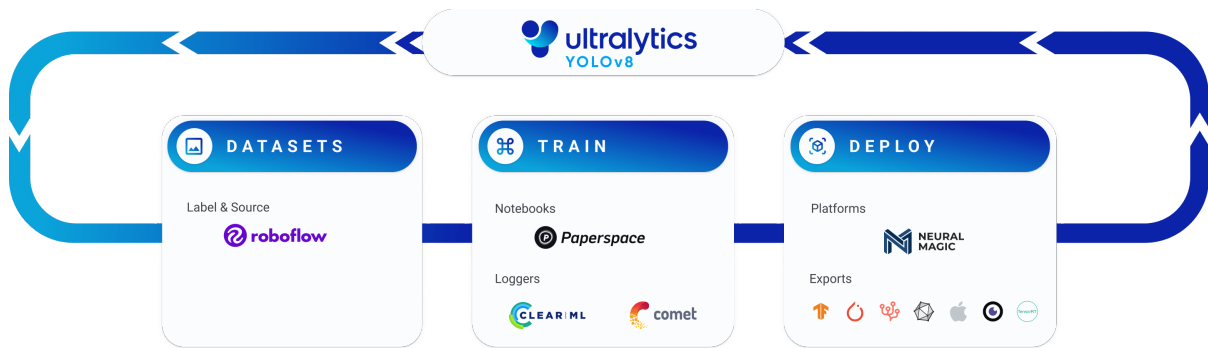


Predict



YOLOv8 **predict mode** can generate predictions for various tasks, returning either a list of `Results` objects or a memory-efficient generator of `Results` objects when using the streaming mode. Enable streaming mode by passing `stream=True` in the predictor's call method.

Predict

Return a list with `Stream=False`

```
inputs = [img, img] # list of numpy arrays
results = model(inputs) # list of Results objects

for result in results:
    boxes = result.boxes # Boxes object for bbox outputs
    masks = result.masks # Masks object for segmentation masks outputs
    probs = result.probs # Class probabilities for classification outputs
```

Return a generator with `Stream=True`


```
inputs = [img, img] # list of numpy arrays
results = model(inputs, stream=True) # generator of Results objects






for result in results:
    boxes = result.boxes # Boxes object for bbox outputs
    masks = result.masks # Masks object for segmentation masks outputs
    probs = result.probs # Class probabilities for classification outputs
```

Tip

Streaming mode with `stream=True` should be used for long videos or large predict sources, otherwise results will accumulate in memory and will eventually cause out-of-memory errors.

Sources

YOLOv8 can accept various input sources, as shown in the table below. This includes images, URLs, PIL images, OpenCV, numpy arrays, torch tensors, CSV files, videos, directories, globs, YouTube videos, and streams. The table indicates whether each source can be used in streaming mode with `stream=True`  and an example argument for each source.

source	model(arg)	type	notes
image	<code>'im.jpg'</code>	<code>str</code> , <code>Path</code>	
URL	<code>'https://ultralytics.com/images/bus.jpg'</code>	<code>str</code>	
screenshot	<code>'screen'</code>	<code>str</code>	
PIL	<code>Image.open('im.jpg')</code>	<code>PIL.Image</code>	HWC, RGB
OpenCV	<code>cv2.imread('im.jpg')</code>	<code>np.ndarray</code>	HWC, BGR
numpy	<code>np.zeros((640, 1280, 3))</code>	<code>np.ndarray</code>	HWC
torch	<code>torch.zeros(16, 3, 320, 640)</code>	<code>torch.Tensor</code>	BCHW, RGB
CSV	<code>'sources.csv'</code>	<code>str</code> , <code>Path</code>	RTSP, RTMP, HTTP
video 	<code>'vid.mp4'</code>	<code>str</code> , <code>Path</code>	
directory 	<code>'path/'</code>	<code>str</code> , <code>Path</code>	
glob 	<code>'path/*.jpg'</code>	<code>str</code>	Use <code>*</code> operator
YouTube 	<code>'https://youtu.be/Zgi9g1ksQHc'</code>	<code>str</code>	
stream 	<code>'rtsp://example.com/media.mp4'</code>	<code>str</code>	RTSP, RTMP, HTTP

Arguments

`model.predict` accepts multiple arguments that control the prediction operation. These arguments can be passed directly to `model.predict`:

Example

```
model.predict(source, save=True, imgsz=320, conf=0.5)
```

All supported arguments:

Key	Value	Description
source	'ultralytics/assets'	source directory for images or videos
conf	0.25	object confidence threshold for detection
iou	0.7	intersection over union (IoU) threshold for NMS
half	False	use half precision (FP16)
device	None	device to run on, i.e. cuda device=0/1/2/3 or device=cpu
show	False	show results if possible
save	False	save images with results
save_txt	False	save results as .txt file
save_conf	False	save results with confidence scores
save_crop	False	save cropped images with results
hide_labels	False	hide labels
hide_conf	False	hide confidence scores
max_det	300	maximum number of detections per image
vid_stride	False	video frame-rate stride
line_width	None	The line width of the bounding boxes. If None, it is scaled to the image size.
visualize	False	visualize model features
augment	False	apply image augmentation to prediction sources
agnostic_nms	False	class-agnostic NMS
retina_masks	False	use high-resolution segmentation masks
classes	None	filter results by class, i.e. class=0, or class=[0,2,3]
boxes	True	Show boxes in segmentation predictions

Image and Video Formats

YOLOv8 supports various image and video formats, as specified in [yolo/data/utils.py](#). See the tables below for the valid suffixes and example predict commands.

Image Suffixes

Image Suffixes	Example Predict Command	Reference
.bmp	yolo predict source=image.bmp	Microsoft BMP File Format
.dng	yolo predict source=image.dng	Adobe DNG
.jpeg	yolo predict source=image.jpeg	JPEG
.jpg	yolo predict source=image.jpg	JPEG
.mpo	yolo predict source=image.mpo	Multi Picture Object
.png	yolo predict source=image.png	Portable Network Graphics
.tif	yolo predict source=image.tif	Tag Image File Format
.tiff	yolo predict source=image.tiff	Tag Image File Format
.webp	yolo predict source=image.webp	WebP
.pfm	yolo predict source=image.pfm	Portable FloatMap

Video Suffixes

Video Suffixes	Example Predict Command	Reference
.asf	yolo predict source=video.asf	Advanced Systems Format
.avi	yolo predict source=video.avi	Audio Video Interleave
.gif	yolo predict source=video.gif	Graphics Interchange Format
.m4v	yolo predict source=video.m4v	MPEG-4 Part 14
.mkv	yolo predict source=video.mkv	Matroska
.mov	yolo predict source=video.mov	QuickTime File Format
.mp4	yolo predict source=video.mp4	MPEG-4 Part 14 - Wikipedia
.mpeg	yolo predict source=video.mpeg	MPEG-1 Part 2
.mpg	yolo predict source=video.mpg	MPEG-1 Part 2
.ts	yolo predict source=video.ts	MPEG Transport Stream
.wmv	yolo predict source=video.wmv	Windows Media Video
.webm	yolo predict source=video.webm	WebM Project

Working with Results

The `Results` object contains the following components:

- `Results.bboxes`: `Boxes` object with properties and methods for manipulating bounding boxes
- `Results.masks`: `Masks` object for indexing masks or getting segment coordinates
- `Results.probs`: `torch.Tensor` containing class probabilities or logits
- `Results.orig_img`: Original image loaded in memory
- `Results.path`: `Path` containing the path to the input image

Each result is composed of a `torch.Tensor` by default, which allows for easy manipulation:

Results

```
results = results.cuda()
results = results.cpu()
results = results.to('cpu')
results = results.numpy()
```

Boxes

`Boxes` object can be used to index, manipulate, and convert bounding boxes to different formats. Box format conversion operations are cached, meaning they're only calculated once per object, and those values are reused for future calls.

- Indexing a `Boxes` object returns a `Boxes` object:

Boxes

```
results = model(img)
boxes = results[0].boxes
box = boxes[0] # returns one box
box.xyxy
```

- Properties and conversions



Boxes Properties

```
boxes.xyxy # box with xyxy format, (N, 4)
boxes.xywh # box with xywh format, (N, 4)
boxes.xyxy # box with xyxy format but normalized, (N, 4)
boxes.xywh # box with xywh format but normalized, (N, 4)
boxes.conf # confidence score, (N, 1)
boxes.cls # cls, (N, 1)
boxes.data # raw bboxes tensor, (N, 6) or boxes.bboxes
```

Masks

`Masks` object can be used index, manipulate and convert masks to segments. The segment conversion operation is cached.



Masks

```
results = model(inputs)
masks = results[0].masks # Masks object
masks.xy # x, y segments (pixels), List[segment] * N
masks.xyn # x, y segments (normalized), List[segment] * N
masks.data # raw masks tensor, (N, H, W) or masks.masks
```

probs

`probs` attribute of `Results` class is a `Tensor` containing class probabilities of a classification operation.



Probs

```
results = model(inputs)
results[0].probs # cls prob, (num_class, )
```

Class reference documentation for `Results` module and its components can be found [here](#)

Plotting results

You can use `plot()` function of `Result` object to plot results on in image object. It plots all components (boxes, masks, classification logits, etc.) found in the results object

Plotting

```
res = model(img)
res_plotted = res[0].plot()
cv2.imshow("result", res_plotted)
```

Argument	Description
<code>conf (bool)</code>	Whether to plot the detection confidence score.
<code>line_width (int, optional)</code>	The line width of the bounding boxes. If None, it is scaled to the image size.
<code>font_size (float, optional)</code>	The font size of the text. If None, it is scaled to the image size.
<code>font (str)</code>	The font to use for the text.
<code>pil (bool)</code>	Whether to use PIL for image plotting.
<code>example (str)</code>	An example string to display. Useful for indicating the expected format of the output.
<code>img (numpy.ndarray)</code>	Plot to another image. if not, plot to original image.
<code>labels (bool)</code>	Whether to plot the label of bounding boxes.
<code>boxes (bool)</code>	Whether to plot the bounding boxes.
<code>masks (bool)</code>	Whether to plot the masks.
<code>probs (bool)</code>	Whether to plot classification probability.

Streaming Source `for`-loop

Here's a Python script using OpenCV (cv2) and YOLOv8 to run inference on video frames. This script assumes you have already installed the necessary packages (opencv-python and ultralytics).



Streaming for-loop

```
import cv2
from ultralytics import YOLO

# Load the YOLOv8 model
model = YOLO('yolov8n.pt')

# Open the video file
video_path = "path/to/your/video/file.mp4"
cap = cv2.VideoCapture(video_path)

# Loop through the video frames
while cap.isOpened():
    # Read a frame from the video
    success, frame = cap.read()

    if success:
        # Run YOLOv8 inference on the frame
        results = model(frame)

        # Visualize the results on the frame
        annotated_frame = results[0].plot()

        # Display the annotated frame
        cv2.imshow("YOLOv8 Inference", annotated_frame)

        # Break the loop if 'q' is pressed
```

Created 2023-03-12, Updated 2023-05-14

Authors: Glenn Jocher (13)



Tweet



Share

Comments

6 reactions



1



1



1



1



1



1

28 comments · 28 replies – powered by giscus

Oldest

Newest

Gautambusa4 23 days ago