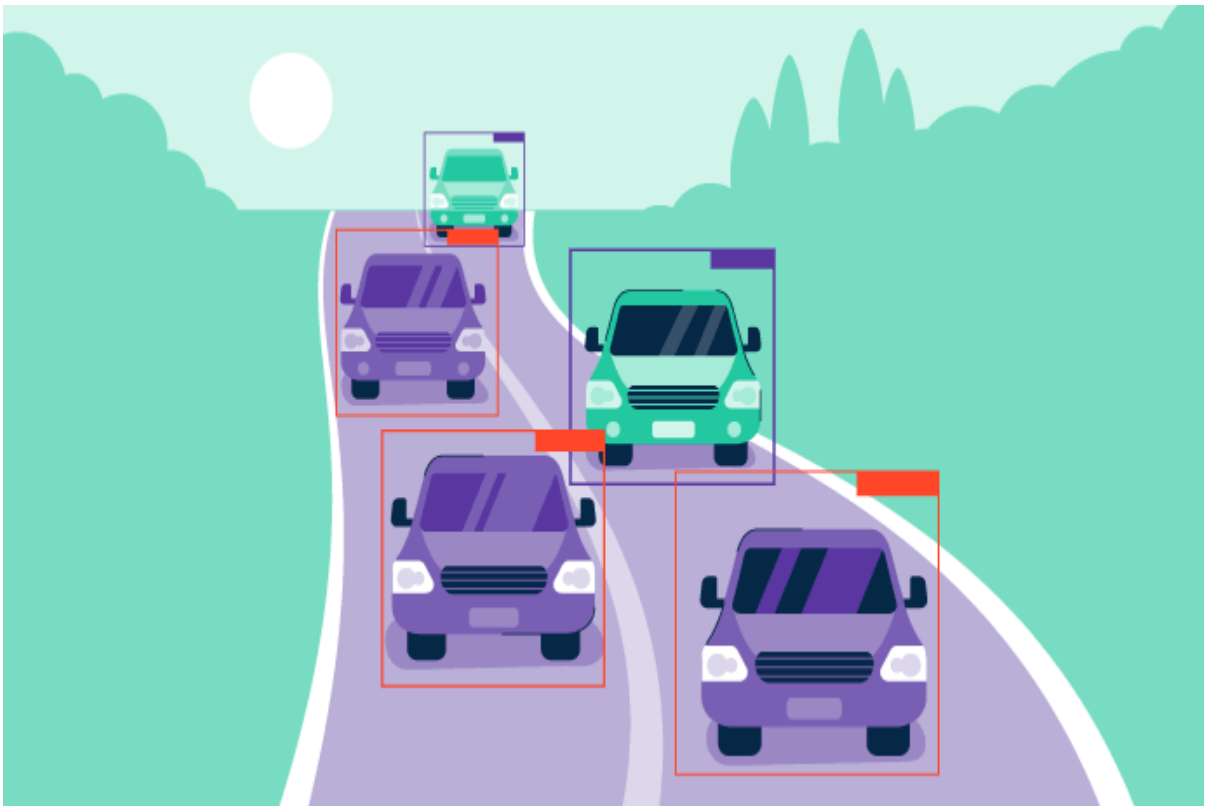
Object Detection



What Is Object Detection? 🤔

Humans are naturally wired to spot objects around us, thanks to our sophisticated visual system 👀. But now, machines are catching up with us! 🚀

With the rapid advancements in technologies like artificial intelligence (AI), Internet of Things (IoT), and quantum computing, scientists are teaching computers to mimic our cognitive abilities 🧠. This allows machines to help us navigate the visual world—**with a little extra help from their digital brains 🖥️.

Object detection is a computer vision technique that uses artificial neural networks to classify objects within images. It has endless applications in autonomous vehicles, security surveillance, retail analytics, augmented reality, medical imaging, and robotics 🤖.

This process leverages advanced image recognition software, which uses machine learning algorithms and neural networks to accurately identify and classify objects in images 🎯. It's a game-changer for industries looking to automate and optimize operations, making tasks faster, smarter, and more accurate! 💡

Introducing YOLOv8 🚀

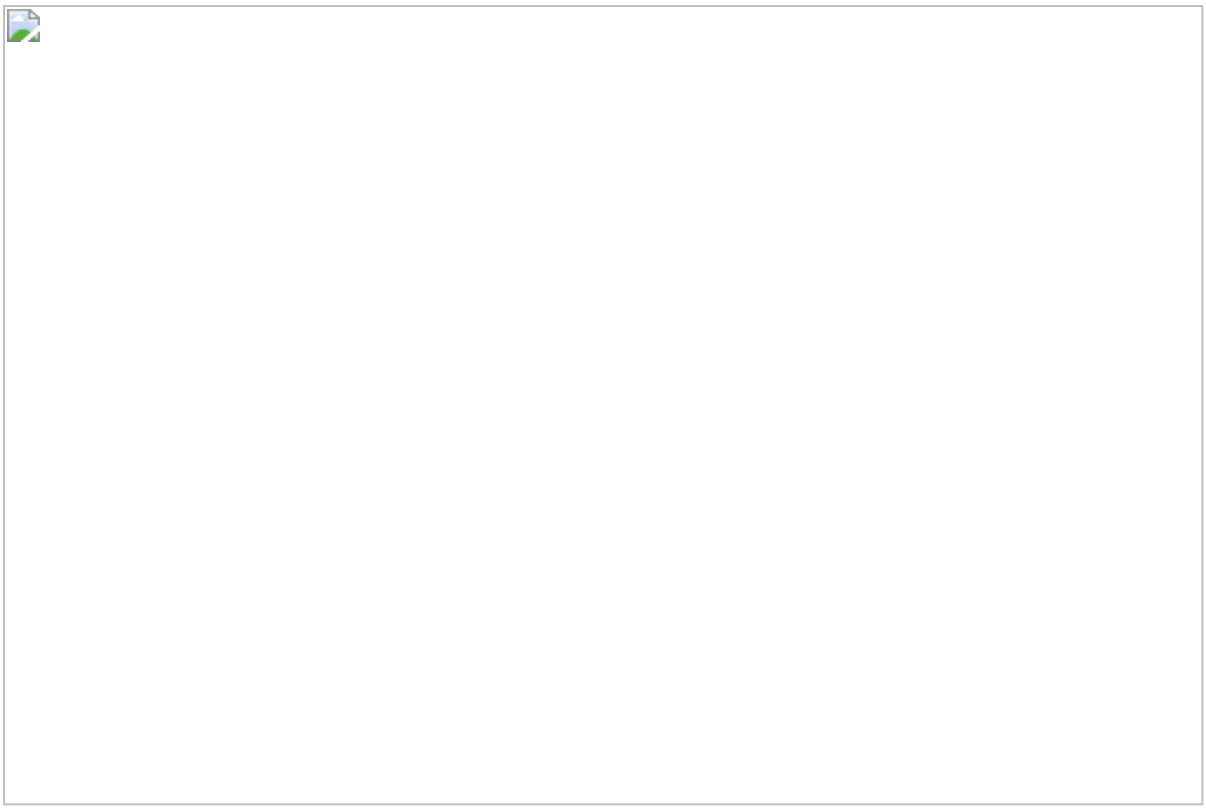
YOLO (You Only Look Once) is a cutting-edge, single-stage object detection framework, built for real-time applications ⚡. It's fast and highly efficient, making it perfect for industries where speed and accuracy are key 🔑.

YOLO is built on Convolutional Neural Networks (CNNs) and trained on large-scale datasets like ImageNet, making it a top contender for real-time applications across various fields 🌍. The model processes images at a mind-blowing speed of up to 45 frames per second 🎬, making it one of the fastest models for object detection! 🚀

From YOLOv1 to YOLOv8, each version has seen incredible improvements in performance, usability, and speed. The latest iteration, YOLOv8, is designed to take object detection to the next level with unmatched accuracy, speed, and flexibility. Let's look at some of the key features of YOLOv8 that make it stand out:

- Optimized Speed & Accuracy ⚡: YOLOv8 delivers lightning-fast processing without sacrificing accuracy, ensuring high-quality detections in real-time environments.
- Enhanced Architecture 🏗️: The YOLOv8 model is optimized to reduce redundancy while boosting overall efficiency for faster and more reliable results.
- Lightweight Desig ⚖️: YOLOv8 is more compact, requiring fewer parameters but still achieving strong detection results, especially for small objects.
- Hardware Efficiency 💻: Optimized for a wide range of hardware platforms, YOLOv8 performs effectively on edge devices and high-end systems alike.
- Improved Small Object Detection 🔍: YOLOv8 excels at detecting small or distant objects, making it perfect for complex and cluttered environments.

YOLOv8 has already been trained on benchmark datasets like MS COCO and VOC, setting new standards in real-time object detection. With various configurations available, it is adaptable to different use cases—from lightweight models** to more powerful versions for specialized applications 🌐.



Timeline of YOLO versions from 2015 to 2024.

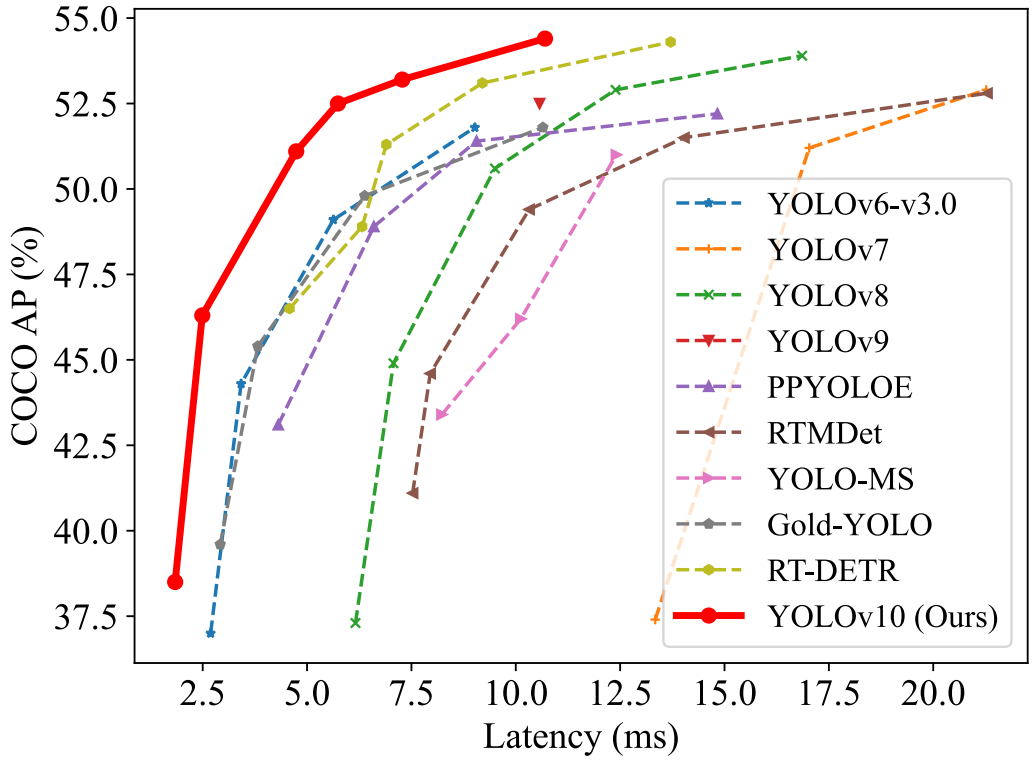
Why YOLOv8? 🤔

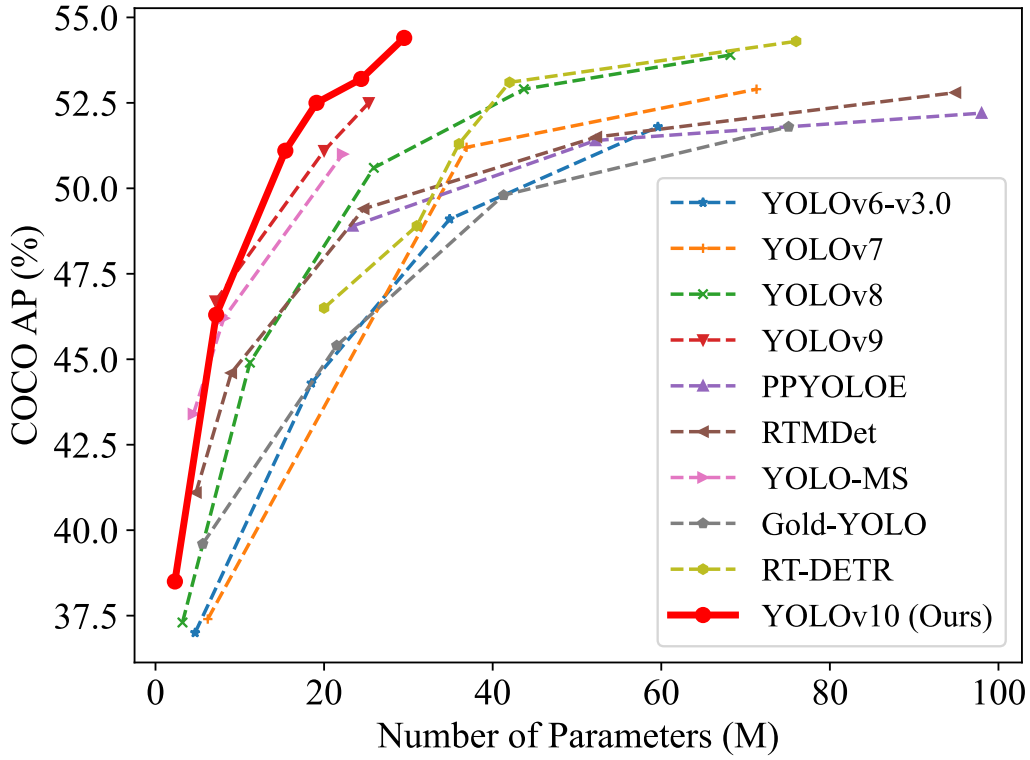
YOLOv8 is a game-changing tool for developers and researchers working with object detection 🔧. But with so many options available, why choose YOLOv8 for your next project?

Here are 5 key reason:

1. High Accuracy 🎯: YOLOv8 consistently outperforms other models on popular benchmarks like COCO and Roboflow 100, guaranteeing precise object detection.
2. Speed ⚡: As a single-stage detector, YOLOv8 is faster than multi-stage models, which makes it ideal for real-time applications like self-driving cars 🚙 and drone surveillance 🛸.
3. Ease of Use 🛠️: YOLOv8 comes with a well-documented Python API and a user-friendly command-line interface (CLI), making it accessible for developers at all levels.
4. Versatility 🔁: YOLOv8 can be used for tasks like object detection, **instance segmentation, and image classification, making it a versatile tool for multiple applications.
5. Large Community 🌐: With a thriving and active developer communit, finding resources and support for YOLOv8 is a breeze!

As YOLOv8 continues to evolve, it's clear that its impact will be monumental across industries. It will revolutionize how we interact with the world, from autonomous vehicles to industrial automation—and beyond 🌟.

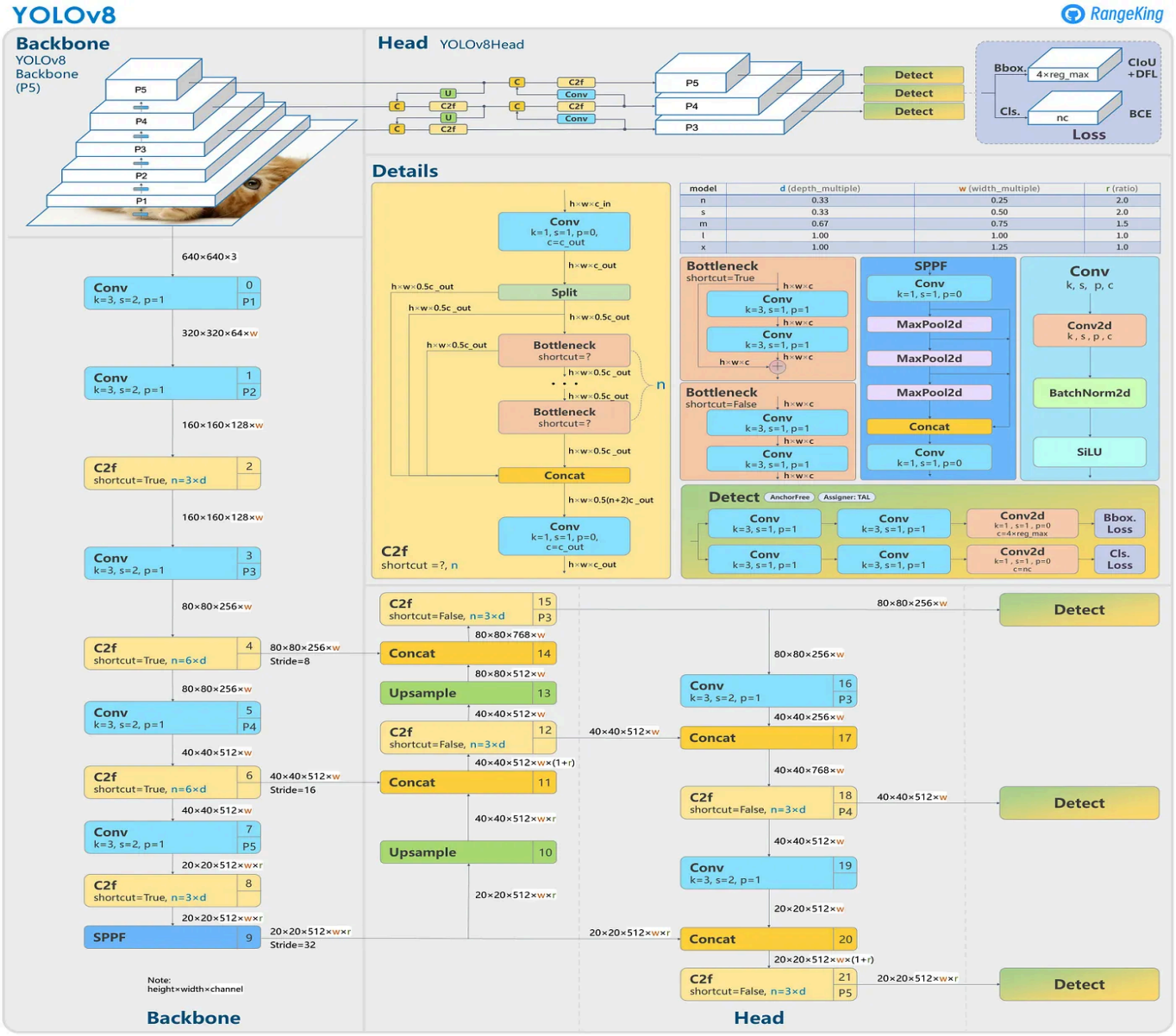




Comparing YOLOv8 with others in terms of latency-accuracy (left) and size-accuracy (right) trade-offs.

- YOLOv8 Architecture

YOLOv8 uses a Convolutional Neural Network (CNN), broken down into two main parts: the backbone and the head.



model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

- Backbone: Based on CSPDarknet53 architecture with 53 convolutional layers, it utilizes cross-stage partial connections to improve data flow.
- Head: This part consists of multiple convolutional and fully connected layers, responsible for predicting bounding boxes, objectness scores, and **class probabilities.
- Self-attention Mechanism: The head uses this mechanism to focus on key areas in the image, improving prediction relevance.
- Multi-scaled Detection: With a feature pyramid network, YOLOv8 can detect objects of varying sizes across different layers, from tiny to large 🌈.

In a world where technology is evolving at breakneck speed, YOLOv8 is at the forefront, ready to change the way we see and interact with the world around us 🌟. 🚀

Importing Libraries

```
!pip install ultralytics
```

Show hidden output

```
import os
import cv2
import glob
import yaml
import random
import pathlib
import numpy as np
import pandas as pd
from PIL import Image
import seaborn as sns
from tqdm import tqdm
from ultralytics import YOLO
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tqdm.notebook import trange, tqdm
from IPython.display import Image, Video
from IPython.display import Image, display
from sklearn.metrics import precision_score, recall_score
!wandb disabled
import warnings
warnings.filterwarnings('ignore')
```

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/setti
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/
W&B disabled.
```

Discription File For The Custom Dataset

```
train_path = '/kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images'
val_path = '/kaggle/input/vehicledetection/VehiclesDetectionDataset/valid/images'
test_path = '/kaggle/input/vehicledetection/VehiclesDetectionDataset/test/images'

yaml_content = f"""
test: {test_path}
train: {train_path}
val: {val_path}

nc: 5  # Number of classes
names: ['Ambulance', 'Bus', 'Car', 'Motorcycle', 'Truck']
"""
output_path = 'data.yaml'
with open(output_path, 'w') as file:
    file.write(yaml_content)
color_code = "\033[38;2;163;196;243m"
reset_code = "\033[0m"
print(f"{color_code}YAML file has been saved to {output_path}{reset_code}")
```

```
YAML file has been saved to data.yaml
```

```
output_path = 'data.yaml'
with open(output_path, 'r') as file:
    data = yaml.safe_load(file)
class_names = data['names']
print(f"{color_code}Class Names:{reset_code} {class_names}")
```

```
Class Names: ['Ambulance', 'Bus', 'Car', 'Motorcycle', 'Truck']
```

```
import os
import random
from PIL import Image
images_folder = '/kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images'
image_files = [f for f in os.listdir(images_folder) if f.endswith(('.jpg', '.png', '.jpeg
random_image = random.choice(image_files)
random_image_path = os.path.join(images_folder, random_image)
image = Image.open(random_image_path)
image_size = image.size
image_mode = image.mode
num_channels = image.layers if hasattr(image, 'layers') else len(image.getbands())
print(f"Random Image: {random_image}")
print(f"Image Size: {image_size}")
print(f"Image Mode (Channels): {image_mode}")
print(f"Number of Channels: {num_channels}")
```

```
Random Image: 076317ab4c925aa9_jpg.rf.2eab8692c9e3035eb3c2dfcd2a07e74c.jpg
Image Size: (416, 416)
Image Mode (Channels): RGB
Number of Channels: 3
```

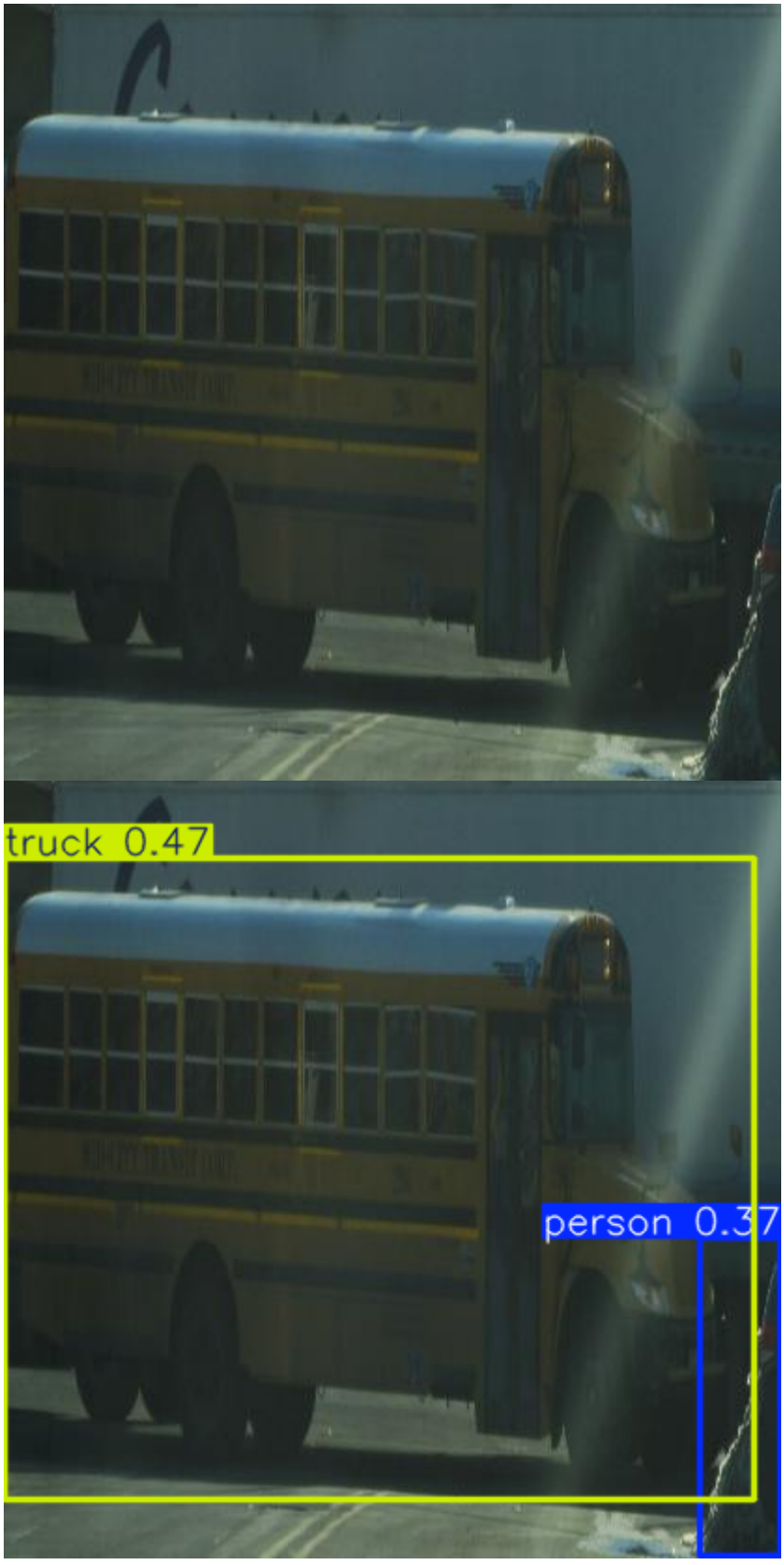
```
image_files = [f for f in os.listdir(train_path) if f.endswith('.jpg')]
selected_images = random.sample(image_files, 16)
fig, axes = plt.subplots(4, 4, figsize=(10, 10))
for i, ax in enumerate(axes.flat):
    img_path = os.path.join(train_path, selected_images[i])
    img = mpimg.imread(img_path)
    ax.imshow(img)
    ax.axis('off')
plt.tight_layout()
plt.show()
```


PHILIPPINES
Savana.O carro 100% brasileiro que conhece a sua potência.
POLYSTILO
43
43

```
images_folder = '/kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images'
image_files = [f for f in os.listdir(images_folder) if f.endswith(('.jpg', '.png', '.jpeg
random_images = random.sample(image_files, 20)
model = YOLO("yolov8n.pt")
for image_file in random_images:
    image_path = os.path.join(images_folder, image_file)
    result_predict = model.predict(source=image_path, imgsz=(416))
    original_image = Image.open(image_path)
    display(original_image)
    for result in result_predict:
        plot = result.plot()
        plot = cv2.cvtColor(plot, cv2.COLOR_BGR2RGB)
        display(Image.fromarray(plot))
```

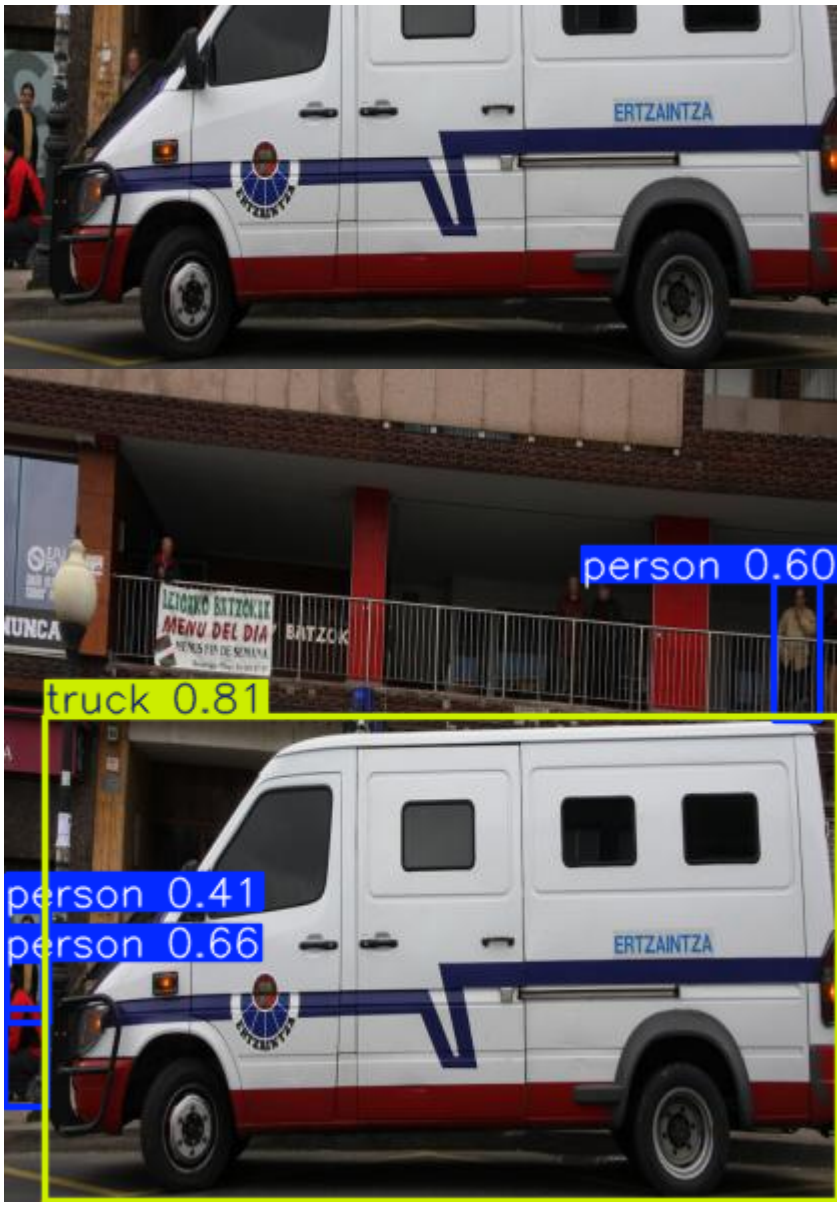
```
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8n.pt
100%|██████████| 6.25M/6.25M [00:00<00:00, 102MB/s]

image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/7bec73
Speed: 6.8ms preprocess, 6.3ms inference, 254.5ms postprocess per image at shape (1,
```




```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/93684b
Speed: 2.2ms preprocess, 9.6ms inference, 1.5ms postprocess per image at shape (1, 3,
```


```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/d7fba6
Speed: 1.2ms preprocess, 6.5ms inference, 1.3ms postprocess per image at shape (1, 3,
```




```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/9c8fc2
Speed: 0.8ms preprocess, 8.7ms inference, 1.3ms postprocess per image at shape (1, 3,
```


```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/b12f44
Speed: 0.8ms preprocess, 6.7ms inference, 1.3ms postprocess per image at shape (1, 3,
```

```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/d8457f
Speed: 0.9ms preprocess, 7.5ms inference, 1.3ms postprocess per image at shape (1, 3,
```

```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/944965
Speed: 0.8ms preprocess, 6.2ms inference, 1.4ms postprocess per image at shape (1, 3,
```


```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/b8e3cf
Speed: 0.8ms preprocess, 6.1ms inference, 1.2ms postprocess per image at shape (1, 3,
```


```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/02178f
Speed: 1.2ms preprocess, 6.8ms inference, 1.2ms postprocess per image at shape (1, 3,
```


```
image 1/1 /kaggle/input/vehicledetection/VehiclesDetectionDataset/train/images/db8edd
Speed: 1.1ms preprocess, 8.6ms inference, 1.6ms postprocess per image at shape (1, 3,
```