**Parking Space Detection**

train YOLO to detect not only vehicles but also parking spaces in real-time.

**How to implement YOLO:**

* Load a pre-trained YOLO model (YOLOv5 or YOLOv8 is recommended for ease of use).
* Detect objects in each frame.
* For each detected object, check if it is a "car" and mark the corresponding parking slot accordingly.

Let's break down the process into actionable steps to help you implement YOLO for parking space detection in your autonomous driving project:

**Step 1: Choose and Set Up YOLO**

We will use YOLOv5 or YOLOv8 due to their ease of use, fast inference time, and good accuracy. You can follow these steps to set up the YOLO model in a Python environment:

**1.1 Install YOLO**

First, you'll need to install the necessary libraries and the YOLO model. YOLOv5 is easier to integrate and has an active community. To set up YOLOv5, follow these steps:

1. **Clone the YOLOv5 repository**:
2. git clone <https://github.com/ultralytics/yolov5.git>
3. cd yolov5
4. **Install dependencies**: You can install the necessary dependencies using pip:
5. pip install -r requirements.txt
6. **Test the YOLO setup**: Run a test to ensure everything is working. This will download a pre-trained model and run inference on some sample images.
7. python detect.py --source data/images --weights yolov5s.pt --conf 0.25 --view-img

If this runs without errors, you have YOLOv5 set up and ready to go.

**1.2 Set Up Python Environment**

If you're using **YOLOv8**, you can directly install it using pip:

pip install ultralytics

Now, YOLOv8 can be used like this:

import torch

# Load YOLOv5 model

model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

# OR for YOLOv8

from ultralytics import YOLO

model = YOLO('yolov8s.pt')

**Step 2: Collect/Label Data**

For parking spot detection, you'll need a labeled dataset of parking spaces and parked cars. There are a couple of approaches:

**2.1 Use Existing Datasets**

You can use existing datasets such as:

* **PKLot dataset**: This contains images of parking lots with annotations of parked cars.
* **CNRPark-EXT dataset**: Another popular dataset for parking spot detection.

You can download these datasets and load them into your YOLO training environment. These datasets contain labeled parking spots and occupancy statuses.

**2.2 Create Your Own Dataset (Optional)**

If you want more specific data, you can capture parking lot images and label them using an annotation tool like **LabelImg**.

* **LabelImg**: This tool allows you to manually annotate your dataset with bounding boxes around cars and empty spots. After labeling, you'll save the annotations in YOLO format (txt files).

You will need to create a dataset with labels for:

* **Cars** (for detecting occupied spaces).
* **Empty spaces** (for unoccupied parking slots).
* Optionally: **People or other objects** if you want to detect temporary obstructions.

**Step 3: Train or Fine-Tune the YOLO Model**

Once you have your dataset, you can train or fine-tune the model to detect parking spaces.

**3.1 Organize the Dataset**

Organize the dataset folder as follows (YOLO format):

dataset/

images/

train/

val/

labels/

train/

val/

* **Images**: Place your training and validation images here.
* **Labels**: Place the corresponding label files (in YOLO format) for each image here.

**3.2 Train YOLOv5 or YOLOv8**

Now that your dataset is ready, you can train YOLO. For example, to train with YOLOv5, use the following command:

python train.py --img 640 --batch 16 --epochs 100 --data your\_dataset.yaml --weights yolov5s.pt --cache

* -img 640: Image size.
* -batch 16: Batch size.
* -epochs 100: Number of training epochs.
* -data: Path to your dataset config file.
* -weights: Pre-trained model to start training with.
* -cache: Cache images for faster training.

The your\_dataset.yaml file should define the paths to your dataset and the number of classes (e.g., cars, empty spots).

For YOLOv8, the command is similar:

model.train(data='your\_dataset.yaml', epochs=100, imgsz=640)

**3.3 Monitor Training**

Training will take some time, depending on your hardware. You can monitor the performance using metrics like mAP (mean Average Precision) and inference speed.

**Step 4: Optimize for Deployment**

Once the model is trained, you will need to optimize it for real-time inference.

**4.1 Model Optimization Techniques**

* **Quantization**: Reduces the size of the model and increases inference speed.
* **Pruning**: Removes unnecessary parts of the model to make it faster.
* **TensorRT or ONNX**: Convert the model to TensorRT or ONNX format for optimized inference on GPUs.

For example, converting to ONNX:

python export.py --weights best.pt --img 640 --batch 1 --device 0 --simplify --include onnx

**4.2 Hardware Acceleration**

Deploy the model on a device with GPU or TPU support. Use frameworks like TensorRT (for NVIDIA GPUs) to achieve low-latency inference.

**Step 5: Test in a Real Environment**

Finally, you will need to test the trained and optimized model in a real parking lot environment.

**5.1 Integrate with the Camera Feed**

Using a camera mounted on the autonomous vehicle, you can pass the video feed to your model and get real-time parking spot detection.

import cv2

# Open the video stream

cap = cv2.VideoCapture(0) # Use the correct camera index for your setup

while cap.isOpened():

ret, frame = cap.read()

if ret:

# Run inference on the frame

results = model(frame)

# Display results

cv2.imshow('Parking Spot Detection', results.imgs[0])

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

break

cap.release()

cv2.destroyAllWindows()

**5.2 Evaluate Model Performance**

* **Detection Accuracy**: Make sure the model accurately detects parked cars and empty spots.
* **Speed**: Measure the inference speed to ensure it’s real-time.
* **Edge Cases**: Test for various conditions (weather, lighting) and ensure the model works consistently.

**Summary of Steps:**

1. Set up YOLO (v5 or v8) in Python.
2. Collect or label parking space data using existing datasets or manually label your own.
3. Train or fine-tune the YOLO model for detecting parked cars and empty spots.
4. Optimize the trained model using techniques like quantization and deploy it on hardware like GPUs or TPUs.
5. Test the model with a real-time video feed in a parking lot to evaluate performance and detection accuracy.

This approach will give you a robust system for detecting parking spaces in autonomous vehicles. Let me know if you need help with any specific part of the process!

**Using yolo v3**

Based on the content from the paper "Vehicle and Parking Space Detection Based on Improved YOLO Network Model" that you've uploaded, I'll guide you step-by-step to better understand and implement it for your task.

### Step-by-Step Understanding and Implementation

#### 1. \*\*Understanding YOLO and the Improvements in the Paper\*\*

- \*\*YOLO v3 Overview\*\*: YOLO (You Only Look Once) is a real-time object detection algorithm that processes an entire image at once, predicting bounding boxes and class probabilities for objects within those boxes. YOLOv3 uses a deep convolutional network (Darknet53) and is known for its speed and decent accuracy.

- \*\*Improvements in the Paper\*\*: The paper focuses on improving YOLO v3 by:

- Adding \*\*residual blocks\*\* to extract deeper vehicle parking space features.

- Using \*\*four different scale feature maps\*\* to capture more granular features.

- The improved model achieves better accuracy and detection speed, especially in parking space scenarios, by reducing the false positive and false negative rates.

#### 2. \*\*YOLO Network Structure\*\*

- \*\*Darknet53\*\*: YOLOv3 is built on this backbone. It consists of residual blocks, which help make the network deeper without suffering from vanishing gradient problems. The residual structure allows the network to handle complex tasks like parking space detection by learning better features.

The residual blocks enable YOLOv3 to capture fine details, essential for distinguishing between occupied and unoccupied parking spaces. The improvements involve using deeper residual layers and multiple feature maps to handle different object scales (cars, trucks, parking spots).

#### 3. \*\*Dataset for Training\*\*

- The paper uses \*\*PASCAL VOC\*\*, \*\*COCO\*\*, and \*\*PKLot\*\* datasets. The PKLot dataset is specifically used for parking space detection and includes images from different weather conditions (sunny, rainy, cloudy) with labeled parking spots as occupied or empty.

- \*\*PKLot dataset\*\*: This is ideal for your parking space detection task, as it provides real-world images with various lighting and weather conditions, making your model more robust.

### Implementation Steps

#### 1. \*\*Set Up YOLO v3\*\*

Start by cloning the YOLOv3 repository and installing dependencies:

```bash

git clone https://github.com/ultralytics/yolov3

cd yolov3

pip install -r requirements.txt

```

Alternatively, you can use \*\*YOLOv5 or YOLOv8\*\* if you're more comfortable with these versions. Follow a similar process as mentioned earlier for YOLOv5 or YOLOv8.

#### 2. \*\*Data Preprocessing\*\*

- Download the \*\*PKLot dataset\*\* and other relevant datasets like PASCAL VOC and COCO if needed.

- Convert the PKLot dataset annotations (if needed) into YOLO format:

- \*\*Label format\*\*: YOLO uses bounding boxes defined by `(class, x\_center, y\_center, width, height)` relative to the image size.

- You can automate the conversion from XML annotations (PASCAL VOC format) to YOLO format using Python scripts.

#### 3. \*\*Model Training\*\*

Fine-tune a pre-trained YOLO model using the PKLot dataset:

```bash

python train.py --img 640 --batch 16 --epochs 50 --data pklot.yaml --weights yolov3.pt

```

- \*\*Pre-training\*\*: Use pre-trained weights (like `yolov3.pt`), as the model has already learned to detect common objects such as cars.

- \*\*Fine-tuning\*\*: Since your task focuses on parking space detection, fine-tune the model on the PKLot dataset to detect parking spots and cars specifically.

#### 4. \*\*K-means Clustering for Anchor Boxes\*\*

The paper mentions using \*\*K-means clustering\*\* to improve the selection of anchor boxes, which are essential for YOLO to propose bounding boxes. This is especially important for parking space detection, as cars and parking spaces may vary in size.

- You can use the `kmeans.py` script (available in YOLO repositories) to perform K-means clustering on your dataset to obtain optimized anchor boxes.

#### 5. \*\*Model Optimization and Deployment\*\*

- \*\*Optimize the Model\*\*: Once trained, optimize your model for real-time deployment using techniques like quantization and model pruning. Tools like \*\*TensorRT\*\* or \*\*OpenVINO\*\* can help optimize YOLO models for fast inference.

- Convert the trained model to ONNX format for hardware-accelerated deployment.

- \*\*Deploying on Edge Devices\*\*: If you’re using low-power devices like Raspberry Pi, the paper suggests reducing the network structure (lighter versions of YOLO) to ensure it runs efficiently on such devices.

#### 6. \*\*Real-Time Testing and Evaluation\*\*

After training and optimizing, you need to test the model in real-time using a camera feed. Integrate the camera into the YOLO detection pipeline, as shown earlier, and evaluate:

- \*\*Accuracy\*\*: Ensure that the model correctly detects occupied and unoccupied spaces.

- \*\*Speed\*\*: Measure the inference speed in real-time environments to confirm it's suitable for autonomous vehicles.

### Summary of Improvements from the Paper:

- \*\*Residual Blocks\*\*: Improved feature extraction for better parking spot and vehicle detection.

- \*\*Clustering Anchor Boxes\*\*: Customizing anchor boxes specifically for parking spaces and vehicles improves detection accuracy.

- \*\*Multiple Scale Feature Maps\*\*: Enhances detection for objects of different sizes, crucial for detecting both small cars and large vehicles.

By following these steps, you'll be able to implement the improved YOLO model described in the paper for parking space detection in your autonomous driving project. Let me know if you need help with specific steps, such as dataset preparation, training, or optimization!

Certainly! Here's a **simplified and concise breakdown of how the model in the paper** performs parking space and vehicle detection using the improved YOLOv3:

### 1. \*\*Input and Preprocessing\*\*

- The model takes an input image (or video frame) of a parking lot.

- The image is processed through the YOLOv3 network, which uses a \*\*Convolutional Neural Network (CNN)\*\* to extract important features (like edges, shapes, and textures).

### 2. \*\*Object Detection (YOLOv3)\*\*

- \*\*YOLOv3 Network\*\*:

- YOLOv3 detects objects (cars, trucks, buses) by dividing the image into a grid and predicting bounding boxes for objects within the grid.

- The network classifies objects into categories (e.g., car, truck, bus) and assigns confidence scores.

- \*\*Residual Blocks\*\* (improved in the paper) are used to extract deeper features, making detection more accurate.

### 3. \*\*Parking Space Detection\*\*

- The model detects \*\*vehicles\*\* in the parking spaces. It identifies whether a parking space is occupied or empty by analyzing the presence of vehicles in the frame.

- The \*\*PKLot dataset\*\* is used for training the model to recognize parking spaces under different weather conditions (sunny, rainy, cloudy).

### 4. \*\*Anchor Boxes with K-means Clustering\*\*

- The model uses \*\*anchor boxes\*\* (pre-defined bounding boxes) to propose candidate locations for vehicles and parking spaces.

- \*\*K-means clustering\*\* is applied to customize anchor boxes for the parking spaces and vehicles, improving accuracy.

### 5. \*\*Multi-Scale Feature Detection\*\*

- The improved model uses \*\*four different scale feature maps\*\*, which allows it to detect objects of varying sizes (small cars, large trucks) and differentiate between them more accurately.

### 6. \*\*Prediction\*\*

- The network predicts:

- \*\*Bounding boxes\*\*: The exact location of each vehicle and parking space in the image.

- \*\*Class labels\*\*: Whether the object is a car, truck, bus, or an empty parking space.

- \*\*Confidence scores\*\*: How confident the model is in its detection.

### 7. \*\*Real-Time Performance\*\*

- The model is optimized for \*\*real-time detection\*\* with fast inference speeds (milliseconds per detection), making it suitable for real-time applications like autonomous parking.

### 8. \*\*Output\*\*

- The output is an image (or video frame) with \*\*bounding boxes\*\* around vehicles and parking spaces. The model also labels each space as occupied or empty, helping manage parking efficiently.

### Overall Process:

1. Input image →

2. YOLOv3 processes the image →

3. Detects vehicles and parking spaces →

4. Classifies parking spaces as \*\*occupied\*\* or \*\*empty\*\* →

5. Outputs real-time results with bounding boxes and labels.

This process is designed to be accurate and fast, making it ideal for parking management in autonomous vehicles.