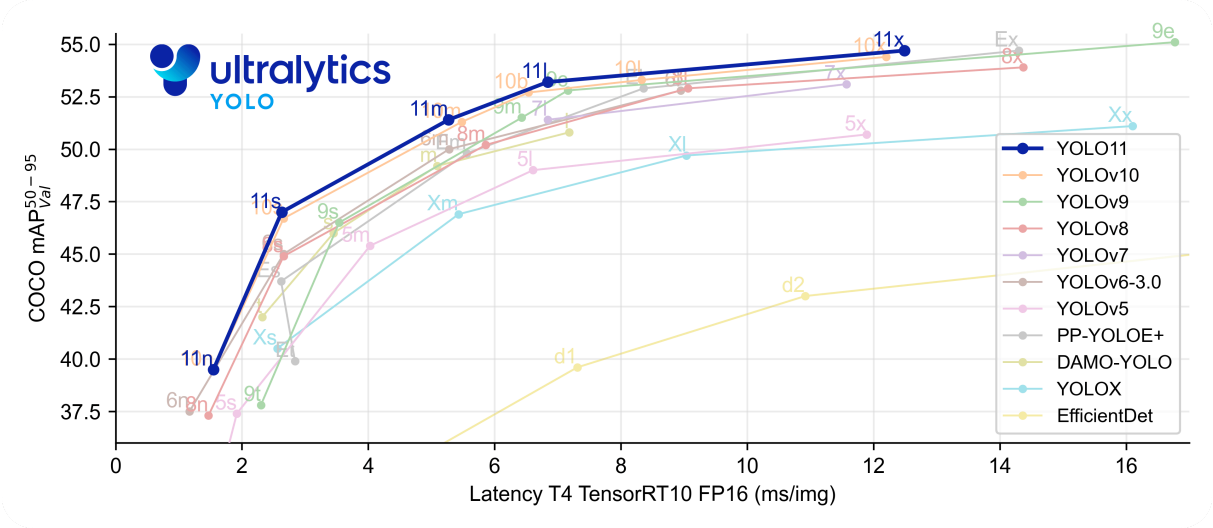
Ultralytics YOLO11

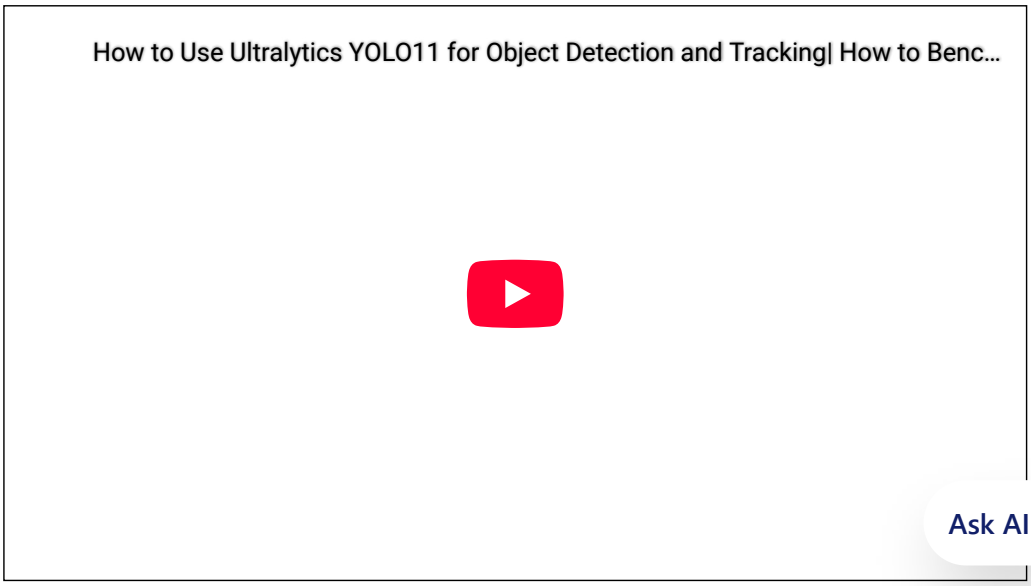
Overview

YOLO11 is the latest iteration in the Ultralytics YOLO series of real-time object detectors, redefining what's possible with cutting-edge accuracy, speed, and efficiency. Building upon the impressive advancements of previous YOLO versions, YOLO11 introduces significant improvements in architecture and training methods, making it a versatile choice for a wide range of computer vision tasks.



0:00 / 18:44

Ultralytics YOLO11 🚀 Podcast generated by NotebookLM



Watch: How to Use Ultralytics YOLO11 for Object Detection and Tracking | How to Benchmark | YOLO11 RELEASED 🚀

Key Features

- **Enhanced Feature Extraction:** YOLO11 employs an improved backbone and neck architecture, which enhances feature extraction capabilities for more precise object detection and complex task performance.
- **Optimized for Efficiency and Speed:** YOLO11 introduces refined architectural designs and optimized training pipelines, delivering faster processing speeds and maintaining an optimal balance between accuracy and performance.
- **Greater Accuracy with Fewer Parameters:** With advancements in model design, YOLO11m achieves a higher mean Average Precision (mAP) on the COCO dataset while using 22% fewer parameters than YOLOv8m, making it computationally efficient without compromising accuracy.
- **Adaptability Across Environments:** YOLO11 can be seamlessly deployed across various environments, including edge devices, cloud platforms, and systems supporting NVIDIA GPUs, ensuring maximum flexibility.
- **Broad Range of Supported Tasks:** Whether it's object detection, instance segmentation, image classification, pose estimation, or oriented object detection (OBB), YOLO11 is designed to cater to a diverse set of computer vision challenges.

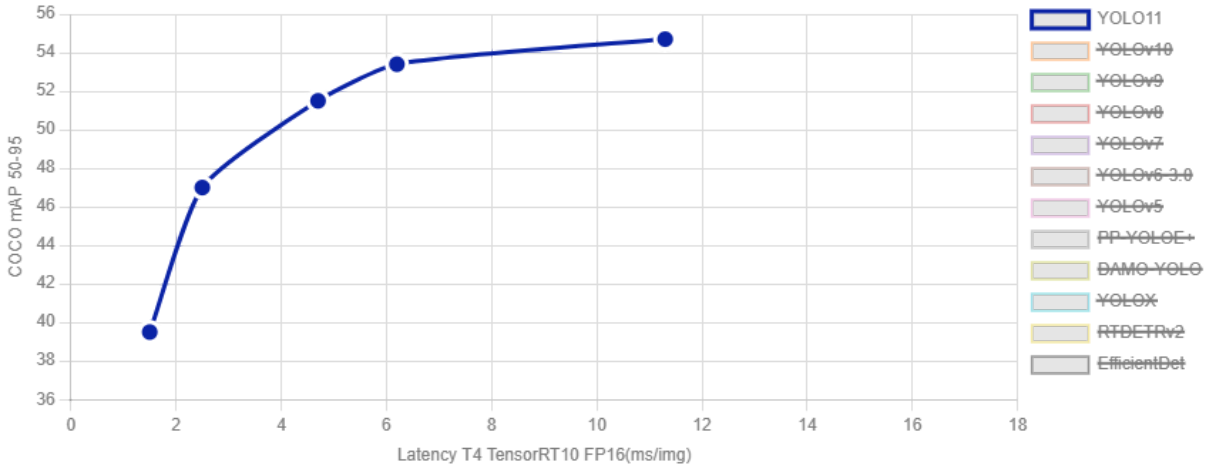
Supported Tasks and Modes

YOLO11 builds upon the versatile model range introduced in YOLOv8, offering enhanced support across various computer vision tasks:

Model	Filenames	Task	Inference	Validation	Training	Export
YOLO11	`yolo11n.pt` `yolo11s.pt` `yolo11m.pt` `yolo11l.pt` `yolo11x.pt`	Detection	✅	✅	✅	✅
YOLO11-seg	`yolo11n-seg.pt` `yolo11s-seg.pt` `yolo11m-seg.pt` `yolo11l-seg.pt` `yolo11x-seg.pt`	Instance Segmentation	✅	✅	✅	✅
YOLO11-pose	`yolo11n-pose.pt` `yolo11s-pose.pt` `yolo11m-pose.pt` `yolo11l-pose.pt` `yolo11x-pose.pt`	Pose/Keypoints	✅	✅	✅	✅
YOLO11-obb	`yolo11n-obb.pt` `yolo11s-obb.pt` `yolo11m-obb.pt` `yolo11l-obb.pt` `yolo11x-obb.pt`	Oriented Detection	✅	✅	✅	✅
YOLO11-cls	`yolo11n-cls.pt` `yolo11s-cls.pt` `yolo11m-cls.pt` `yolo11l-cls.pt` `yolo11x-cls.pt`	Classification	✅	✅	✅	✅

This table provides an overview of the YOLO11 model variants, showcasing their applicability in specific tasks and compatibility with operational modes such as Inference, Validation, Training, and Export. This flexibility makes YOLO11 suitable for a wide range of applications in computer vision, from real-time detection to complex segmentation tasks.

Performance Metrics


COCO mAP 50-95
56
54
52
50
48
46
44
42
40
38
36
0
2
4
6
8
10
12
14
16
18
Latency T4 TensorRT10 FP16(ms/img)
YOLO11
YOLOv10
YOLOv9
YOLOv8
YOLOv7
YOLOv6-3.0
YOLOv5
PP-YOLOE+
DAMO-YOLO
YOLOX
RTDETRv2
EfficientDet

Performance

Detection (COCO)

See Detection Docs for usage examples with these models trained on COCO, which include 80 pre-trained classes.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

Segmentation (COCO)

See Segmentation Docs for usage examples with these models trained on COCO, which include 80 pre-trained classes.

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n-seg	640	38.9	32.0	65.9 ± 1.1	1.8 ± 0.0	2.9	10.4
YOLO11s-seg	640	46.6	37.8	117.6 ± 4.9	2.9 ± 0.0	10.1	35.5
YOLO11m-seg	640	51.5	41.5	281.6 ± 1.2	6.3 ± 0.1	22.4	123.3
YOLO11l-seg	640	53.4	42.9	344.2 ± 3.2	7.8 ± 0.2	27.6	142.2
YOLO11x-seg	640	54.7	43.8	664.5 ± 3.2	15.8 ± 0.7	62.1	319.0

Classification (ImageNet)

See Classification Docs for usage examples with these models trained on ImageNet, which include 1000 pre-trained classes.

Model	size (pixels)	acc top1	acc top5	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B) at 224
YOLO11n-cls	224	70.0	89.4	5.0 ± 0.3	1.1 ± 0.0	1.6	0.5
YOLO11s-cls	224	75.4	92.7	7.9 ± 0.2	1.3 ± 0.0	5.5	1.6
YOLO11m-cls	224	77.3	93.9	17.2 ± 0.4	2.0 ± 0.0	10.4	5.0
YOLO11l-cls	224	78.3	94.3	23.2 ± 0.3	2.8 ± 0.0	12.9	6.2
YOLO11x-cls	224	79.5	94.9	41.4 ± 0.9	3.8 ± 0.0	28.4	13.7

Pose (COCO)

See Pose Estimation Docs for usage examples with these models trained on COCO, which include 1 pre-trained class, 'person'.

Model	size (pixels)	mAP^{pose} 50-95	mAP^{pose} 50	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n-pose	640	50.0	81.0	52.4 ± 0.5	1.7 ± 0.0	2.9	7.6
YOLO11s-pose	640	58.9	86.3	90.5 ± 0.6	2.6 ± 0.0	9.9	23.2
YOLO11m-pose	640	64.9	89.4	187.3 ± 0.8	4.9 ± 0.1	20.9	71.7
YOLO11l-pose	640	66.1	89.9	247.7 ± 1.1	6.4 ± 0.1	26.2	90.7
YOLO11x-pose	640	69.5	91.1	488.0 ± 13.9	12.1 ± 0.2	58.8	203.3

OBB (DOTAv1)

See Oriented Detection Docs for usage examples with these models trained on DOTAv1, which include 15 pre-trained classes.

Model	size (pixels)	mAP^{test} 50	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n-obb	1024	78.4	117.6 ± 0.8	4.4 ± 0.0	2.7	17.2
YOLO11s-obb	1024	79.5	219.4 ± 4.0	5.1 ± 0.0	9.7	57.5
YOLO11m-obb	1024	80.9	562.8 ± 2.9	10.1 ± 0.4	20.9	183.5
YOLO11l-obb	1024	81.0	712.5 ± 5.0	13.5 ± 0.6	26.2	232.0
YOLO11x-obb	1024	81.3	1408.6 ± 7.7	28.6 ± 1.0	58.8	520.2

Usage Examples

This section provides simple YOLO11 training and inference examples. For full documentation on these and other modes, see the Predict, Train, Val, and Export docs pages.

Note that the example below is for YOLO11 Detect models for object detection. For additional supported tasks, see the Segment, Classify, OBB, and Pose docs.

Example

Python

PyTorch pretrained `*.pt` models as well as configuration `*.yaml` files can be passed to the `YOLO()` class to create a model instance in Python:

```python
from ultralytics import YOLO

# Load a COCO-pretrained YOLO11n model
model = YOLO("yolo11n.pt")

# Train the model on the COCO8 example dataset for 100 epochs
results = model.train(data="coco8.yaml", epochs=100, imgsz=640)

# Run inference with the YOLO11n model on the 'bus.jpg' image
results = model("path/to/bus.jpg")
```

CLI

CLI commands are available to directly run the models:

```bash
# Load a COCO-pretrained YOLO11n model and train it on the COCO8 example dataset for 100 epochs
yolo train model=yolo11n.pt data=coco8.yaml epochs=100 imgsz=640

# Load a COCO-pretrained YOLO11n model and run inference on the 'bus.jpg' image
yolo predict model=yolo11n.pt source=path/to/bus.jpg
```

Citations and Acknowledgements

Ultralytics YOLO11 Publication

Ultralytics has not published a formal research paper for YOLO11 due to the rapidly evolving nature of the models. We focus on advancing the technology and making it easier to use, rather than producing static documentation. For the most up-to-date information on YOLO architecture, features, and usage, please refer to our GitHub repository and documentation.

If you use YOLO11 or any other software from this repository in your work, please cite it using the following format:

BibTeX

```bibtex
@software{yolo11_ultralytics,
  author = {Glenn Jocher and Jing Qiu},
  title = {Ultralytics YOLO11},
  version = {11.0.0},
  year = {2024},
  url = {https://github.com/ultralytics/ultralytics},
  orcid = {0000-0001-5950-6979, 0000-0002-7603-6750, 0000-0003-3783-7069},
  license = {AGPL-3.0}
}
```

Please note that the DOI is pending and will be added to the citation once it is available. YOLO11 models are provided under AGPL-3.0 and Enterprise licenses.

FAQ

What are the key improvements in Ultralytics YOLO11 compared to previous versions?

Ultralytics YOLO11 introduces several significant advancements over its predecessors. Key improvements include:

- **Enhanced Feature Extraction:** YOLO11 employs an improved backbone and neck architecture, enhancing feature extraction capabilities for more precise object detection.
- **Optimized Efficiency and Speed:** Refined architectural designs and optimized training pipelines deliver faster processing speeds while maintaining a balance between accuracy and performance.
- **Greater Accuracy with Fewer Parameters:** YOLO11m achieves higher mean Average Precision (mAP) on the COCO dataset with 22% fewer parameters than YOLOv8m, making it computationally efficient without compromising accuracy.
- **Adaptability Across Environments:** YOLO11 can be deployed across various environments, including edge devices, cloud platforms, and systems supporting NVIDIA GPUs.
- **Broad Range of Supported Tasks:** YOLO11 supports diverse computer vision tasks such as object detection, instance segmentation, image classification, pose estimation, and oriented object detection (OBB).

How do I train a YOLO11 model for object detection?

Training a YOLO11 model for object detection can be done using Python or CLI commands. Below are examples for both methods:

Example

Python

```python
from ultralytics import YOLO

# Load a COCO-pretrained YOLO11n model
model = YOLO("yolo11n.pt")

# Train the model on the COCO8 example dataset for 100 epochs
results = model.train(data="coco8.yaml", epochs=100, imgsz=640)
```

CLI

```bash
# Load a COCO-pretrained YOLO11n model and train it on the COCO8 example dataset for 100 epochs
yolo train model=yolo11n.pt data=coco8.yaml epochs=100 imgsz=640
```

For more detailed instructions, refer to the Train documentation.

What tasks can YOLO11 models perform?

YOLO11 models are versatile and support a wide range of computer vision tasks, including:

- **Object Detection:** Identifying and locating objects within an image.
- **Instance Segmentation:** Detecting objects and delineating their boundaries.
- **Image Classification:** Categorizing images into predefined classes.
- **Pose Estimation:** Detecting and tracking keypoints on human bodies.
- **Oriented Object Detection (OBB):** Detecting objects with rotation for higher precision.

For more information on each task, see the Detection, Instance Segmentation, Classification, Pose Estimation, and Oriented Detection documentation.

How does YOLO11 achieve greater accuracy with fewer parameters?

YOLO11 achieves greater accuracy with fewer parameters through advancements in model design and optimization techniques. The improved architecture allows for efficient feature extraction and processing, resulting in higher mean Average Precision (mAP) on datasets like COCO while using 22% fewer parameters than YOLOv8m. This makes YOLO11 computationally efficient without compromising on accuracy, making it suitable for deployment on resource-constrained devices.

Can YOLO11 be deployed on edge devices?

Yes, YOLO11 is designed for adaptability across various environments, including edge devices. Its optimized architecture and efficient processing capabilities make it suitable for deployment on edge devices, cloud platforms, and systems supporting NVIDIA GPUs. This flexibility ensures that YOLO11 can be used in diverse applications, from real-time detection on mobile devices to complex segmentation tasks in cloud environments. For more details on deployment options, refer to the Export documentation.