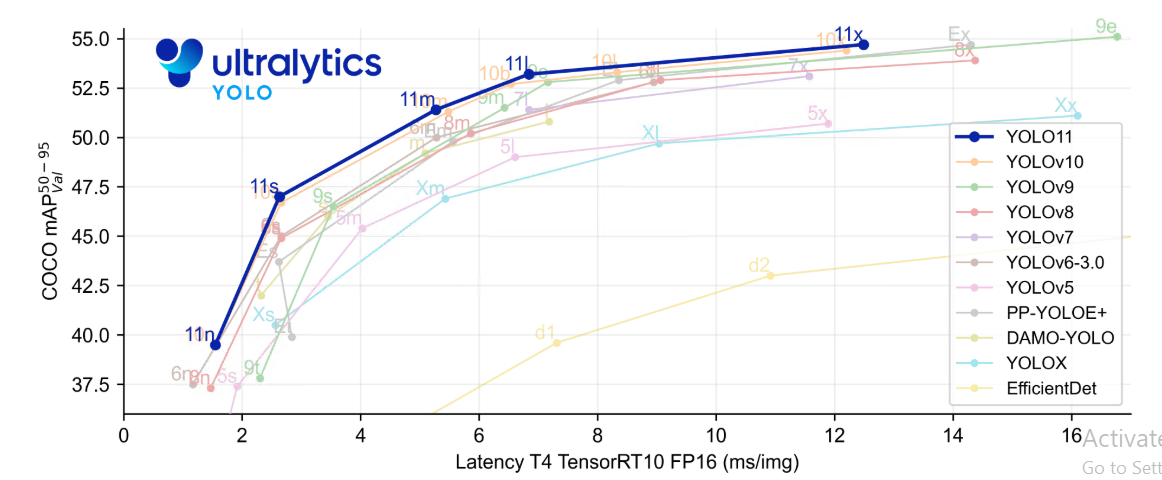
Overview

YOLO11 is the latest iteration in the [Ultralytics](https://www.ultralytics.com/) YOLO series of real-time object detectors, redefining what's possible with cutting-edge [accuracy](https://www.ultralytics.com/glossary/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](https://www.ultralytics.com/glossary/computer-vision-cv) tasks.



Key Features

* **Enhanced Feature Extraction:** YOLO11 employs an improved [backbone](https://www.ultralytics.com/glossary/backbone) and neck architecture, which enhances [feature extraction](https://www.ultralytics.com/glossary/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](https://www.ultralytics.com/glossary/mean-average-precision-map) (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](https://docs.ultralytics.com/tasks/detect/) | ✅ | ✅ | ✅ | ✅ |
| YOLO11-seg | yolo11n-seg.pt yolo11s-seg.pt yolo11m-seg.pt yolo11l-seg.pt yolo11x-seg.pt | [Instance Segmentation](https://docs.ultralytics.com/tasks/segment/) | ✅ | ✅ | ✅ | ✅ |
| YOLO11-pose | yolo11n-pose.pt yolo11s-pose.pt yolo11m-pose.pt yolo11l-pose.pt yolo11x-pose.pt | [Pose/Keypoints](https://docs.ultralytics.com/tasks/pose/) | ✅ | ✅ | ✅ | ✅ |
| YOLO11-obb | yolo11n-obb.pt yolo11s-obb.pt yolo11m-obb.pt yolo11l-obb.pt yolo11x-obb.pt | [Oriented Detection](https://docs.ultralytics.com/tasks/obb/) | ✅ | ✅ | ✅ | ✅ |
| YOLO11-cls | yolo11n-cls.pt yolo11s-cls.pt yolo11m-cls.pt yolo11l-cls.pt yolo11x-cls.pt | [Classification](https://docs.ultralytics.com/tasks/classify/) | ✅ | ✅ | ✅ | ✅ |

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

**Performance**



[Detection (COCO)](https://docs.ultralytics.com/models/yolo11/#__tabbed_1_1)[Segmentation (COCO)](https://docs.ultralytics.com/models/yolo11/#__tabbed_1_2)[Classification (ImageNet)](https://docs.ultralytics.com/models/yolo11/#__tabbed_1_3)[Pose (COCO)](https://docs.ultralytics.com/models/yolo11/#__tabbed_1_4)[OBB (DOTAv1)](https://docs.ultralytics.com/models/yolo11/#__tabbed_1_5)

See [Detection Docs](https://docs.ultralytics.com/tasks/detect/) for usage examples with these models trained on [COCO](https://docs.ultralytics.com/datasets/detect/coco/), which include 80 pre-trained classes.

| **Model** | **size (pixels)** | **mAPval 50-95** | **Speed CPU ONNX (ms)** | **Speed T4 TensorRT10 (ms)** | **params (M)** | **FLOPs (B)** |
| --- | --- | --- | --- | --- | --- | --- |
| [YOLO11n](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt) | 640 | 39.5 | 56.1 ± 0.8 | 1.5 ± 0.0 | 2.6 | 6.5 |
| [YOLO11s](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s.pt) | 640 | 47.0 | 90.0 ± 1.2 | 2.5 ± 0.0 | 9.4 | 21.5 |
| [YOLO11m](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11m.pt) | 640 | 51.5 | 183.2 ± 2.0 | 4.7 ± 0.1 | 20.1 | 68.0 |
| [YOLO11l](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11l.pt) | 640 | 53.4 | 238.6 ± 1.4 | 6.2 ± 0.1 | 25.3 | 86.9 |
| [YOLO11x](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11x.pt) | 640 | 54.7 | 462.8 ± 6.7 | 11.3 ± 0.2 | 56.9 | 194.9 |

Usage Examples

This section provides simple YOLO11 training and inference examples. For full documentation on these and other [modes](https://docs.ultralytics.com/modes/), see the [Predict](https://docs.ultralytics.com/modes/predict/), [Train](https://docs.ultralytics.com/modes/train/), [Val](https://docs.ultralytics.com/modes/val/), and [Export](https://docs.ultralytics.com/modes/export/) docs pages.

Note that the example below is for YOLO11 [Detect](https://docs.ultralytics.com/tasks/detect/) models for [object detection](https://www.ultralytics.com/glossary/object-detection). For additional supported tasks, see the [Segment](https://docs.ultralytics.com/tasks/segment/), [Classify](https://docs.ultralytics.com/tasks/classify/), [OBB](https://docs.ultralytics.com/tasks/obb/), and [Pose](https://docs.ultralytics.com/tasks/pose/) docs.

**Example**

[Python](https://docs.ultralytics.com/models/yolo11/#__tabbed_2_1)

[PyTorch](https://www.ultralytics.com/glossary/pytorch) pretrained \*.pt models as well as configuration \*.yaml files can be passed to the YOLO() class to create a model instance in 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](https://docs.ultralytics.com/models/yolo11/#__tabbed_2_2)

# Build a new model from YAML and start training from scratch

yolo detect train data=coco8.yaml model=yolo11n.yaml epochs=100 imgsz=640

# Start training from a pretrained \*.pt model

yolo detect train data=coco8.yaml model=yolo11n.pt epochs=100 imgsz=640

# Build a new model from YAML, transfer pretrained weights to it and start training

yolo detect train data=coco8.yaml model=yolo11n.yaml pretrained=yolo11n.pt epochs=100 imgsz=640

yolo task=detect mode=train model=yolov8n.pt imgsz=640 data=custom\_data.yaml epochs=10 batch=8 name=yolov8n\_custom

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](https://github.com/ultralytics/ultralytics) and [documentation](https://docs.ultralytics.com/).

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

[BibTeX](https://docs.ultralytics.com/models/yolo11/#__tabbed_3_1)

@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}

}

Environment setting for Windows 10

1. Install Cuda11.8.0 and CuDNN 8.9.7.29 for training by using GPU.

1) Run cuda\_11.8.0\_522.06\_windows.exe to install CUDA

2) Extract cudnn-windows-x86\_64-8.9.7.29\_cuda11-archive.zip into the CUDA Installed directory (C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.8) to install CuDNN 8.9.7.29

1. Install Python 3.9.13 (Any 3.9 python version is OK)

Run Anaconda3-2022.10-Windows-x86\_64.exe

1. Install Visual Studio 2019 (or 2017, 2022) to install pandas python package --> optional, can be skipped
2. Install labelimg python package
3. Open Anaconda command prompt
4. Run following command

pip install labelimg --no-index --find-links D:/python3.9-package

here D:/python3.9-package is path of folder which contains python packages.

--find-links means offline install from specified package folder

1. Install torch python package

To install cuda version

pip install torch-2.6.0+cu118-cp39-cp39-win\_amd64.whl --no-index --find-links D:/python3.9-package

To install cpu version

pip install torch-2.6.0-cp39-cp39-win\_amd64.whl --no-index --find-links D:/python3.9-package

1. Install torchvision python package

To install cuda version

pip install torchvision-0.21.0+cu118-cp39-cp39-win\_amd64.whl --no-index --find-links D:/python3.9-package

To install cpu version

pip install torchvision==0.21.0 --no-index --find-links D:/python3.9-package

1. Install ultralytics python package

pip install ultralytics --no-index --find-links D:/python3.9-package