**1. 📖 Introduction to Object Detection**

Object detection is a computer vision task that detects and classifies multiple objects in an image or video and draws bounding boxes around them.

Two tasks:

* Classification (What is in the image?)
* Localization (Where is the object?)

**2. ⚙️ What is YOLO?**

**YOLO** (You Only Look Once) is a single-stage object detection algorithm that predicts bounding boxes and class probabilities in a single evaluation.

**3. 🧬 YOLO Evolution: From v1 to v8**

| **Version** | **Release Year** | **Key Features** |
| --- | --- | --- |
| YOLOv1 | 2016 | Fast but less accurate |
| YOLOv2/v3 | 2017-18 | Improved accuracy & support for multi-label |
| YOLOv4 | 2020 | Speed + Accuracy, supported by OpenCV |
| YOLOv5 | 2020 | Written in PyTorch, modular, scalable |
| YOLOv6 | 2022 | Industrial-grade, optimized for edge devices |
| YOLOv7 | 2022 | State-of-the-art accuracy-speed tradeoff |
| **YOLOv8** | 2023 | New Ultralytics framework, better integration, more modular, supports segmentation, classification, detection |

**4. ✅ Why YOLOv8?**

* Built and maintained by **Ultralytics**
* Native support for:
  + Object Detection
  + Image Classification
  + Instance Segmentation
  + Pose Estimation
* Pythonic API (from ultralytics import YOLO)
* Easy deployment (ONNX, TensorRT, CoreML, TFLite, etc.)
* Actively maintained, lightweight, and beginner-friendly

**Compared to YOLOv5:**

* Cleaner codebase
* PyTorch-only (no custom framework)
* Better modularity and export options

**5. 🧰 Installation and Setup**

pip install ultralytics

code :

from ultralytics import YOLO

model = YOLO('yolov8n.pt') # load nano model

**6. 🧠 YOLOv8 Model Types**

| **Model** | **Size** | **Speed** | **Accuracy** | **Use Case** |
| --- | --- | --- | --- | --- |
| yolov8n | Nano | ⚡⚡⚡⚡⚡ | ⭐⭐ | Edge Devices |
| yolov8s | Small | ⚡⚡⚡⚡ | ⭐⭐⭐ | Fast Inference |
| yolov8m | Medium | ⚡⚡⚡ | ⭐⭐⭐⭐ | Balanced |
| yolov8l | Large | ⚡⚡ | ⭐⭐⭐⭐⭐ | Accuracy |
| yolov8x | X-Large | ⚡ | ⭐⭐⭐⭐⭐⭐ | Best accuracy |

**7. 📂 Dataset Preparation**

Use **Roboflow**, **LabelImg**, or any annotation tool.

* Format: YOLO format (.txt file with class\_id x\_center y\_center width height)
* Structure:

datasets/

├── images/

│ ├── train/

│ ├── val/

├── labels/

│ ├── train/

│ ├── val/

**8. 🏋️ Training a Custom YOLOv8 Model**

**data.yaml file**

yaml

path: datasets/

train: images/train

val: images/val

names:

0: person

1: car

2: bag

**Command**

yolo detect train data=data.yaml model=yolov8n.pt epochs=50 imgsz=640

**9. 📸 Inference**

yolo detect predict model=weights/best.pt source=test.jpg

Or using Python:

model = YOLO("weights/best.pt")

results = model("test.jpg", save=True)

**10. 📊 Evaluation Metrics**

* mAP@0.5: mean average precision at 0.5 IoU
* Precision
* Recall
* FPS (Frames per second)

**11. ⚖️ Comparison with Alternatives**

| **Metric** | **YOLOv8** | **Faster R-CNN** | **SSD** | **EfficientDet** |
| --- | --- | --- | --- | --- |
| Speed | 🔥 Fast | ❌ Slow | ⚡ Medium | ⚡ Medium |
| Accuracy | ✅ High | ✅ High | ✅ Moderate | ✅ High |
| Ease of Use | ✅ Easy | ❌ Complex | ⚠️ Moderate | ⚠️ Moderate |
| Deployment | ✅ Easy | ⚠️ Hard | ✅ Easy | ✅ Easy |
| Codebase | Modern | Outdated | Legacy | TensorFlow-based |

**Why YOLOv8?**  
YOLOv8 offers the best mix of:

* Speed (real-time)
* Accuracy
* Simplicity
* Deployment readiness

**12. 🏭 Real-World Use Cases**

* Vehicle detection
* Industrial safety
* Retail analytics
* Face mask detection
* Wildlife monitoring

**13. 🛠️ Challenges & Best Practices**

**Common Challenges:**

* Small dataset → underfitting
* Imbalanced class data
* Poor labeling

**Best Practices:**

* Use Roboflow for preprocessing
* Augment your data
* Start with yolov8n.pt → iterate
* Use validation to avoid overfitting