A README

The Choice of using YOLOv11 Nano

The choice of YOLOv11 Nano for this study was driven by its remarkable balance of accuracy, speed, and efficiency. Traditional disease detection methods often require extensive resources and sophisticated equipment that are not feasible for medium-scale farms. The YOLOv11 Nano, however, offers a transformative approach due to its lightweight architecture specifically designed to perform real-time object detection even in resource-limited settings.

One of the primary strengths of YOLOv11 Nano is its streamlined convolutional network, which allows for rapid processing of input data. This is critical for early-stage disease detection in orange fruits where timely identification can prevent the spread and severity of infections. The model's efficiency ensures that it can be deployed on edge devices like smartphones and drones, which are increasingly accessible to farmers. This allows for immediate, on-the-spot analysis in the field, reducing delays associated with traditional lab-based diagnostics.

Problem Statement

The manuscript begins by establishing the importance of disease detection in oranges to mitigate crop loss and improve productivity. Disease identification, particularly in resource-constrained environments, requires models that are:

- Lightweight and computationally efficient.

- Accurate in detection.

- Compatible with low-power devices.

The problem statement emphasizes the challenges in deploying traditional deep learning models in agricultural settings due to their computational and hardware demands.

Model Architecture

The manuscript introduces the YOLOv11 Nano model, highlighting its lightweight design and adaptability:

- **Depthwise Convolution Layers**: Incorporated to reduce model complexity, achieving a 4.2× reduction in FLOPs (floating-point operations per second) and a 68% smaller model size compared to traditional approaches.

- **Optimization Strategy**: Replacing the default Adam optimizer with Adamax (a variant leveraging the infinity norm) to handle sparse gradients more effectively.

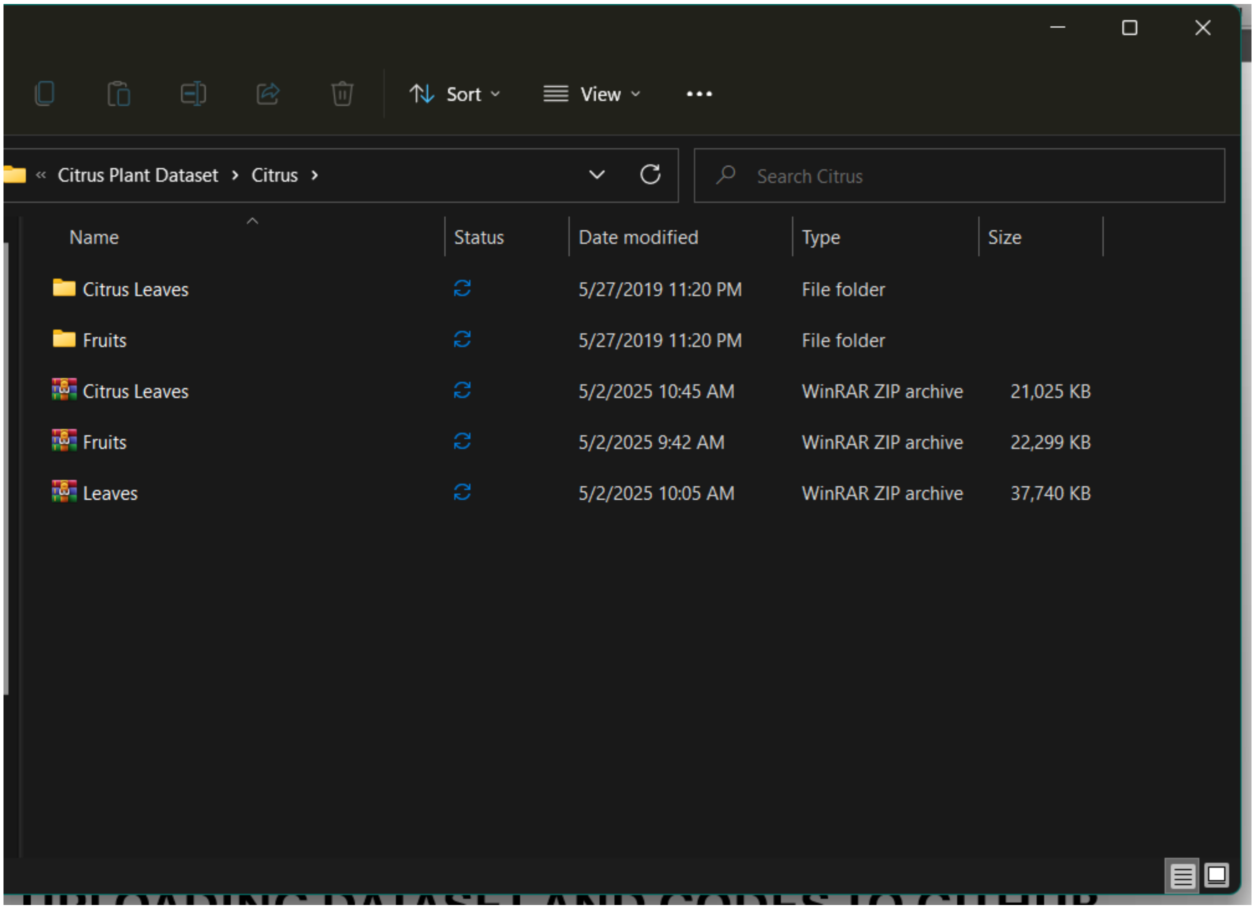
- **Dynamic Parameter Adjustment**: A novel dynamic optimization strategy was implemented to adaptively modify the learning rate, beta1, and beta2 parameters during training, enhancing convergence stability.

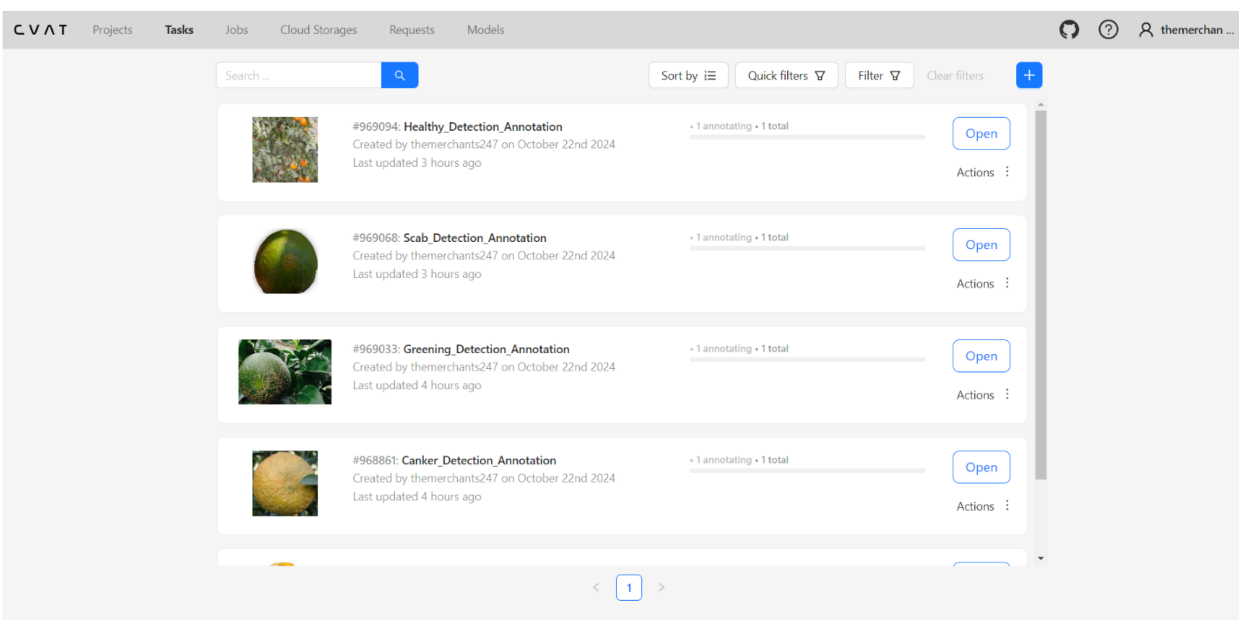
Dataset Curation

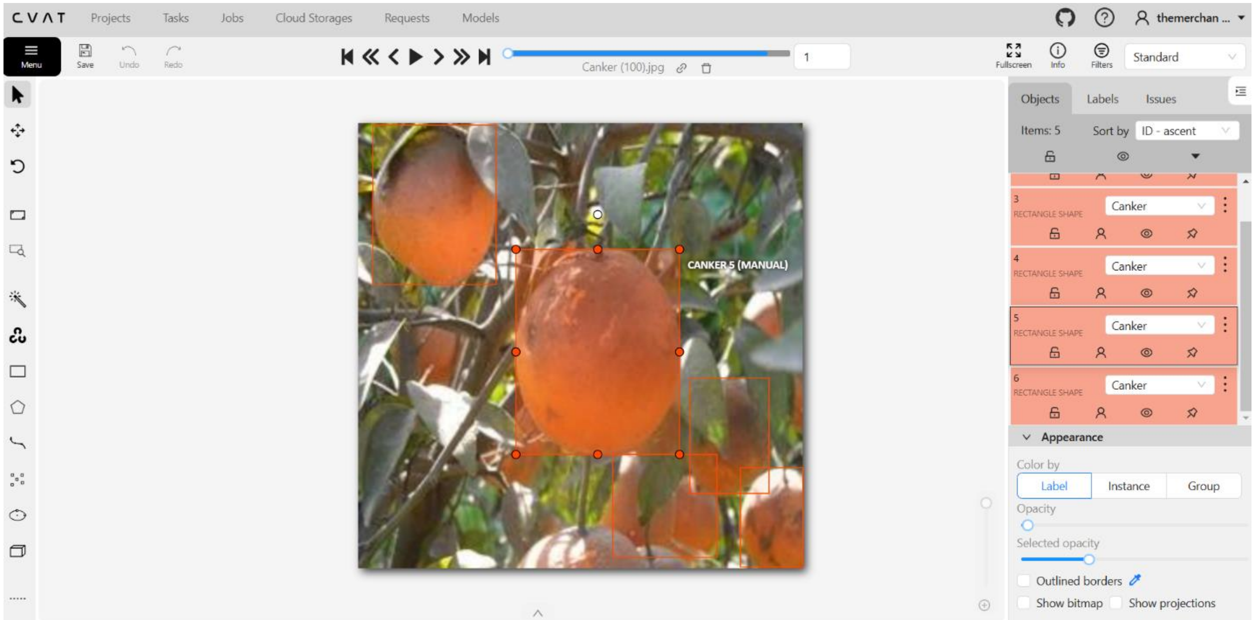
The manuscript details the creation of a dataset specifically tailored for the study:

- **Scope**: A curated dataset of 700 annotated images of diseased and healthy oranges was used, covering conditions like black spots, cankers, greening and scab — prevalent diseases affecting orange fruits. The dataset was obtained from the Mendeley dataset repository.

- **Annotation Process**: Annotation was performed using bounding boxes to mark regions of interest, ensuring high-quality labels for training and evaluation. Annotation was done using Cvat annotation tool. The annotated files were then converted into a Yolo format.


Sort
View
Citrus Plant Dataset › Citrus ›
Search Citrus
Name
Status
Date modified
Type
Size
Citrus Leaves 5/27/2019 11:20 PM File folder
Fruits 5/27/2019 11:20 PM File folder
Citrus Leaves 5/2/2025 10:45 AM WinRAR ZIP archive 21,025 KB
Fruits 5/2/2025 9:42 AM WinRAR ZIP archive 22,299 KB
Leaves 5/2/2025 10:05 AM WinRAR ZIP archive 37,740 KB


CVAT
Projects
Tasks
Jobs
Cloud Storages
Requests
Models
themerchan ...
Sort by
Quick filters
Filter
Clear filters
#969094: Healthy_Detection_Annotation
Created by themerchants247 on October 22nd 2024
Last updated 3 hours ago
• 1 annotating • 1 total
Open
Actions
#969068: Scab_Detection_Annotation
Created by themerchants247 on October 22nd 2024
Last updated 3 hours ago
• 1 annotating • 1 total
Open
Actions
#969033: Greening_Detection_Annotation
Created by themerchants247 on October 22nd 2024
Last updated 4 hours ago
• 1 annotating • 1 total
Open
Actions
#968861: Canker_Detection_Annotation
Created by themerchants247 on October 22nd 2024
Last updated 4 hours ago
• 1 annotating • 1 total
Open
Actions
1



-**Diversity:** The dataset includes images captured under varying lighting conditions and angles to simulate real-world scenarios.

Experimental Results

The results section of the manuscript focuses on the performance metrics and comparative analysis:

- **Mean Average Precision (mAP):** The Adamax-optimized YOLOv11 Nano achieved **95% mAP accuracy**, surpassing baseline models in precision.

- **Inference Latency:** A 21% reduction in inference latency compared to YOLOv8 Nano is reported, demonstrating the model's suitability for real-time applications.

- **Resource Efficiency:** The model requires only **1.8 GB RAM**, ensuring compatibility with low-power devices commonly available in agricultural environments.

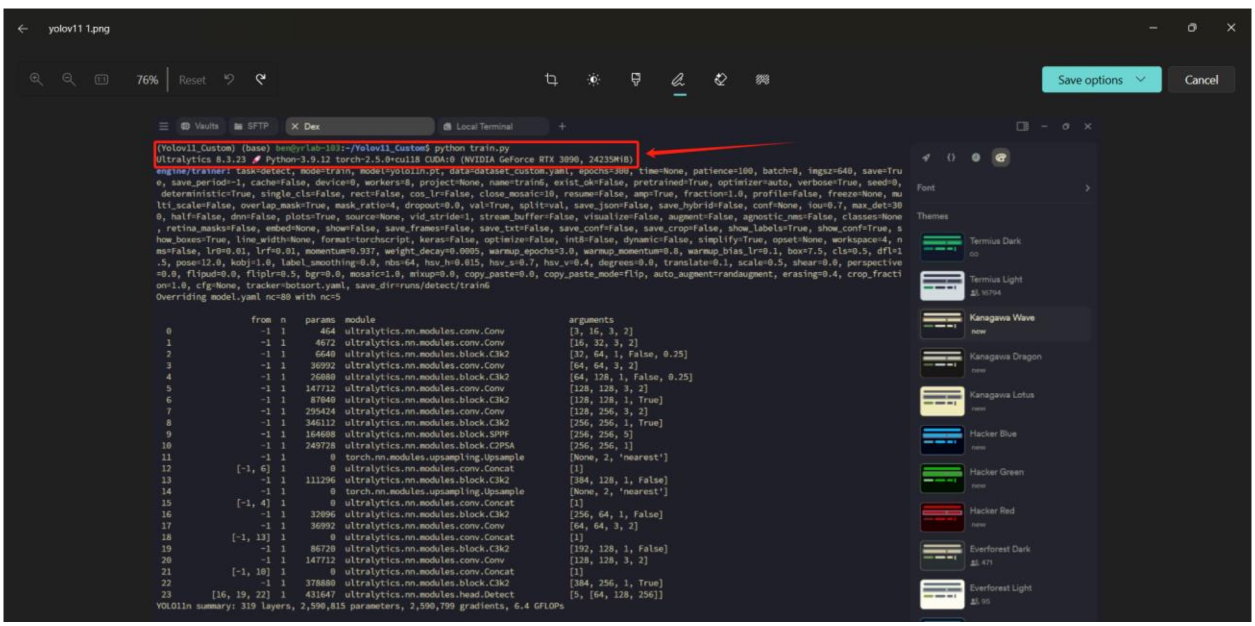
Training on YOLOv11

Prerequisites

Server Setup: NVIDIA GeForce RTX 3090 graphics card, which is highly compatible with YOLOv11. The YOLOv11 is compatible with CUDA and the CuDNN and features a strong parallel processing architecture.   Having 62.5 GB of RAM and 10 GB of hard drive space allows our system to quickly load models and handle massive amounts of data. Ubuntu 20.04.5 LTS (GNU/Linux 5.15.0-76-generic x86_64)

Python Environment: Python -3.9.12, torch-2.5.0+cu118, Ultralytics – 8.3.23

YOLOv9 Source Code: Obtain the YOLOv9 implementation.



Setting up the python environment:

sudo apt update

sudo apt install python3-venv # Installing python3-venv if not installed

python3 -m venv yolov11_custom-env

source yolov11_custom-env/bin/activate

pip install --upgrade pip

pip install torch torchvision opencv-python

Cloning the YOLOv11 Repository from github

git clone https://github.com/ultralytics/yolov11.git

cd yolov11

pip install -r requirements.txt

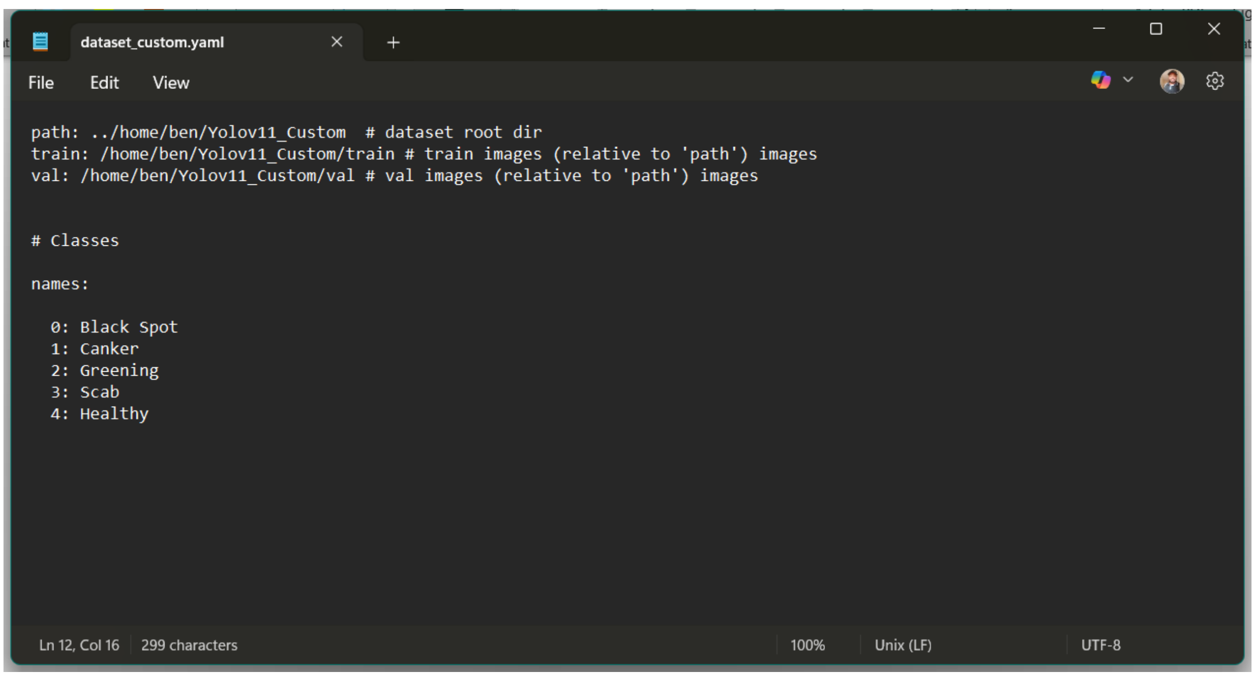
Creating directory and uploading the dataset via SFTP

Using the Termius's SFTP client to upload the dataset to the server.
../home/ben/Yolov11_Custom

mkdir -p datasets/ **home/ben/Yolov11_Custom** /images

mkdir -p datasets/ **home/ben/Yolov11_Custom** /val

images and label files were uploaded to respective directories

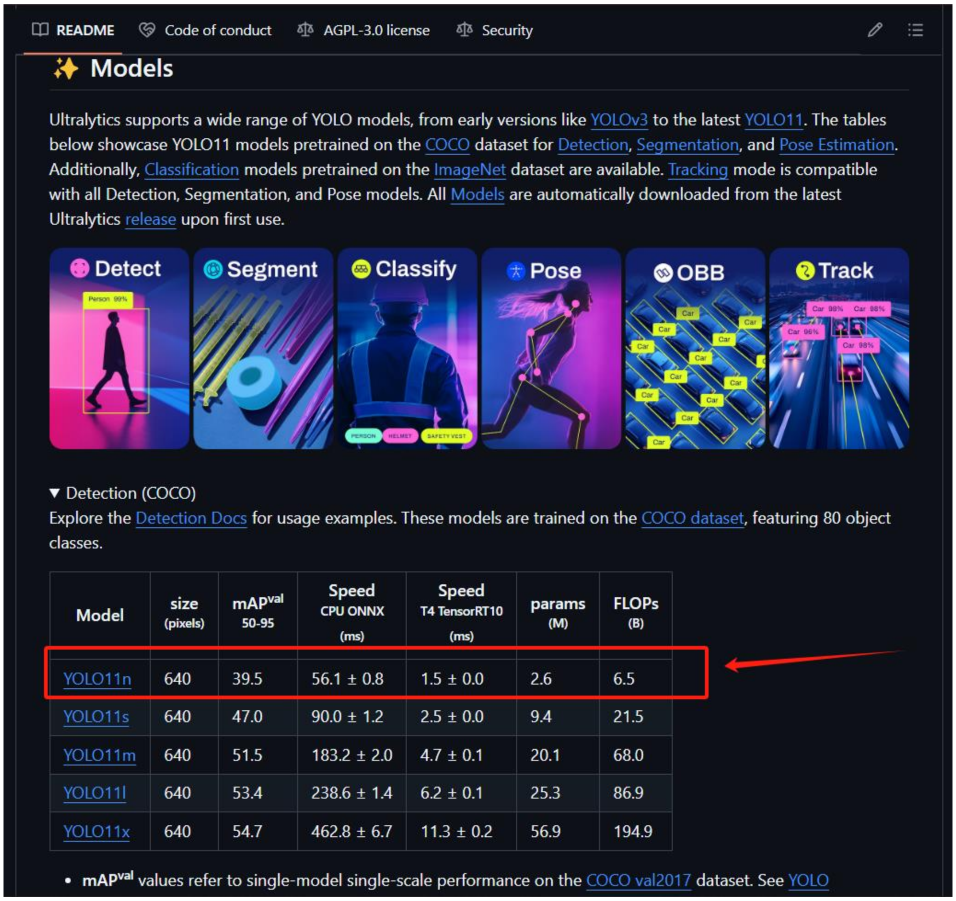



```
path: ../home/ben/Yolov11_Custom  # dataset root dir
train: /home/ben/Yolov11_Custom/train # train images (relative to 'path') images
val: /home/ben/Yolov11_Custom/val # val images (relative to 'path') images


# Classes

names:

  0: Black Spot
  1: Canker
  2: Greening
  3: Scab
  4: Healthy
```

Training the Model

Parameter adjustments

Downloading Yolov11nano from github



README   Code of conduct   AGPL-3.0 license   Security

✨ Models

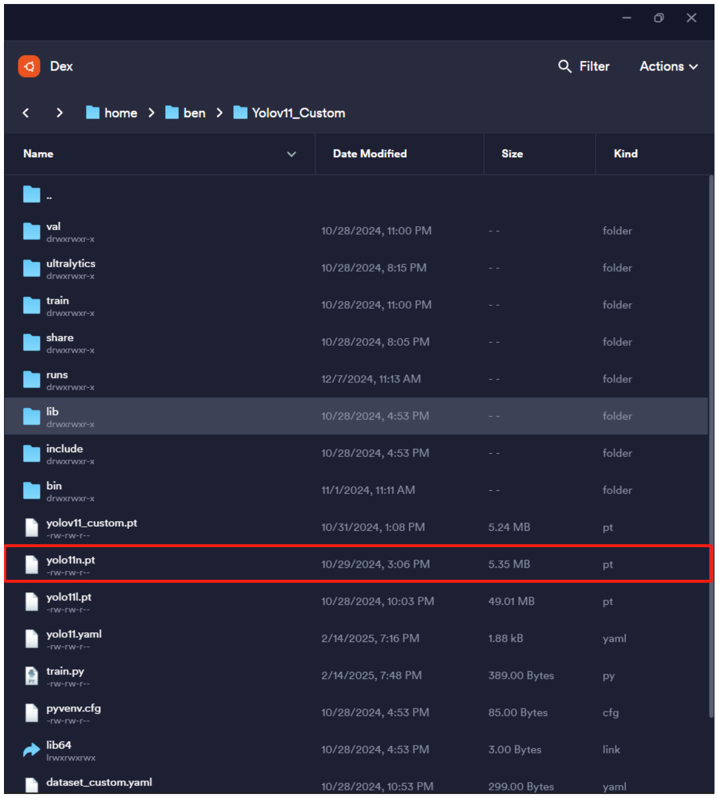
Ultralytics supports a wide range of YOLO models, from early versions like YOLOv3 to the latest YOLO11. The tables below showcase YOLO11 models pretrained on the COCO dataset for Detection, Segmentation, and Pose Estimation. Additionally, Classification models pretrained on the ImageNet dataset are available. Tracking mode is compatible with all Detection, Segmentation, and Pose models. All Models are automatically downloaded from the latest Ultralytics release upon first use.

▼ Detection (COCO)

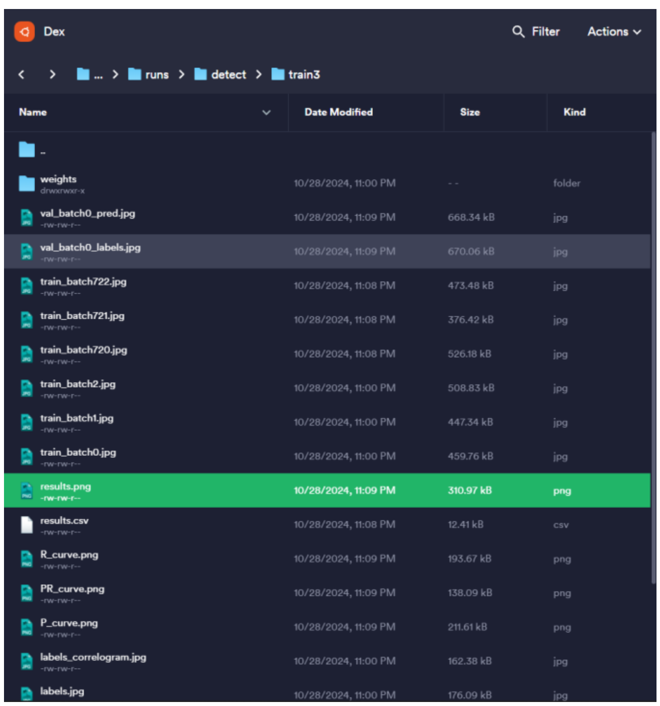
Explore the Detection Docs for usage examples. These models are trained on the COCO dataset, featuring 80 object classes.

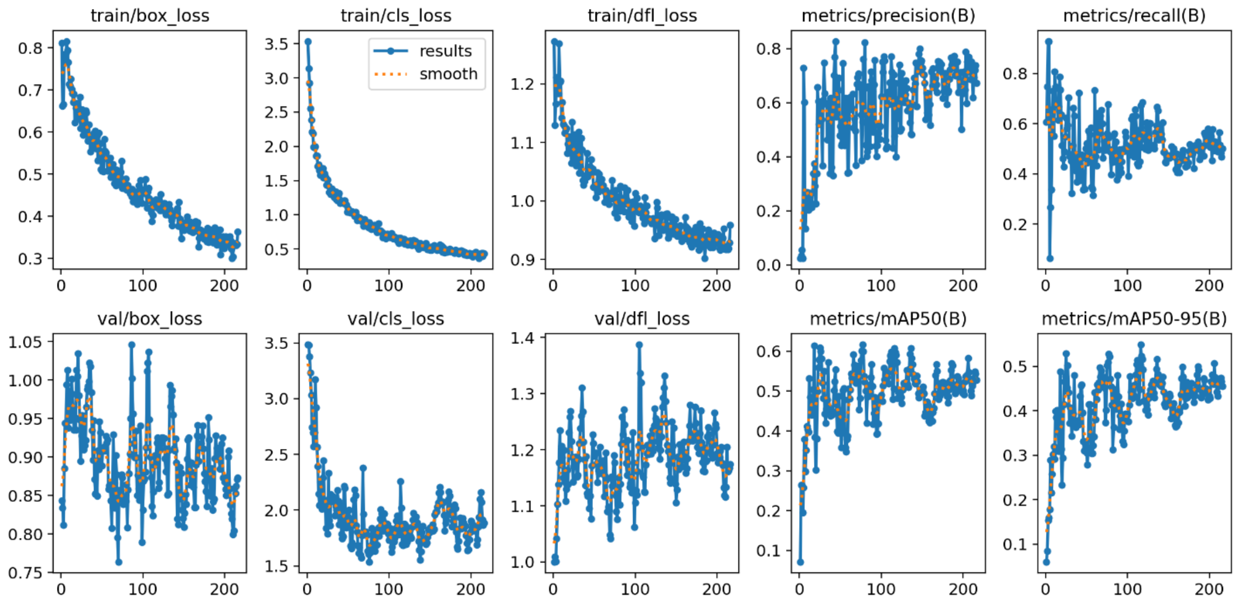
Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
YOLO11n	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x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

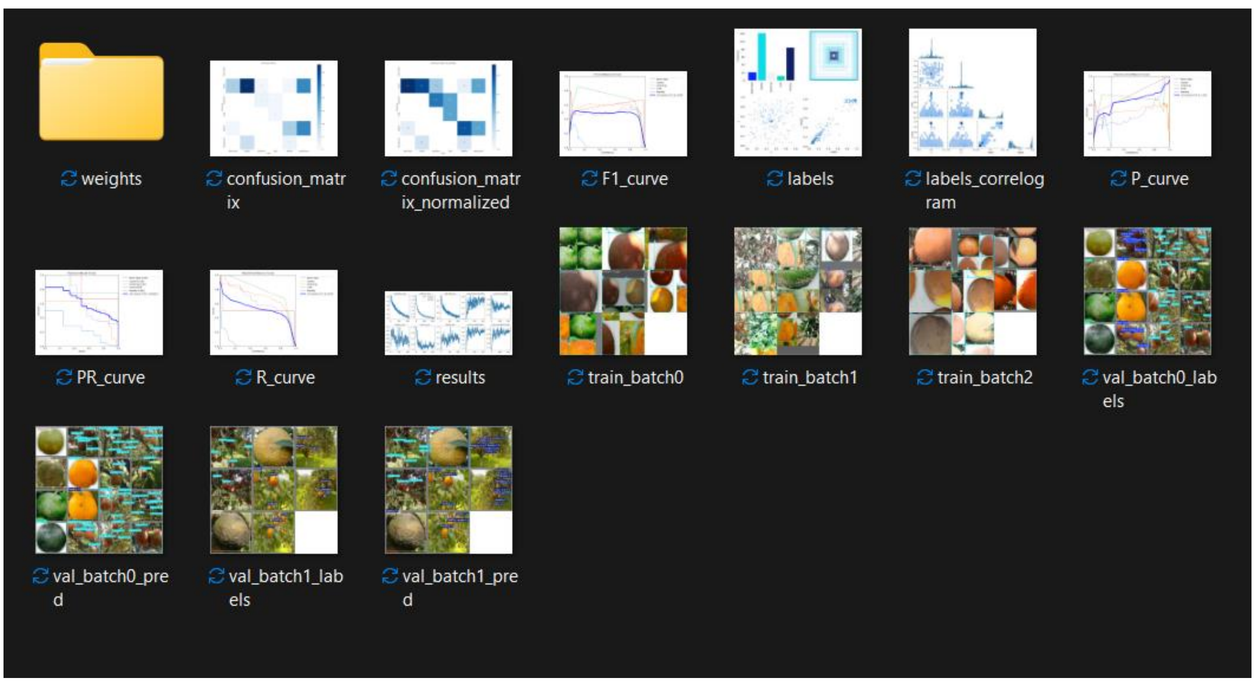
- **mAP^{val}** values refer to single-model single-scale performance on the COCO val2017 dataset. See YOLO



Initial training result



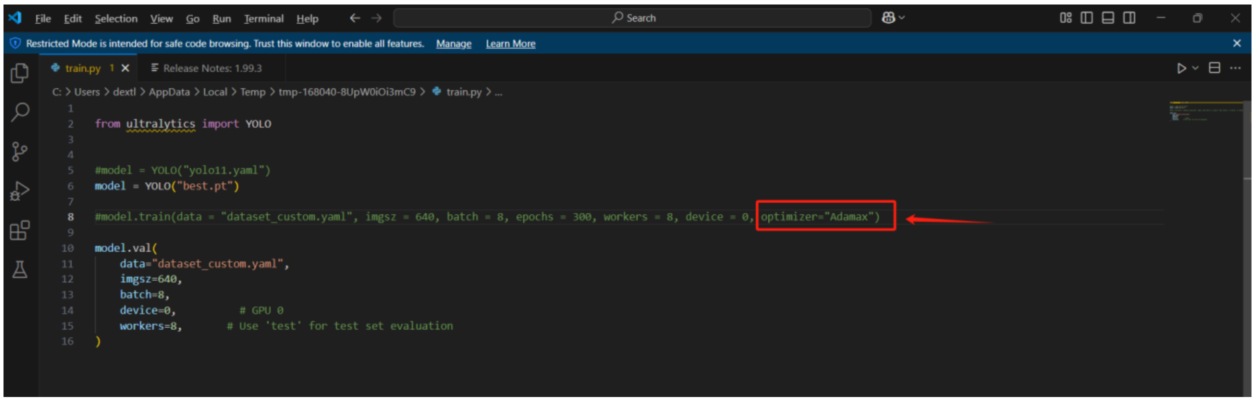




Introducing the Adamax Optimizer

Adamax is an optimizer that builds upon the Adam optimization algorithm. It uses the infinity norm (maximum) of the gradient instead of the L2 norm. This provides superior stability and robustness against sparse gradients, making it suitable for tasks with noisy or irregular updates.

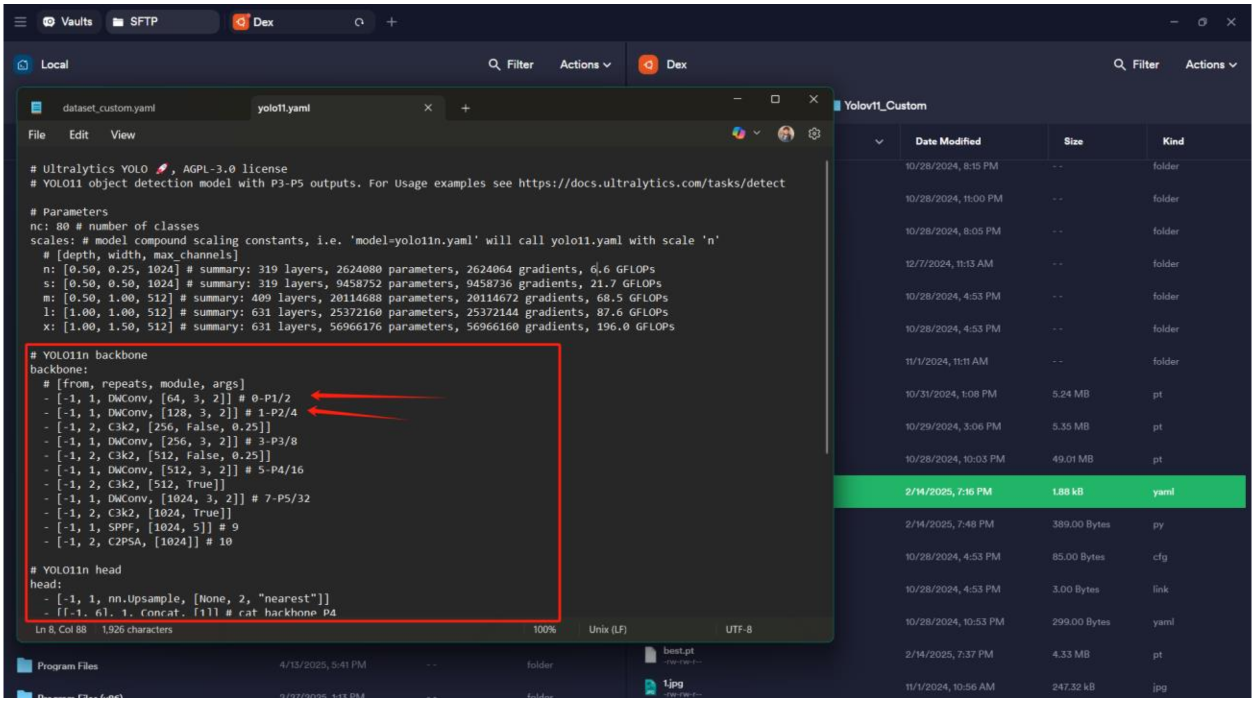
The Adamax optimizer in YOLOv11 was modified in the training script and configuration files.



The parameters for the Adamax optimizer, such as learning rate, beta1, and beta2 were modified in the configuration python file. Other parameters such as batch, epochs, were continuously modified.

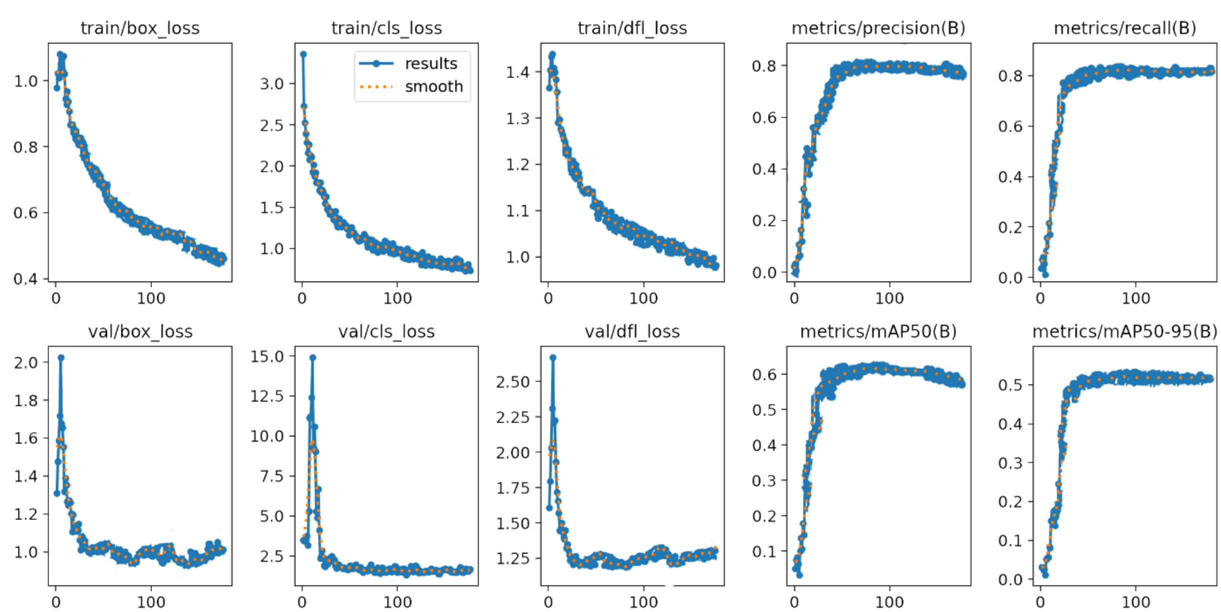
```
# In trainer.py
class MyAdamaxOptimizer(torch.optim.Optimizer):
    # ... (implementation of Adamax algorithm) ...

def get_optimizer(model, config):
    if config['optimizer'] == 'Adamax':
        return MyAdamaxOptimizer(model.parameters(), lr=config['lr'], beta1=conf


# In default.yaml
# ...
# optimizer: "Adamax"
# lr: 0.001
# beta1: 0.9
# beta2: 0.999
# ...
```

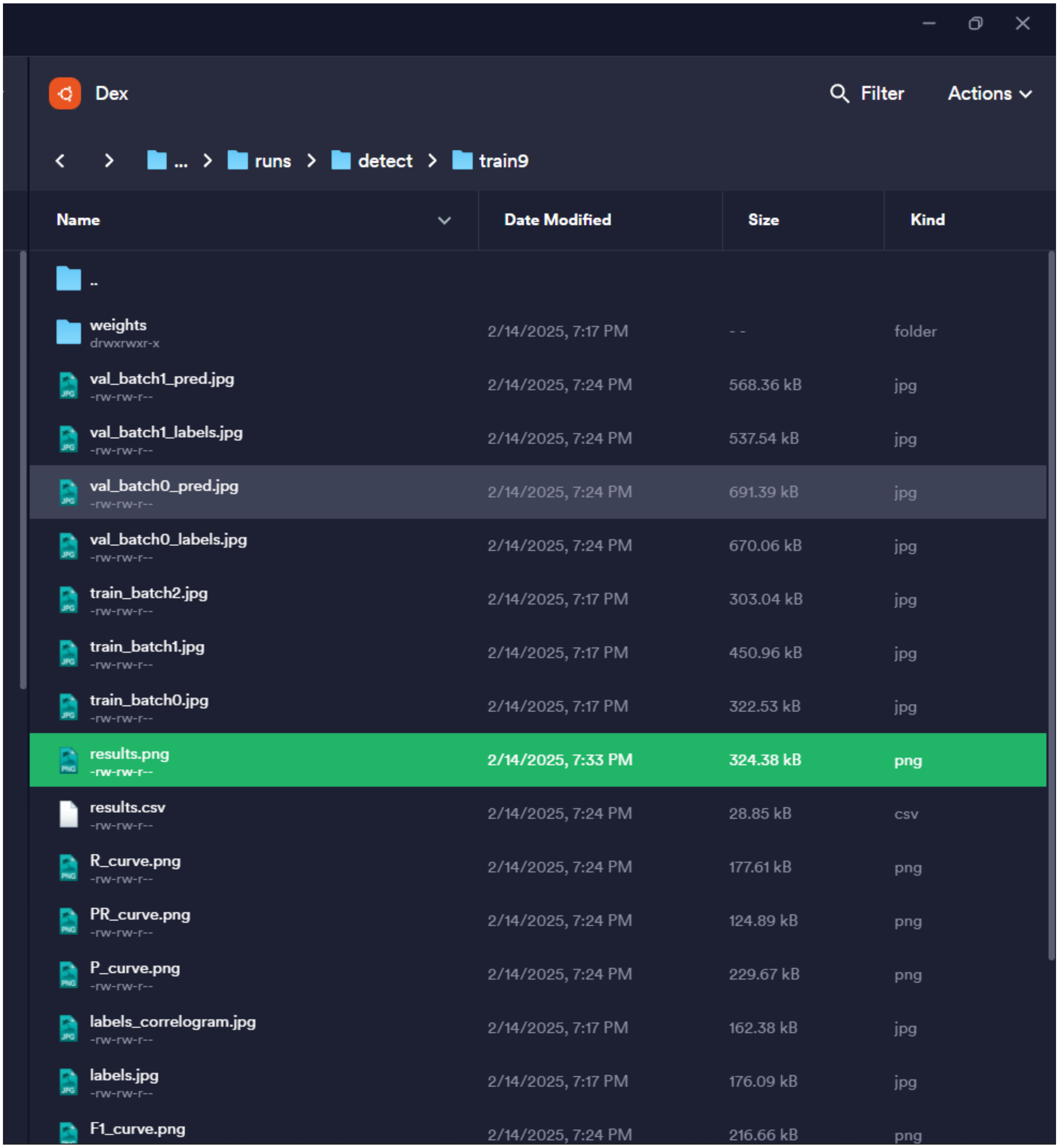
Depthwise convolution layers



The depthwise convolution layer applies a single convolutional filter per input channel, significantly reducing computational cost while maintaining performance.

Enhanced training result




Dex
Filter
Actions
... > runs > detect > train9
Name
Date Modified
Size
Kind
..
weights
drwxrwxr-x
2/14/2025, 7:17 PM
- -
folder
val_batch1_pred.jpg
-rw-rw-r--
2/14/2025, 7:24 PM
568.36 kB
jpg
val_batch1_labels.jpg
-rw-rw-r--
2/14/2025, 7:24 PM
537.54 kB
jpg
val_batch0_pred.jpg
-rw-rw-r--
2/14/2025, 7:24 PM
691.39 kB
jpg
val_batch0_labels.jpg
-rw-rw-r--
2/14/2025, 7:24 PM
670.06 kB
jpg
train_batch2.jpg
-rw-rw-r--
2/14/2025, 7:17 PM
303.04 kB
jpg
train_batch1.jpg
-rw-rw-r--
2/14/2025, 7:17 PM
450.96 kB
jpg
train_batch0.jpg
-rw-rw-r--
2/14/2025, 7:17 PM
322.53 kB
jpg
results.png
-rw-rw-r--
2/14/2025, 7:33 PM
324.38 kB
png
results.csv
-rw-rw-r--
2/14/2025, 7:24 PM
28.85 kB
csv
R_curve.png
-rw-rw-r--
2/14/2025, 7:24 PM
177.61 kB
png
PR_curve.png
-rw-rw-r--
2/14/2025, 7:24 PM
124.89 kB
png
P_curve.png
-rw-rw-r--
2/14/2025, 7:24 PM
229.67 kB
png
labels_correlogram.jpg
-rw-rw-r--
2/14/2025, 7:17 PM
162.38 kB
jpg
labels.jpg
-rw-rw-r--
2/14/2025, 7:17 PM
176.09 kB
jpg
F1_curve.png
2/14/2025, 7:24 PM
216.66 kB
png