YOLO

YOLO (You Only Look Once) - Real-Time Object Detection 🚀

YOLO is a **deep learning-based object detection model** that processes an image in a **single forward pass**, making it **fast and efficient** compared to older models like R-CNN, Fast R-CNN, and Faster R-CNN.

1️⃣ What is YOLO?

- **YOLO (You Only Look Once)** is an object detection model that detects multiple objects in an image **at once** instead of **sliding a window** over the image like older methods.
- Unlike traditional methods (**R-CNN, Faster R-CNN**), YOLO is a **single-shot detector (SSD)**, meaning:
  - It predicts **bounding boxes + class labels** in one pass.
  - It treats object detection as a **regression problem**.

2️⃣ Evolution of YOLO Models

YOLO Version	Key Features
YOLOv1 (2016)	First YOLO model, fast but less accurate
YOLOv2 (2017)	Better accuracy, supports multi-scale detection
YOLOv3 (2018)	Uses Darknet-53, improved multi-scale detection
YOLOv4 (2020)	CSPDarknet backbone, faster and more accurate
YOLOv5 (2020, unofficial)	Faster, more optimized, PyTorch-based
YOLOv7 (2022)	Most optimized real-time detection
YOLOv8 (2023, Ultralytics)	Latest version with better accuracy & ease of use

🔹 **YOLOv5 & YOLOv8** are the most commonly used today.

3️⃣ How YOLO Works

🔹 YOLO **divides an image into an S×S grid**.
🔹 Each grid cell **predicts bounding boxes + confidence scores**.
🔹 Uses **Non-Maximum Suppression (NMS)** to remove duplicate boxes.

💡 **Example: A 3x3 grid on a 640x640 image → Each cell predicts objects inside it.**

YOLO Outputs

Each grid cell predicts:
1️⃣ Bounding box coordinates: (x, y, w, h)
2️⃣ Objectness score (probability that an object is present)
3️⃣ Class probabilities (e.g., "car", "dog", "person")

4️⃣ Implementing YOLOv5 in Python

Step 1: Install YOLOv5

```
git clone https://github.com/ultralytics/yolov5
cd yolov5
pip install -r requirements.txt
```

Step 2: Load YOLOv5 Pretrained Model

```
import torch
from PIL import Image
import cv2

# Load YOLOv5 model (pretrained on COCO dataset)
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Load an image
img = Image.open("image.jpg")

# Run inference
results = model(img)

# Show results
results.show()
```

Step 3: Process Video in Real-Time

```
cap = cv2.VideoCapture(0)  # Webcam input
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    results = model(frame)
    results.render()  # Draw bounding boxes
    cv2.imshow('YOLOv5', frame)

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

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

5️⃣ Custom Training on Your Own Dataset

To train YOLOv5 on a custom dataset (e.g., detecting cats and dogs):

Step 1: Install Dependencies

```
pip install -U ultralytics
```

Step 2: Prepare Dataset

- Label your images using **LabelImg**.
- Format dataset in YOLO format:

```
dataset/
├── images/
│   ├── train/ (Training images)
│   ├── val/ (Validation images)
├── labels/
│   ├── train/ (Bounding box labels)
│   ├── val/ (Validation labels)
```

- Create a `data.yaml` file:

```
train: ./dataset/images/train
val: ./dataset/images/val
nc: 2  # Number of classes
names: ["cat", "dog"]
```

Step 3: Train YOLOv5

```
python train.py --img 640 --batch 16 --epochs 100 --data data.yaml --weights yolov5s.pt --device 0
```

Step 4: Test the Trained Model

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path='runs/train/exp/weights/best.pt')
results = model("test.jpg")
results.show()
```

6️⃣ Differences Between YOLO and Other Object Detection Models

Model	Speed (FPS)	Accuracy	Real-Time Usage
YOLOv5	**Faster**	**Good**	**Yes** ✅
Faster R-CNN	**Slower**	**High**	X No
SSD (Single Shot MultiBox)	**Fast**	**Medium**	✅ Yes
RetinaNet	**Medium**	**High**	X No

🔹 **YOLO is the best choice for real-time detection** (CCTV, self-driving cars, drones).

7️⃣ Applications of YOLO

✅ **Self-Driving Cars** 🚗
✅ **Security & Surveillance (CCTV)** 📷
✅ **Autonomous Drones** 🚁
✅ **Medical Image Analysis (X-ray, MRI)** 🏥
✅ **Retail (Checkout-free stores like Amazon Go)** 🏪

8️⃣ Final Thoughts

- **YOLO is the fastest real-time object detection model** 🏆.
- **YOLOv5 & YOLOv8** are the most widely used today.
- Can be **fine-tuned on custom datasets**.
- Perfect for **real-time applications like security, robotics, and self-driving cars**.

🚀 **To train YOLO on your dataset?**

Training YOLOv5 on Your Custom Dataset 🚀

These are the **steps** to train **YOLOv5** on your **custom dataset** using PyTorch.

📌 Step 1: Install Dependencies

Run the following commands in **Google Colab** or your local system:

```
# Clone the YOLOv5 repository
git clone https://github.com/ultralytics/yolov5.git
cd yolov5

# Install dependencies
pip install -r requirements.txt
pip install ultralytics
```

📌 Step 2: Prepare Your Dataset

💡 YOLO Dataset Format

- Each image must have a corresponding `.txt` file with bounding box information.

- Each line in the `.txt` file represents an **object**, formatted as:

  ```
  class_id center_x center_y width height
  ```

  - `class_id`: The class index (e.g., 0 for cats, 1 for dogs).
  - `center_x`, `center_y`, `width`, `height`: **Normalized values** (between `0-1`).

🔹 Dataset Folder Structure

```
dataset/
├── images/
│   ├── train/  # Training images
│   ├── val/    # Validation images
│   ├── test/   # Testing images (optional)
├── labels/
│   ├── train/  # Bounding box annotations for training
│   ├── val/    # Bounding box annotations for validation
│   ├── test/   # Bounding box annotations for testing (optional)
└── data.yaml   # Configuration file
```

🔹 Example: Annotation File (`dog.txt`)

For an image of a **dog (class 0)** with a bounding box:

```
0 0.45 0.60 0.30 0.50
```

This means:

- **Class 0** (dog)
- Bounding box at (0.45, 0.60) with width `0.30` and height `0.50`.

🔹 Label Your Images

Use **LabelImg** to annotate images and save labels in YOLO format:

```
pip install labelImg
labelImg
```

📌 Step 3: Create `data.yaml` Configuration File

Create a `data.yaml` file inside the dataset directory:

```
train: ./dataset/images/train
val: ./dataset/images/val
test: ./dataset/images/test  # Optional
nc: 2  # Number of classes
names: ["cat", "dog"]  # Class names
```

📌 Step 4: Train YOLOv5 on Custom Dataset

Run the training script:

```
python train.py --img 640 --batch 16 --epochs 100 --data dataset/data.yaml --weights yolov5s.pt --device 0
```

🔹 Explanation

- `--img 640` → Image size **640x640**
- `--batch 16` → Batch size **16**
- `--epochs 100` → Train for **100 epochs**
- `--data dataset/data.yaml` → Use the dataset
- `--weights yolov5s.pt` → Use a **pretrained model**

- `--device 0` → Use **GPU (CUDA 0)**

📌 Step 5: Evaluate the Model

Run evaluation on the validation set:

```
python val.py --data dataset/data.yaml --weights runs/train/exp/weights/best.pt --img 640
```

📌 Step 6: Test on New Images

Use the trained model to **detect objects** in images:

```
import torch
from PIL import Image

# Load trained YOLO model
model = torch.hub.load('ultralytics/yolov5', 'custom', path='runs/train/exp/weights/best.pt')

# Load and run inference on an image
img = Image.open("test.jpg")
results = model(img)

# Show detected objects
results.show()
```

📌 Step 7: Deploy YOLO in Real-Time (Webcam)

```
import cv2

# Load YOLO model
model = torch.hub.load('ultralytics/yolov5', 'custom', path='runs/train/exp/weights/best.pt')

cap = cv2.VideoCapture(0)  # Open webcam

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

    results = model(frame)  # Run YOLO detection
    results.render()  # Draw bounding boxes

    cv2.imshow('YOLOv5 Detection', frame)

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

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

🚀 Final Notes

✅ **YOLOv5 is fast and easy to train on custom datasets**
✅ Can be used for **real-time applications like surveillance, self-driving cars, and medical imaging**
✅ Can be fine-tuned for **small, medium, or large datasets**

Setting Up LabelImg for YOLO Dataset Annotation 🏷️

To train YOLO on your custom dataset, you need to annotate images and create `.txt` files with bounding box coordinates. **LabelImg** is a great tool for this.

📌 Step 1: Install LabelImg

Run the following command in **Windows/Linux/Mac**:

```
pip install labelImg
labelImg
```

📌 Step 2: Open LabelImg

After installation, run:

```
labelImg
```

- The interface will open where you can load and annotate images.

📌 Step 3: Change the Save Format to YOLO

1️⃣ Click **"View"** > Select **"YOLO"**
2️⃣ Click **"Open Dir"** and select the folder where your images are stored.
3️⃣ Click **"Change Save Dir"** and choose the directory to save label files.

📌 Step 4: Annotate Images

1️⃣ Select an image and **draw bounding boxes** around objects.
2️⃣ Enter the **class name** (e.g., "cat", "dog").
3️⃣ Save annotations. It will create `.txt` files in **YOLO format**.

📌 Step 5: Verify YOLO Annotations

Each **image must have a corresponding `.txt` file** with the same name.
Example for `dog.jpg` → `dog.txt`:

```
0 0.45 0.60 0.30 0.50
```

- `0` → Class ID (dog)
- `0.45` → Bounding box center X (normalized)
- `0.60` → Bounding box center Y (normalized)
- `0.30` → Width
- `0.50` → Height

📌 Step 6: Organize Your Dataset

Structure should be:

```
dataset/
├── images/
│   ├── train/
│   ├── val/
│   ├── test/  # Optional
├── labels/
│   ├── train/
│   ├── val/
│   ├── test/  # Optional
└── data.yaml  # Configuration file
```

📌 Step 7: Train YOLOv5

Once annotations are ready, run:

```
python train.py --img 640 --batch 16 --epochs 100 --data dataset/data.yaml --weights yolov5s.pt --device 0
```