Technische Universität Braunschweig

15-Sept-23

Object Detection for DIOR Dataset

Group E

Raissa Magana Ugarte 5262878
Karam Mawas

Outline

1. Introduction
2. DIOR Dataset
3. Models
4. Experimental Setup
5. Results
6. Summary

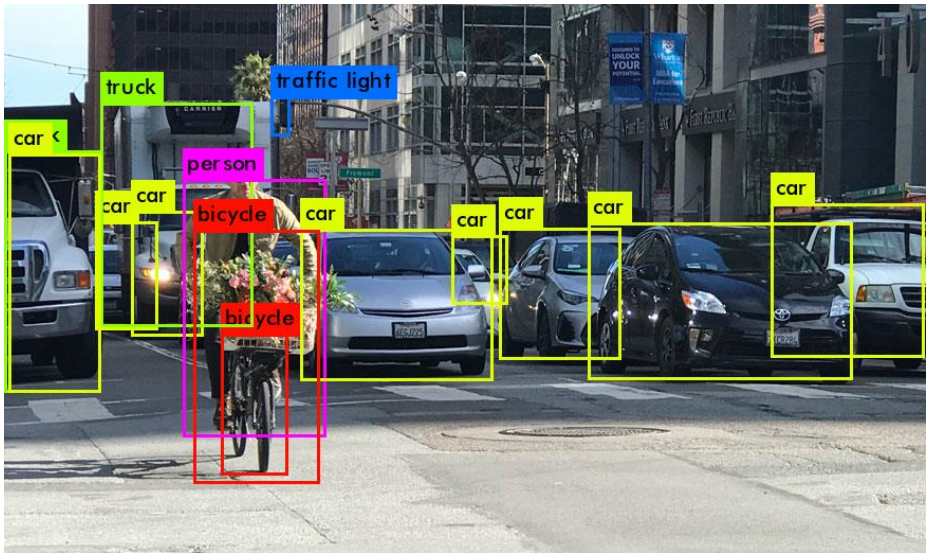
Outline

Introduction

Object Detection is a computer vision task in which the goal is to detect and locate objects of interest in an image or video. The task involves identifying the position and boundaries of objects in an image and classifying the objects into different categories.

The state-of-the-art methods can be categorized into two main types:

- One-stage methods prioritize inference speed, models like YOLO, SSD and RetinaNet.
- Two-stage methods prioritize detection accuracy, including Faster R-CNN, Mask R-CNN.



Object Detection (Source: Medium)

Outline

DIOR Dataset

The DIOR Dataset is a large-scale, publicly available benchmark for object Detection in Optical Remote sensing images, which named as DIOR (Li et al., 2019)

Its main objective is to be used to develop and validate data-driven methods to classify and detect multiple different objects from a Satellite Imagery/Optical Remote Sensing image.

The dataset contains 23463 images and 192472 instances, covering 20 object classes.

800×800 pixels with 3 channels each and the spatial resolutions range from 0.5m to 30m.

Data Classes in DIOR

20 Objects:

- Golf field
- Expressway-toll-station
- Vehicle
- Train station
- Chimney
- Storage tank
- Ship
- Harbor
- Airplane
- Ground track field
- Tennis court
- Dam
- Basketball court
- Expressway-Service-area
- Stadium
- Airport
- Baseball field
- Bridge
- Windmill
- Overpass

Fig. 2. Example images taken from the proposed DIOR dataset, which were obtained with different imaging conditions, weathers, seasons, and image quality.

Credit: Li et al., 2019

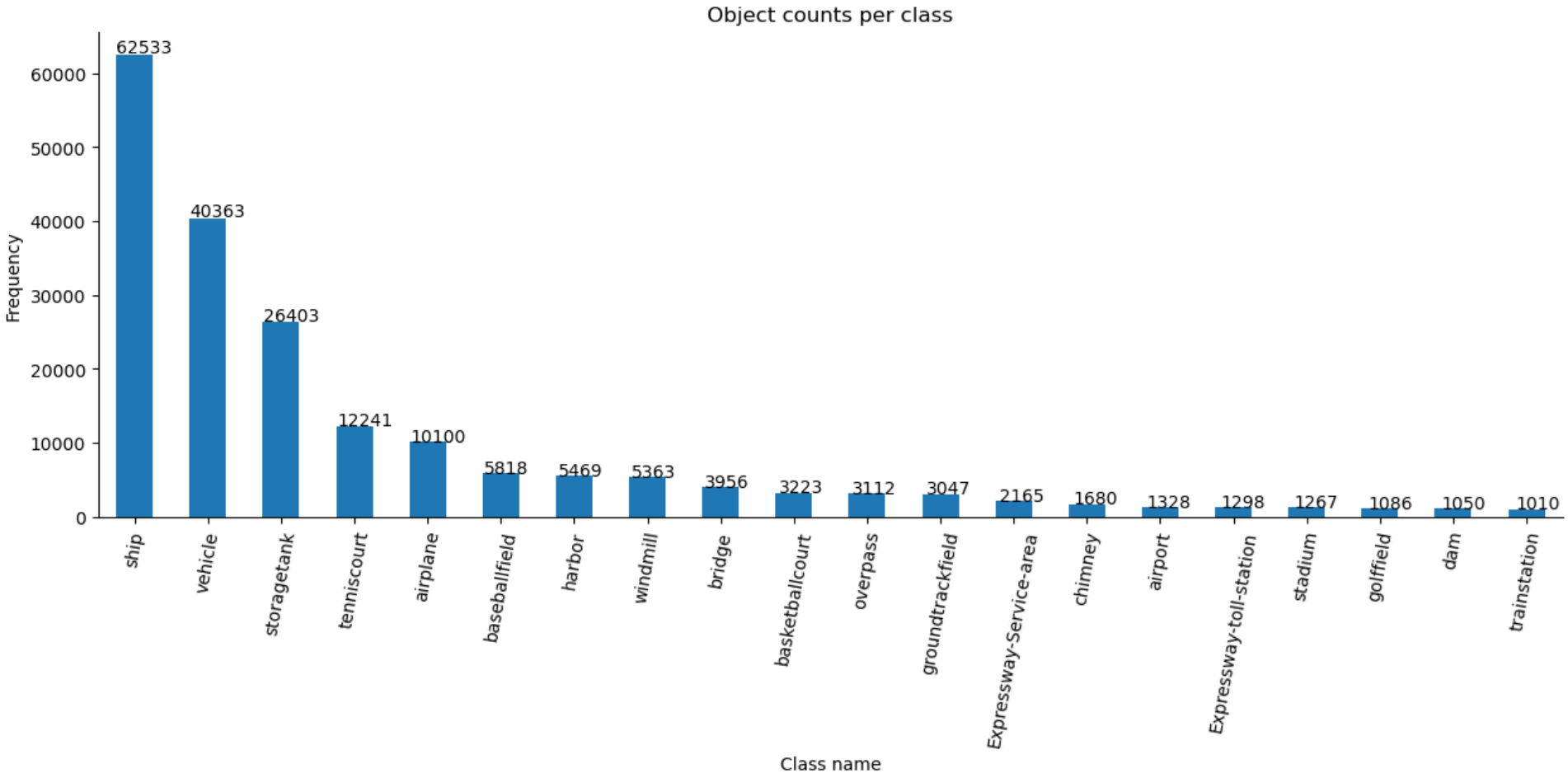
EDA
Different Annotations Results

The objects can be found by two means within the DIOR dataset:

Horizontal Bounding Boxes

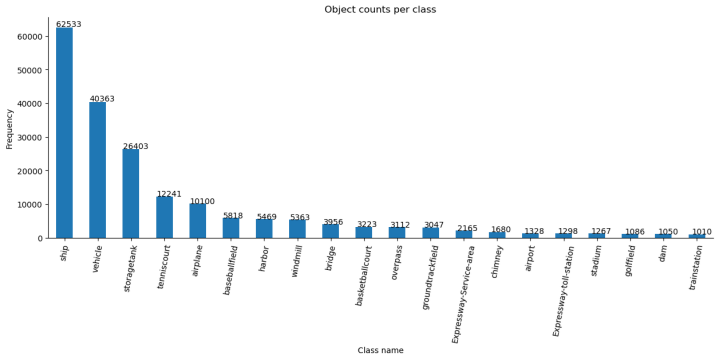
Oriented Bounding Boxes

EDA
Number of Classes in the Dataset



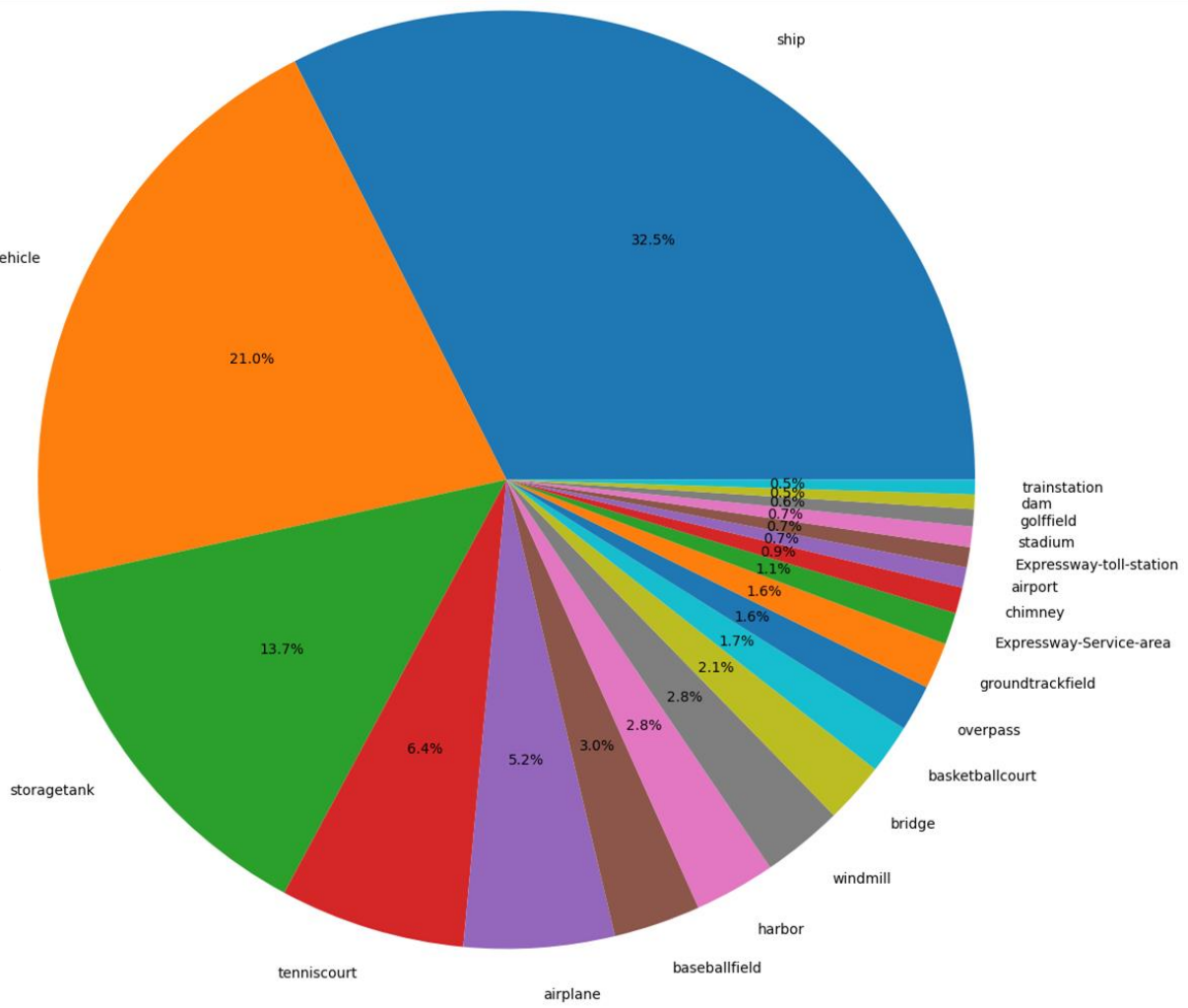
Frequency distribution of objects per class

EDA
Number of Classes in the Dataset



Frequency distribution of objects per class

Imbalanced Classes

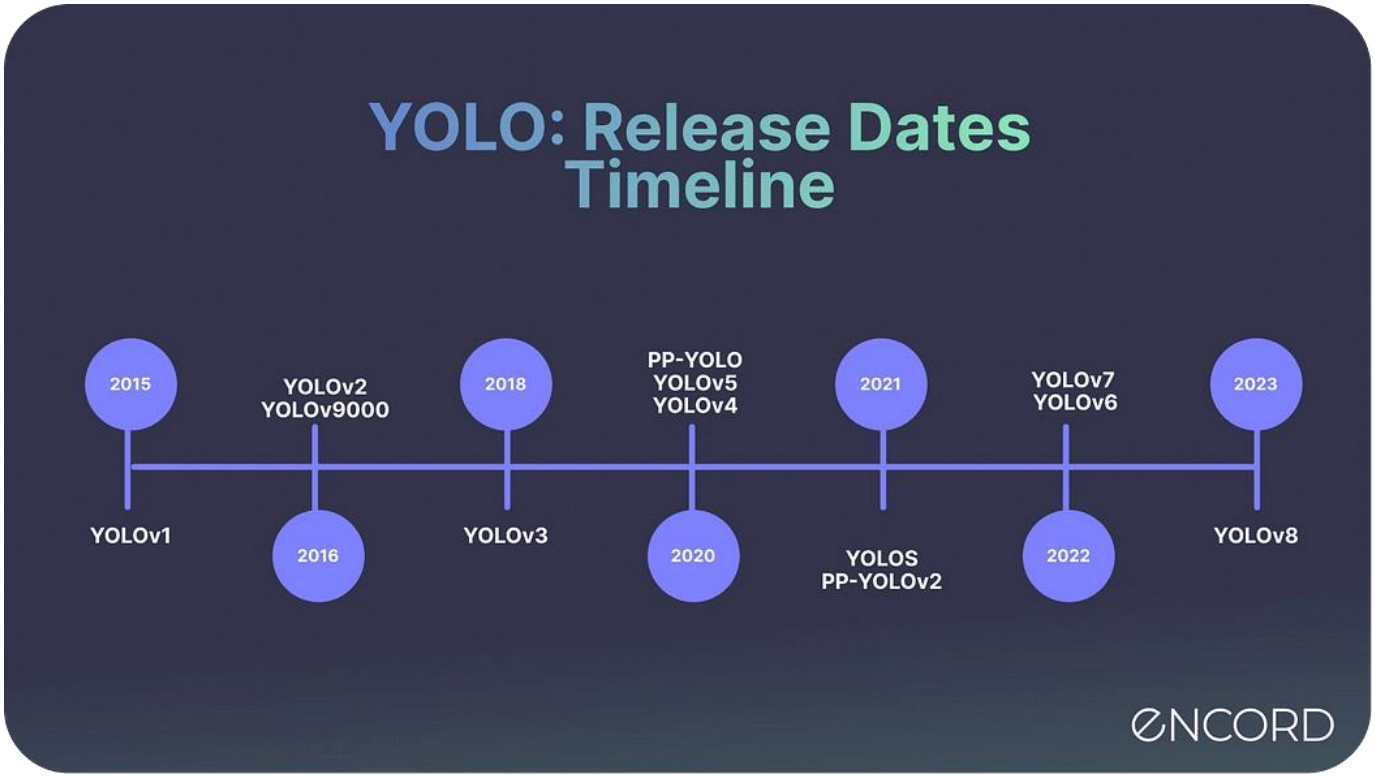


Frequency distribution of objects per class in %

Outline

YOLO: Introduction

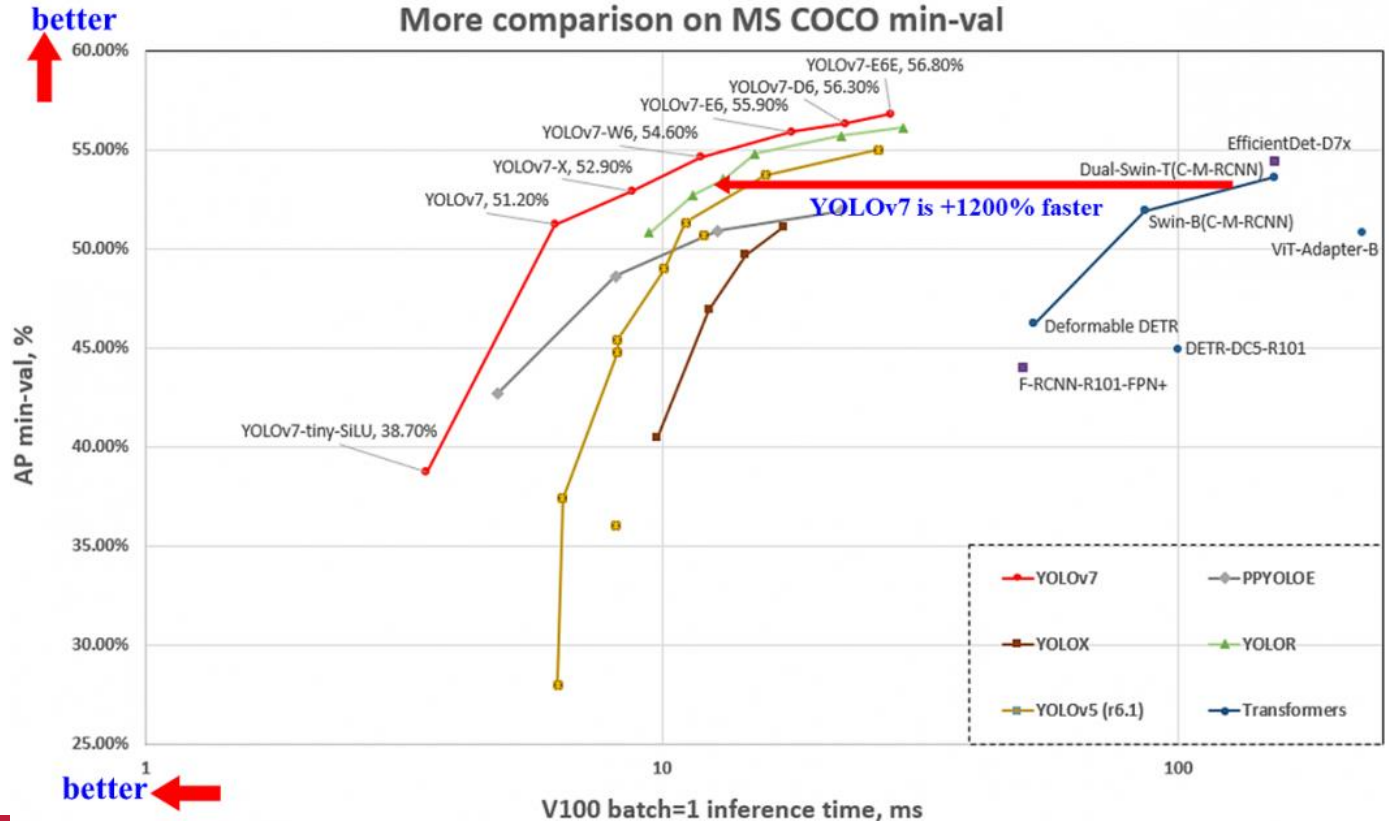
You Only Look Once (YOLO) is a state-of-the-art, real-time object detection algorithm introduced in 2015 by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in their famous research paper “You Only Look Once: Unified, Real-Time Object Detection”.



YOLO Timeline from 2015 to 2022 (Source: Medium)

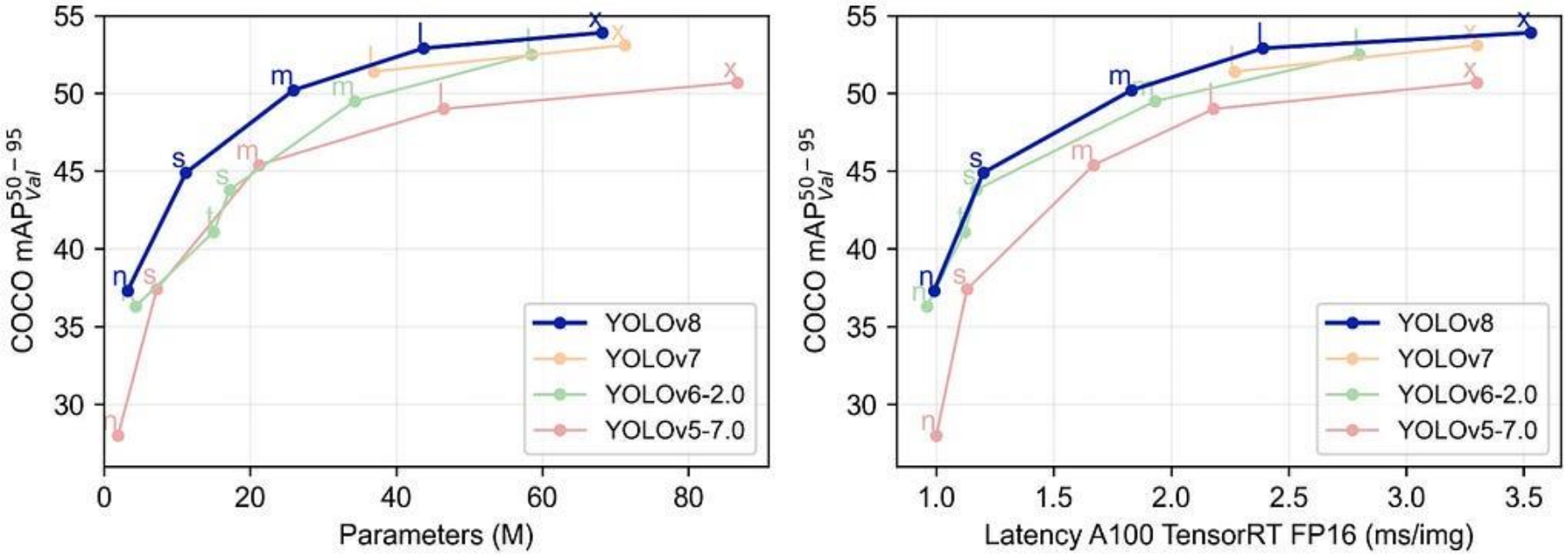
YOLO: Performance

The authors frame the object detection problem as a regression problem instead of a classification task. The network does not look at the complete image. Instead, parts of the image which have high probabilities of containing the object. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes.



Comparison between YOLO and other Object Detection models (Source: Medium)

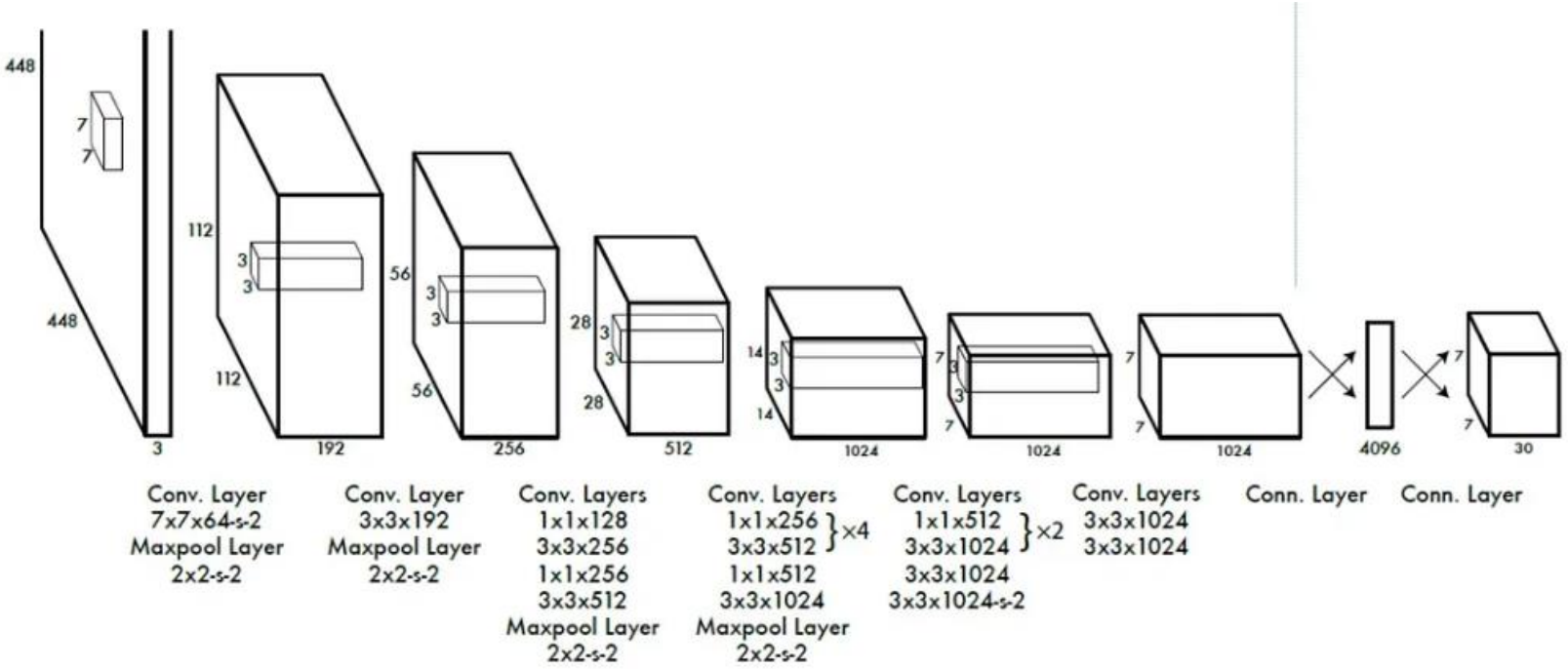
YOLO: Performance



Comparison between YOLO models (Source: Medium)

YOLO: Architecture

YOLOv1 has overall 24 convolutional layers, four max-pooling layers, and two fully connected layers.



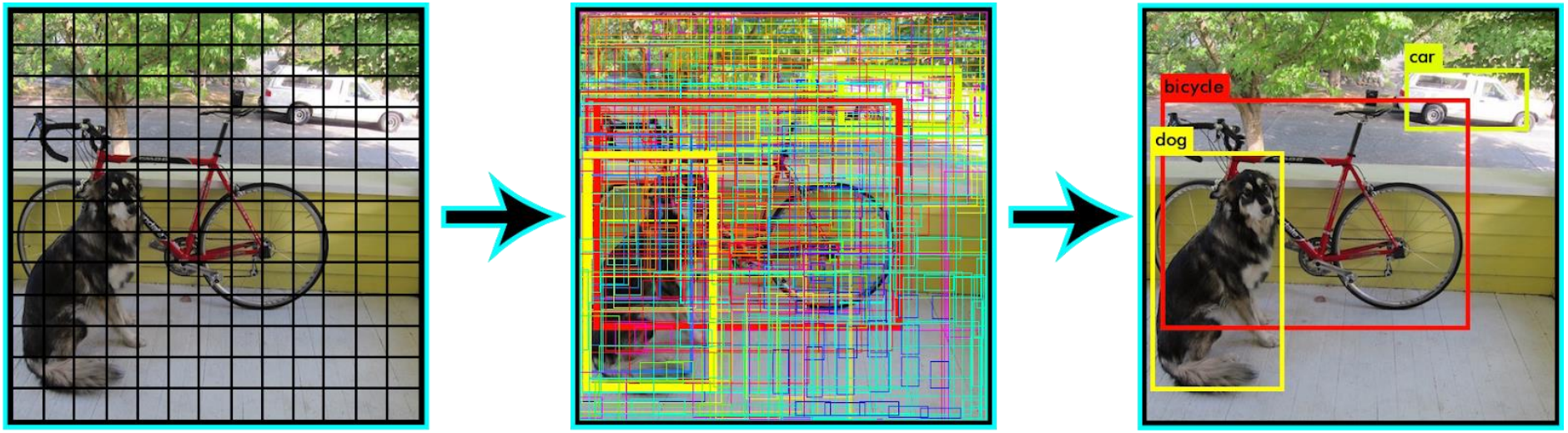
YOLO original architecture (Source: YOLO paper)

The first 20 convolutional layers from the backbone of the network and, with the addition of an average pooling layer and a single fully connected layer, it was pre-trained and validated on the ImageNet 2012 dataset. During inference, the final four layers and 2 FC layers are added to the network; all initialized randomly.

YOLO: How it works

The algorithm works based on the following four approaches:

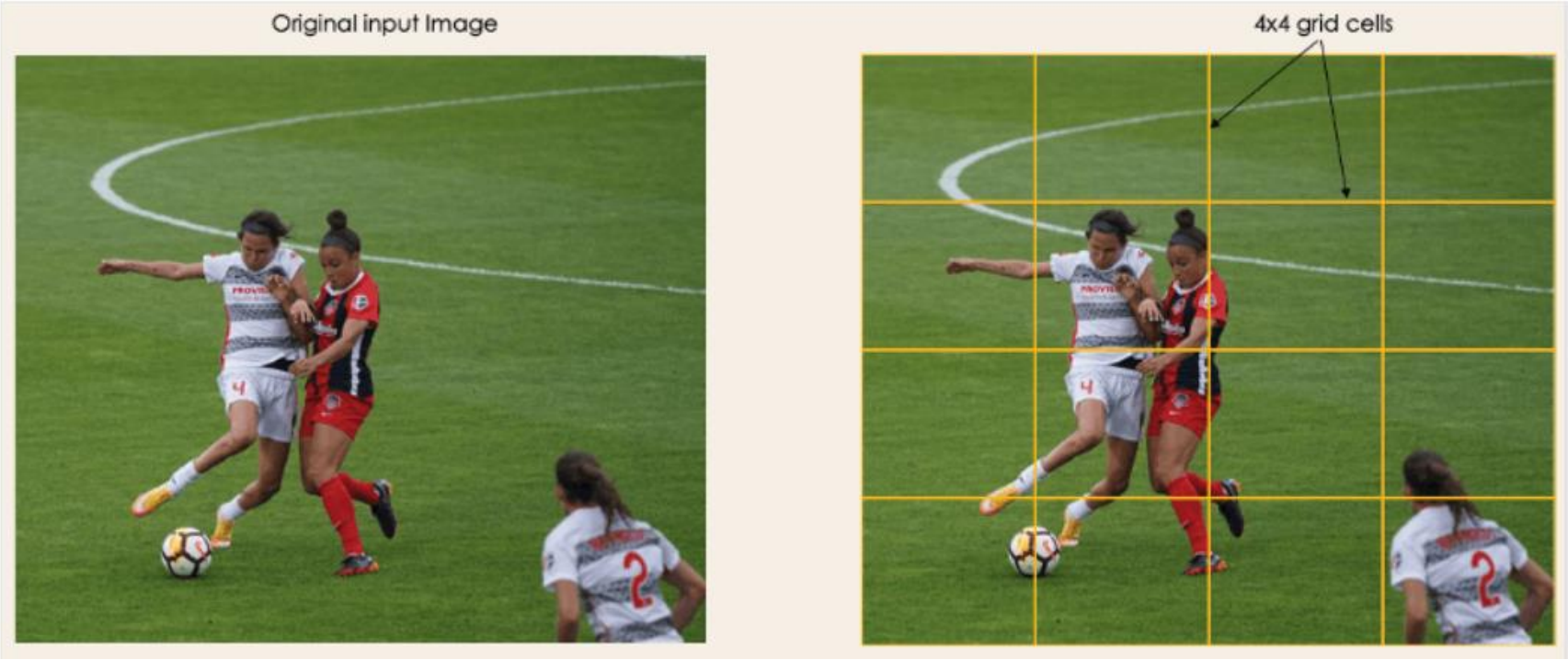
- Residual blocks
- Bounding box regression
- Intersection Over Unions or IOU for short
- Non-Maximum Suppression.



YOLO Timeline from 2015 to 2022 (Source: Datacamp)

YOLO: How it works – Residual Blocks

- This first step starts by dividing the original image (A) into NxN grid cells of equal shape
- Each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value.



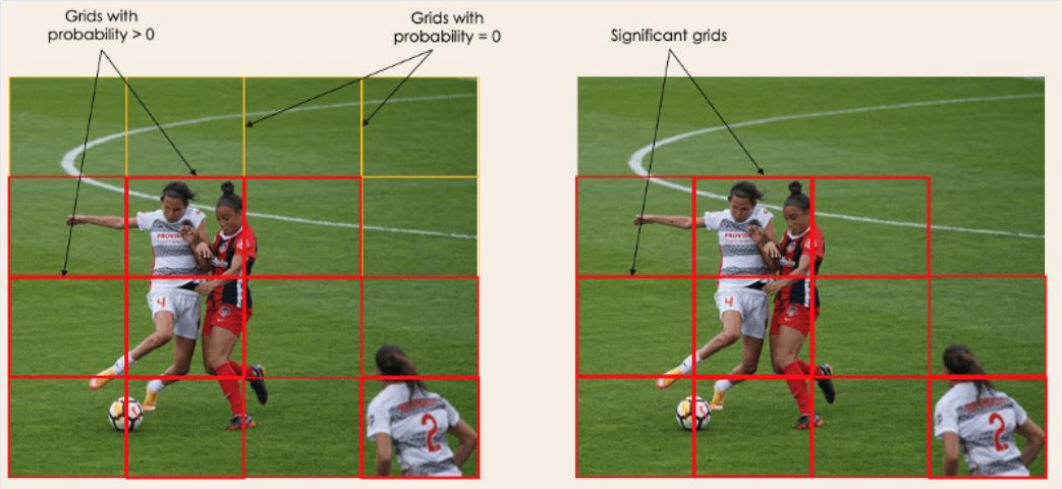
Input image grid division for Residual Blocks (Source: Datacamp)

YOLO: How it works – Bounding Box Regression

YOLO determines the attributes of these bounding boxes using a single regression module in the following format, where Y is the final vector representation for each bounding box.

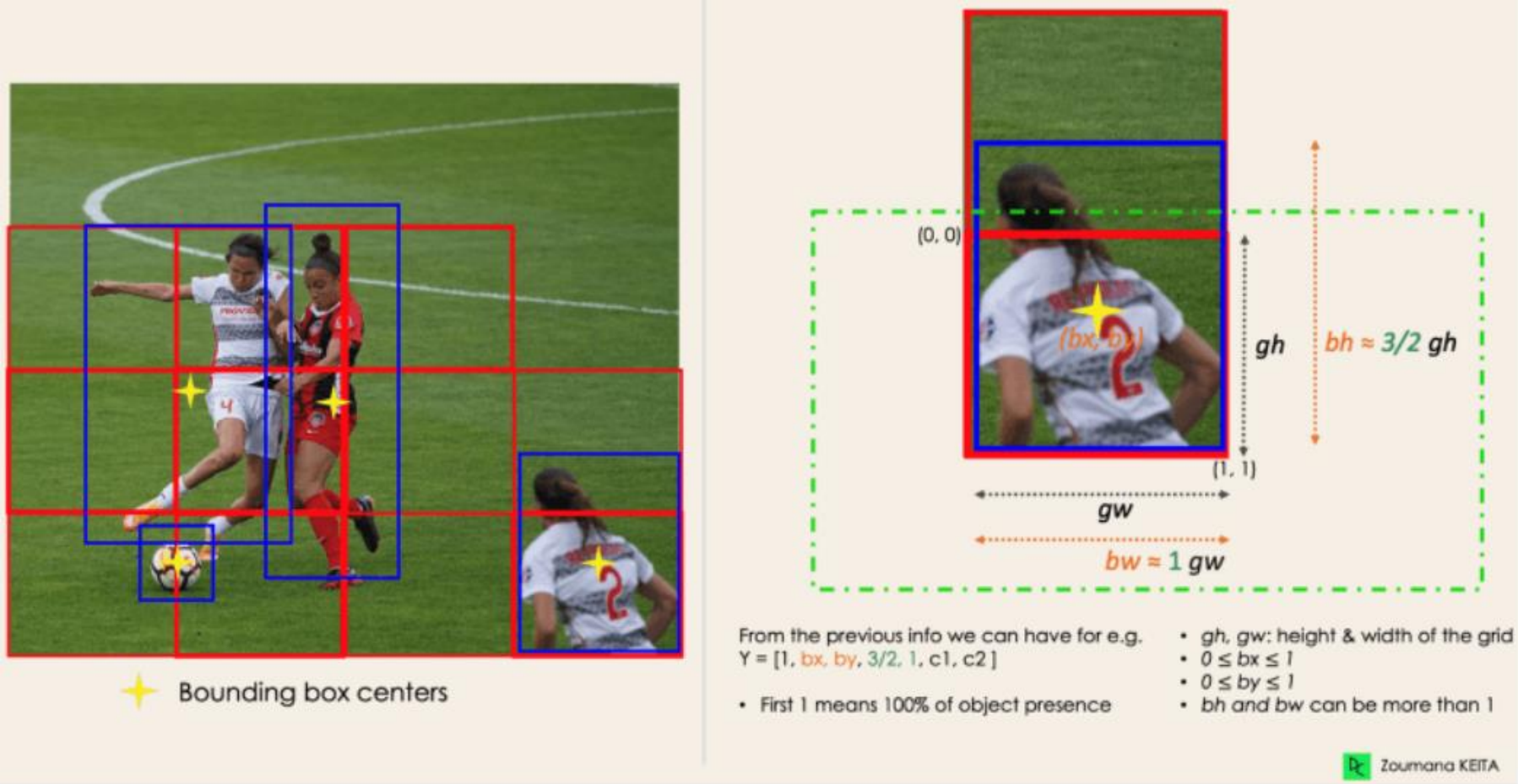
$$Y = [pc, bx, by, bh, bw, c1, c2]$$

- pc corresponds to the probability score of the grid containing an object.
- bx, by are the x and y coordinates of the center of the bounding box w.r.t. the enveloping grid cell.
- bh, bw correspond to the height and the width of the bounding box w.r.t. the enveloping grid cell.
- c1 and c2 correspond to the two classes Player and Ball.



Grid Division according to objects' probabilities (Source: Datacamp)

YOLO: How it works – Bounding Box Regression

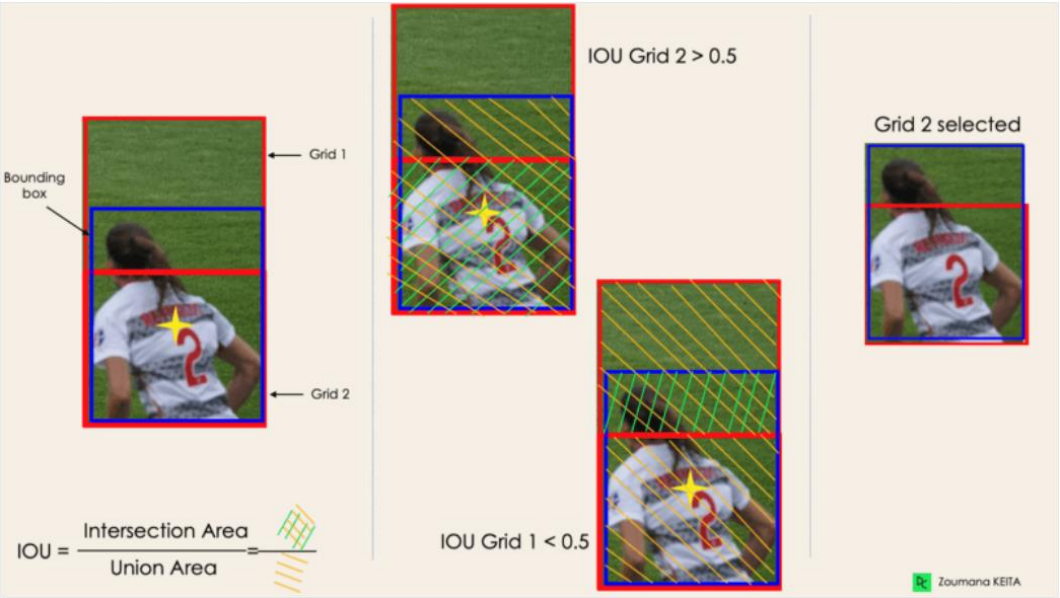


Bounding Box Regression Applied to Bottom Player (Source: Datacamp)

YOLO: How it works – IoU

Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant.

- The user defines its IOU selection threshold, which can be, for instance, 0.5.
- Computes the IOU of each grid cell which is the Intersection area divided by the Union Area.
- It ignores the prediction of the grid cells having an IOU ≤ threshold and considers those with an IOU > threshold.



IoU functioning of YOLO (Source: Datacamp)

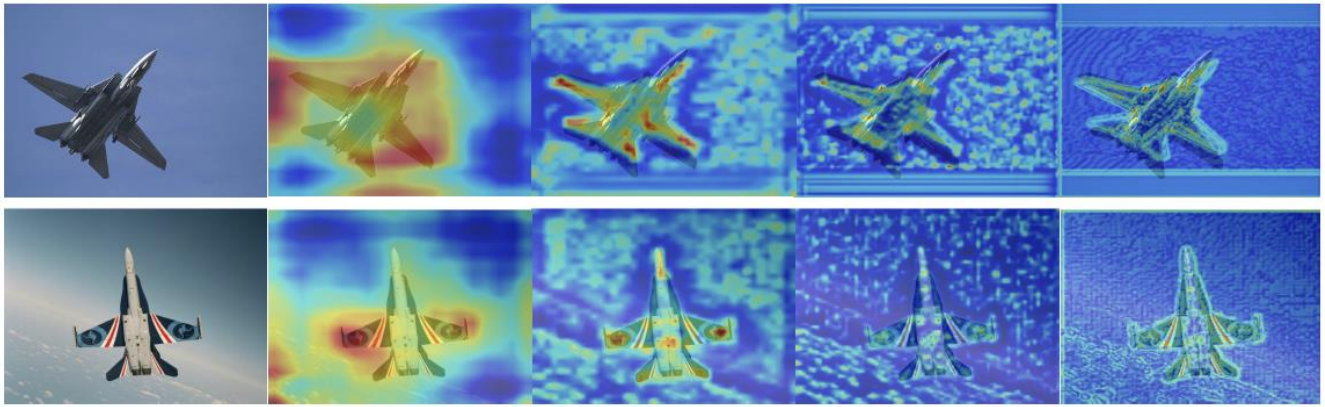
YOLO: How it works – Non-Max Suppression or NMS

Non-Maximum Suppression (NMS) is a post-processing technique used in object detection algorithms to reduce the number of overlapping bounding boxes and improve the overall detection quality. Object detection algorithms typically generate multiple bounding boxes around the same object with different confidence scores. NMS filters out redundant and irrelevant bounding boxes, keeping only the most accurate ones.

Non-Maximum Suppression (NMS). a) Shows the typical output of an object detection model containing multiple overlapping boxes. b) Shows the output after NMS (Source: A Comprehensive Review of YOLO)

YOLOv5

- The backbone of YOLOv5 is Darknet53, a new network architecture that focuses on feature extraction
- Cross Stage Partial connections enable the architecture to achieve a richer gradient flow
- The neck aggregates and refines the features extracted by the backbone
- YOLOv5's head consists of three branches, each predicting a different feature scale. Each head produces bounding boxes, class probabilities, and confidence scores.
- Finally, the network uses Non-maximum Suppression (NMS) to filter out overlapping bounding boxes.



The four stages of the Feature Activation maps of CSPDarkNet53 backbone (Source: Real-Time Flying Object Detection with YOLOv8)

YOLOv5

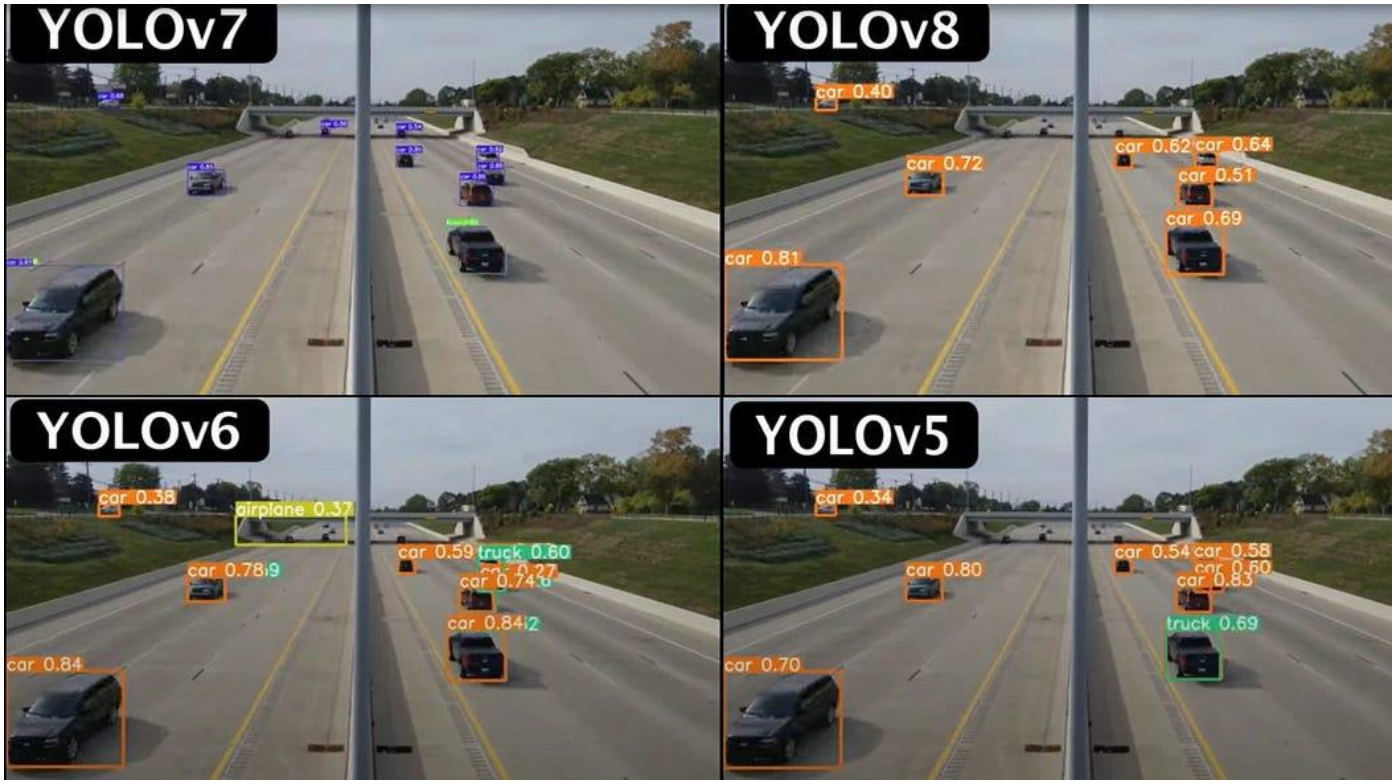
The different models among version 5:

Model	YAML	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
yolov5nu.pt	yolov5n.yaml	640	34.3	73.6	1.06	2.6	7.7
yolov5su.pt	yolov5s.yaml	640	43.0	120.7	1.27	9.1	24.0
yolov5mu.pt	yolov5m.yaml	640	49.0	233.9	1.86	25.1	64.2
yolov5lu.pt	yolov5l.yaml	640	52.2	408.4	2.50	53.2	135.0
yolov5xu.pt	yolov5x.yaml	640	53.2	763.2	3.81	97.2	246.4

Specifications Table (Source: Ultralytics)

YOLOv7: Focal Loss

YOLOv7 incorporates a new technique called “Focal Loss”, which is designed to address the class imbalance problem that often arises in object detection tasks. The Focal Loss function gives more weight to hard examples and reduces the influence of easy examples.



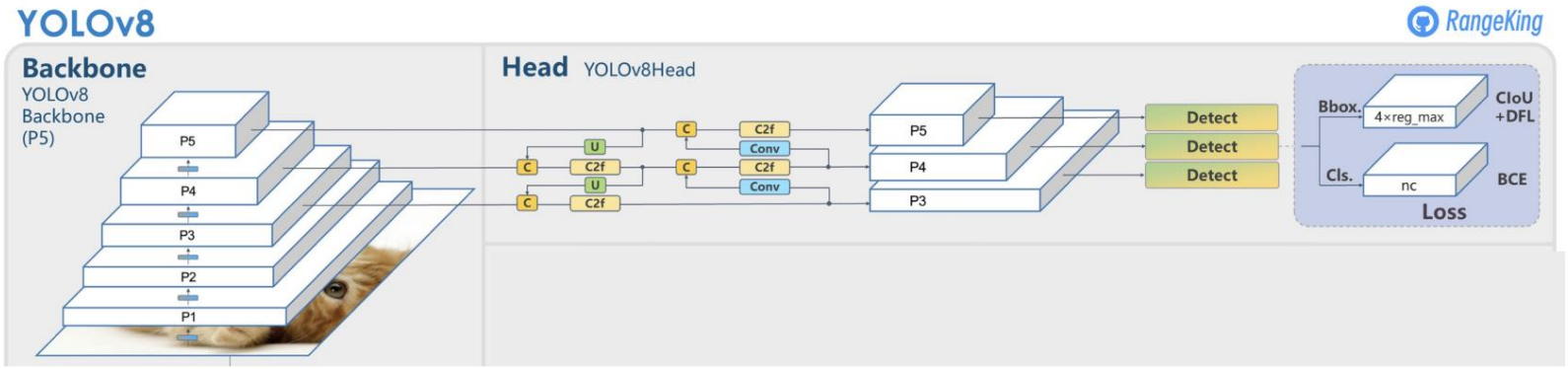
YOLO v5 to v8 Comparison (Source: Medium)

YOLOv8

Some of the improvements in this version are: new neural network architecture that utilizes both Feature Pyramid Network (FPN) and Path Aggregation Network (PAN) and a new labeling tool that simplifies the annotation process

- The FPN works by gradually reducing the spatial resolution of the input image while increasing the number of feature channels. This results in the creation of feature maps that are capable of detecting objects at different scales and resolutions.

- The PAN architecture, on the other hand, aggregates features from different levels of the network through skip connections. By doing so, the network can better capture features at multiple scales and resolutions, which is crucial for accurately detecting objects of different sizes and shapes.



YOLOv8 Architecture (Source: Real-Time Flying Object Detection with YOLOv8)

YOLOv8

The different models among version 8:

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

Specifications table (Source: Ultralytics)

Outline

1. Introduction
2. DIOR Dataset
3. Models
4. Experimental Setup
5. Results
6. Summary

Experiments

Model	V5		V8	
Nano	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2		epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2
Small	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-3
Medium	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	

Experimental Protocol

Dataset Distribution:

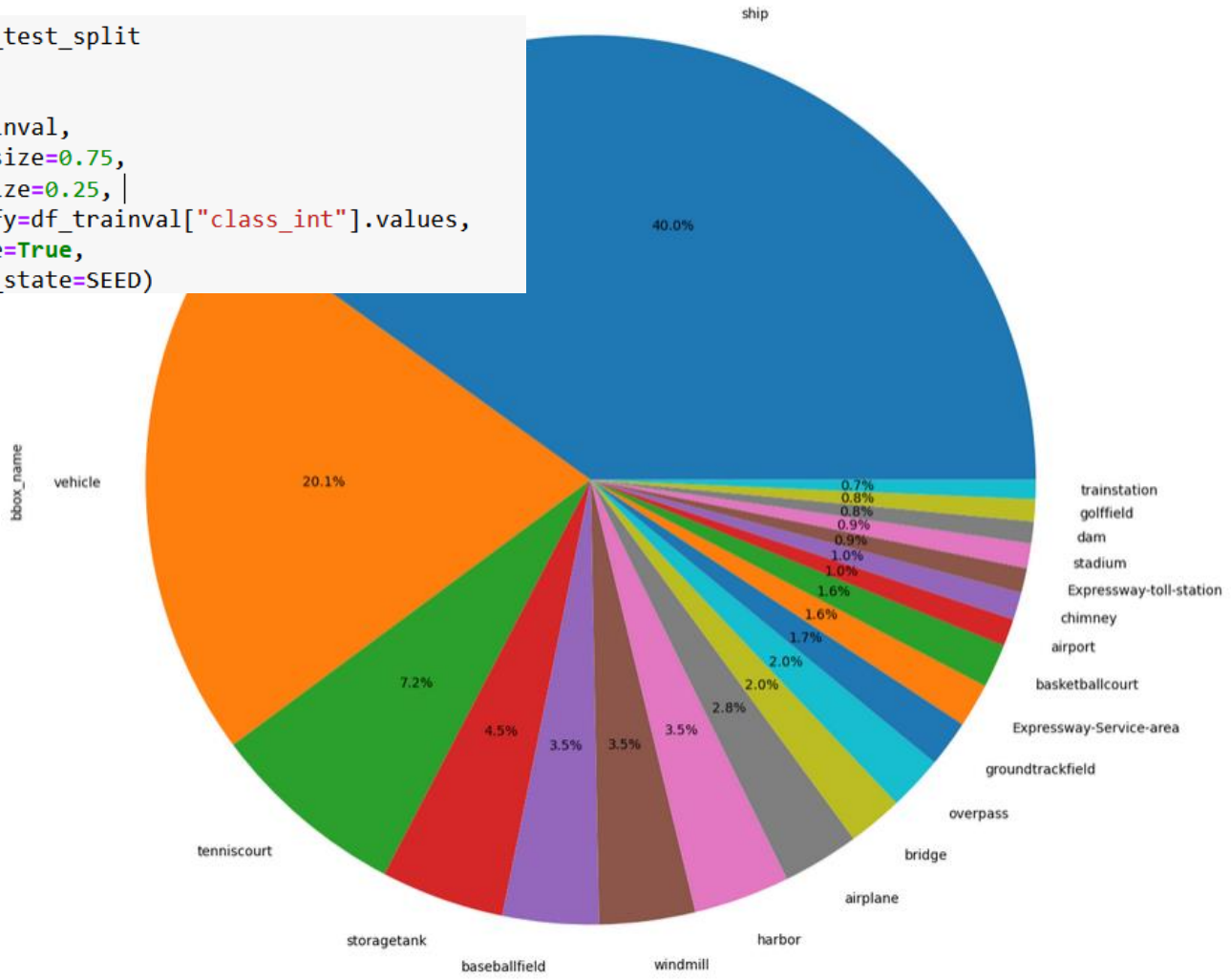
	Recommended Distribution (EDA)	Used Distribution for YOLO
Training set	**25%** (1 – 5862)	**45%** (10590)
Validation set	**25%** (5863 – 11725)	6387 images
Test set	**50%** (11726 – 23463)	**50%** (11738)

Metrics:

- Recall
- Precision
- mAP

EDA
Number of Classes in the Dataset

```python
from sklearn.model_selection import train_test_split
# Split train, val
SEED = 42
train_df,val_df = train_test_split(df_trainval,
                                   train_size=0.75,
                                   test_size=0.25,
                                   stratify=df_trainval["class_int"].values,
                                   shuffle=True,
                                   random_state=SEED)
```



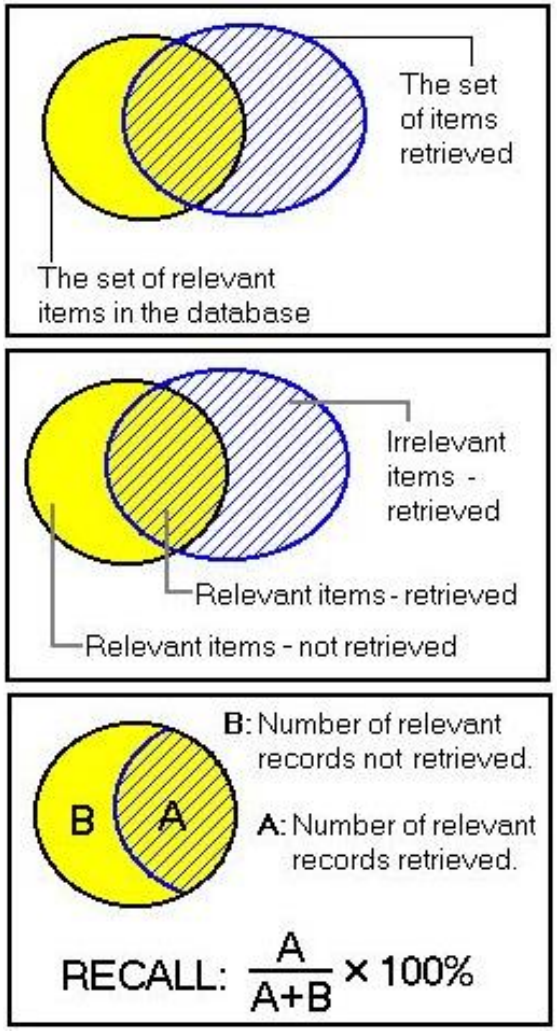
Frequency distribution of objects per class in %

Metrics: Recall

The recall measures the model's ability to detect Positive samples. The higher the recall, the more positive samples detected.

The recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ NEgatives}$$



Recall explanation (Source: Slide Player)

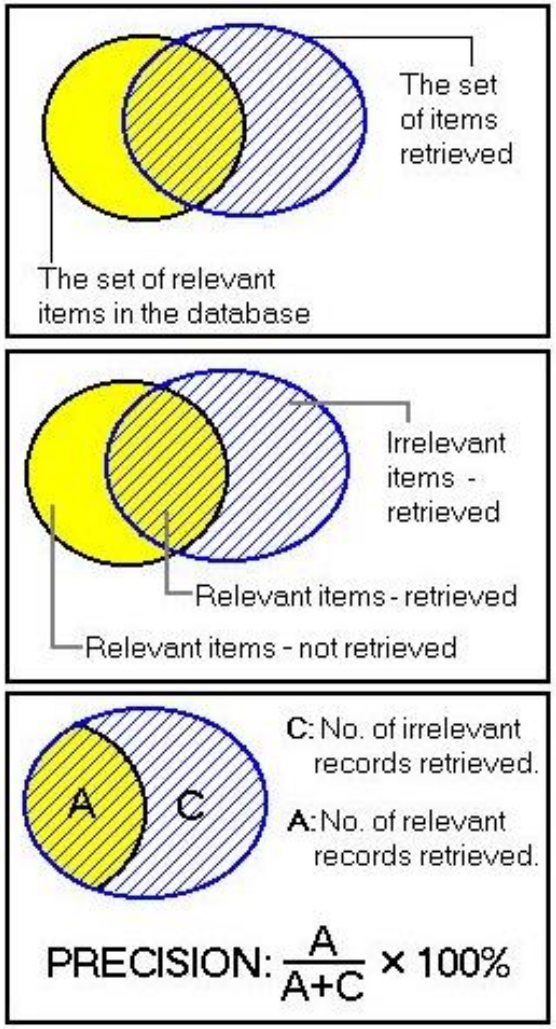
Metrics: Precision

The precision is calculated as the ratio between the number of Positive samples correctly classified to the total number of samples classified as Positive (either correctly or incorrectly).

When the model makes many incorrect Positive classifications, or few correct Positive classifications, this increases the denominator and makes the precision small. On the other hand, the precision is high when:

- The model makes many correct Positive classifications (maximize True Positive).
- The model makes fewer incorrect Positive classifications (minimize False Positive).

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$



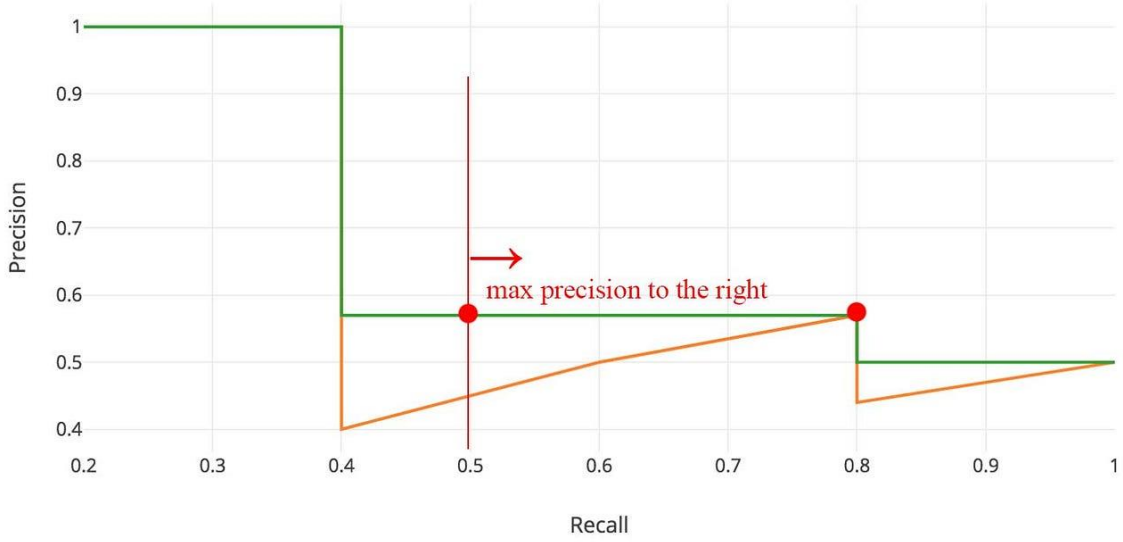
Precision explanation (Source: Slide Player)

Metrics: mAP

AP (Average Precision)

Represents the area under the curve Precision Recall Curve. The higher the curve is in the upper right corner, the larger the area, so the higher the AP, and the better the machine learning model.

$$mAP = \frac{1}{N}\sum_{N} AP_n$$



Pecision vs Recall Cruve (Source: Medium)

Metrics: F1 Score

F1 score is a machine learning evaluation metric that measures a model's accuracy. It combines the precision and recall scores of a model. The accuracy metric computes how many times a model made a correct prediction across the entire dataset.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Outline

1. Introduction
2. DIOR Dataset
3. Models
4. Experimental Setup
5. Results
6. Summary

Experiments

Model	V5		V8	
Nano	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2		epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2
Small	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-3
Medium	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	

Configuration: YOLO - v8 & v5 – 5 epochs

Google Colab free version

NVIDIA-SMI 525.105.17
Driver Version: 525.105.17
CUDA Version: 12.0
Tesla T4 15360MiB

Model	V5	V8
Nano	214 Layers 1,790,977 Parameters 1,790,977 Gradients 02:08:33 Duration	225 Layers 3,014,748 Parameters 3,014,732 Gradients 01:29:01 Duration
Small	214 Layers 7,073,569 Parameters 7,073,569 Gradients 02:09:12 Duration	225 Layers 11,143,340 Parameters 11,143,324 Gradients 01:46:36 Duration
Medium	291 Layers 20,948,097 Parameters 20,948,097 Gradients 02:09:52 Duration	295 Layers 25,867,900 Parameters 25,867,884 Gradients 02:22:53 Duration

Results: YOLO - v8 & v5 – 5 epochs

Model	V5				V8			
	P	R	mAP 50	mAP 50-95	P	R	mAP 50	mAP 50-95
Nano	0.243	0.393	0.169	0.086	0.313	**0.466**	**0.197**	**0.119**
Small	0.214	0.407	0.172	0.085	0.322	0.373	0.168	0.104
Medium	**0.401**	0.350	0.164	0.085	0.231	0.331	0.137	0.091

Experiments

Model	V5		V8	
Nano	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2		epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2
Small	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 **lr: 1e-3**
Medium	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	

YOLO v8s: 50 epochs vs 5 epochs

of epochs: 50
# of batches: 16
Seed: 42
Optimizer: Adam
Image_Size: 800
lr: 1e-3

Python 3.11.4
Torch 2.0.1
Cuda 117
Quadro P2000, 5120 MiB

# of epochs: 5
# of batches: 16
Seed: 42
Optimizer: Adam
Image_Size: 800
lr: 1e-2

Google Colab free version

NVIDIA-SMI 525.105.17
Driver Version: 525.105.17
CUDA Version: 12.0
Tesla T4 15360MiB

Model	V8s – 50 epochs | lr 1e-3				V8s – 5 epochs | lr 1e-2			
	P	R	mAP50	mAP50-95	P	R	mAP50	mAP50-95
Small	0.393	0.79	0.394	0.295	0.322	0.373	0.168	0.104

50 epochs: 290.028 h ~ 12 days
5 epochs: 1.657 h

Experiments

Model	V5		V8	
Nano	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2		epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2
Small	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-3
Medium	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	

YOLO v5s: 50 epochs vs 5 epochs

of epochs: 50
# of batches: 16
Seed: 42
Optimizer: Adam
Image_Size: 800
lr: 1e-2

Python 3.11.4
Torch 2.0.1
Cuda 117
Quadro P2000, 5120 MiB

# of epochs: 5
# of batches: 16
Seed: 42
Optimizer: Adam
Image_Size: 800
lr: 1e-2

Google Colab free version

NVIDIA-SMI 525.105.17
Driver Version: 525.105.17
CUDA Version: 12.0
Tesla T4 15360MiB

Model	V5s– 50 epochs				V5s – 5 epochs			
	P	R	mAP50	mAP50-95	P	R	mAP50	mAP50-95
Small	0.317	0.682	0.313	0.188	0.214	0.407	0.172	0.085

50 epochs: 258.602 h ~ 11 days
5 epochs: 1.951 h

Experiments

Model	V5		V8	
Nano	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2		epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2
Small	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 50 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-3
Medium	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	epochs: 10 batches: 16 optimizer: SGD image_size: 800 lr: 1e-2	epochs: 5 batches: 16 seed: 42 optimizer: adam image_size: 800 lr: 1e-2	

YOLO v8s: 50 epochs vs 5 epochs

Model	V5m – 5epochs | Adam				V5m – 10epochs | SGD			
	P	R	mAP50	mAP50-95	P	R	mAP50	mAP50-95
Small	**0.401**	0.350	0.164	0.085	**0.389**	0.763	**0.391**	**0.288**

Model	V8n – 5epochs | Adam				V8n – 10epochs | SGD			
	P	R	mAP50	mAP50-95	P	R	mAP50	mAP50-95
Small	0.313	**0.466**	0.197	0.119	**0.896**	0.248	**0.566**	**0.475**

YOLOv5m-10epochs: Training

```
Validating runs/detect/train2/weights/best.pt...
Ultralytics YOLOv8.0.178 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla V100-SXM2-16GB, 16151MiB)
YOLOv5m summary (fused): 248 layers, 25056796 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 200/200 [01:00<00:00,  3.30it/s]
                   all       6387      17018      0.389      0.763      0.391      0.288
             golffield       6387        128      0.571      0.781      0.585      0.407
               vehicle       6387       3431      0.269      0.675      0.255      0.171
Expressway-toll-station       6387        152      0.477      0.684       0.41      0.323
           trainstation       6387        125      0.443      0.448      0.373      0.188
                chimney       6387        162      0.492      0.901      0.482      0.429
           storagetank       6387        761      0.303      0.765      0.339      0.299
                  ship       6387       6837      0.243      0.925      0.257      0.178
                harbor       6387        591      0.191      0.635      0.175      0.114
              airplane       6387        472      0.359      0.922      0.359      0.315
           tenniscourt       6387       1225      0.297      0.956      0.317      0.298
      groundtrackfield       6387        291      0.404       0.85      0.458      0.393
                   dam       6387        128      0.553      0.633      0.524      0.274
        basketballcourt       6387        269      0.339      0.847      0.349      0.312
Expressway-Service-area       6387        270      0.335      0.804      0.335      0.235
               stadium       6387        149      0.511      0.924      0.527       0.42
               airport       6387        165      0.565      0.835      0.658      0.407
          baseballfield       6387        596      0.359      0.953      0.365      0.333
                 bridge       6387        342      0.349      0.418      0.284      0.176
              windmill       6387        591      0.374      0.729      0.479       0.28
              overpass       6387        333      0.348      0.575      0.291      0.202
Speed: 0.2ms preprocess, 3.9ms inference, 0.0ms loss, 0.8ms postprocess per image
Results saved to runs/detect/train2
```

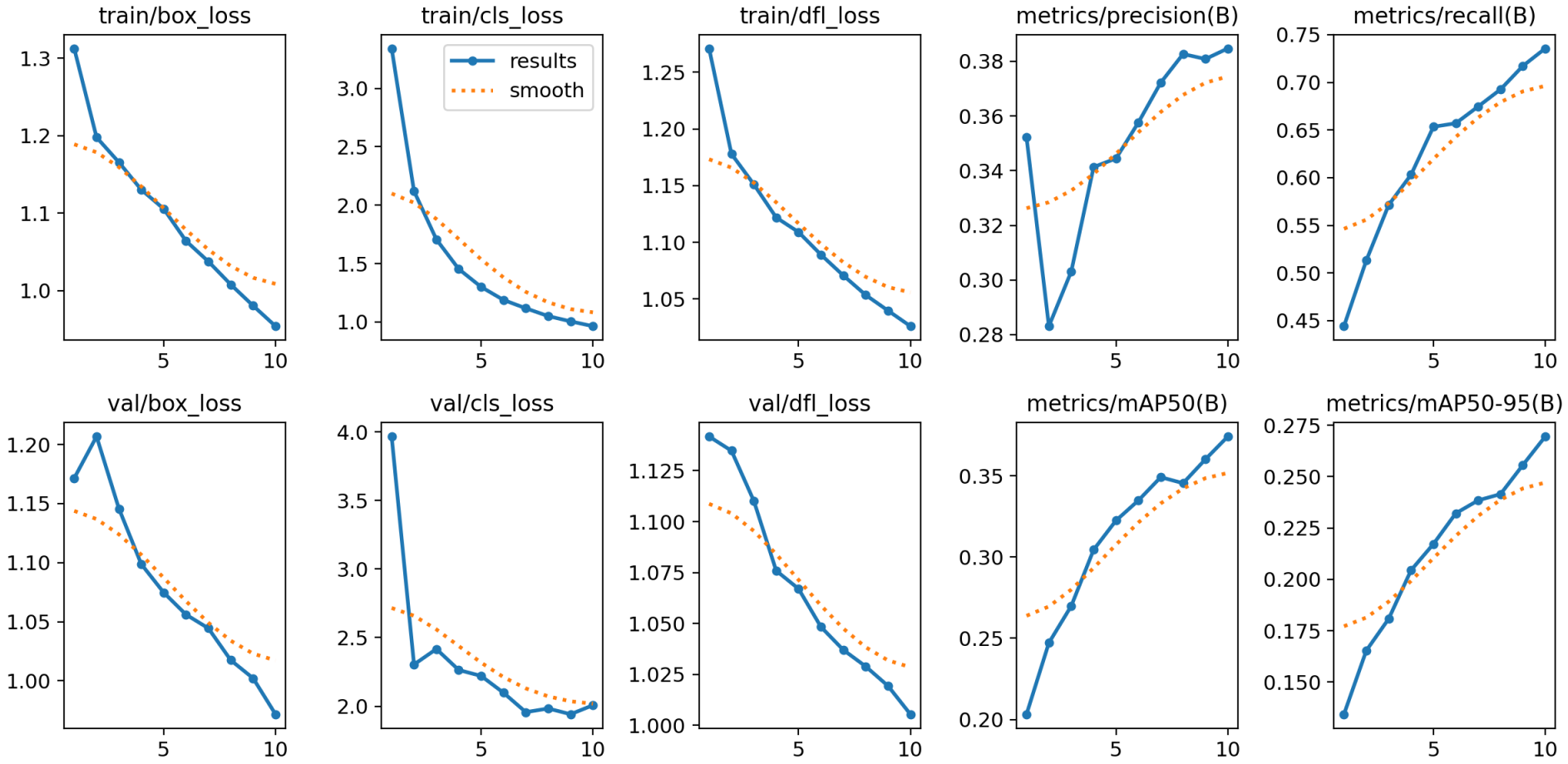
Results table (Source: Ultralytics)

YOLOv5m-10epochs: Inference

```
Ultralytics YOLOv8.0.178 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla V100-SXM2-16GB, 16151MiB)
val: Scanning /content/drive/MyDrive/Colab Notebooks/FinalProject/Yolov8/yolo_datasets/test/labels... 10902 images, 836 backgrounds, 0 corrupt
val: New cache created: /content/drive/MyDrive/Colab Notebooks/FinalProject/Yolov8/yolo_datasets/test/labels.cache
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 734/734 [02:15<00:00,  5.42it/s]
                   all      11738     114556      0.896      0.248      0.566      0.475
             golffield      11738        542       0.91      0.299      0.594      0.464
               vehicle      11738      24864      0.925     0.0663      0.494      0.415
Expressway-toll-station      11738        635       0.93      0.376      0.642      0.586
           trainstation      11738        468      0.731     0.0406      0.385       0.26
                chimney      11738        946      0.903      0.441      0.661      0.597
           storagetank      11738      21281      0.939     0.0942      0.514      0.475
                  ship      11738      32066      0.926     0.0279      0.477      0.407
                harbor      11738       2869      0.878     0.0802      0.476      0.396
              airplane      11738       7423      0.876      0.357      0.602      0.508
           tenniscourt      11738       6816      0.923      0.238      0.574      0.531
      groundtrackfield      11738       1735      0.902       0.33      0.607      0.541
                   dam      11738        512      0.844      0.307      0.575      0.371
        basketballcourt      11738       2016      0.931      0.655      0.778      0.728
Expressway-Service-area      11738       1009       0.89     0.0882      0.488      0.391
               stadium      11738        623      0.886      0.376      0.627      0.541
               airport      11738        618      0.894      0.286      0.583      0.426
          baseballfield      11738       3251      0.934       0.53      0.723      0.649
                 bridge      11738       2414      0.847     0.0961      0.468      0.369
              windmill      11738       2837      0.934     0.0892       0.51      0.397
              overpass      11738       1631      0.913      0.174      0.537      0.446
Speed: 0.3ms preprocess, 8.1ms inference, 0.0ms loss, 0.5ms postprocess per image
Results saved to runs/detect/val2
```

Results table (Source: Ultralytics)

YOLOv5m-10epochs: Training

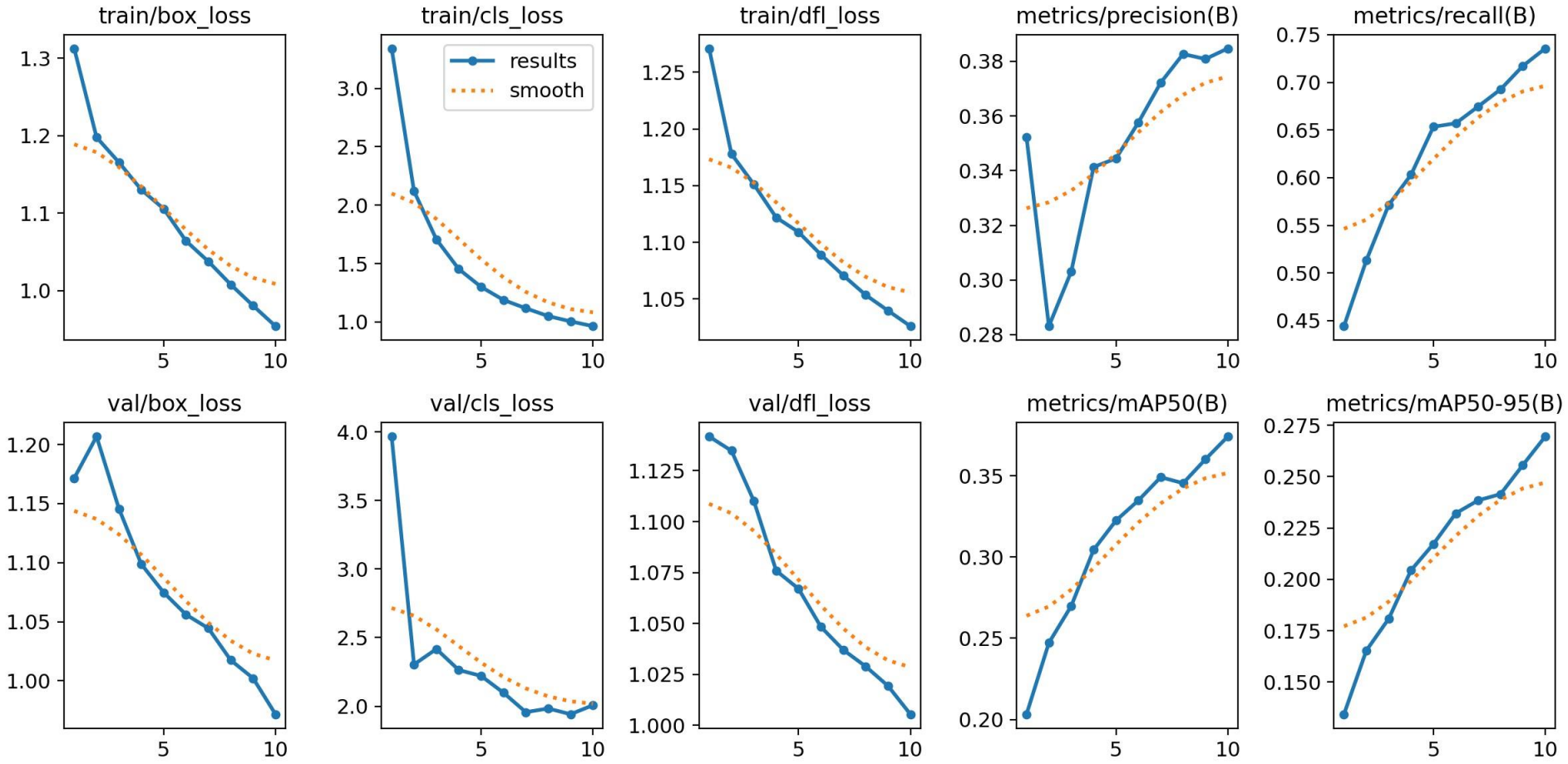


YOLOv8n-10epochs: Training

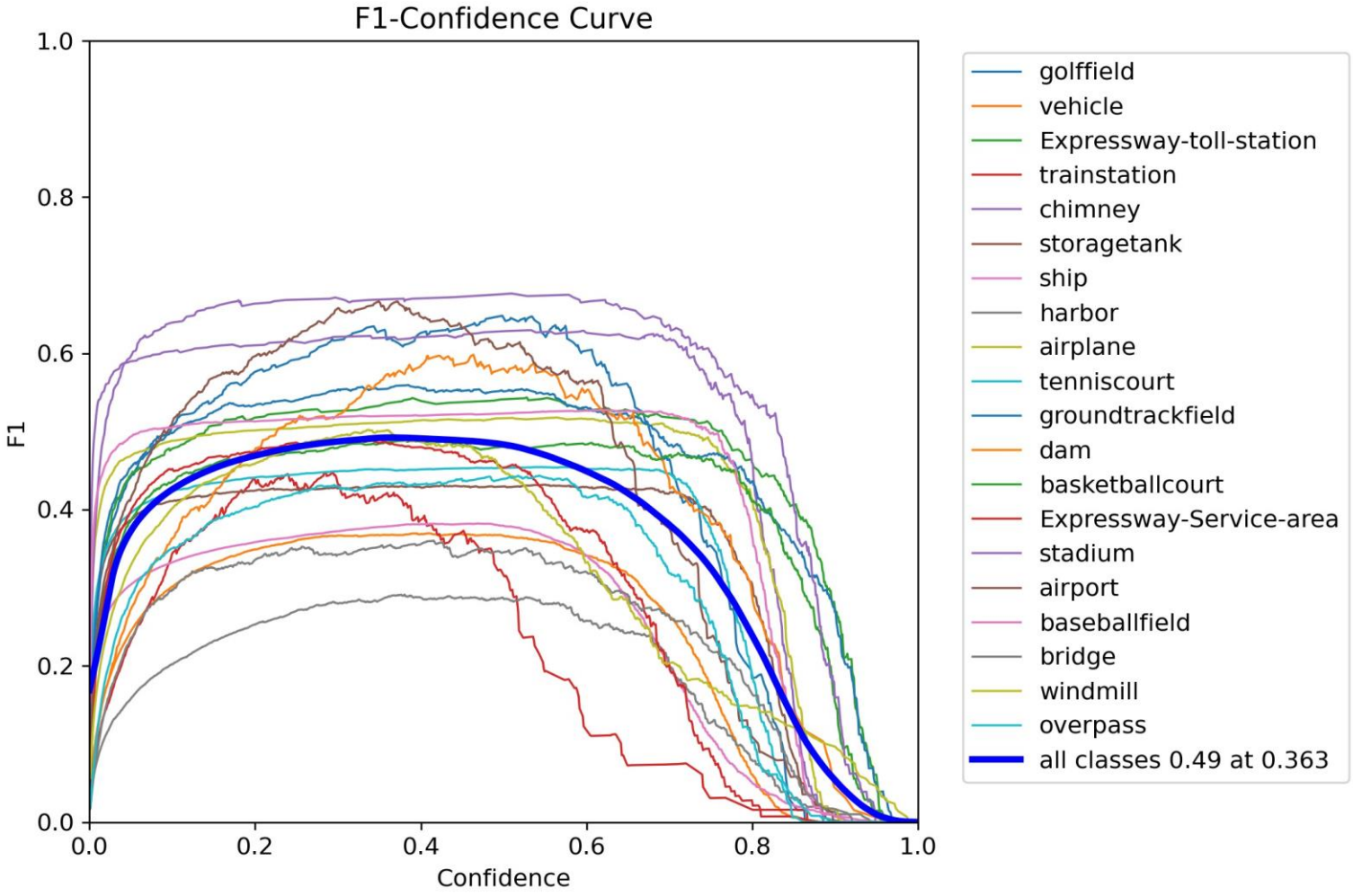
```
Validating runs/detect/train4/weights/best.pt...
Ultralytics YOLOv8.0.173 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla V100-SXM2-16GB, 16151MiB)
Model summary (fused): 168 layers, 3009548 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 200/200 [01:02<00:00,  3.21it/s]
                   all       6387      17018      0.385      0.736      0.374       0.27
             golffield       6387        128      0.506      0.767      0.565      0.382
               vehicle       6387       3431      0.268      0.582      0.226      0.144
Expressway-toll-station       6387        152      0.453      0.664      0.338      0.287
           trainstation       6387        125      0.444      0.384      0.369      0.165
                chimney       6387        162      0.476      0.883      0.483      0.427
           storagetank       6387        761      0.308      0.711      0.297      0.252
                  ship       6387       6837      0.242      0.886      0.253      0.167
                harbor       6387        591      0.191      0.599      0.171      0.112
              airplane       6387        472      0.354      0.926      0.347        0.3
           tenniscourt       6387       1225      0.297      0.934      0.312      0.286
      groundtrackfield       6387        291      0.423      0.821      0.437       0.37
                   dam       6387        128      0.501      0.675      0.561      0.288
        basketballcourt       6387        269      0.348      0.818      0.335      0.291
Expressway-Service-area       6387        270      0.343      0.811      0.319      0.205
               stadium       6387        149      0.527      0.919      0.536       0.44
               airport       6387        165      0.585       0.76      0.608      0.371
          baseballfield       6387        596       0.36      0.935      0.365      0.333
                 bridge       6387        342      0.346       0.36       0.25      0.149
              windmill       6387        591      0.371      0.719       0.44      0.245
              overpass       6387        333      0.351      0.565      0.276      0.182
Speed: 0.3ms preprocess, 1.0ms inference, 0.0ms loss, 1.0ms postprocess per image
Results saved to runs/detect/train4
```

Results table (Source: Ultralytics)

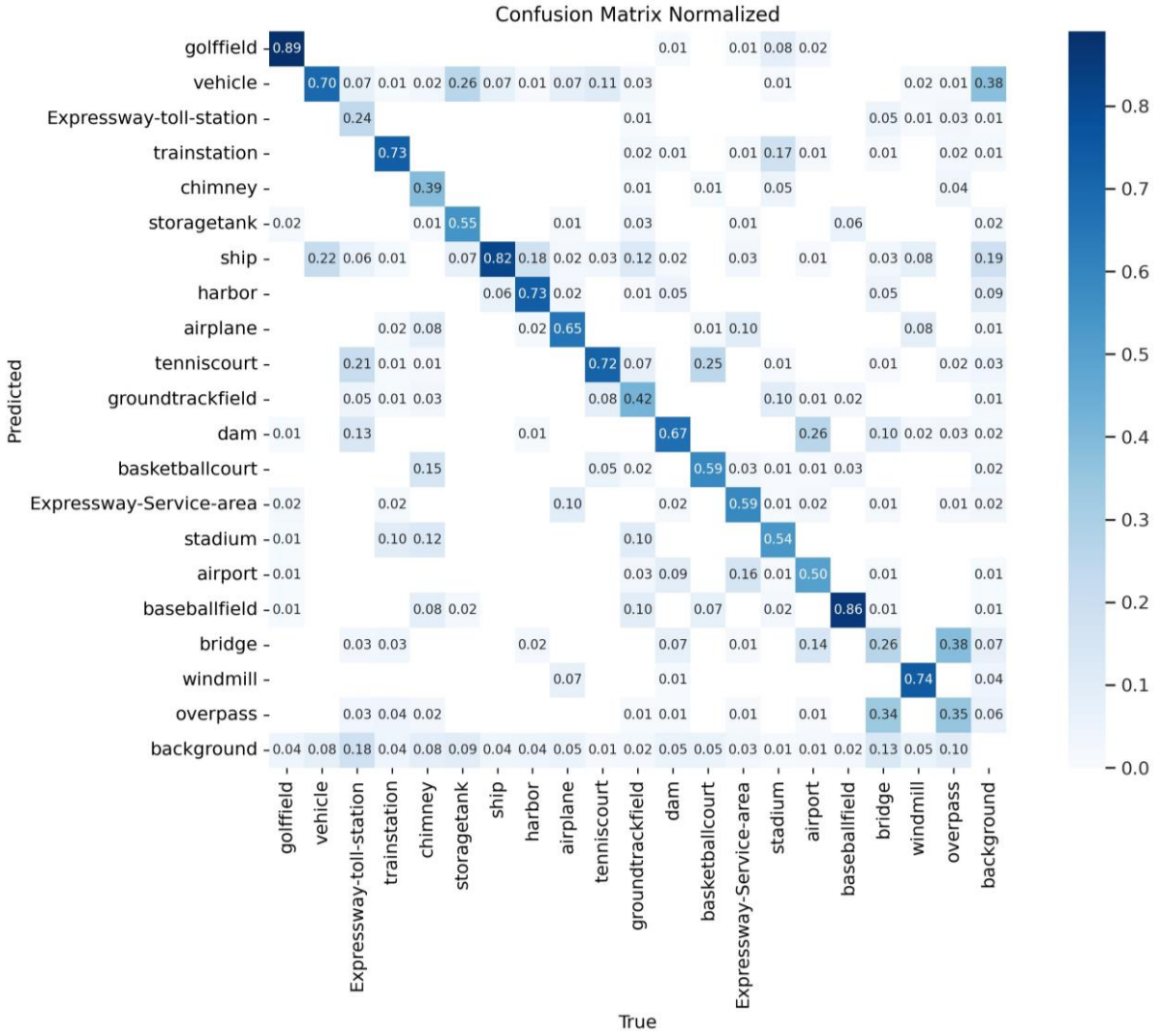
YOLOv8n-10epochs: Training



YOLOv8n-10epochs: Training



YOLOv8n-10epochs: Training

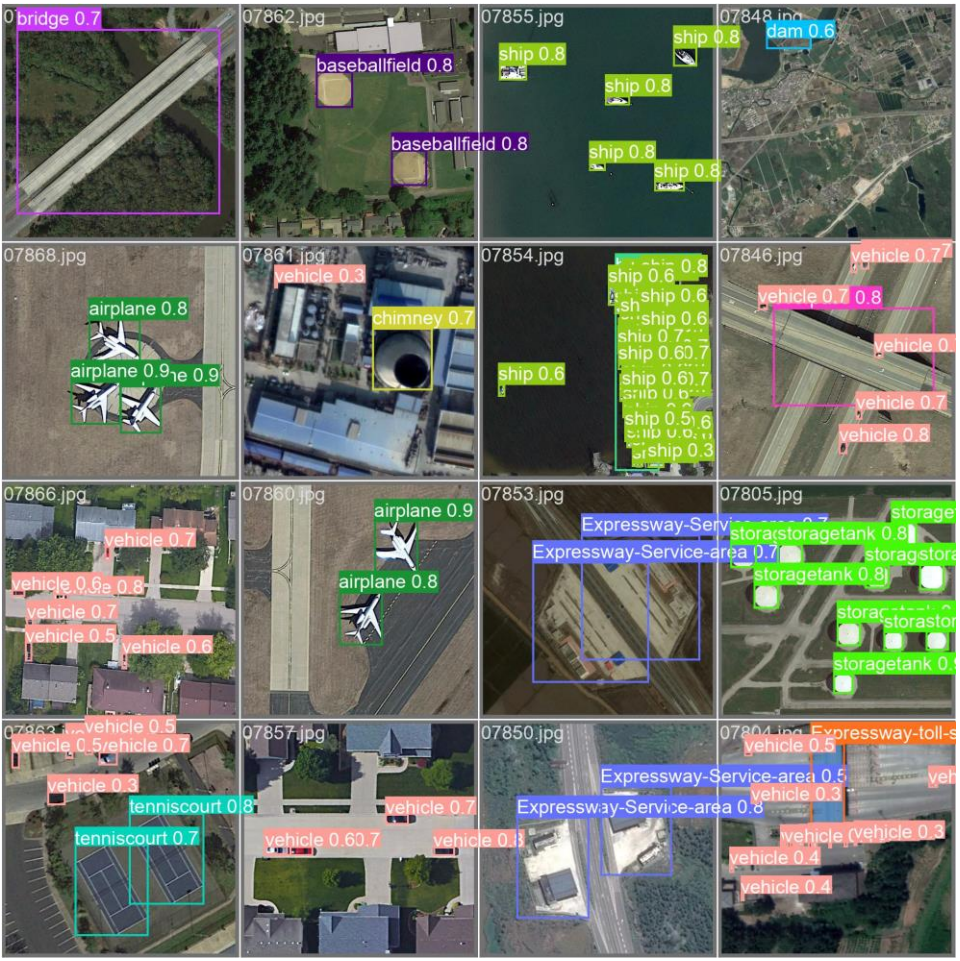


YOLOv8n: Prediction vs Labels

Predictions from YOLOv8n-based model

Ground Truth Labels

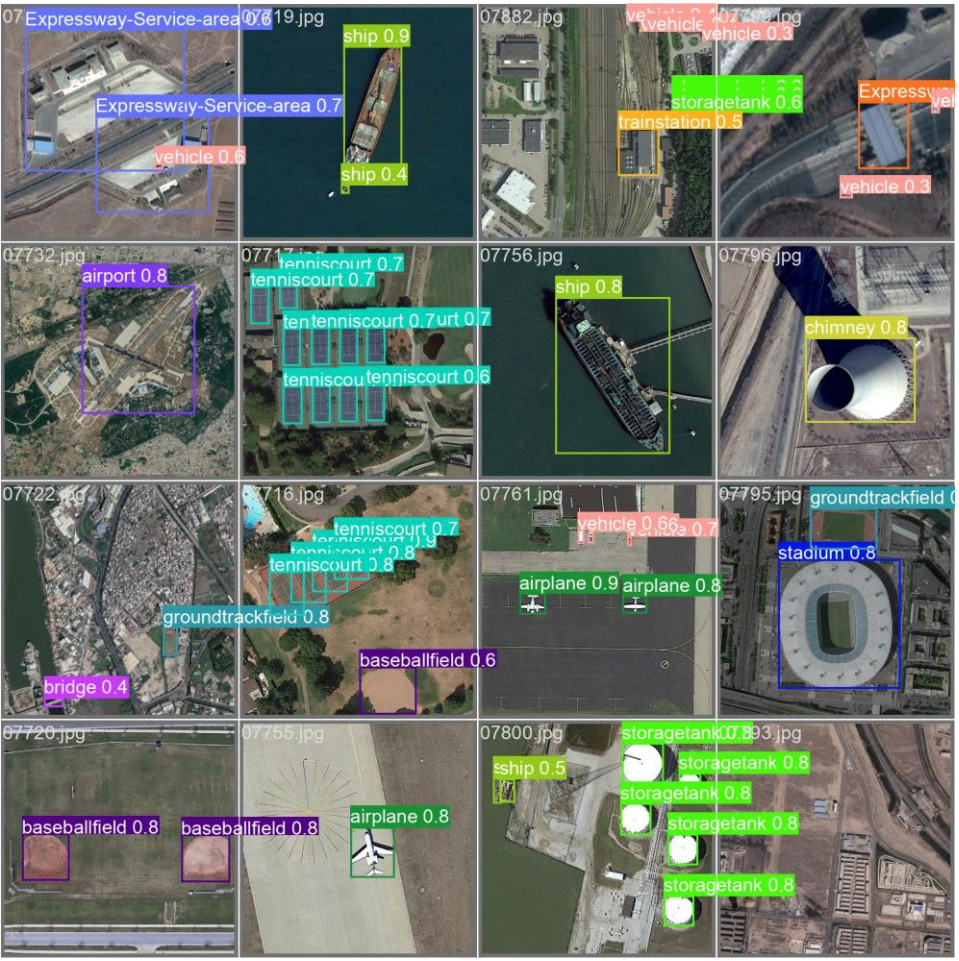
YOLOv8n: Prediction vs Labels



Predictions from YOLOv8n-based model

Ground Truth Labels

YOLOv8n: Prediction vs Labels

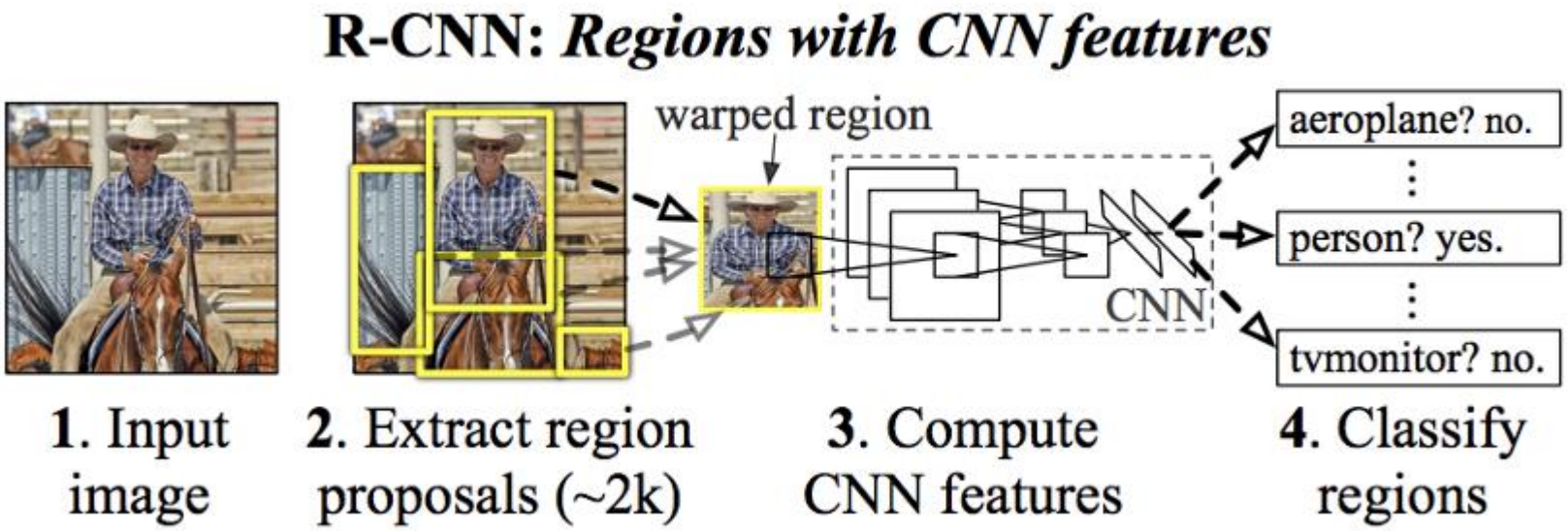


Predictions from YOLOv8n-based model

Ground Truth Labels

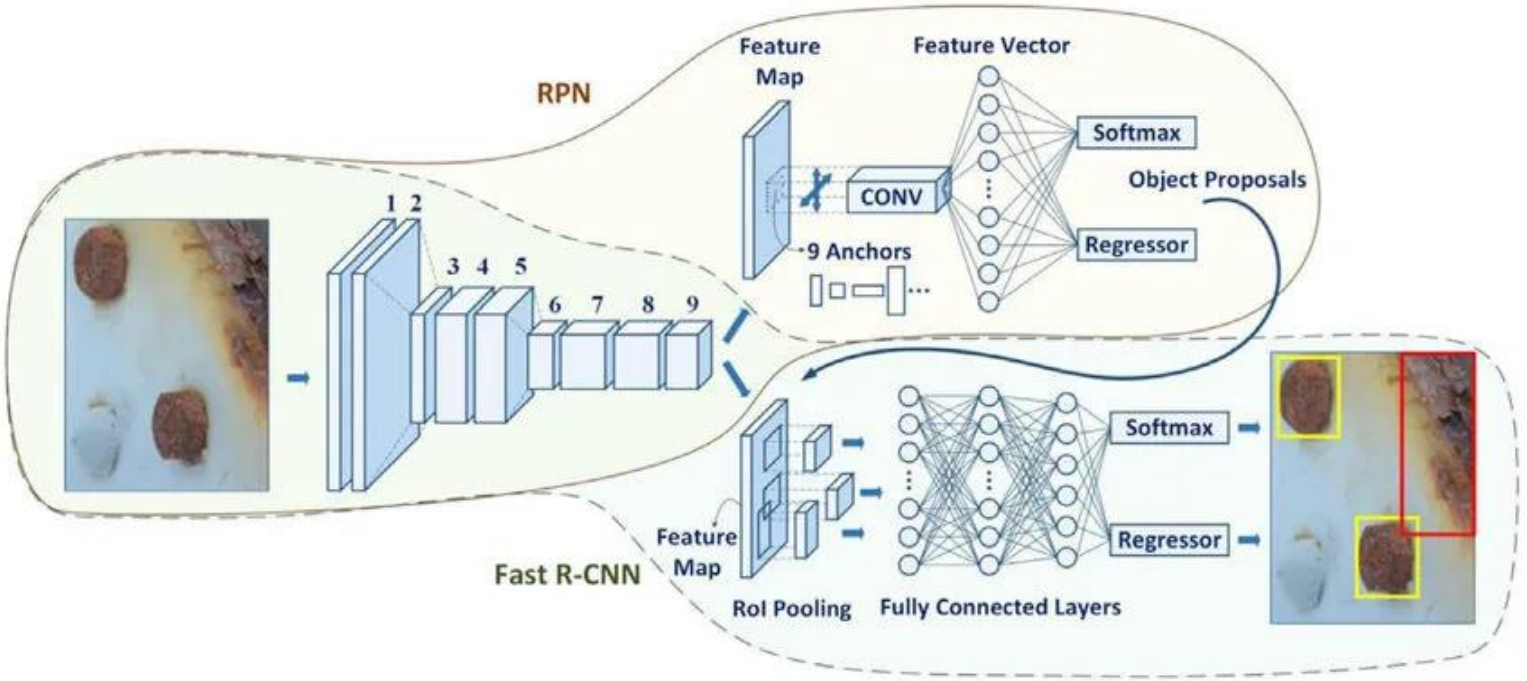
Other Models: Faster R-CNN

Unlike two-stage detection models, such as R-CNN, that first propose regions of interest and then classify these regions, YOLO processes the entire image in a single pass, making it faster and more efficient.



Faster RCNN Object detection (Source: Analytics Vidhya)

Other Models: Faster R-CNN



Faster RCNN Object detection (Source: Towards Data Science)

Outline

1. Introduction
2. DIOR Dataset
3. Models
4. Experimental Setup
5. Results
6. Summary

Summary

- Our aim to build robust and efficient object detection systems, therefore YOLO was used in different version
- One of the biggest challenges was preparing the data for the Training of the model
- According to the results, the model can be further enhanced for an improved functioning. Hyperparameter tunning would be a proposal.

References

- Reis, D., Kupec, J., Hong, J., & Daoudi, A. (2023). Real-Time Flying Object Detection with YOLOv8. arXiv preprint arXiv:2305.09972.

- Terven, J., & Cordova-Esparza, D. (2023). A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond. arXiv preprint arXiv:2304.00501, https://arxiv.org/pdf/2304.00501.pdf