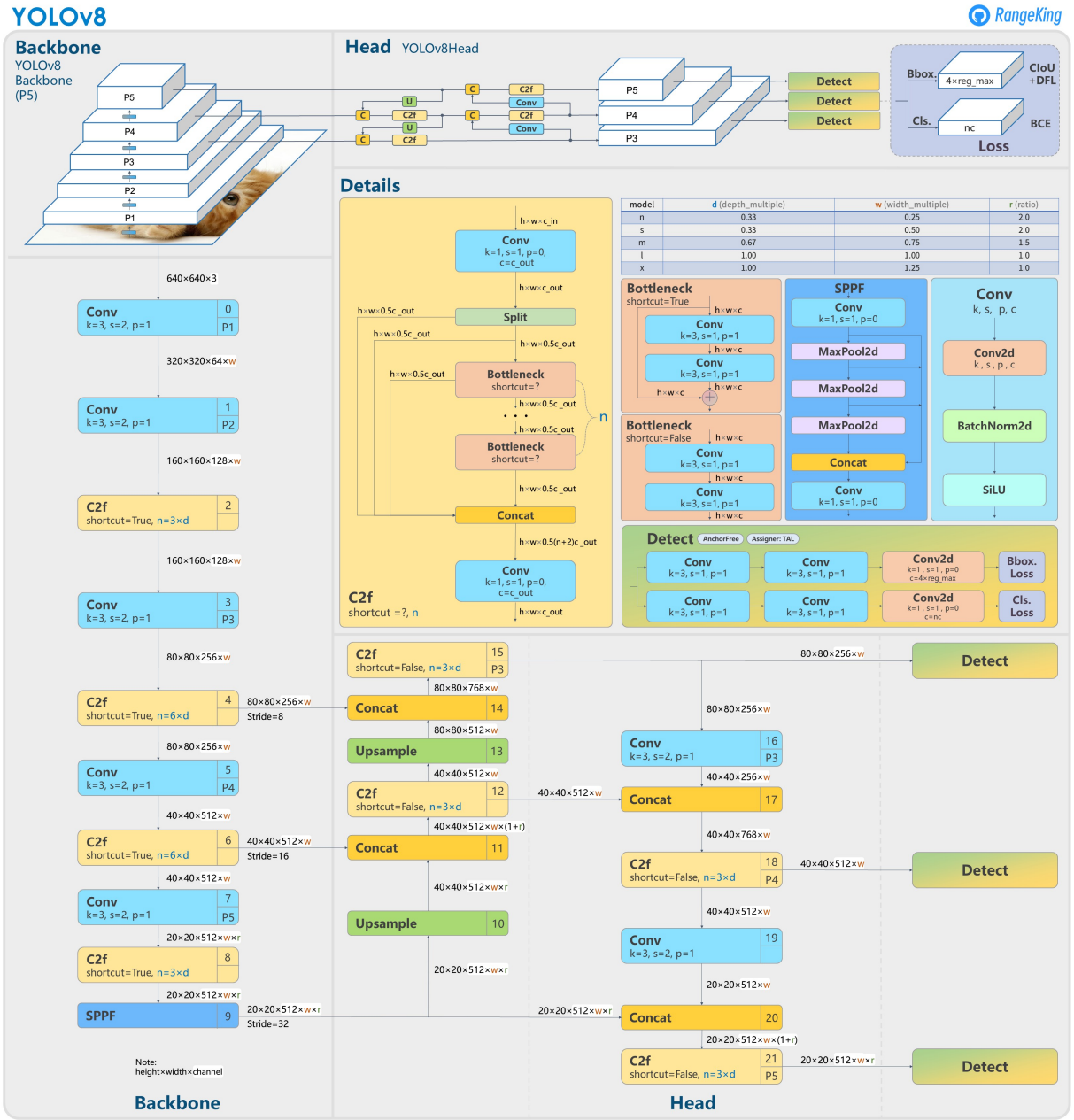

Figure 2.15. Visualisation of YOLOv8 architecture.

Source: https://github.com/ultralytics/ultralytics/issues/189

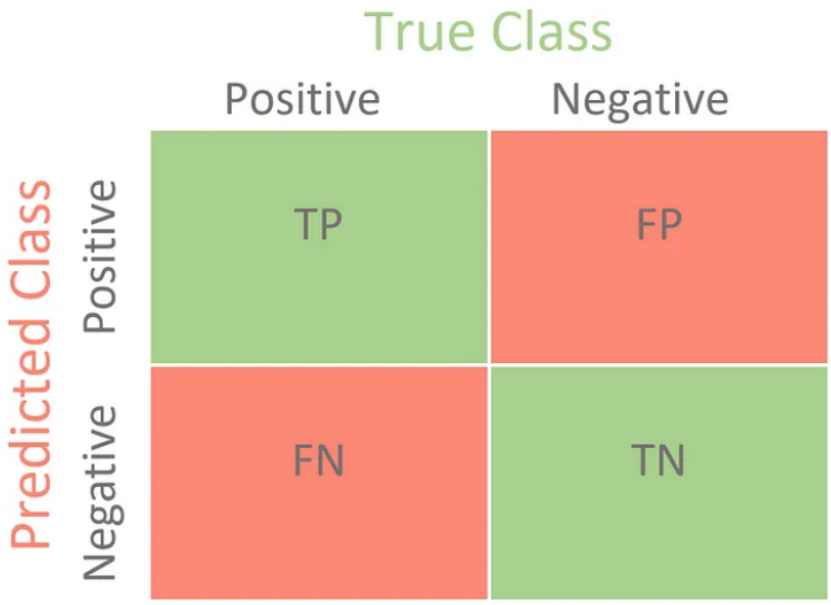

Figure 2.16. Confusion Matrix for Binary Classification.

Source: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826

A higher precision value indicates that the model is effective in minimizing false positives and accurately predicting positive instances. (H. Li et al., 2021).

Recall. Its purpose is to assess sensitivity of the model by measuring the proportion of correctly predicted positive instances out of all the actual positive instances present in the dataset. To calculate recall, one divides the number of true positives (TP) by the sum of true positives and false negatives (FN):

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

A high recall value indicates that the model exhibits a low rate of falsely labeling positive instances as negative. This signifies the model's effectiveness in capturing and identifying positive instances accurately, without missing them (H. Li et al., 2021).

F1 score. It combines precision and recall into a single value to provide a assessment of the model's performance. It is calculated as the harmonic mean of precision and recall:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.3}$$

It ranges from 0 to 1, where a score of 1 indicates perfect precision and recall, while a score of 0 indicates the worst possible performance (H. Li et al., 2021).

Mean average precision. In object detection tasks, mAP takes into account the precision-recall trade-off by considering the precision values at different recall levels. It assesses how well the model can identify objects of interest across various levels of recall (H. Li et al., 2021).

Within the mAP metric, two popular variants are mAP50 and mAP50-95:

- mAP50 refers to the mean Average Precision at an IoU (Intersection over Union) threshold of 0.5. This threshold considers a relatively lenient overlap between the predicted bounding boxes and the ground truth objects. It measures how well the model can accurately detect and localize objects when allowing for a moderate level of bounding box overlap.
- mAP50-95 extends the evaluation to a range of IoU thresholds from 0.5 to 0.95. This wider range of thresholds requires better bounding box overlap, providing a more comprehensive assessment of the model's detection performance across various levels of overlap.

A higher mAP score indicates superior performance, signifying a model with heightened precision in identifying objects across diverse recall values. mAP stands as a valuable metric for comparing and benchmarking object detection models, providing a comprehensive evaluation of their detection accuracy and localization capabilities.

Frames per second. It measures the speed at which consecutive frames are displayed or processed in video playback or graphics rendering. It is an important indicator of the smoothness and quality of visual content. FPS quantifies the number of frames that can be shown or processed within one second. A higher FPS value indicates a faster rate of frame display or processing, resulting in a smoother and more fluid visual experience. For instance, a video with a higher FPS will appear more seamless and realistic compared to a video with a lower FPS (H. Li et al., 2021).

Chapter 3

Implementation of YOLOv8

3.1 Platform

Colab is an online platform by Google for writing, executing, and collaborating on Python code using Jupyter notebooks. It is designed for machine learning and data analysis, providing access to powerful hardware resources like GPUs and TPUs.

Colab gives access to Tesla T4 GPU developed by NVIDIA. It is designed specifically for DL and AI inference. It is based on the Turing architecture and features 16 GB of high-speed GDDR6 memory.

Table 3.1. Version of used tools

Tool	Version
GPU	Tesla T4
Python	3.10.12
torch	torch-2.0.1+cu118
YOLOv8	YOLOv8.0.119

3.2 Dataset

3.2.1 SeaShips

The dataset used as a basis for learning our model is a large-scale ship dataset called SeaShips, which consists of 31,455 meticulously annotated, high-resolution images measuring 1920 × 1080 pixels (Shao et al., 2018). These images captured six principal ship types:

- ore carrier
- bulk cargo carrier
- general cargo ship
- container ship
- fishing boat
- passenger ship.

Figure 3.1. Example images of six ship types.

Source: (Shao et al., 2018)

The creation of the SeaShips dataset was facilitated by the utilization of an in-field video monitoring system deployed around the captivating Hengqin Island, situated in Zhuhai city, China. Within the project, a network of 156 cameras was strategically positioned across 50 distinct locations along the island's northwest border. This deployment effectively encompassed a vast coastal area spanning a total of 53 square kilometers.

A subset of 45 cameras, situated in various seashore locations, played a pivotal role in capturing the selected images for the SeaShips dataset. This deliberate selection process ensured a comprehensive representation of diverse maritime scenarios. Videos were diligently recorded from each camera during multiple time periods, including January, April, August, and

October of both Year 2017 and Year 2018. The recording duration ranged from 6:00 a.m. to 20:00 p.m. daily, resulting in an extensive and varied collection of visual data.

To ensure the SeaShips dataset encompassed a wide spectrum of ship-related characteristics and complexities, the chosen images deliberately incorporated various features. These features included different ship types, hull parts, scales, viewpoints, illumination conditions, and varying degrees of occlusion. The aim was to create a dataset that accurately represented the intricacies encountered in real-world maritime environments.

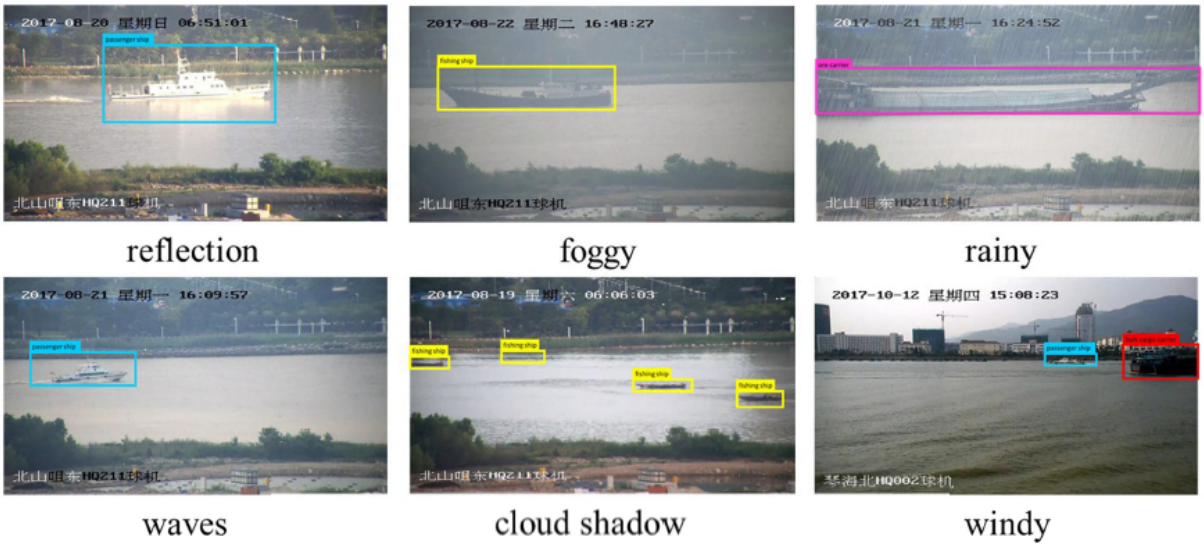

Figure 3.2. The ship detection results under some extreme weather conditions.

Source: (Shao et al., 2018)

3.2.2 SeaShips from roboflow.com

However, the dataset used in this experiment is only a subset of the one just mentioned from roboflow.com. It is called SeaShips Dataset and is available for use to everyone (YOLOv5Seaships, 2022). It contains 6979 photos and the tags are already annotated according to the YOLOv8 format. The only changes made to each image are auto-orientation of pixel data (with EXIF-orientation stripping) and resizing to 640x640, without using any augmentation techniques. Another significant difference from the perspective of teaching a convolutional neural network model is the fact that it has already been divided into 2 sets: training set and validation set, where the sizes are respectively 5592 and 1387 already labeled images.

The dataset was already prepared specifically for learning YOLOv8 model, according to the documentation provided by the authors[1]:

- Each image is 640 pixels in size, the same resolution that the model was trained on using benchmark COCO dataset. Naturally, greater levels might also be employed, such as a level of 1280 pixels, particularly if there are a lot of little things.
- Since every picture has been correctly labelled, each label snugly encloses every object, and the class names are all consistent. Open one of the mosaics made at the start of training (`train_batch*.jpg`), as the one in Figure 3.3, to verify that all labels are shown correctly.
- Unfortunately, only in one instance (fishing boat) does the total number of instances for a class reach 1500, the quantity that is suggested for every class. Bulk cargo and ore carriers are the next two to have more than 1000 photographs, followed by container and general cargo ships with just over 600 images, and passenger ships, which have only over 200 images. Even though it does not adhere to the YOLOv8 dataset recommendations, it will be quite instructive to observe how this object identification model handles such diverse numbers of photos per class (see Figure 3.4).

3.2.3 wsodd-usv-dataset from roboflow.com

Another dataset used was wsodd-usv-dataset from roboflow.com that is full of images very similar to the SeaShips dataset from roboflow.com, but was created for a more general purpose as there are classes such as: animal, ball, boat, bridge, buoy, grass, harbor, mast, person, platform, rock, rubbish, ship and tree (lyx.limyixian, 2022). There are 6057 images in train set and 1409 in validation set.

3.3 Enhancement of Dataset

Initially, the complete SeaShips dataset, including images, was downloaded to the local hard drive for further processing. Since the dataset obtained from roboflow.com was divided only into a train set and a validation set, it was decided to create a separate test set. To accomplish this, a Python package called split-folders was employed. The dataset was split randomly, with

[1] https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/

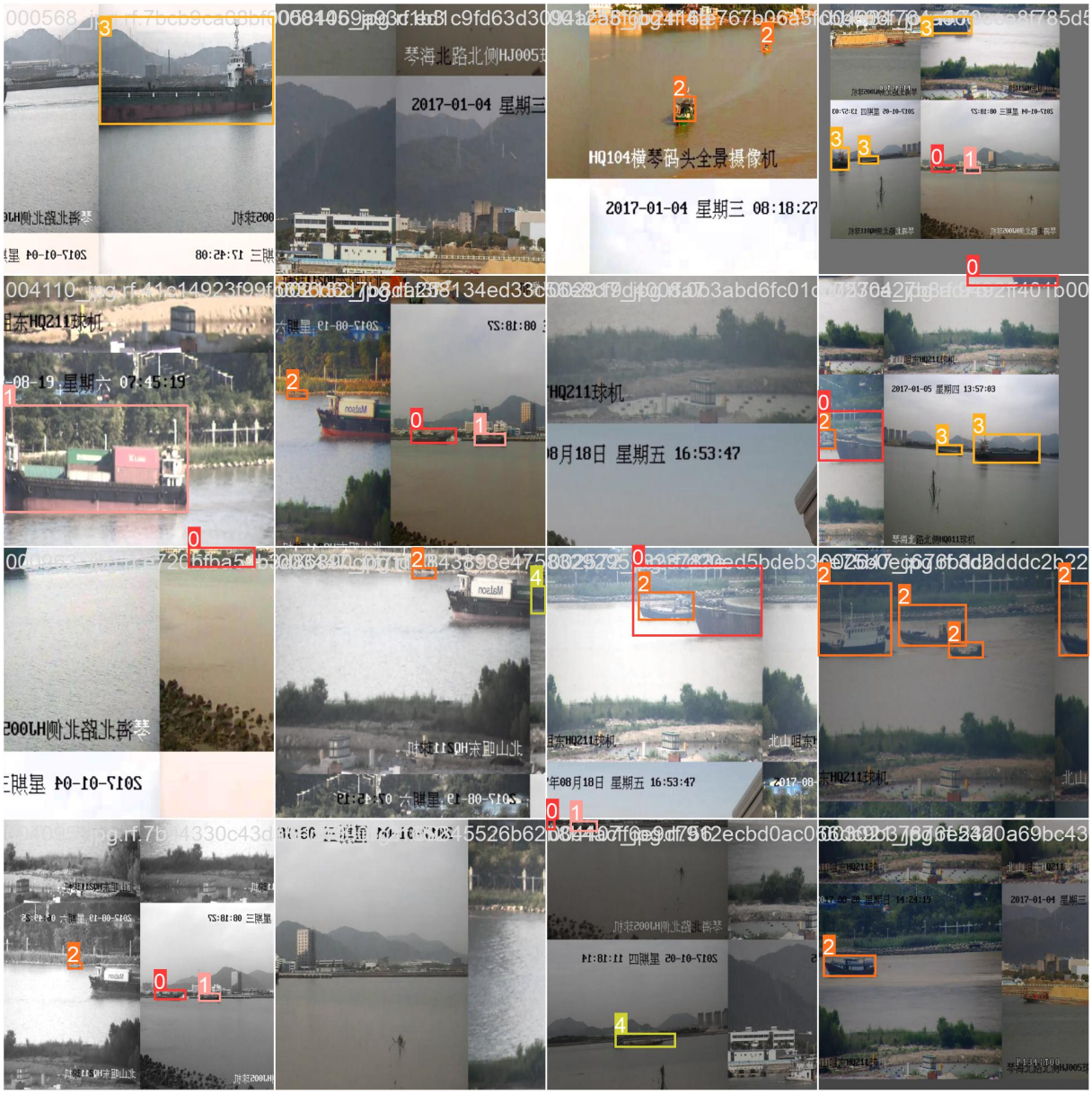

Figure 3.3. Example mosaic with labels.

Source: own work

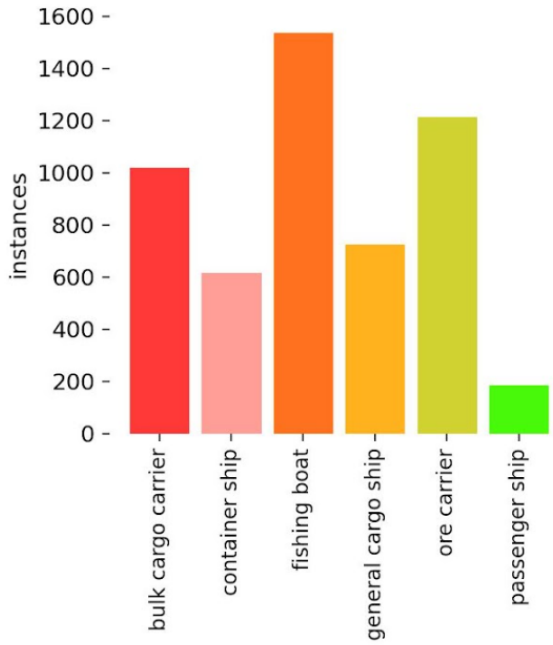

Figure 3.4. Instances per class.

Source: own work

10% of the images (560 images) being allocated to the test set, while the remaining 90% were retained in the train set.

Another crucial enhancement made to the original dataset was to add between 1-10% of background images to the train set, as all images always had at least one instance of a class in it. Such practice is recommended for object detection datasets, as it lets to decrease False Positives[2]. In benchmark datasets like COCO val2017, it is common to include a small percentage of background images. These background images do not contain any labeled instances of classes and serve as a representation of the background context. To integrate background images into the training set using YOLOv8, the process is straightforward. The wsodd-usv-dataset from roboflow.com was used for this purpose.

To obtain the background images, the ”Marked null” option was selected while browsing the dataset on the roboflow.com website. Additionally the images were resized into 640 x 640 resolution, so it will be the same as original train set. This ensured that only background images were provided, which were then downloaded separately to the computer. Initially, there were 380 such images. However, upon reviewing them, it was observed that many of them did not fit well with the original dataset.

[2] https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/

For instance, Figure 3.5a displays an unfit image where the camera is too close to the shore, and the buildings are highly detailed. In Figure 3.5b, despite being a background image, there are clearly two ships present. Including such unfit images could potentially degrade the overall accuracy of the model. Therefore, a manual review was conducted, and the unsuitable images were removed from the additional dataset.

After the review, 273 background images that fit well with the original dataset were directly added to the train set for further training.

(a) close city **(b) blurred ships**

Figure 3.5. Examples of bad background images

Source: own work

3.4 Training a Model

The initial decision was to select a model for training the dataset and performing inference. Considering that YOLOv8 is currently considered the most advanced and state-of-the-art model for real-time object detection, the focus was on experimenting with the three smallest models provided by Ultralytics. This approach aimed to find the optimal balance between frames per second (FPS) and accuracy within these models:

- YOLOv8n
- YOLOv8s
- YOLOv8m.

We trained all three of them with default parameters on our dataset. In Table 1 we can see the results given by testing the best parameters for each of the models tested on validation set:

Table 3.2. Results for default models

Parameter	Yolov8n	Yolov8s	Yolov8m
mAP50	0.880	0.899	0.911
mAP50-95	0.669	0.658	0.711
FPS	96	74	47
Size (MB)	6	21.5	49.6

Based on the results presented in the table, we can justify using the YOLOv8s model over the other variants (YOLOv8n and YOLOv8m) for deploying it in practice, for example in surveillance cameras at the shore, particularly when considering the Frames Per Second (FPS) metric, because although most of the cameras operate at 30 FPS, for some scenarios there is a need for higher FPS, especially in case when we want to capture fast movements of detailed actions, such cameras operate at even 60 FPS, thus the use of YOLOv8s seems like an rational trade-off between an amount of FPS and overall accuracy, that we will try to further improve.

Then we proceeded with enhancing chosen model mAP50 and mAP50-95 by changing some of the default parameters, while leaving the same rest. In Table 3.3 are the parameters that were left the same in all trained models:

The following were the reasons we decided to leave these values intact:

- epoch: We decided to perform 100 iterations on the dataset during training, mostly because more iterations would take a lot of time and be impractical given that accuracy converges fairly quickly and Google Colab's free version only allows for limited access to GPUs.
- imgsz: The 640x640 resolution of the data utilized for the YOLOv8 pretrained models, which were trained using COCO val2017 data, was chosen in order to maintain accuracy. Larger image sizes would also demand more computing power, which was already constrained by Colab's free version, even while they made it easier to capture more detailed features.
- optimizer: SGD (Stochastic Gradient Descent) was chosen as the optimizer for its popularity and widespread use in adjusting model parameters. It offers the flexibility to man-

Table 3.3. Default training parameters

Parameter	Value
epochs	100
imgsz	640
optimizer	SGD
cos_lr	False
amp	True
iou	0.7
lr0	0.01
momentum	0.937
warmup_epochs	3.0
weight_decay	0.0005
translate	0.1
scale	0.5
fliplr	0.5
mosaic	1.0

ually control the learning rate, which can be beneficial in scenarios with large datasets and high-dimensional data like the SeaShips dataset. Additionally, SGD is known to perform well in such cases, making it a suitable choice for optimizing the model's performance. Another option considered was the Adam optimizer, which has demonstrated better generalization capabilities in some empirical studies (Choi et al., 2019). However, Adam requires careful selection of hyperparameters to ensure optimal performance, as suboptimal choices may lead to worse results.

- amp: Automatic Mixed Precision (AMP) is a technique employed to accelerate training and inference by utilizing lower precision data types, such as float16. By incorporating AMP, we can minimize the demand for computational resources while still achieving efficient performance. In our case, we specifically adopted AMP to restrict the usage of GPU and optimize resource utilization.
- iou: The term "iou=0.7" represents the threshold value for Intersection over Union (IoU). By setting it to 0.7, we prioritize more accurate detections over a higher number of de-

tections. This ensures that the model focuses on correctly identifying objects rather than maximizing the overall detection count.

- lr0: With regards to the learning rate, the value lr0 = 0.01 denotes the initial rate employed in the training procedure. By setting it to 0.01, we adopt a strategy of larger step sizes, facilitating quicker convergence and early reduction of the loss function.
- momentum: The momentum coefficient, specified as 0.937, plays a crucial role in the optimization process. With this value, the algorithm considers 93.7% of the previous update direction when calculating the current update. By incorporating a significant portion of the past information, it promotes smoother and more consistent progress throughout the training process.
- warmup_epochs: The warm-up epochs in the training process denote the initial phase where the learning rate gradually increases over a specified number of epochs. By default, this number is set to 3, allowing the model to explore the parameter space and adapt to the training data before reaching the desired learning rate (lr0). With a total of 100 epochs, this choice of 3 warm-up epochs provides sufficient time for the regular training process to yield satisfactory results.
- weight_decay: Weight decay is a well-known regularization technique employed to counteract overfitting in models. It accomplishes this by adding a penalty term to the loss function, discouraging the model from having excessively large weights. In our case, the weight decay is set at 0.0005, which is a relatively small value. However, based on the performance of our models, overfitting is not observed, and therefore it was deemed appropriate to keep the weight decay at its default value.
- translate: The ”translate” parameter in YOLO denotes the translation augmentation applied to the training data. With a value of 0.1, it implies that objects are randomly translated up to 10% of their width and height. This augmentation technique enhances the model’s ability to handle objects that may appear in various positions, contributing to its adaptability in real-world scenarios.
- scale: The ”scale” parameter in YOLO represents the scaling augmentation applied to the training data. When set to 0.5, it indicates that objects in the training images can be randomly scaled down to 50% of their original size. This augmentation technique enhances the model’s robustness in detecting objects of different sizes, including smaller or more

distant ships in our case. By incorporating this scaling variation, we improve the model's capability to handle diverse scenarios and increase the overall detection performance.

- fliplr: The "fliplr" parameter in YOLO pertains to the augmentation technique of horizontal flipping. With a value of 0.5, there is a 50% probability of each training image being horizontally flipped during the training process. By incorporating this augmentation, the model is trained to handle scenarios where the camera undergoes horizontal flipping, which can occur due to factors like wind or other environmental conditions. This augmentation enhances the model's adaptability and robustness, enabling it to effectively detect and classify objects in various orientations.
- mosaic: The "mosaic" parameter in YOLO controls the application of the mosaic augmentation technique during training. Setting it to 1.0 applies the augmentation to all training samples. Mosaic augmentation combines multiple images into a single one, improving the model's ability to detect ships in close proximity or densely populated areas.

Table 3.4 displays our experimental approach to improve the accuracy by modifying certain parameters from their default values. We aimed to surpass the performance achieved using the default parameter settings.

Table 3.4. Changed training parameters

Models	**batch**	**lrf**	**cos_lr**
Model 1	32	0.01	False
Default model	16	0.01	False
Model 2	16	0.0001	False
Model 3	8	0.001	False
Model 4	16	0.001	False
Model 5	8	unknown	True
Best model	8	0.0001	False

In Table 3.5, are provided a detailed overview of the best results obtained for the mAP50 and mAP50-95 metrics across the trained models.

The following were the reasons it was decided to change those values and explanation how it could influence the results:

Table 3.5. Indicators for trained models

Models	mAP50	mAP50-95
Model 1	0.868	0.663
Default model	0.899	0.658
Model 2	0.897	0.679
Model 3	0.891	0.681
Model 4	0.890	0.689
Model 5	0.903	0.694
Best model	0.894	0.697

- batch: During our experiments with different batch sizes, we found that using a smaller batch size of 8 yielded better results for aforementioned dataset containing ships. While smaller batch sizes require less memory and computational resources, they also introduce some potential challenges.

  One such challenge is the impact on generalization performance. With smaller batch sizes, the model may encounter fewer diverse examples in each iteration, potentially limiting its ability to learn robust representations of ships. However, in this specific case, the smaller batch size of 8 was sufficient to capture the necessary ship variations present in utilized dataset. It suggests that this dataset may have contained a reasonable level of diversity even within smaller batches.

  Additionally, smaller batch sizes can introduce noise in the gradient estimation, which affects the stability of weight updates during training. This noise can lead to less consistent and reliable updates, potentially hindering the learning process. However, in the experiments, it was observed that the model trained with a batch size of 8 achieved better performance on ship detection, indicating that the noise introduced by the smaller batch size did not negatively impact the learning process significantly.

  Overall, the choice of a batch size of 8 for SeaShips dataset proved to be better in terms of performance. It effectively balanced the memory requirements, model generalization, and stability of weight updates, leading to improved ship detection accuracy.
- lrf: During our experiments, we found that adjusting the learning rate factor (lrf) to 0.0001 instead of the default value of 0.01 resulted in the best performance for this spe-

cific ship dataset. This lower learning rate proved to be more suitable for our specific scenario, and there are several reasons behind this observation.

Firstly, ships in used dataset exhibit complex visual characteristics, such as variations in size, shape, and appearance. The higher learning rate of 0.01 could potentially lead to overshooting optimal solutions during the model's optimization process, making it challenging to capture fine-grained details of the ships accurately.

Secondly, by using a smaller learning rate of 0.0001, the model was able to make more precise adjustments to its parameters. This finer control allowed it to learn more accurate representations of ships, which is crucial for detecting similar ships of different types. Precise localization and precise bounding box predictions are essential in such cases.

Moreover, employing a lower learning rate contributes to the stability of the model during training. It helps prevent large fluctuations in the loss function, ensuring a smoother optimization process. This stability allows the model to converge to a better solution and avoid getting stuck in suboptimal states.

- cos_lr: The use of a cosine learning rate schedule was explored, which gradually reduces the learning rate following a cosine curve. This technique aims to achieve a smoother decrease in the learning rate throughout training, potentially facilitating better convergence of the model. However, contrary to expectations, it was found that using a constant learning rate (cos_lr=False) was more effective for ship detection in our dataset.

  Several factors could explain why a constant learning rate yielded better results in this experiments. Firstly, ships in our dataset exhibit diverse visual characteristics, and maintaining a stable learning rate throughout training allowed the model to focus on learning and adapting to these specific ship features without being influenced by the fluctuations introduced by the cosine learning rate schedule.

  Secondly, precise localization and accurate bounding box predictions are crucial for ship detection tasks. By using a constant learning rate, a consistent and reliable optimization process was provided, enabling the model to concentrate on refining these critical aspects without being affected by variations in the learning rate.

  Furthermore, the evaluation metric, specifically mAP50-95, considers accuracy across a range of confidence thresholds. The fact that cosine learning rate schedule performed

worse in mAP50-95 suggests that it struggled to maintain high precision at higher confidence thresholds, potentially resulting in more false positives.

Training the model returns two models, one with best parameters observed and the other one with the parameters achieved in last epoch, we focused on the one with best parameters overall, such model was found by default using this Python code:

```python
def fitness(x):
  # Model fitness as a weighted combination of metrics
  w = [0.0, 0.0, 0.1, 0.9] # weights for [P, R, mAP@0.5, mAP@0.5:0.95]
  return (x[:, :4] * w).sum(1)
```

Code Listing 3.1. Finding best.pt

The model fitness calculation places significant emphasis on the mAP50-95 metric, which evaluates the model's performance across a range of intersection over union (IoU) thresholds. This metric provides a comprehensive assessment of the model's ability to accurately detect and localize objects across different IoU thresholds. In contrast, the mAP50 metric has a lesser impact on the overall model fitness. Precision and Recall metrics are not given significant weight, indicating their relatively minor contribution to the evaluation. This approach aligns with the need for a robust ship detection model, as the mAP50-95 metric takes into account a wider range of IoU thresholds, ensuring accurate localization under varying conditions.

3.5 Analysis of The Enhanced YOLOv8s Model

In this section, a detailed analysis of our best trained model will be conducted. We will examine the confusion matrix to gain insights into the model's performance across different classes. Additionally, we will explore various curves that depict the progress of the model during training and evaluation. Furthermore, we will closely examine specific challenges or issues that the model encountered during the process. By delving into these aspects, we aim to gain a comprehensive understanding of the strengths, weaknesses, and limitations of our best trained model. The green color on some of the metrics indicates that the training was resumed, it does not make any difference to the results.

3.5.1 mAP Performance

The optimal performance for a model was achieved at epoch 94, where the mAP50 reached 0.894 and the mAP50-95 reached 0.697. It is worth noting that two other models had higher

mAP50 values, particularly the model trained using cosine learning rate, which had a higher mAP50 by 0.009. However, it is important to consider the mAP50-95 metric, which measures accuracy across a wider range of confidence thresholds. In this regard, the model with the higher mAP50 actually performed worse, with a lower mAP50-95 by about 0.003.

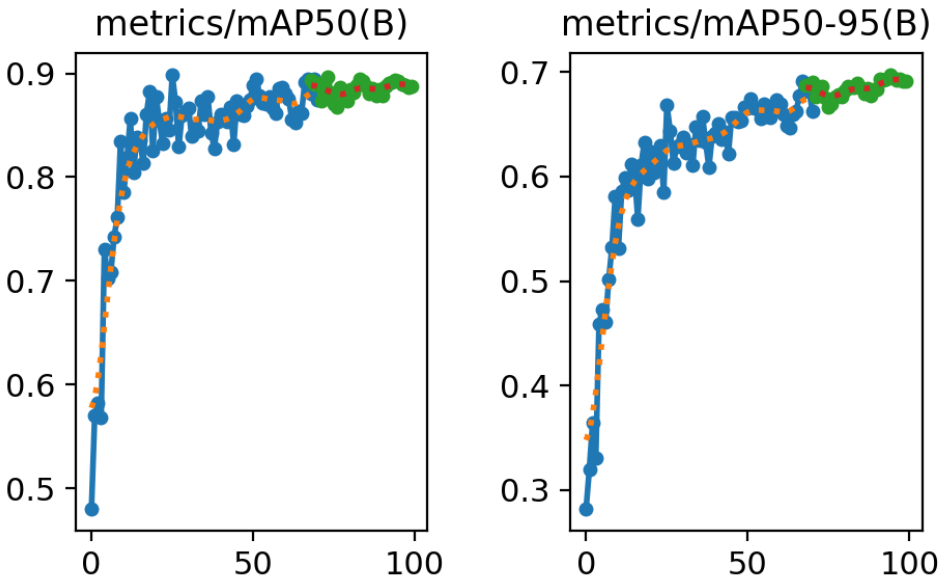

Figure 3.6. mAP50 and mAP50-95 for our best model.

Source: own work

The graph presented in Figure 3.6 illustrates the progression of the two metrics throughout the training of the model. Overall, there was a consistent upward trend, indicating improvement over time, with occasional volatility. However, the most significant increase occurred around the 25th epoch, after which the rate of progress gradually slowed down. The mAP50 and mAP50-95 in the last epoch for our best model turned out to be 0.888 and 0.691, so slightly less than the ones get from our optimum.

3.5.2 Loss for Validation Set

We will analyse 3 types of losses for our model, in all cases, the smaller the loss the better:

- Box Loss: The box loss evaluates the disparity between the predicted bounding box coordinates and the actual bounding box coordinates in the training data.
- Class Loss: The class loss quantifies the dissimilarity between the predicted class probabilities and the true class labels assigned to the objects.

- DFL Loss: DFL stands for Deformable Convolutional Networks (DCN) Feature Loss they allow the network to learn and apply adaptive filters that can be spatially shifted and deformed. The DFL loss measures the discrepancy between the predicted deformable convolutional features and the ground truth features.

The Figure 3.7 provides an overview of how the losses have evolved during the training process of the object detection model. The plot demonstrates that these losses exhibit variations across different epochs, indicating the model's learning progress.

When examining the figure, it becomes evident that there are no significant or abrupt fluctuations in the values of the three losses. Instead, they show minor fluctuations without any discernible pattern. The class loss might exhibit a subtle trend, but for the box loss and DFL loss, the fluctuations are more random and do not follow a clear pattern.

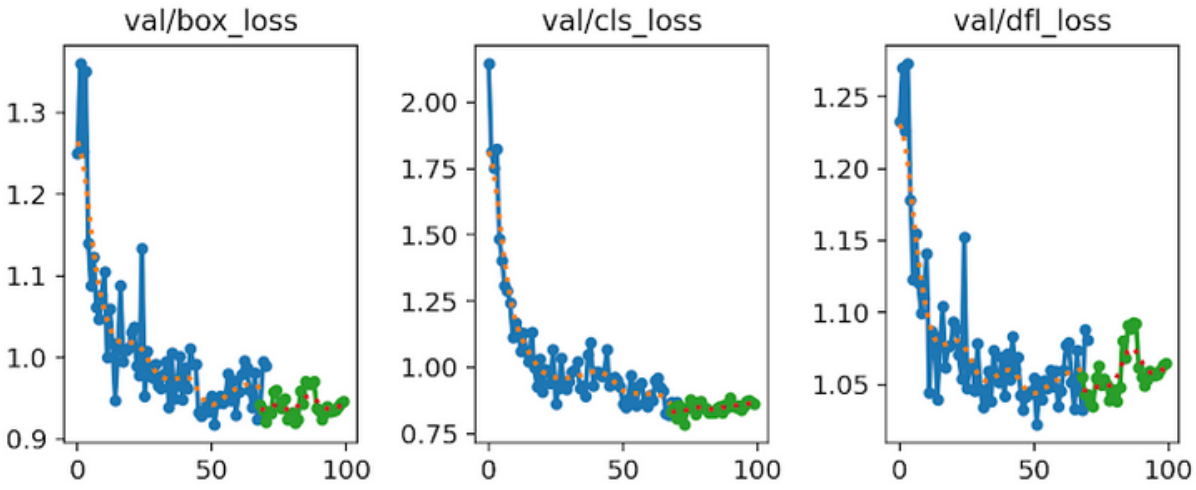

Figure 3.7. Box, cls, dfl loss for our validation set.

Source: own work

One notable finding from these observations is that as the mAP50 and mAP50-95 metrics increased over time, there was no evidence of overfitting in the model. This conclusion is supported by the fact that the losses initially decreased and then reached a plateau. If there had been a continuous increase in the losses at some moment, it would have indicated overfitting. However, as explained earlier, this was not the case during the training process.

Therefore, the absence of a sustained rise in the losses suggests that the model did not overfit the training data. Instead, it demonstrated a positive correlation between the increasing mAP scores and the stable or decreasing losses, indicating successful training without overfitting.

3.5.3 Loss for Training Set

The provided training set losses have the same explanation as the validation set losses. The observations from Figure 3.8 reveal a consistent downward trend in the training losses throughout the training process, indicating an improvement in the model's performance. Although there are occasional fluctuations, the overall trajectory shows a gradual decrease, this trend suggests that the model is learning from the training data and making better predictions over time. It implies an increasing accuracy and a better fit to the desired output.

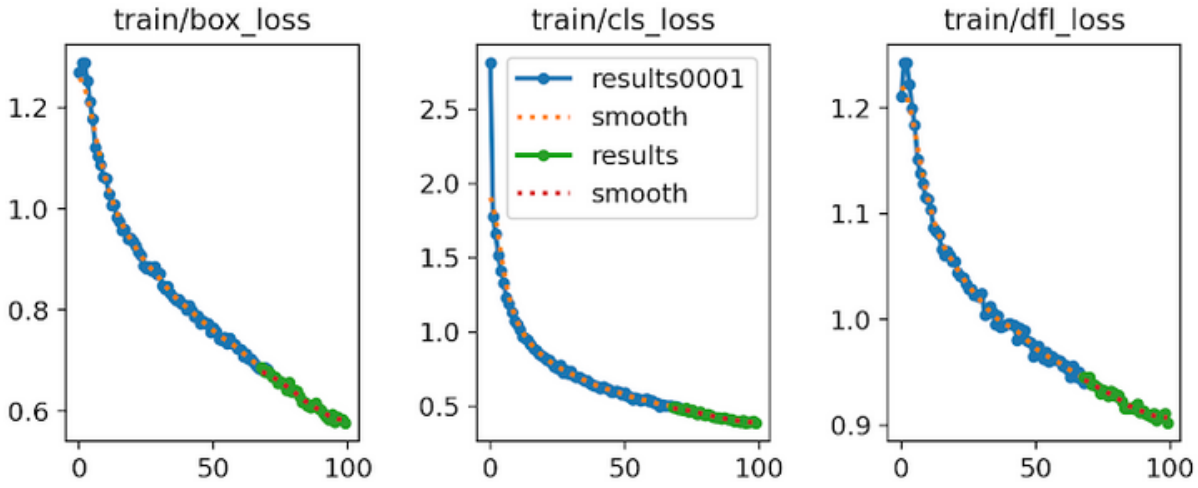

Figure 3.8. Box, cls, dfl loss for our training set.

Source: own work

3.5.4 F1-Confidence Curve

As mentioned earlier, the F1 score is a metric that considers both precision and recall to assess the overall performance of a model in binary classification tasks.

Based on the findings presented in Figure 3.9, we have determined that the optimal confidence threshold for all classes is 0.482. When utilizing this threshold, our model achieves an impressive F1 score of 0.85. The high F1 score achieved by our model indicates its proficiency in accurately identifying positive instances (precision) while also capturing a substantial portion of all positive instances (recall). By achieving a high F1 score, we can have confidence in our model's ability to effectively perform its designated tasks, such as ship recognition, with a favourable trade-off between precision and recall.

By examining Figure 3.9, we can also gain insights into the relative ease with which our enhanced YOLOv8s model was able to accurately classify different types of ships. It is evident

that certain categories were more easily recognizable than others. For instance, ore carriers and container ships consistently achieved higher F1 scores, indicating that our model had a relatively easier time correctly identifying them. On the other hand, passenger ships posed a greater challenge, as their F1 scores were generally lower compared to other ship types.

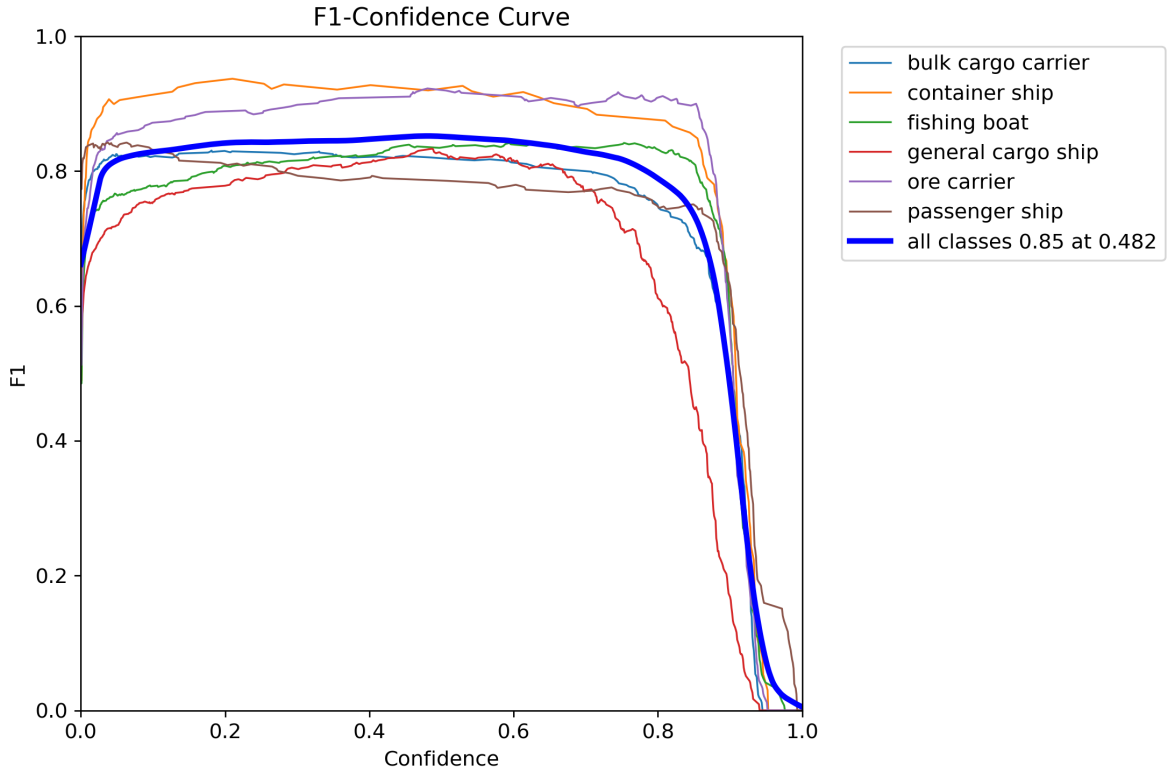

Figure 3.9. F1 Confidence Curve for for our best model.

Source: own work

3.5.5 Confusion Matrix

Already explained confusion matrix provides a visual representation of the model's performance by showing which classes have been predicted correctly and where misclassifications have occurred. In Figure 3.10 we can see the confusion matrix for our specific model validation set.

Even more clearly we can see with what ships the model had problems with on normalized confusion matrix, where the values are normalized to show the percentage of predictions for each class relative to the actual instances. For our model we can see such matrix in Figure 3.11.

Let's analyse in sequence all variables on the "True" label and the reasons why this specific class might have such results:

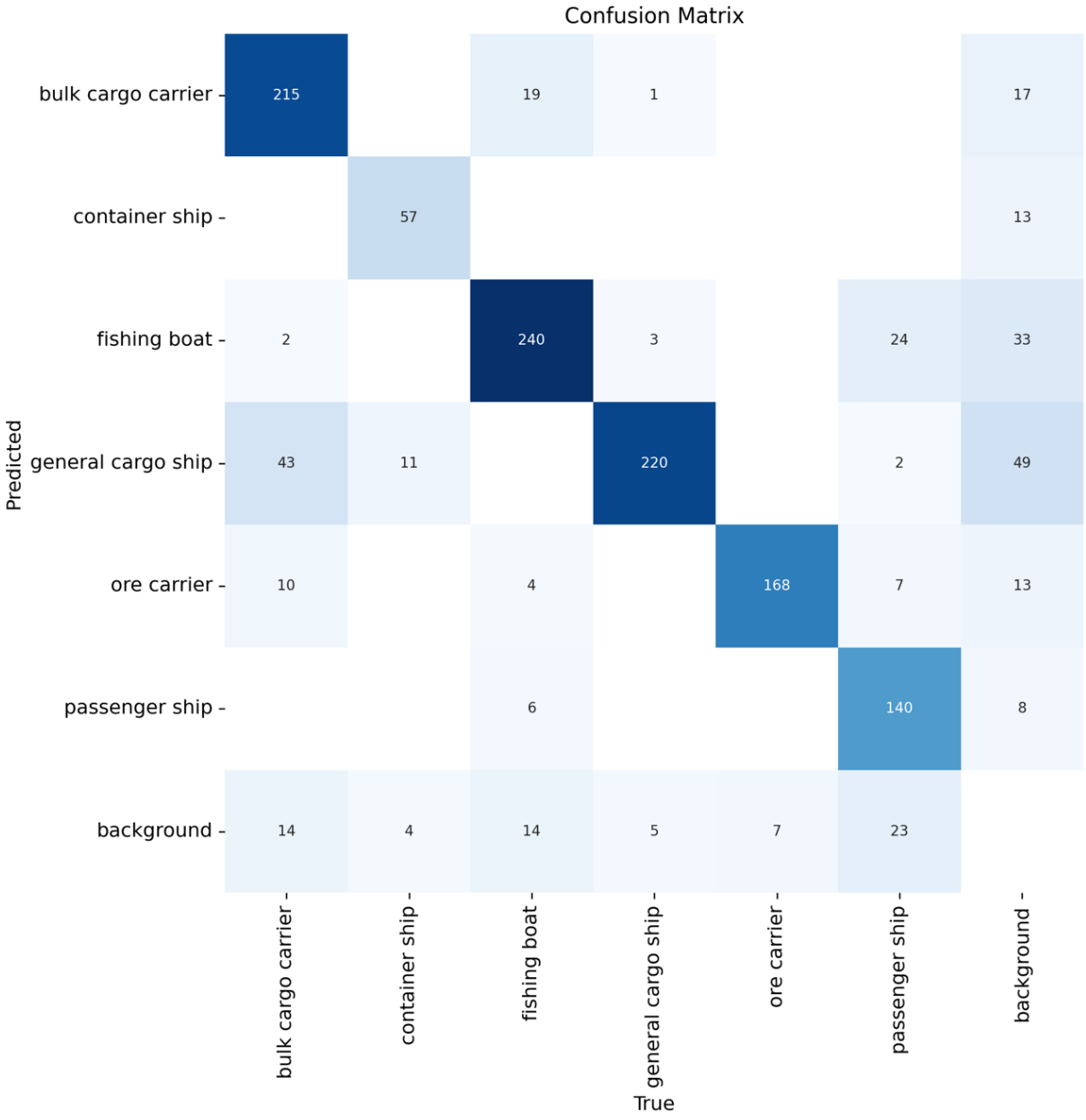

Figure 3.10. Confusion Matrix for our best model.

Source: own work

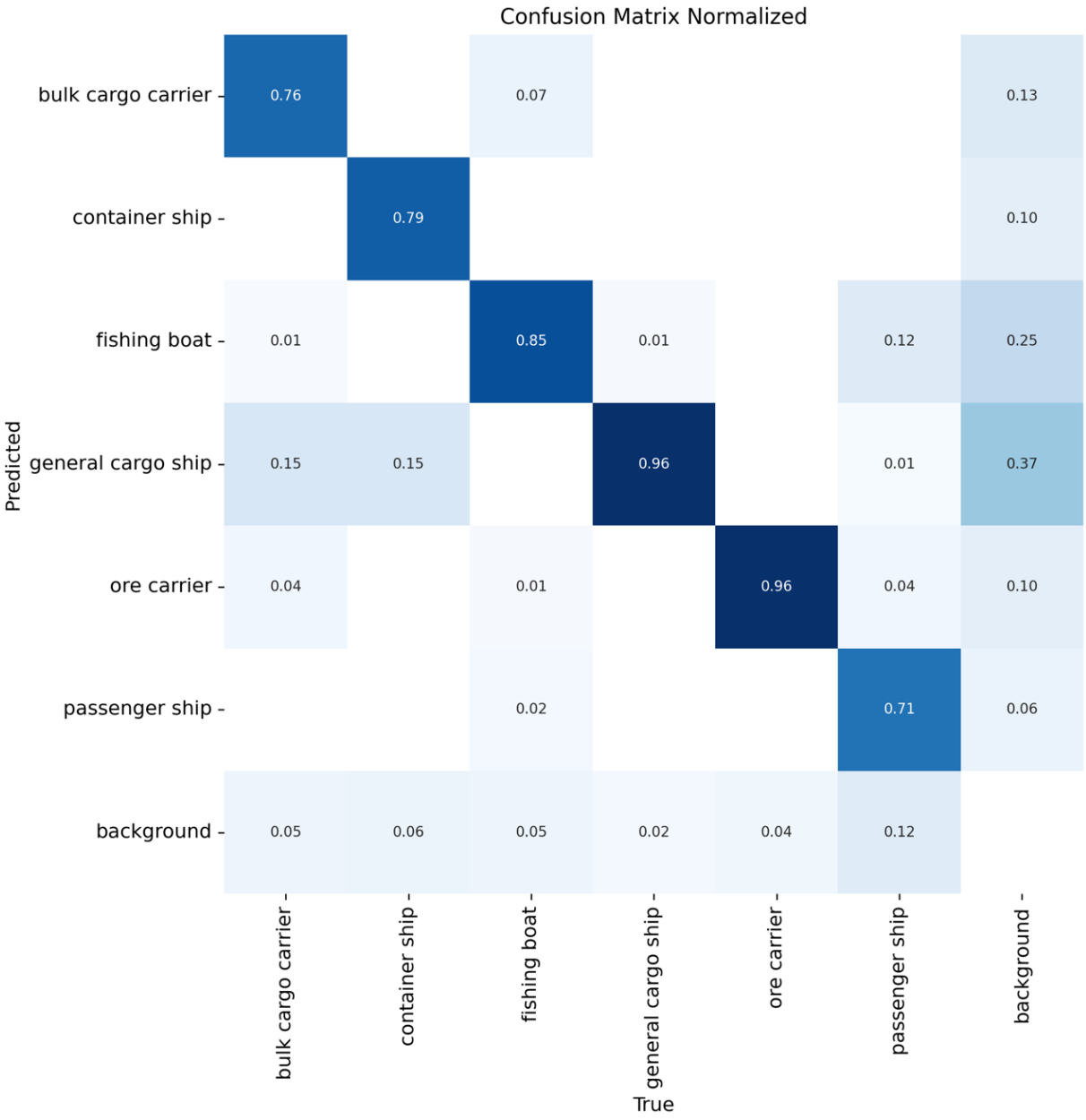

Figure 3.11. Normalized Confusion Matrix for our best model.

Source: own work

- Bulk cargo carrier: The model correctly predicted 76% of instances of the ship type under consideration. This percentage is slightly lower compared to the accuracy achieved for general cargo ships. The lower accuracy for this ship type could be attributed to the similarities between the two types, which exhibit slight differences in shape, as illustrated in Figure 3.12.

(a) bulk cargo ship **(b) general cargo ship**

Figure 3.12. Example of correctly detected bulk cargo and general cargo ship.

Source: own work

- Container ship: A similar situation was observed when distinguishing between container ships and general cargo ships. Our model correctly predicted 79% of instances as container ships, but 15% of container ships were misclassified as general cargo ships. This pattern can be observed in Figure 3.13. It is possible that the model confused the containers carried by container ships either as part of the background or as integral components of general cargo ships.
- Fishing boat: The model achieved a significant 85% accuracy in predicting instances of fishing boats. This high accuracy could be attributed to the fact that fishing boats constituted a large portion of the SeaShips dataset, providing ample opportunities for the model to learn their distinguishing features. However, despite the high accuracy, the model still made some mistakes in classification. 5% of the detections for fishing boats were falsely classified as background even though there was actual boat on the image. Another common confusion was observed with bulk cargo ships, which could be caused by challenging perspectives, as exemplified in Figure 3.14.

(a) incorrectly detected container **(b) correctly detected container**

Figure 3.13. Example of both wrongly and correctly detected container ship.

Source: own work

- General cargo ship: This particular type of ship achieved an impressive 96% accuracy in predictions and was rarely confused with other ship types. It is worth noting that this high accuracy was achieved despite the relatively smaller number of instances in the dataset, with only around 700 instances compared to over 1000 instances for bulk cargo carriers. The model demonstrated a strong ability to accurately identify and classify this ship type, showing its effectiveness in distinguishing it from other types in the dataset.
- Ore carrier: The ore carrier class achieved an equally impressive 96% accuracy in predictions, which is the same as the general cargo ship class. The model rarely confused ore carriers with other types of ships, and when it did, it was typically with the background class. This exceptional performance can be attributed to the relatively large number of instances in the training set, with around 1200 instances for the ore carrier class. The abundance of training data for this class likely contributed to the model's ability to accurately distinguish and classify ore carriers with high precision.
- Passenger ship: The class of passenger ships had the lowest percentage of correctly predicted detections, with an accuracy rate of approximately 71%. These ships were frequently confused with the background class and fishing boats. Several factors may have contributed to this phenomenon. One of the most significant factors was the relatively small number of instances of passenger ships in our dataset, with only 200 instances available for training.

(a) incorrectly detected fishing boat **(b) correctly detected fishing boat**

Figure 3.14. Example of confusing fishing boat with general cargo ship.

Source: own work

Figure 3.15 illustrates an example where a passenger ship was falsely classified as a fishing boat. This misclassification may be attributed to the frontal view of the ship, its similarity in size and shape to a fishing boat, and the limited representation of passenger ships in the training data. Additionally, Figure 3.16 demonstrates how, in some cases, the model fails to detect passenger ships correctly, even when they are present in the image.

- Background: Upon examining Figure 3.10, it is evident that the model frequently misclassified the background as certain classes. This is particularly noticeable in the case of fishing boats, with 33 instances of misclassification, and general cargo ships, with 49 instances. The confusion between fishing boats and the background can be attributed to their relatively small size, leading the model to occasionally misidentify other objects as fishing boats, as depicted in Figure 3.17.

  An interesting case arises with general cargo ships, as shown in Figure 3.18. It appears that the model assigned two bounding boxes to a single ship, resulting in a false prediction where the background class is incorrectly assigned. This anomaly may be due to the model perceiving certain elements of the shore as ship components, leading to the prediction of another ship ”on top” of the existing one.

(a) incorrectly detected passenger ship

(b) correctly detected passenger ship

Figure 3.15. Example of confusing passenger ship with fishing boat.

Source: own work

In order to assess the practical application of our model on real-world videos, we obtained a selection of videos from the publicly accessible collection on the website Pexels[3]. We applied our model to these videos and the results can be viewed in the following YouTube playlist. For detecting fishing boats, we used the confidence threshold of 0.482, determined as the best value based on the F1 score. https://www.youtube.com/playlist?list=PL4ZepVAM6v08cO-c8yeMF5ko3lyDpr4yo.

[3] https://www.pexels.com/

(a) incorrectly detected passenger ship

(b) correctly detected passenger ship

Figure 3.16. Example of confusing passenger ship with background.

Source: own work

(a) incorrectly detected fishing boat

(b) incorrectly detected fishing boat

Figure 3.17. Example of confusing passenger ship with background.

Source: own work

(a) incorrectly detected 2 general cargo ships

(b) incorrectly detected 2 general cargo ships

Figure 3.18. Example of confusing background with general cargo ship.

Source: own work

Chapter 4

Conclusion and Future Work

In conclusion, our research paper aims to address the problems outlined in the introduction. By doing so, we provide a comprehensive review of our findings throughout the course of this study.

4.1 Overview of Enhancements in YOLO Series

During our exploration of the YOLO (You Only Look Once) models, which included the most recent version known as YOLOv8, we identified several trends within the YOLO series. One significant trend revolves around the evolution of the anchor mechanism. In the initial YOLO version, anchors were not utilized. However, in subsequent iterations leading up to YOLOv5, anchor boxes were introduced to enhance the performance of object detection. Interestingly, in more recent versions like YOLOX and subsequent iterations, including YOLOv8, the use of anchors was once again eliminated, while still achieving state-of-the-art accuracy levels.

The utilization of a different framework marked a significant advancement in the development of these models. At the beginning the YOLO models, mainly developed by Redmond, were constructed using the Darknet framework. However, Ultralytics, the creators of YOLOv8, introduced a shift to PyTorch as the framework with the release of YOLOv3. This transition to PyTorch greatly facilitated the progress of the object detection community in advancing these models, making the development process considerably easier.

The backbone architecture of YOLO models underwent significant changes. YOLOv8 reduced the reliance on maxpool and basic convolutional layers, introducing advanced techniques like the C2f module and complex convolutional layers. These layers involve for example

2D convolution, 2D batch normalization, and SiLU activation, enhancing the model's capabilities for object detection (Figure 2.15).

In summary, significant advancements have been made in just a few years, with more developments on the horizon as new models are rapidly introduced. It is important to note that the objective of the YOLO series extends beyond increasing detection accuracy; it also aims to strike a trade-off between accuracy and speed.

4.2 Performance of YOLOv8 on Detecting Ships

Our training results yielded impressive achievements, with 89.4% mAP50 and 69.7% mAP50-95, indicating the exceptional performance of YOLOv8 in detecting subtle differences among various ships. Remarkably, we did not observe any signs of overfitting, demonstrating the suitability of the original YOLOv8 architecture employed in our research paper for effectively handling diverse environmental conditions in the images.

One notable challenge encountered by our model was the limited number of instances available for certain classes, such as passenger ships, as discussed in Subchapter 3.5.5. However, despite this limitation, the YOLOv8 model was still able to correctly detect these types of ships, highlighting the overall capabilities and potential of the YOLOv8 model.

4.3 Feasibility of Fine-Tuning YOLOv8

During our experiment, we discovered that we could achieve a significant improvement in the overall mAP50-95, increasing it from 65.8% to 69.7%, by making careful adjustments to a few training parameters, specifically the batch size and the final learning rate (Figure3.4). Interestingly, we found that using the cosine learning rate yielded nearly optimal results, indicating its great potential in enhancing model performance.

By conducting multiple experiments and selectively modifying training parameters while keeping the hyperparameters and architecture unchanged, we were able to achieve substantial improvements in model accuracy. This highlights the importance of fine-tuning training parameters to maximize performance.

4.4 Future Work

There are several potential avenues for advancing our work. Firstly, we could enhance the ship database by balancing the number of instances for each class, aiming for at least 10,000 instances per class. This would improve the training data and potentially enhance the model's performance across all classes.

Another possibility is to consider modifying the YOLOv8 architecture specifically for the detection of ships from shore camera perspectives. However, this is a complex change that requires careful consideration and expertise, as it could have a significant impact on the accuracy and effectiveness of our model.

Additionally, further fine-tuning of training parameters and hyperparameters could also lead to improvements in our model. Leveraging automated tools such as Amazon Sagemaker[1] or Ray Tune[2] may facilitate this process, but it is important to note that these tools can be computationally demanding.

Overall, there are opportunities to enhance our work through database enrichment, architectural modifications, and parameter fine-tuning, but careful consideration and appropriate computational resources are required to effectively explore these possibilities.

[1] https://aws.amazon.com/sagemaker/
[2] https://docs.ray.io/en/latest/tune/index.html

Bibliography

Ablan, M. C. A., Garces, L. R. (2005). Exclusive Economic Zones and the Management of Fisheries in the South China Sea. Springer Netherlands. https://doi.org/10.1007/1-4020-3133-5_9

Aboah, A., Wang, B., Bagci, U., Adu-Gyamfi, Y. (2023). Real-time Multi-Class Helmet Violation Detection Using Few-Shot Data Sampling Technique and YOLOv8.

Ahmed, D., Sapkota, R., Churuvija, M., Karkee, M. (2023). Machine Vision-Based Crop-Load Estimation Using YOLOv8.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*, 53. https://doi.org/10.1186/s40537-021-00444-8

Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., Modi, K., Ghayvat, H. (2021). CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope. *Electronics*, *10*, 2470. https://doi.org/10.3390/electronics10202470

Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., Dahl, G. E. (2019). On Empirical Comparisons of Optimizers for Deep Learning.

Diwan, T., Anirudh, G., Tembhurne, J. V. (2023). Object detection using YOLO: challenges, architectural successors, datasets and applications. *Multimedia Tools and Applications*, *82*, 9243–9275. https://doi.org/10.1007/s11042-022-13644-y

Dubey, S. R., Singh, S. K., Chaudhuri, B. B. (2021). Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark.

Ge, Z., Liu, S., Wang, F., Li, Z., Sun, J. (2021). YOLOX: Exceeding YOLO Series in 2021.

Gholamalinezhad, H., Khosravi, H. (2020). Pooling Methods in Deep Neural Networks, a Review.

Girshick, R. (2015). Fast R-CNN.

Han, X., Chang, J., Wang, K. (2021). Real-time object detection based on YOLO-v2 for tiny vehicle object. *Procedia Computer Science*, *183*, 61–72. https://doi.org/10.1016/j.procs.2021.02.031

Imran, I. H., Das, T., Galib, A. H., Roy, T. (2023). Unleashing the Power of YOLOv8 for Accurate Bengali Mathematical Expression Detection. *Research Square*.

Jain, A., Mao, J., Mohiuddin, K. (1996). Artificial neural networks: a tutorial. *Computer*, *29*, 31–44. https://doi.org/10.1109/2.485891

Jocher, G., Chaurasia, A., Qiu, J. (2023). YOLO by Ultralytics. https://github.com/ultralytics/ultralytics

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., Kwon, Y., Michael, K., Fang, J., NanoCode012, TaoXie, imyhxy, Lorna, Yifu, Z., Wong, C., V, A., Montes, D., Wang, Z., Fati, C., Nadar, J., Laughing, UnglvKitDe, Sonck, V., tkianai, yxNONG, Skalski, P., Hogan, A., Nair, D., Strobel, M., Jain, M. (2022). ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation. *Zenodo*. https://doi.org/10.5281/zenodo.7347926

Kim, J.-H., Kim, N., Won, C. S. (2023). High-Speed Drone Detection Based On Yolo-V8. *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1–2. https://doi.org/10.1109/ICASSP49357.2023.10095516

Krenker, A., Bester, J., Kos, A. (2011). Introduction to the Artificial Neural Networks. InTech. https://doi.org/10.5772/15751

Krogh, A. (2008). What are artificial neural networks? *Nature Biotechnology*, *26*, 195–197. https://doi.org/10.1038/nbt1386

Laroca, R., Severo, E., Zanlorensi, L. A., Oliveira, L. S., Gonçalves, G. R., Schwartz, W. R., Menotti, D. (2018). A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector. https://doi.org/10.1109/IJCNN.2018.8489629

Li, H., Deng, L., Yang, C., Liu, J., Gu, Z. (2021). Enhanced YOLO v3 Tiny Network for Real-Time Ship Detection From Visual Image. *IEEE Access*, *9*, 16692–16706. https://doi.org/10.1109/ACCESS.2021.3053956

Li, Y., Fan, Q., Huang, H., Han, Z., Gu, Q. (2023). A Modified YOLOv8 Detection Network for UAV Aerial Image Recognition. *Drones*, *7*, 304. https://doi.org/10.3390/drones7050304

Liang, D., Geng, Q., Wei, Z., Vorontsov, D. A., Kim, E. L., Wei, M., Zhou, H. (2022). Anchor Retouching via Model Interaction for Robust Object Detection in Aerial Images. *IEEE Transactions on Geoscience and Remote Sensing*, *60*, 1–13. https://doi.org/10.1109/TGRS.2021.3136350

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S. (2016). Feature Pyramid Networks for Object Detection.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., Pietikäinen, M. (2018). Deep Learning for Generic Object Detection: A Survey.

lyx.limyixian. (2022). wsodd-usv dataset Dataset. *Roboflow Universe*. https://universe.roboflow.com/lyx-limyixian-gmail-com/wsodd-usv-dataset

Munin, A., Folarin, A., Munin-Doce, A., Alonso-Garcia, L., Diaz-Casas, V., Ferreno-Gonzalez, S., Ciriano-Palacios, J. M. (2023). Real Time Vessel Detection Model Using Deep Learning Algorithms for Controlling a Barrier System. *SSRN*.

Pahde, F., Ostapenko, O., Jähnichen, P., Klein, T., Nabi, M. (2018). Self Paced Adversarial Training for Multimodal Few-shot Learning.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection.

Redmon, J., Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger.

Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement.

Ren, S., He, K., Girshick, R., Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.

Shao, Z., Wu, W., Wang, Z., Du, W., Li, C. (2018). SeaShips: A Large-Scale Precisely Annotated Dataset for Ship Detection. *IEEE Transactions on Multimedia*, *20*, 2593–2604. https://doi.org/10.1109/TMM.2018.2865686

Terven, J., Cordova-Esparza, D. (2023). A Comprehensive Review of YOLO: From YOLOv1 and Beyond.

Tian, Y., Yang, G., Wang, Z., Wang, H., Li, E., Liang, Z. (2019). Apple detection during different growth stages in orchards using the improved YOLO-V3 model. *Computers and Electronics in Agriculture*, *157*, 417–426. https://doi.org/10.1016/j.compag.2019.01.012

Wang, C.-Y., Bochkovskiy, A., Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.

Wu, J. (2017). Introduction to Convolutional Neural Networks.

Yao, J., Qi, J., Zhang, J., Shao, H., Yang, J., Li, X. (2021). A Real-Time Detection Algorithm for Kiwifruit Defects Based on YOLOv5. *Electronics*, *10*, 1711. https://doi.org/10.3390/electronics10141711

YOLOv5Seaships. (2022). SeaShips Dataset. *Roboflow Universe*. https://universe.roboflow.com/yolov5seaships/seaships-mcvwt

Yoon, S. (2021). South Korea and the South China Sea: A Middle-Power Model for Practical Policies? Bristol University Press. https://doi.org/10.51952/9781529213478.ch018

Zhang, H., Cisse, M., Dauphin, Y. N., Lopez-Paz, D. (2017). mixup: Beyond Empirical Risk Minimization.

Zhang, S., Yang, H., Yang, C., Yuan, W., Li, X., Wang, X., Zhang, Y., Cai, X., Sheng, Y., Deng, X., Huang, W., Li, L., He, J., Wang, B. (2023). Edge Device Detection of Tea Leaves with One Bud and Two Leaves Based on ShuffleNetv2-YOLOv5-Lite-E. *Agronomy*, *13*, 577. https://doi.org/10.3390/agronomy13020577

Zhang, Y., Guo, Z., Wu, J., Tian, Y., Tang, H., Guo, X. (2022). Real-Time Vehicle Detection Based on Improved YOLO v5. *Sustainability*, *14*, 12274. https://doi.org/10.3390/su141912274

Zhao, Z.-Q., Zheng, P., Xu, S.-T., Wu, X. (2019). Object Detection With Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, *30*, 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865

Zou, J., Han, Y., So, S.-S. (2008). Overview of Artificial Neural Networks. https://doi.org/10.1007/978-1-60327-101-1_2

Zou, Z., Chen, K., Shi, Z., Guo, Y., Ye, J. (2019). Object Detection in 20 Years: A Survey.

List of Tables

List of Figures

Code Listings