YOLO-11 dustbin overflow detection

## Project Description:

This is a personal computer vision project developed using Python and the YOLOv11 object detection framework. The aim of the project is to detect the status of dustbins in public or household environments.

The model is trained to classify dustbins into two categories:

Empty Dustbin

Overflowing Dustbin (with garbage also detected outside the bin)

Additionally, the project also identifies if any garbage is lying outside the dustbin, which is important for monitoring cleanliness in smart city or waste management systems.

YOLOv11's advanced hyperparameter tuning was used to improve the model’s accuracy and generalization. The training was done using a custom-labeled dataset, and all implementation was done purely in Python.

## Developed By:

Mayank Manjhi  
 (Personal Research Project)

[Project Description: 1](#_Toc2008781100)

[️Technologies Used: 1](#_Toc647929924)

[Developed By: 1](#_Toc972117403)

[Definition 3](#_Toc564679372)

[Installation 3](#_Toc1550311569)

[Detection Model 4](#_Toc90993152)

[Image segmentation model 6](#_Toc614864132)

[Image classification 8](#_Toc883846911)

[Pose estimation 11](#_Toc1225205487)

[Oriented object detection 13](#_Toc474056313)

[Some hyperparameters: 15](#_Toc1028067518)

[Conclusion 18](#_Toc560538993)

Yolo-model

Definition : Ultralytics YOLO is the latest advancement in the acclaimed YOLO (You Only Look Once) series for real-time object detection and image segmentation. It builds on previous versions by introducing new features and improvements for enhanced performance, flexibility, and efficiency. YOLO supports various [vision AI tasks](https://docs.ultralytics.com/tasks/) such as detection, segmentation, pose estimation, tracking, and classification. Its state-of-the-art architecture ensures superior speed and accuracy, making it suitable for diverse applications, including edge devices and cloud APIs.

Installation: Getting started with YOLO is quick and straightforward. You can install the Ultralytics package using [pip](https://pypi.org/project/ultralytics/) and get up and running in minutes. Here's a basic installation command:

Cmnd— pip install ultralytics

If you want full control on dependencies, then install the ultralytics without dependencies and install the library according to your project

Cmnd- pip install ultralytics --no-deps

Some libraries imp libraries

dependencies

"numpy>=1.23.0",

"matplotlib>=3.3.0",

"opencv-python>=4.6.0",

"pillow>=7.1.2",

"pyyaml>=5.3.1",

"requests>=2.23.0",

"scipy>=1.4.1",

"torch>=1.8.0",

"torch>=1.8.0,!=2.4.0; sys\_platform == 'win32'", # Windows CPU errors w/ 2.4.0 https://github.com/ultralytics/ultralytics/issues/15049

"torchvision>=0.9.0",

"tqdm>=4.64.0", # progress bars

"psutil", # system utilization

"py-cpuinfo", # display CPU info

"pandas>=1.1.4",

"ultralytics-thop>=2.0.0", # FLOPs computation https://github.com/ultralytics/thop

This method gives full control but requires careful management of dependencies. Ensure all required packages are installed with compatible versions [click here for more](https://docs.ultralytics.com/quickstart/#custom-installation-methods)

Model: Ultralytics supports a wide range of YOLO models, from early versions like [YOLOv3](https://docs.ultralytics.com/models/yolov3/) to the latest [YOLO11](https://docs.ultralytics.com/models/yolo11/). The tables below showcase YOLO11 models pretrained on the [COCO](https://docs.ultralytics.com/datasets/detect/coco/) dataset for [Detection](https://docs.ultralytics.com/tasks/detect/), [Segmentation](https://docs.ultralytics.com/tasks/segment/), and [Pose Estimation](https://docs.ultralytics.com/tasks/pose/). Additionally, [Classification](https://docs.ultralytics.com/tasks/classify/) models pretrained on the [ImageNet](https://docs.ultralytics.com/datasets/classify/imagenet/)

Detection Model: Detection is the primary task supported by YOLO11. It involves identifying objects in an image or video frame and drawing bounding boxes around them. The detected objects are classified into different categories based on their features. YOLO11 can detect multiple objects in a single image or video frame with high accuracy and speed, making it ideal for real-time applications like surveillance systems and autonomous vehicles.YOLO11 pretrained Detect models are shown here

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 |

mAPval values are for single-model single-scale on COCO val2017 dataset.  
Reproduce by yolo val detect data=coco.yaml device=0

Speed averaged over COCO val images using an Amazon EC2 P4d instance.  
Reproduce by yolo val detect data=coco.yaml batch=1 device=0|cpu

Yolo11n, n means nano it’s a lightweight model use if you have low power machine

Yolo11s, s means small its’s a small model but faster the yolo11n and more accurate then

Yolo11m, m means medium more powerful then prev two models and need a powerfull machine to run this model

And other two models yolo11l and yolo11x they both are the most powerful model need most powerful machine to run model

Basically, the yolo divide the model into category n,s,m,l,x not only in detection model but also for image segmentation, classification, pose estimation and obb model

Image segmentation model: Segmentation takes object detection further by segmenting an image into different regions based on content. Each region is assigned a label, providing pixel-level precision for applications such as medical imaging, agricultural analysis, and manufacturing quality control. YOLO11 implements a variant of the U-Net architecture to perform efficient and accurate segmentation.

Instance segmentation goes a step further than object detection and involves identifying individual objects in an image and segmenting them from the rest of the image.

The output of an instance segmentation model is a set of masks or contours that outline each object in the image, along with class labels and confidence scores for each object. Instance segmentation is useful when you need to know not only where objects are in an image, but also what their exact shape is.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **size** **(pixels)** | **mAPbox** **50-95** | **mAPmask** **50-95** | **Speed** **CPU ONNX** **(ms)** | **Speed** **T4 TensorRT10** **(ms)** | **params** **(M)** | **FLOPs** **(B)** |
| [YOLO11n-seg](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n-seg.pt) | 640 | 38.9 | 32.0 | 65.9 ± 1.1 | 1.8 ± 0.0 | 2.9 | 10.4 |
| [YOLO11s-seg](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s-seg.pt) | 640 | 46.6 | 37.8 | 117.6 ± 4.9 | 2.9 ± 0.0 | 10.1 | 35.5 |
| [YOLO11m-seg](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11m-seg.pt) | 640 | 51.5 | 41.5 | 281.6 ± 1.2 | 6.3 ± 0.1 | 22.4 | 123.3 |
| [YOLO11l-seg](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11l-seg.pt) | 640 | 53.4 | 42.9 | 344.2 ± 3.2 | 7.8 ± 0.2 | 27.6 | 142.2 |
| [YOLO11x-seg](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11x-seg.pt) | 640 | 54.7 | 43.8 | 664.5 ± 3.2 | 15.8 ± 0.7 | 62.1 | 319.0 |

mAPval values are for single-model single-scale on COCO val2017 dataset.  
Reproduce by yolo val segment data=coco.yaml device=0

Speed averaged over COCO val images using an Amazon EC2 P4d instance.  
Reproduce by yolo val segment data=coco.yaml batch=1 device=0|cpu

Image classification: The output of an image classifier is a single class label and a confidence score. Image classification is useful when you need to know only what class an image belongs to and don't need to know where objects of that class are located or what their exact shape is.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **size** **(pixels)** | **acc** **top1** | **acc** **top5** | **Speed** **CPU ONNX** **(ms)** | **Speed** **T4 TensorRT10** **(ms)** | **params** **(M)** | **FLOPs** **(B) at 224** |
| [YOLO11n-cls](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n-cls.pt) | 224 | 70.0 | 89.4 | 5.0 ± 0.3 | 1.1 ± 0.0 | 1.6 | 0.5 |
| [YOLO11s-cls](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s-cls.pt) | 224 | 75.4 | 92.7 | 7.9 ± 0.2 | 1.3 ± 0.0 | 5.5 | 1.6 |
| [YOLO11m-cls](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11m-cls.pt) | 224 | 77.3 | 93.9 | 17.2 ± 0.4 | 2.0 ± 0.0 | 10.4 | 5.0 |
| [YOLO11l-cls](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11l-cls.pt) | 224 | 78.3 | 94.3 | 23.2 ± 0.3 | 2.8 ± 0.0 | 12.9 | 6.2 |
| [YOLO11x-cls](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11x-cls.pt) | 224 | 79.5 | 94.9 | 41.4 ± 0.9 | 3.8 ± 0.0 | 28.4 | 13.7 |

acc values are model accuracy on the ImageNet dataset validation set.  
Reproduce by yolo val classify data=path/to/ImageNet device=0

Speed averaged over ImageNet val images using an Amazon EC2 P4d instance.  
Reproduce by yolo val classify data=path/to/ImageNet batch=1 device=0|cpu

Pose estimation : detects specific keypoints in images or video frames to track movements or estimate poses. These keypoints can represent human joints, facial features, or other significant points of interest. YOLO11 excels at keypoint detection with high accuracy and speed, making it valuable for [fitness applications](https://www.ultralytics.com/blog/ai-in-our-day-to-day-health-and-fitness), [sports analytics](https://www.ultralytics.com/blog/exploring-the-applications-of-computer-vision-in-sports), and [human-computer interaction](https://www.ultralytics.com/blog/custom-training-ultralytics-yolo11-for-dog-pose-estimation).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **size** **(pixels)** | **mAPpose** **50-95** | **mAPpose** **50** | **Speed** **CPU ONNX** **(ms)** | **Speed** **T4 TensorRT10** **(ms)** | **params** **(M)** | **FLOPs** **(B)** |
| [YOLO11n-pose](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n-pose.pt) | 640 | 50.0 | 81.0 | 52.4 ± 0.5 | 1.7 ± 0.0 | 2.9 | 7.6 |
| [YOLO11s-pose](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s-pose.pt) | 640 | 58.9 | 86.3 | 90.5 ± 0.6 | 2.6 ± 0.0 | 9.9 | 23.2 |
| [YOLO11m-pose](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11m-pose.pt) | 640 | 64.9 | 89.4 | 187.3 ± 0.8 | 4.9 ± 0.1 | 20.9 | 71.7 |
| [YOLO11l-pose](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11l-pose.pt) | 640 | 66.1 | 89.9 | 247.7 ± 1.1 | 6.4 ± 0.1 | 26.2 | 90.7 |
| [YOLO11x-pose](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11x-pose.pt) | 640 | 69.5 | 91.1 | 488.0 ± 13.9 | 12.1 ± 0.2 | 58.8 | 203.3 |

mAPval values are for single-model single-scale on COCO Keypoints val2017 dataset.  
Reproduce by yolo val pose data=coco-pose.yaml device=0

Speed averaged over COCO val images using an Amazon EC2 P4d instance.  
Reproduce by yolo val pose data=coco-pose.yaml batch=1 device=0|cpu

Oriented object detection: goes a step further than standard object detection by introducing an extra angle to locate objects more accurately in an image.

The output of an oriented object detector is a set of rotated bounding boxes that precisely enclose the objects in the image, along with class labels and confidence scores for each box. Oriented bounding boxes are particularly useful when objects appear at various angles, such as in aerial imagery, where traditional axis-aligned bounding boxes may include unnecessary background.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **size** **(pixels)** | **mAPtest** **50** | **Speed** **CPU ONNX** **(ms)** | **Speed** **T4 TensorRT10** **(ms)** | **params** **(M)** | **FLOPs** **(B)** |
| [YOLO11n-obb](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n-obb.pt) | 1024 | 78.4 | 117.6 ± 0.8 | 4.4 ± 0.0 | 2.7 | 17.2 |
| [YOLO11s-obb](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11s-obb.pt) | 1024 | 79.5 | 219.4 ± 4.0 | 5.1 ± 0.0 | 9.7 | 57.5 |
| [YOLO11m-obb](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11m-obb.pt) | 1024 | 80.9 | 562.8 ± 2.9 | 10.1 ± 0.4 | 20.9 | 183.5 |
| [YOLO11l-obb](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11l-obb.pt) | 1024 | 81.0 | 712.5 ± 5.0 | 13.5 ± 0.6 | 26.2 | 232.0 |
| [YOLO11x-obb](https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11x-obb.pt) | 1024 | 81.3 | 1408.6 ± 7.7 | 28.6 ± 1.0 | 58.8 | 520.2 |

mAPtest values are for single-model multiscale on DOTAv1 dataset.  
Reproduce by yolo val obb data=DOTAv1.yaml device=0 split=test and submit merged results to DOTA evaluation.

Speed averaged over DOTAv1 val images using an Amazon EC2 P4d instance.  
Reproduce by yolo val obb data=DOTAv1.yaml batch=1 device=0|cpu

Some hyperparameters: it's very important to know the parameters of yolo models

Because it helps to build a better model according to your needs.

1.Epoch Explained: In machine learning (ML), particularly in the context of training deep learning models, an epoch represents one complete pass of the entire training dataset through the learning algorithm. Training models is an iterative process where the model learns patterns by repeatedly processing the data. Epochs are a fundamental hyperparameter that defines the number of times the algorithm will work through the entire dataset, allowing the model to learn from each example within the data multiple times

During the training process, a model's internal parameters, or weights, are adjusted based on the errors it makes in its predictions. This adjustment typically happens using an optimization algorithm like Gradient Descent or its variants (e.g., Adam Optimizer). One epoch means that every sample in the training dataset has had an opportunity to update the model's internal parameters once. For large datasets, processing the entire dataset at once is computationally expensive, so the data is often divided into smaller chunks called batches.

2.Batch Size and GPU Utilization

When training models on large datasets, efficiently utilizing your GPU is key. Batch size is an important factor. It is the number of data samples that a machine learning model processes in a single training iteration. Using the maximum batch size supported by your GPU, you can fully take advantage of its capabilities and reduce the time model training takes. However, you want to avoid running out of GPU memory. If you encounter memory errors, reduce the batch size incrementally until the model trains smoothly.

With respect to YOLO11, you can set the batch\_size parameter in the [training configuration](https://docs.ultralytics.com/modes/train/) to match your GPU capacity. Also, setting batch=-1 in your training script will automatically determine the [batch size](https://www.ultralytics.com/glossary/batch-size) that can be efficiently processed based on your device's capabilities. By fine-tuning the batch size, you can make the most of your GPU resources and improve the overall training process.

3. Early Stopping

patience=5 means training will stop if there's no improvement in validation metrics for 5 consecutive epochs. Using this method ensures the training process remains efficient and achieves optimal performance without excessive computation.

4.Selecting an Optimizer

An optimizer is an algorithm that adjusts the weights of your neural network to minimize the loss function, which measures how well the model is performing. In simpler terms, the optimizer helps the model learn by tweaking its parameters to reduce errors. Choosing the right optimizer directly affects how quickly and accurately the model learns.

You can also fine-tune optimizer parameters to improve model performance. Adjusting the learning rate sets the size of the steps when updating parameters. For stability, you might start with a moderate learning rate and gradually decrease it over time to improve long-term learning. Additionally, setting the momentum determines how much influence past updates have on current updates. A common value for momentum is around 0.9. It generally provides a good balance.

Common Optimizers

Different optimizers have various strengths and weaknesses. Let's take a glimpse at a few common optimizers.

SGD (Stochastic Gradient Descent):

Updates model parameters using the gradient of the loss function with respect to the parameters.

Simple and efficient but can be slow to converge and might get stuck in local minima.

Adam (Adaptive Moment Estimation):

Combines the benefits of both SGD with momentum and RMSProp.

Adjusts the learning rate for each parameter based on estimates of the first and second moments of the gradients.

Well-suited for noisy data and sparse gradients.

Efficient and generally requires less tuning, making it a recommended optimizer for YOLO11.

RMSProp (Root Mean Square Propagation):

Adjusts the learning rate for each parameter by dividing the gradient by a running average of the magnitudes of recent gradients.

Helps in handling the vanishing gradient problem and is effective for recurrent neural networks.

For YOLO11, the optimizer parameter lets you choose from various optimizers, including SGD, Adam, AdamW, NAdam, RAdam, and RMSProp, or you can set it to auto for automatic selection based on model configuration.

They are some important parameters for training the model. If you want to learn more parameters the visit [click here](https://docs.ultralytics.com/modes/train/#resuming-interrupted-trainings)

Conclusion: Ultralytics YOLO11 supports multiple computer vision tasks, including detection, segmentation, classification, oriented object detection, and keypoint detection. Each task addresses specific needs in the computer vision landscape, from basic object identification to detailed pose analysis. By understanding the capabilities and applications of each task, you can select the most appropriate approach for your specific computer vision challenges and leverage YOLO11's powerful features to build effective solutions.