

## 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

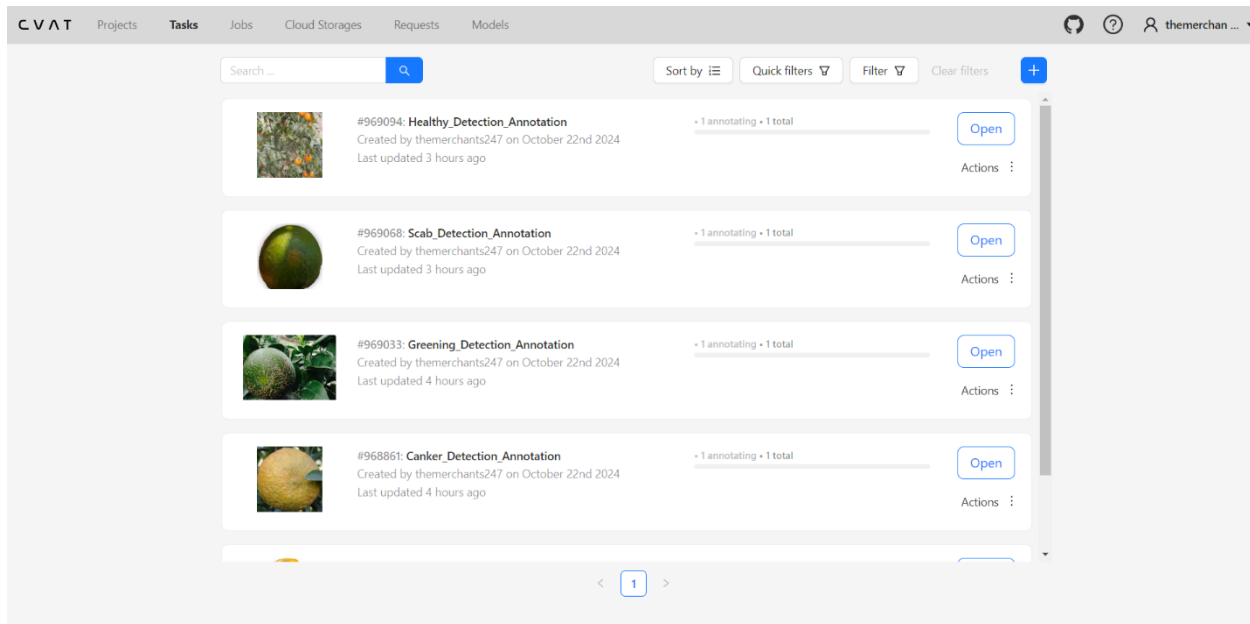
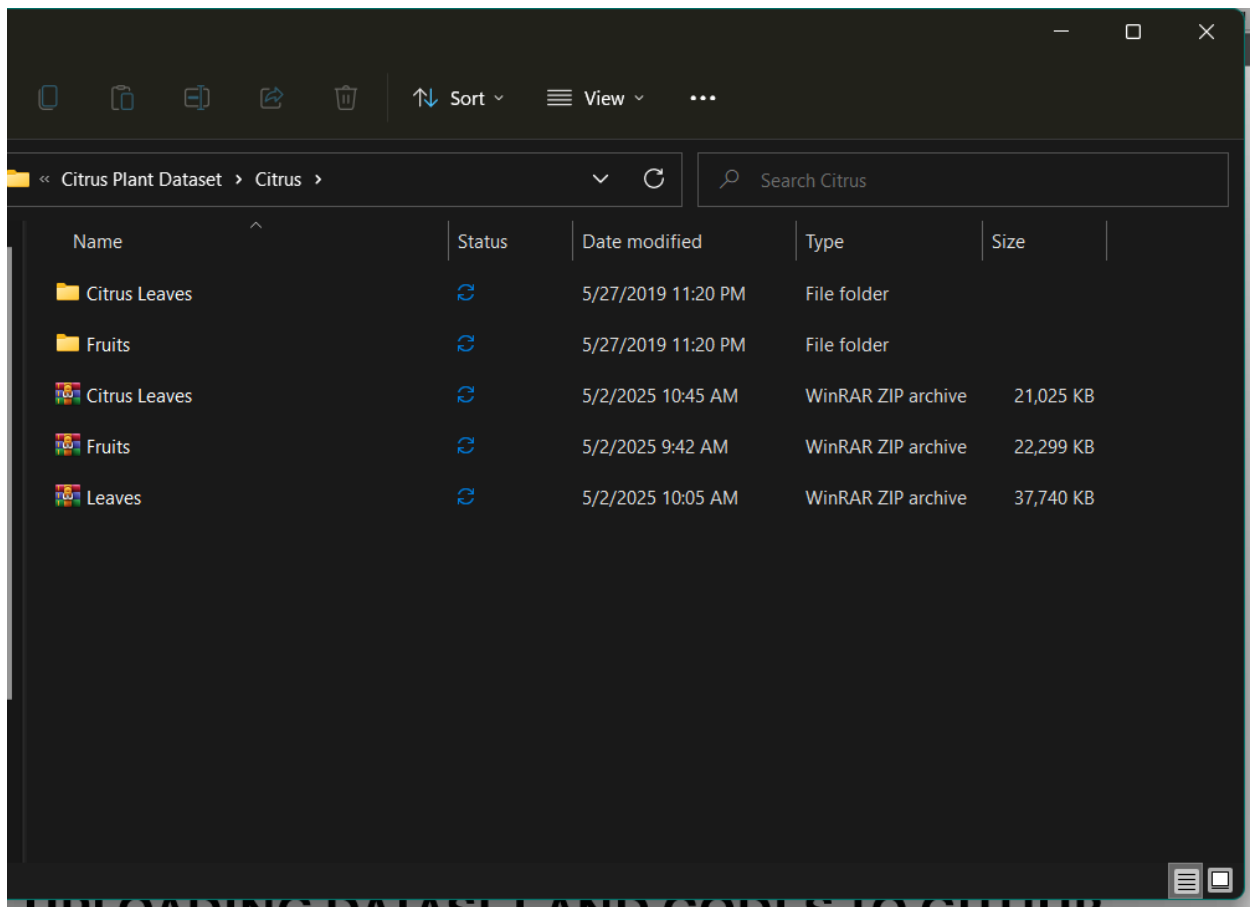
The manuscript introduces the YOLOv11 Nano model, highlighting its lightweight design and adaptability:

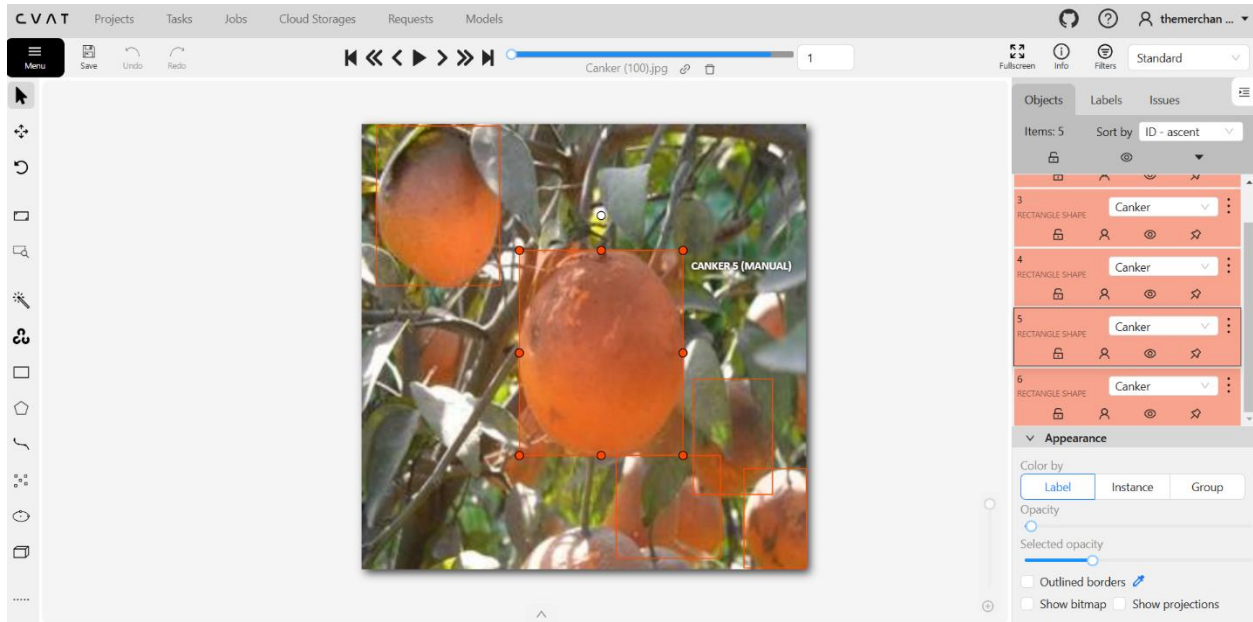
- **Depthwise Convolution Layers:** Incorporated to reduce model complexity, achieving a 4.2× reduction in FLOPs (floating-point operations per second) and a 68% smaller model size compared to traditional approaches.
- **Optimization Strategy:** Replacing the default Adam optimizer with Adamax (a variant 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.





-**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:

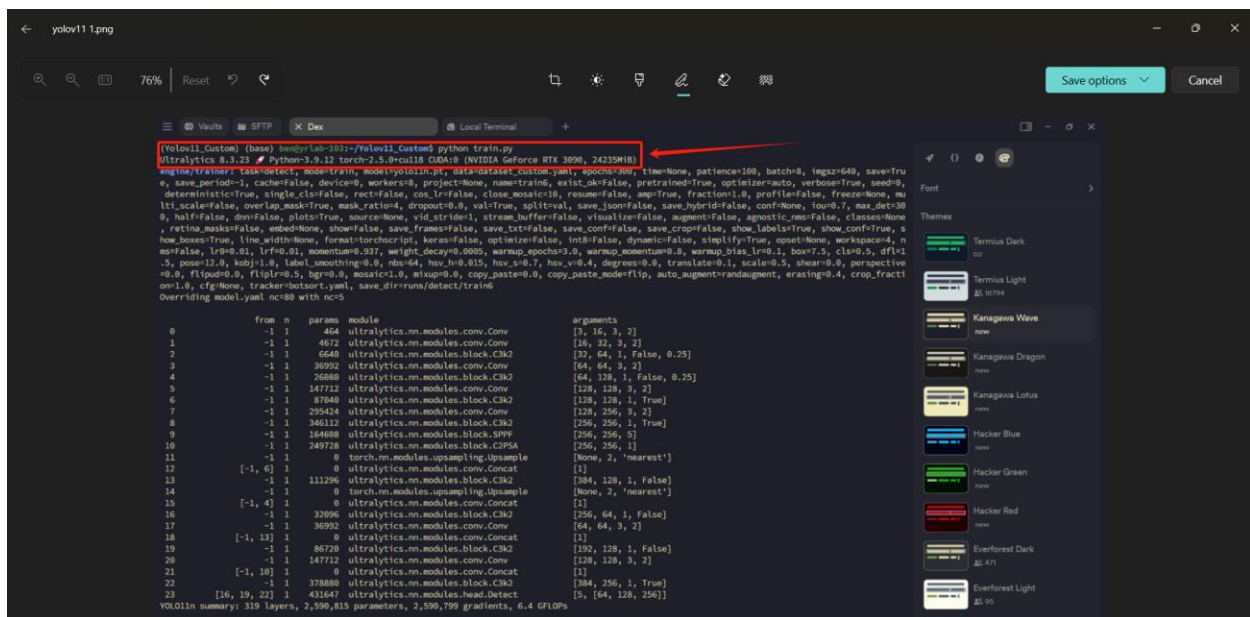
- **Mean Average Precision (mAP):** The Adamax-optimized YOLOv11 Nano achieved **95% mAP accuracy**, surpassing baseline models in precision.
- **Inference Latency:** A 21% reduction in inference latency compared to YOLOv8 Nano is reported, demonstrating the model's suitability for real-time applications.
- **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.



```
(Yolov11 Custom) (base) benji@lan-3911:~/Yolov11_Custom$ python train.py
Ultralytics 8.3.23 Python-3.9.12 torch-2.5.0+cu118 CUDA:0 (NVIDIA GeForce RTX 3090, 24239MiB)
#name/trainer: task=detect, mode=train, model=yolov11n.pt, data=dataset.yaml, epochs=100, time=None, patience=100, batch=16, imgsz=640, save=True, save_period=1, cache_swap=0, device=0, workers=8, project=None, name=train, exist_ok=False, pretrained=True, optimizer=auto, verbose=True, seed=0, deterministic=True, single_cls=False, rect=True, cos_lr=False, close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False, freeze=None, multiscale=False, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True, split_val=True, save_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=30, half=False, den=False, plot=True, source=None, vid_stride=1, stream_buffer=False, visualize=False, augment=False, agnostic_nms=False, classes=None, retina_masks=False, embeds=None, show=False, save_frames=False, save_iter=False, save_conf=False, save_crop=False, show_labels=True, show_conf=True, show_boxes=True, line_width=None, format=torchscript, keras=False, optimize=False, int8=False, dynamic=False, simplify=True, opset=None, workspace=4, nms=False, lr=0.01, lr_f=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, bb=64, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.2, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.0, hgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.0, auto_augment=randaugment, erasing=0.4, crop_fraction=1.0, cfg=None, tracker=botsort, save_dir=runs/detect/train6, overriding model.yaml nc=88 with nc=9

from n      params  module                                arguments
0      -1      1      464      ultralytics.nn.modules.conv.Conv      [3, 16, 3, 2]
1      -1      1      4672     ultralytics.nn.modules.conv.Conv      [16, 32, 3, 2]
2      -1      1      6640     ultralytics.nn.modules.block.C3k2     [32, 64, 1, False, 0.25]
3      -1      1      36992    ultralytics.nn.modules.conv.Conv      [64, 64, 3, 2]
4      -1      1      26880    ultralytics.nn.modules.block.C3k2     [64, 128, 1, False, 0.25]
5      -1      1      147732   ultralytics.nn.modules.conv.Conv      [128, 128, 3, 2]
6      -1      1      87840    ultralytics.nn.modules.block.C3k2     [128, 128, 1, True]
7      -1      1      295424   ultralytics.nn.modules.conv.Conv      [128, 256, 3, 2]
8      -1      1      246112   ultralytics.nn.modules.block.C3k2     [256, 256, 1, True]
9      -1      1      164800   ultralytics.nn.modules.block.SPP      [256, 256, 5]
10     -1      1      249728   ultralytics.nn.modules.block.C2PSA   [256, 256, 1]
11     -1      1      0      torch.nn.modules.upsampling.Upsample  [None, 2, 'nearest']
12     [-1, 4]  1      0      ultralytics.nn.modules.conv.Concat    [1]
13     -1      1      111296   ultralytics.nn.modules.block.C3k2     [384, 128, 1, False]
14     -1      1      0      torch.nn.modules.upsampling.Upsample  [None, 2, 'nearest']
15     [-1, 4]  1      0      ultralytics.nn.modules.conv.Concat    [1]
16     -1      1      32896   ultralytics.nn.modules.block.C3k2     [256, 64, 1, False]
17     -1      1      36992    ultralytics.nn.modules.conv.Conv      [64, 64, 3, 2]
18     [-1, 13] 1      0      ultralytics.nn.modules.conv.Concat    [1]
19     -1      1      86720    ultralytics.nn.modules.block.C3k2     [192, 128, 1, False]
20     -1      1      147732   ultralytics.nn.modules.conv.Conv      [128, 128, 3, 2]
21     [-1, 18] 1      0      ultralytics.nn.modules.conv.Concat    [1]
22     -1      1      378880   ultralytics.nn.modules.block.C3k2     [384, 256, 1, True]
23     [16, 19, 22] 1 431647 ultralytics.nn.modules.head.Detect    [5, [64, 128, 256]]

YOLOv11n summary: 319 layers, 2,590,815 parameters, 2,590,799 gradients, 6.4 GFLOPs
```

### 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

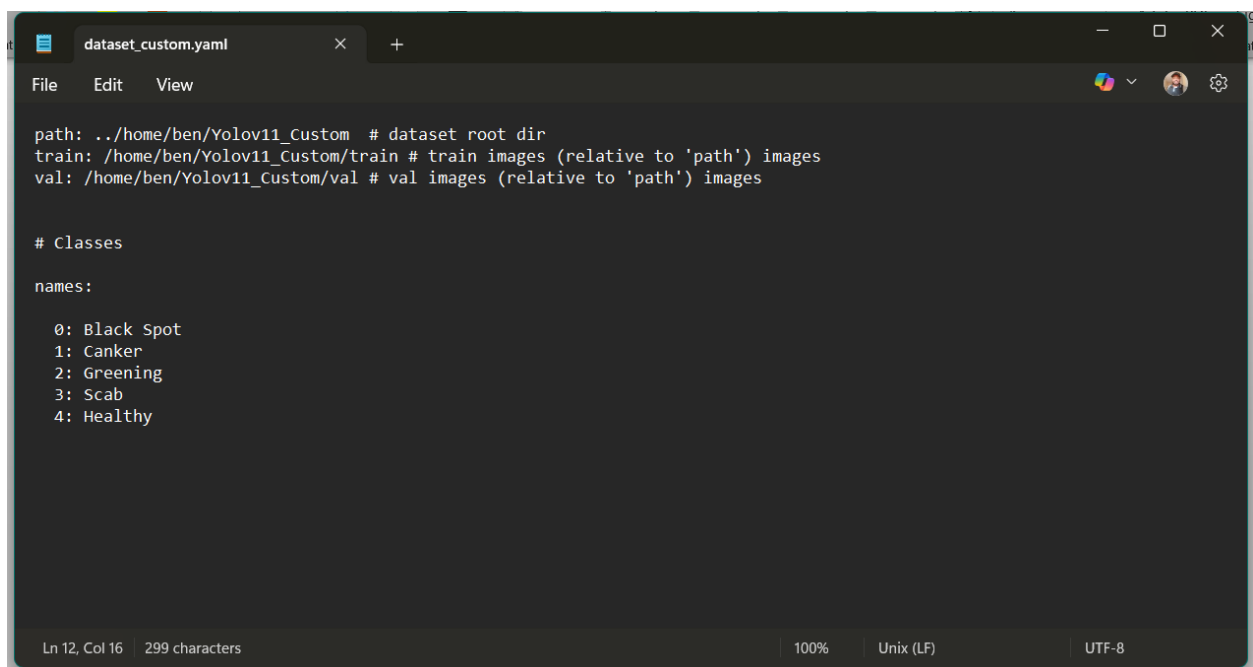
Using the Termius's SFTP client to upload the dataset to the server.

```
../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



```
dataset_custom.yaml

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 Yolov11 nano 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.

**Detect**

**Segment**

**Classify**

**Pose**

**OBB**

**Track**

▼ 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 <sup>val</sup> 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	params (M)	FLOPs (B)
<a href="#">YOLO11n</a>	640	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
<a href="#">YOLO11s</a>	640	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
<a href="#">YOLO11m</a>	640	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
<a href="#">YOLO11l</a>	640	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
<a href="#">YOLO11x</a>	640	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

• mAP<sup>val</sup> values refer to single-model single-scale performance on the [COCO val2017](#) dataset. See [YOLO](#)

Dex

home > ben > YOLOv11\_Custom

Name	Date Modified	Size	Kind
..			
val	10/28/2024, 11:00 PM	--	folder
ultralytics	10/28/2024, 8:15 PM	--	folder
train	10/28/2024, 11:00 PM	--	folder
share	10/28/2024, 8:05 PM	--	folder
runs	12/7/2024, 11:13 AM	--	folder
lib	10/28/2024, 4:53 PM	--	folder
include	10/28/2024, 4:53 PM	--	folder
bin	11/1/2024, 11:11 AM	--	folder
yolo11n.pt	10/31/2024, 1:08 PM	5.24 MB	pt
yolo11n.pt	10/29/2024, 3:06 PM	5.35 MB	pt
yolo11l.pt	10/28/2024, 10:03 PM	49.01 MB	pt
yolo11.yaml	2/14/2025, 7:16 PM	1.88 kB	yaml
train.py	2/14/2025, 7:48 PM	389.00 Bytes	py
pyyaml.cfg	10/28/2024, 4:53 PM	85.00 Bytes	cfg
lib64	10/28/2024, 4:53 PM	3.00 Bytes	link
dataset_custom.yaml	10/28/2024, 10:53 PM	299.00 Bytes	yaml

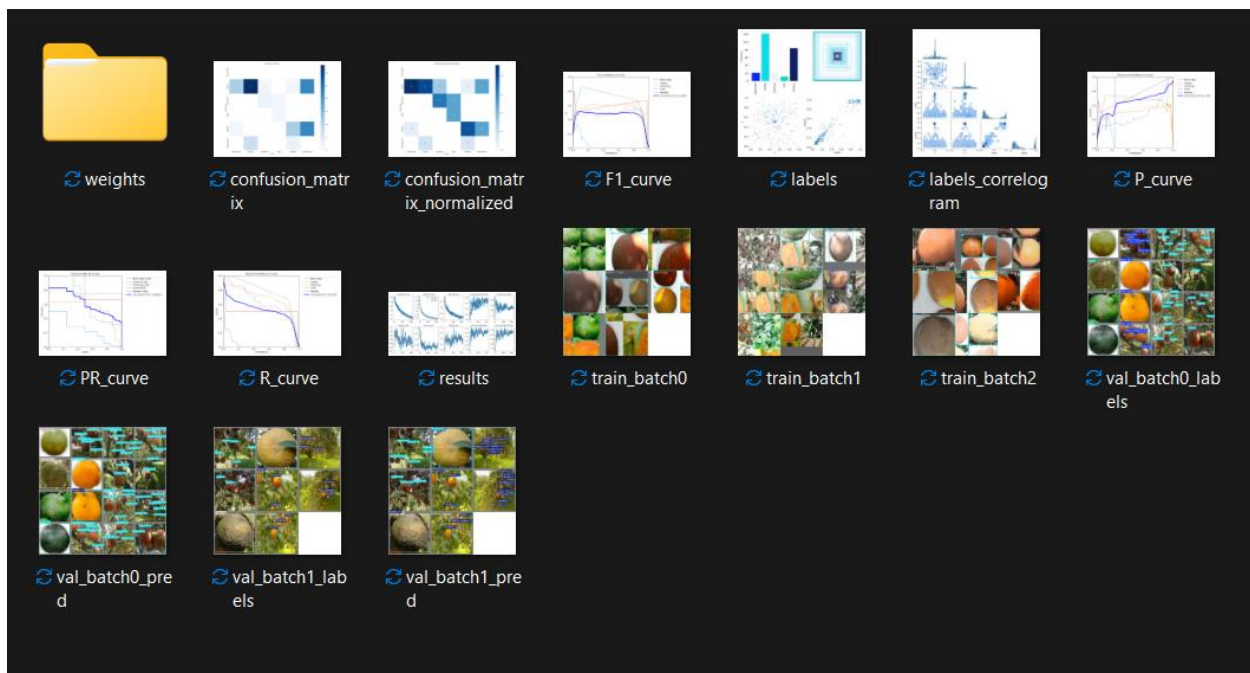
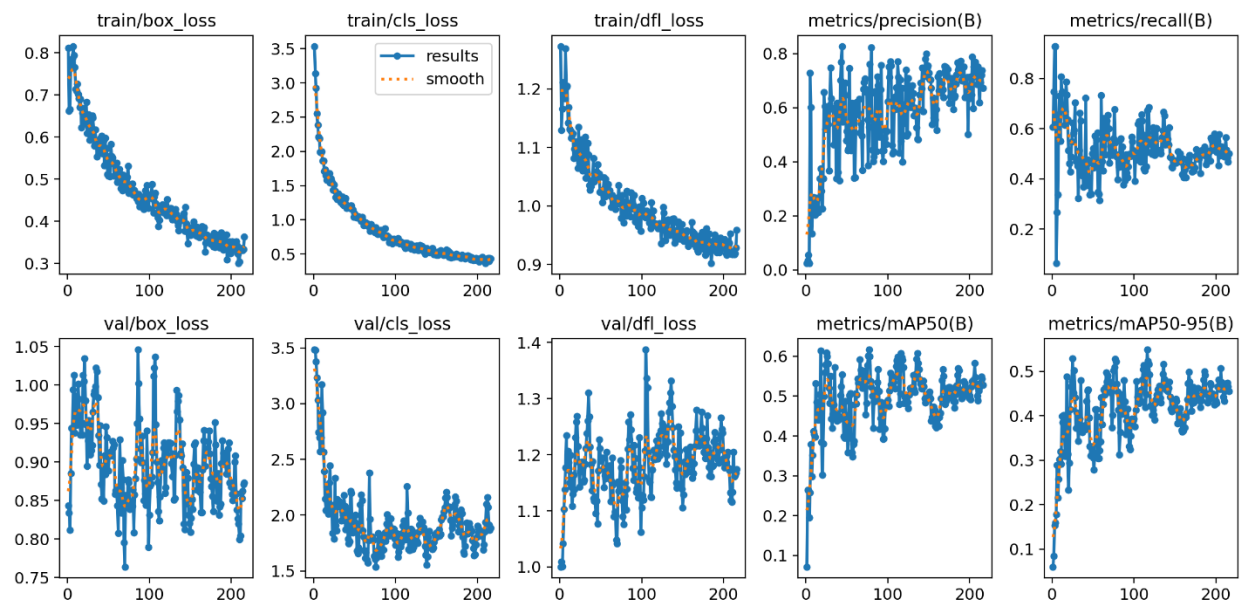
Initial training result

Dex

... > runs > detect > train3

Name	Date Modified	Size	Kind
..			
weights	10/28/2024, 11:00 PM	--	folder
val_batch0_pred.jpg	10/28/2024, 11:09 PM	668.34 kB	jpg
val_batch0_labels.jpg	10/28/2024, 11:09 PM	670.06 kB	jpg
train_batch722.jpg	10/28/2024, 11:08 PM	473.48 kB	jpg
train_batch721.jpg	10/28/2024, 11:08 PM	376.42 kB	jpg
train_batch720.jpg	10/28/2024, 11:08 PM	526.18 kB	jpg
train_batch2.jpg	10/28/2024, 11:00 PM	508.83 kB	jpg
train_batch1.jpg	10/28/2024, 11:00 PM	447.34 kB	jpg
train_batch0.jpg	10/28/2024, 11:00 PM	459.76 kB	jpg
results.png	10/28/2024, 11:09 PM	310.97 kB	png
results.csv	10/28/2024, 11:08 PM	12.41 kB	csv
R_curve.png	10/28/2024, 11:09 PM	193.67 kB	png
PR_curve.png	10/28/2024, 11:09 PM	138.09 kB	png
P_curve.png	10/28/2024, 11:09 PM	211.61 kB	png
labels_correlogram.jpg	10/28/2024, 11:00 PM	162.38 kB	jpg
labels.jpg	10/28/2024, 11:00 PM	176.09 kB	jpg

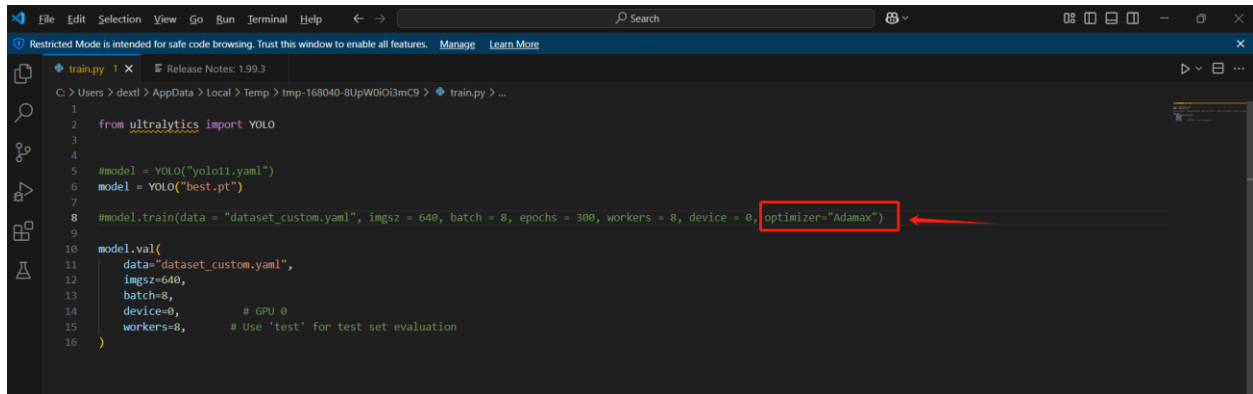




## Introducing the Adamax Optimizer

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.



```
1 from ultralytics import YOLO
2
3
4
5 #model = YOLO("yolo11.yaml")
6 model = YOLO("best.pt")
7
8 #model.train(data = "dataset_custom.yaml", imgsz = 640, batch = 8, epochs = 300, workers = 8, device = 0, optimizer="Adamax")
9
10 model.val(
11     data="dataset_custom.yaml",
12     imgsz=640,
13     batch=8,
14     device=0,          # GPU 0
15     workers=8,         # Use 'test' for test set evaluation
16 )
```

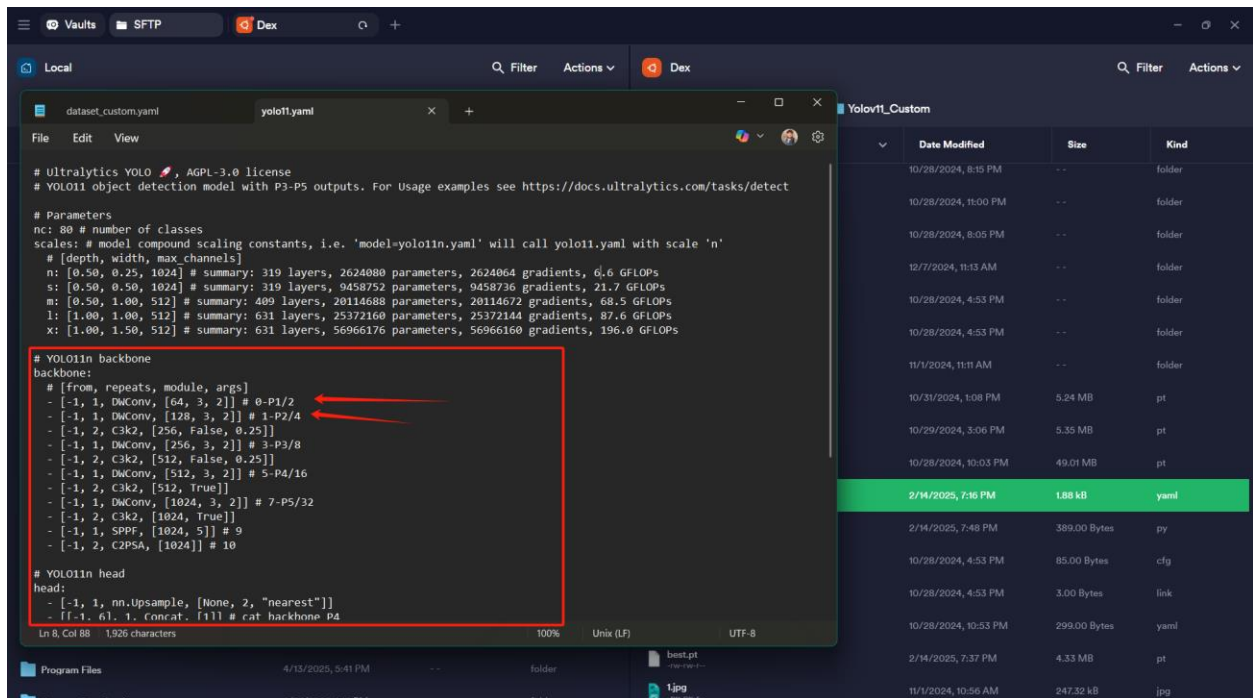
The parameters for the Adamax optimizer, such as learning rate, beta1, and beta2 were modified in the configuration python file. Other parameters such as batch, epochs, were continuously modified.

```
# 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=config

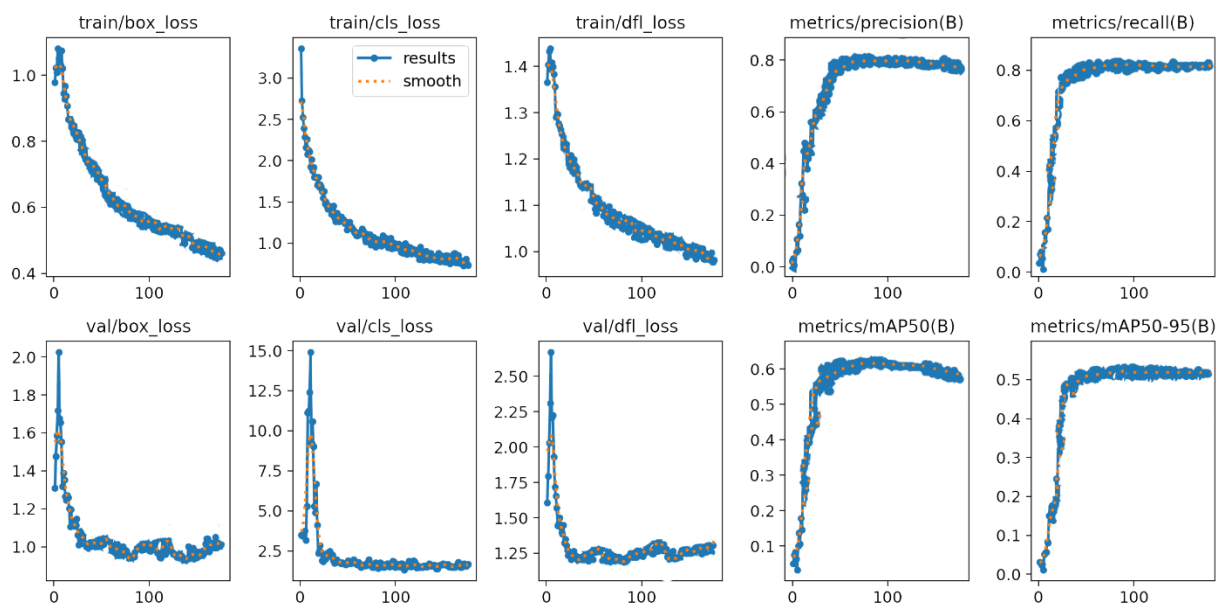
# In default.yaml
# ...
# optimizer: "Adamax"
# lr: 0.001
# beta1: 0.9
# beta2: 0.999
# ...
```

## 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 -rw-rw-r--	2/14/2025, 7:24 PM	216.66 kB	png