YOLO – You only Look once

1. Introduction to Object Detection and YOLO

What is Object Detection?

Object detection is a fundamental computer vision task that involves identifying and locating objects within images or videos. Unlike simpler tasks, object detection provides both what is in an image and where each object is located.

Key Differences:

Task	Description	Output
Classification	Assigns a single label to an entire image	Class label (e.g., "cat")
Object Detection	Identifies objects and their locations	Bounding boxes + class labe
Segmentation	Identifies objects at pixel level	Pixel-wise masks + class lab

Why YOLO = "You Only Look Once"

YOLO revolutionized object detection by treating it as a single regression problem. Traditional methods required multiple passes through the network, but YOLO predicts bounding boxes and class probabilities directly from the full image in one evaluation.

Key Advantages:

- Real-time performance: Can process 30+ FPS on decent hardware
- Single-stage detection: Unlike R-CNN family that requires separate classification and localization
- Global context: Sees the entire image during training and testing

- 2. YOLO Evolution Overview
- The YOLO family has evolved significantly since 2015, with each version introducing improvements in accuracy, speed, and functionality.

Version	Year	Key Highlights
YOLOv1	2015	First real-time object detector, grid-based approach
YOLOv2	2016	Batch normalization, anchor boxes, Darknet-19
YOLOv3	2018	Darknet-53, multi-scale predictions, FPN-inspired design
YOLOv4	2020	CSPDarknet, SPP, PAN, improved speed/accuracy
YOLOv5	2020	PyTorch implementation, easy training, SPPF
YOLOv6	2022	RepVGG architecture, industry optimized
YOLOv7	2022	E-ELAN, better real-time accuracy
YOLOv8	2023	Anchor-free, segmentation support, exportable formats

Why YOLOv8 is Superior

YOLOv8 represents a significant advancement with several key innovations:

- Anchor-free detection: Simplifies training and improves generalization
- Multi-task capabilities: Detection, segmentation, classification, pose estimation
- Modular architecture: Easy to customize and extend

- Export flexibility: Supports ONNX, TensorRT, CoreML formats
- Strong performance: Better speed/accuracy trade-off than predecessors

3. YOLOv8 Architecture Deep Dive

YOLOv8 follows a three-component architecture designed for maximum efficiency:

text

Input Image → Backbone → Neck → Head → Output (bbox, confidence, class)

Key Features:

- Anchor-free predictions: Eliminates need for predefined anchor boxes
- Multi-output prediction: Simultaneous bounding box and class prediction
- Loss optimization: Uses CIoU and DFL (Distribution Focal Loss)

4. YOLOv8 Model Variants

YOLOv8 offers five model sizes to balance speed and accuracy for different use cases:

Model	Size	Speed	mAP@0.5-0.95	Parameters	FLOPs	Use Case
YOLOv8n	Nano	Fastest	37.3	3.2M	8.7B	Real-time on lo GPU/mobile
YOLOv8s	Small	Fast	44.9	11.2M	28.6B	Drones, embed devices
YOLOv8m	Medium	Balanced	50.2	25.9M	78.9B	General purpos applications
YOLOv8l	Large	Accurate	52.9	43.7M	165.2B	Surveillance, hi accuracy needs

Model	Size	Speed	mAP@0.5-0.95	Parameters	FLOPs	Use Case
YOLOv8x	Extra Large	Most Accurate	53.9	68.2M	257.8B	Research, maximum precision

Selection Guidelines:

- Nano/Small: Resource-constrained environments, real-time applications
- Medium: Balanced performance for most applications
- Large/XLarge: When accuracy is more important than speed

6. Custom Dataset Creation

Step-by-Step Dataset Preparation

1. Data Collection

- Gather diverse images representing your target classes
- Aim for 500-1000+ images per class for good performance
- Include various lighting conditions, angles, and backgrounds

2. Data Annotation

Tools: LabelImg, Roboflow, CVAT, or Labelbox

3. Dataset Structure

dataset/

├— images/

| ├— train/

| | ├— img1.jpg

| | ├— img2.jpg

| └— val/

| ├— img3.jpg

| ├— img4.jpg

└── labels/

    ├── train/

    | ├── img1.txt

    | ├── img2.txt

    └── val/

        ├── img3.txt

        ├── img4.txt

4. Configuration File (data.yaml)

train: dataset/images/train

val: dataset/images/val

nc: 2 # number of classes

names: ['Helmet', 'NoHelmet']

7. Training YOLOv8 on Custom Dataset

Environment Setup

bash

 # Install required packages

pip install ultralytics

pip install opencv-python

pip install matplotlib

Basic Training Code

python

from ultralytics import YOLO

 # Load a pretrained model (recommended for transfer learning)

model = YOLO("yolov8s.pt") *# or yolov8n.pt for faster training*

```
# Train the model
results = model.train(
    data="data.yaml",       # path to dataset config
    epochs=50,              # number of training epochs
    imgsz=640,              # image size
    batch=8,                # batch size
    lr0=0.01,               # initial learning rate
    optimizer='Adam',       # optimizer choice
    device=0,               # GPU device (0 for first GPU)
    project='runs/train',   # project directory
    name='custom_model',    # experiment name
    save=True,              # save checkpoints
    verbose=True            # verbose output
)
```

Alternative: Command Line Interface

```
bash
yolo task=detect mode=train model=yolov8s.pt \
    data=data.yaml epochs=50 imgsz=640 batch=8 \
    lr0=0.01 optimizer=Adam device=0 \
    project=runs/train name=custom_model
```

Training Parameters Explained

Parameter	Purpose	Typical Values
epochs	Training duration	50-300 (more for complex tasks)

Parameter	Purpose	Typical Values
imgsz	Input image resolution	640 (standard), 416 (faster), 1280 (accuracy)
batch	Batch size for training	8-32 (depends on GPU memory)
lr0	Initial learning rate	0.001-0.01
optimizer	Optimization algorithm	Adam, SGD, AdamW
device	Hardware for training	0 (GPU), cpu

8. Hyperparameter Fine-Tuning

Critical Hyperparameters

1. Learning Rate (lr0)

Impact: Controls how quickly the model adapts to training data

Guidelines:

- Too high: Model overshoots optimal parameters
- Too low: Very slow convergence
- Recommended: Start with 0.01, reduce to 0.001 if unstable

2. Batch Size

Impact: Affects memory usage, training speed, and gradient stability

Trade-offs:

- Larger batches: More stable training, faster convergence, more memory
- Smaller batches: Less memory, noisier gradients, may need more epochs

3. Image Size (imgsz)

Impact: Resolution affects detection accuracy and computational cost

Image Size	Speed	Accuracy	Memory	Use Case
416	Fastest	Lower	Least	Real-time applications
640	Balanced	Good	Moderate	Standard training
1280	Slowest	Highest	Most	High-precision tasks

4. Epochs and Early Stopping

```
model.train(
    epochs=200,
    patience=30,      # Early stopping patience
    save_best=True    # Save best model
) 9. Model Evaluation and Metrics
```

Key Performance Metrics

Mean Average Precision (mAP)

Most important metric for object detection evaluation

- mAP@0.5: Average precision at IoU threshold of 0.5
- mAP@0.5:0.95: Average precision across IoU thresholds 0.5 to 0.95
- Higher values indicate better performance
- Performance Analysis

Metric	Good Range	Interpretation
mAP@0.5	>0.5	General detection capability

Metric	Good Range	Interpretation
mAP@0.5:0.95	>0.3	Localization accuracy
Precision	>0.8	Few false positives
Recall	>0.8	Few missed detections

10. Real-Time Inference and Deployment

Webcam Inference Example

python

```
from ultralytics import YOLO
import cv2

# Load trained model
model = YOLO("path/to/best.pt")

# Initialize webcam
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break
```

```
    # Run inference
    results = model(frame)

    # Visualize results
    annotated_frame = results[0].plot()

    # Display frame
    cv2.imshow('YOLOv8 Detection', annotated_frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Command Line Inference

bash

```
# Webcam inference
yolo predict model=best.pt source=0 show=True conf=0.5

# Video file inference
yolo predict model=best.pt source=video.mp4 save=True

# Image inference
yolo predict model=best.pt source=image.jpg save=True
```

Batch Processing

python

```
# Process multiple images
results = model.predict(
    source="path/to/images/",
    conf=0.25,       # Confidence threshold
    iou=0.45,        # NMS IoU threshold
    save=True,       # Save results
    save_txt=True,   # Save labels
    save_conf=True   # Save confidence scores
)
```

🧩 11. Model Export and Deployment

Export Formats

YOLOv8 supports multiple export formats for different deployment scenarios:

```
python
from ultralytics import YOLO

model = YOLO("best.pt")

# Export to different formats
model.export(format="onnx")     # ONNX for cross-platform
model.export(format="engine")   # TensorRT for NVIDIA GPUs
model.export(format="coreml")   # CoreML for Apple devices
model.export(format="tflite")   # TensorFlow Lite for mobile
model.export(format="pb")       # TensorFlow SavedModel
```

TensorRT Optimization (NVIDIA GPUs)

```
bash
```

 # Export to TensorRT engine

yolo export model=best.pt format=engine device=0 half=True

 # Use optimized model

yolo predict model=best.engine source=0 show=True

Performance Benefits:

- Up to 5x GPU speedup with TensorRT
- 3x CPU speedup with ONNX optimization
- Reduced memory footprint