YOLOv11 官方介绍文档中文译本

YOLOv11 官方介绍文档中文译本
Chapter 1. 如何使用已有权重文件进行推理(Predict)
1.1 `predict` 模式的关键特性
1.2 推理结果的返回格式
1.3 推理输入源
1.4 YOLO11 支持的输入源
1.5 推理输入源示例
1.5.1 单张图像
1.5.2 截图
1.5.3 URL
1.5.4 PIL 图像
1.5.5 OpenCV 图像
1.5.6 NumPy 数组
1.5.7 PyTorch 张量
1.5.8 CSV 文件
1.5.9 视频文件
1.5.10 目录
1.5.11 文件模式
1.5.12 YouTube 视频
1.5.13 流媒体
1.5.14 多流媒体
1.5.15 网络摄像头
1.6 推理参数配置与流式推理
1.7 YOLOv11所有推理参数说明(重要)
1.7.1 推理参数(用于控制输入源和推理行为)
1.7.2 与可视化有关的参数(用于控制输出的呈现方式)
1.7.3 使用示例(重要，不知道怎么傻瓜式使用的点这里复制下面的代码)
基础推理
保存推理结果
自定义参数
实时摄像头推理
1.8 图像与视频格式
1.8.1 Ultralytics 支持的有效图像格式
1.8.2 Ultralytics 支持的有效视频格式
1.9 YOLO11 结果处理与使用指南
1.9.1 Results 对象
1.9.1.1 Results 属性
1.9.1.2 Results 方法
1.9.2 附加对象类型
1.9.2.1 Boxes（检测框）
1.9.2.2 Masks（检测掩码）
1.9.2.3 Keypoints（关键点）
1.9.3 结果可视化
1.9.3.1 使用 `plot()` 方法
1.9.3.2 参数说明
1.9.4. 视频流推理示例
1.10 总结

Chapter 1. 如何使用已有权重文件进行推理(Predict)

1.1 `predict` 模式的关键特性

YOLO11 的 `predict` 模式设计为强大且多功能，主要特点包括：

- **支持多种数据来源**：无论你的数据是单张图像、图像集合、视频文件，还是实时视频流，`predict` 模式都能满足需求。
- **流式模式**：通过启用 `stream=True`，可以生成高效内存占用的结果对象生成器。
- **批量处理**：支持对多张图像或多帧视频进行批量处理，从而进一步加快推理速度。
- **易于集成**：灵活的 API 设计，便于与现有的数据管道和其他软件组件集成。

1.2 推理结果的返回格式

Ultralytics YOLO 模型在推理时返回两种格式：

- **Python 列表**：包含 `Results` 对象。
- **内存高效的生成器**：当在调用模型时设置 `stream=True` 时，返回生成器形式的 `Results` 对象。

这些特性使得 YOLO11 在各种应用场景下都具备强大的适应性和灵活性。

1.3 推理输入源

YOLO11 支持多种推理输入源，包括静态图像、视频流和其他数据格式。下表展示了不同输入源的支持情况，并说明它们是否可以通过设置参数 `stream=True` 使用流式模式（✅）。流式模式特别适合处理视频或实时流，它会生成一个结果的生成器，而不是将所有帧加载到内存中。

【提示】

- 使用 `stream=True` 的场景
  - **长视频**：避免将所有帧加载到内存，节省资源。
  - **大型数据集**：按需生成结果，有效管理内存。
- `stream=False` 的影响
  - 默认情况下，所有帧或数据点的结果都会存储在内存中。
  - 对于大型输入数据，可能导致内存快速耗尽，引发内存溢出错误（Out-of-Memory）。
- `stream=True` 的优势
  - 利用生成器机制，仅将当前帧或数据点的结果保留在内存中。
  - 大幅降低内存消耗，防止内存溢出问题，适合处理长视频或大规模数据。

通过灵活选择流式模式与否，YOLO11 可以根据具体应用场景实现高效推理。

1.4 YOLO11 支持的输入源

输入源	示例	类型	备注
单张图像	`'image.jpg'`	`str` 或 `Path`	单个图像文件路径。
URL	`'https://ultralytics.com/images/bus.jpg'`	`str`	图像的 URL 地址。

输入源	示例	类型	备注
截图	`'screen'`	`str`	捕获屏幕截图作为输入。
PIL 图像	`Image.open('image.jpg')`	`PIL.Image`	HWC 格式，RGB 通道。
OpenCV 图像	`cv2.imread('image.jpg')`	`np.ndarray`	HWC 格式，BGR 通道，数据类型为 `uint8 (0-255)`。
NumPy 数组	`np.zeros((640, 1280, 3))`	`np.ndarray`	HWC 格式，BGR 通道，数据类型为 `uint8 (0-255)`。
PyTorch 张量	`torch.zeros(16, 3, 320, 640)`	`torch.Tensor`	BCHW 格式，RGB 通道，数据类型为 `float32 (0.0-1.0)`。
CSV 文件	`'sources.csv'`	`str` 或 `Path`	包含图像、视频或目录路径的 CSV 文件。
视频文件 ✅	`'video.mp4'`	`str` 或 `Path`	支持 MP4、AVI 等格式的视频文件。
目录 ✅	`'path/'`	`str` 或 `Path`	包含图像或视频的目录路径。
文件模式 ✅	`'path/*.jpg'`	`str`	使用 `*` 通配符匹配多个文件的路径模式，例如批量加载图片。
YouTube 视频 ✅	`'https://youtu.be/LNwODJXcvt4'`	`str`	YouTube 视频的 URL 地址。
流媒体 ✅	`'rtsp://example.com/media.mp4'`	`str`	流媒体协议的 URL，例如 RTSP、RTMP、TCP 或 IP 地址。
多流媒体 ✅	`'list.streams'`	`str` 或 `Path`	包含每行一个流 URL 的 `.streams` 文件（如 8 个流可按批量大小 8 运行推理）。
网络摄像头 ✅	`0`	`int`	连接的摄像头设备索引（如 `0` 表示默认摄像头）。

说明

1. **流式模式支持（✅）**：使用 `stream=True` 参数启用流式处理，返回一个内存高效的生成器，适用于视频流和大规模输入源，减少内存占用。
2. **兼容多种格式**：YOLO11 的 `predict` 模式可兼容从单张图像到多流视频的各种输入类型。
3. **输入预处理**：输入源的格式会被自动转换为模型所需的格式（如 BCHW 和 RGB 通道顺序）。

推荐实践

- **实时流媒体**：对于实时流或长视频，建议启用 `stream=True` 以节省内存。
- **大批量推理**：使用 `directory` 或 `glob` 批量处理多张图片时，可结合批量处理功能优化推理时间。
- **灵活数据集加载**：使用 `CSV` 文件或 Python 脚本动态生成输入路径，以便适配复杂的输入场景。

1.5 推理输入源示例

1.5.1 单张图像

运行推理，输入单张图片：

```
from ultralytics import YOLO

# 加载预训练的 YOLO11n 模型
model = YOLO("yolo11n.pt")

# 指定图片路径
source = "path/to/image.jpg"

# 对输入图片运行推理
results = model(source)  # 返回 Results 对象的列表
```

1.5.2 截图

运行推理，输入屏幕截图：

```
source = "screen"
results = model(source)
```

1.5.3 URL

运行推理，输入图片的 URL：

```
source = "https://ultralytics.com/images/bus.jpg"
results = model(source)
```

1.5.4 PIL 图像

运行推理，输入 PIL 图像对象：

```
from PIL import Image

source = Image.open("path/to/image.jpg")
results = model(source)
```

1.5.5 OpenCV 图像

运行推理，输入 OpenCV 图像对象：

```
import cv2

source = cv2.imread("path/to/image.jpg")
results = model(source)
```

1.5.6 NumPy 数组

运行推理，输入 NumPy 数组：

```
import numpy as np

source = np.zeros((640, 1280, 3), dtype=np.uint8)  # 示例数组
results = model(source)
```

1.5.7 PyTorch 张量

运行推理，输入 PyTorch 张量：

```
import torch

source = torch.zeros((1, 3, 320, 640), dtype=torch.float32)  # BCHW 格式
results = model(source)
```

1.5.8 CSV 文件

运行推理，输入一个包含图片路径的 CSV 文件：

```
source = "path/to/sources.csv"
results = model(source)
```

1.5.9 视频文件

运行推理，输入视频文件：

```
source = "path/to/video.mp4"
results = model(source)
```

1.5.10 目录

运行推理，输入包含图片或视频的目录：

```
source = "path/to/directory/"
results = model(source)
```

1.5.11 文件模式

运行推理，输入匹配多个文件的通配符模式：

```
source = "path/to/*.jpg"
results = model(source)
```

1.5.12 YouTube 视频

运行推理，输入 YouTube 视频 URL：

```
source = "https://youtu.be/LNwODJXcvt4"
results = model(source)
```

1.5.13 流媒体

运行推理，输入流媒体 URL：

```
source = "rtsp://example.com/media.mp4"
results = model(source)
```

1.5.14 多流媒体

运行推理，输入多个流媒体 URL 的文本文件：

```
source = "path/to/list.streams"
results = model(source)
```

1.5.15 网络摄像头

运行推理，输入摄像头设备索引（如 0 表示默认摄像头）：

```
source = 0
results = model(source)
```

1.6 推理参数配置与流式推理

YOLO11 提供了多种参数，可以在推理时动态调整。常见参数包括：

参数	描述
`save`	是否保存推理结果（True/False）。
`imgsz`	输入图像的分辨率大小，默认为训练时使用的分辨率。
`conf`	置信度阈值（0-1），用于过滤低置信度目标。
`iou`	非极大值抑制（NMS）的 IoU 阈值（0-1）。
`device`	指定推理使用的设备（如 "cpu" 或 "cuda"）。
`stream`	是否启用流式模式（True/False），建议用于长视频或实时流以减少内存消耗。

参数示例

```
from ultralytics import YOLO

# 加载预训练的 YOLO11n 模型
model = YOLO("yolo11n.pt")

# 使用自定义参数运行推理
results = model.predict(
    source="bus.jpg",  # 输入源
    save=True,         # 保存推理结果
    imgsz=320,         # 指定输入图片分辨率
    conf=0.5           # 设置置信度阈值
)
```

对于大规模数据或长时间视频，可以启用 `stream=True`，生成内存高效的结果生成器。

```
results = model.predict("path/to/video.mp4", stream=True)

for result in results:  # 逐帧处理推理结果
    print(result.boxes)
```

1.7 YOLOv11所有推理参数说明(重要)

1.7.1 推理参数(用于控制输入源和推理行为)

参数	类型	默认值	描述
`source`	`str`	`'ultralytics/assets'`	指定推理的数据源，可以是图像路径、视频文件、目录、URL 或设备 ID（如摄像头）。支持多种输入类型和格式。
`conf`	`float`	`0.25`	最低置信度阈值，置信度低于此值的检测结果会被忽略。调整此值可以减少误检（降低假阳性）。
`iou`	`float`	`0.7`	非极大值抑制（NMS）的 IoU 阈值，较低的值会删除更多重叠框，用于减少重复检测。
`imgsz`	`int or tuple`	`640`	推理的图像大小，可以是单个整数（正方形大小）或元组（高度，宽度）。调整大小可影响精度和速度。
`half`	`bool`	`False`	启用半精度（FP16）推理，在支持的 GPU 上加速推理，但对精度影响很小。

参数	类型	默认值	描述
`device`	`str`	`None`	指定推理设备，如 `cpu`、`cuda:0` 或 `0`。用户可以选择在 CPU 或特定 GPU 上运行模型。
`max_det`	`int`	`300`	每张图像允许的最大检测数量，用于限制密集场景中的输出数量，避免过多目标导致混乱。
`vid_stride`	`int`	`1`	视频输入的帧间距。设置较高的值可跳过帧，提高处理速度，但会降低时间分辨率。
`stream_buffer`	`bool`	`False`	是否为视频流缓存帧。如果设置为 `False`，旧帧会被丢弃（适合实时应用）。设置为 `True` 会排队所有帧，但可能增加延迟。
`visualize`	`bool`	`False`	可视化模型特征，帮助理解模型的特征提取和决策过程。常用于调试和模型解释。
`augment`	`bool`	`False`	启用测试时增强（TTA），提高预测的鲁棒性，但推理速度会变慢。
`agnostic_nms`	`bool`	`False`	启用类别无关的非极大值抑制（NMS），即合并不同类别的重叠框。适用于多类别检测场景。
`classes`	`list[int]`	`None`	筛选特定类别的检测结果。只返回属于指定类别的检测目标，适合于多类别任务中的目标筛选。
`retina_masks`	`bool`	`False`	返回高分辨率的分割掩码。如果启用，掩码大小与原始图像一致，否则与推理图像大小一致。
`embed`	`list[int]`	`None`	指定提取特征向量或嵌入向量的层，用于聚类或相似性搜索等下游任务。
`project`	`str`	`None`	设置项目目录名称。如果启用保存功能（`save=True`），推理输出会保存到该目录下。

参数	类型	默认值	描述
`name`	`str`	`None`	设置推理运行的名称，在项目目录中创建子目录以保存结果。

1.7.2 与可视化有关的参数(用于控制输出的呈现方式)

参数	类型	默认值	描述
`show`	`bool`	`False`	是否在窗口中显示标注后的图像或视频，适合开发和测试时的即时可视化反馈。
`save`	`bool`	`False`	是否保存标注后的图像或视频。CLI 默认启用（`True`），Python 脚本中默认关闭（`False`）。
`save_frames`	`bool`	`False`	处理视频时是否保存每一帧为单独的图像，适合逐帧分析或特定帧提取。
`save_txt`	`bool`	`False`	是否将检测结果保存为文本文件，格式为 `[class] [x_center] [y_center] [width] [height] [confidence]`。
`save_conf`	`bool`	`False`	是否在保存的文本文件中包含置信度分数，便于后续分析或集成到其他工具中。
`save_crop`	`bool`	`False`	是否保存检测目标的裁剪图像，适合数据集增强或针对特定目标创建数据集。
`show_labels`	`bool`	`True`	是否在可视化输出中显示每个检测目标的标签，便于快速理解检测结果。
`show_conf`	`bool`	`True`	是否在可视化输出中显示每个目标的置信度分数，有助于了解模型的预测可信度。
`show_boxes`	`bool`	`True`	是否在可视化输出中绘制检测目标的边界框，是检测任务的核心功能之一。
`line_width`	`None` or `int`	`None`	设置边界框线条的宽度。如果设置为 `None`，会根据图像大小自动调整，适合可视化清晰度的自定义需求。

1.7.3 使用示例(重要，不知道怎么傻瓜式使用的点这里复制下面的代码)

基础推理

```
from ultralytics import YOLO

# 加载模型
model = YOLO("yolo11n.pt")

# 推理并显示结果
model.predict("path/to/image.jpg", conf=0.5, imgsz=640, show=True)
```

保存推理结果

```
model.predict("path/to/video.mp4", save=True, save_crop=True, save_txt=True)
```

自定义参数

```
model.predict(
    source="rtsp://example.com/media.mp4",  # 流媒体输入
    conf=0.3,                               # 设置置信度阈值
    iou=0.5,                                # IoU 阈值
    augment=True,                           # 启用增强推理
    classes=[0, 1],                         # 仅检测类别 0 和 1
    save=True                               # 保存检测结果
)
```

实时摄像头推理

```
model.predict(0, show=True, vid_stride=2)  # 使用默认摄像头，跳帧推理
```

1.8 图像与视频格式

1.8.1 Ultralytics 支持的有效图像格式

> **注意**
> HEIC 格式的图像仅支持推理，不支持训练。

图像后缀	推理命令示例	参考
`.bmp`	`yolo predict source=image.bmp`	Microsoft BMP 文件格式
`.dng`	`yolo predict source=image.dng`	Adobe DNG 格式
`.jpeg`	`yolo predict source=image.jpeg`	JPEG 格式
`.jpg`	`yolo predict source=image.jpg`	JPEG 格式
`.mpo`	`yolo predict source=image.mpo`	多图像对象格式（Multi Picture Object）
`.png`	`yolo predict source=image.png`	便携式网络图形（Portable Network Graphics）
`.tif`	`yolo predict source=image.tif`	标记图像文件格式（Tag Image File Format）
`.tiff`	`yolo predict source=image.tiff`	标记图像文件格式（Tag Image File Format）

图像后缀	推理命令示例	参考
`.webp`	`yolo predict source=image.webp`	WebP 图像格式
`.pfm`	`yolo predict source=image.pfm`	可移植浮点地图（Portable FloatMap）
`.HEIC`	`yolo predict source=image.HEIC`	高效图像格式（High Efficiency Image Format）

1.8.2 Ultralytics 支持的有效视频格式

视频后缀	推理命令示例	参考
`.asf`	`yolo predict source=video.asf`	高级系统格式（Advanced Systems Format）
`.avi`	`yolo predict source=video.avi`	音频视频交错（Audio Video Interleave）
`.gif`	`yolo predict source=video.gif`	图形交换格式（Graphics Interchange Format）
`.m4v`	`yolo predict source=video.m4v`	MPEG-4 Part 14
`.mkv`	`yolo predict source=video.mkv`	Matroska 格式
`.mov`	`yolo predict source=video.mov`	QuickTime 文件格式
`.mp4`	`yolo predict source=video.mp4`	MPEG-4 Part 14
`.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 传输流（MPEG Transport Stream）
`.wmv`	`yolo predict source=video.wmv`	Windows 媒体视频（Windows Media Video）
`.webm`	`yolo predict source=video.webm`	WebM 项目

1.9 YOLO11 结果处理与使用指南

1.9.1 Results 对象

所有的 `model.predict()` 调用都会返回一个或多个 `Results` 对象。

1.9.1.1 Results 属性

属性名	类型	描述
`orig_img`	`numpy.ndarray`	原始图像，以 NumPy 数组形式表示。
`orig_shape`	`tuple`	原始图像的形状，格式为 `(height, width)`。
`boxes`	`Boxes` (可选)	包含检测框的 `Boxes` 对象。
`masks`	`Masks` (可选)	包含检测掩码的 `Masks` 对象。
`probs`	`Probs` (可选)	包含分类任务的类别概率的 `Probs` 对象。
`keypoints`	`Keypoints` (可选)	包含检测关键点的 `Keypoints` 对象。
`obb`	`OBB` (可选)	包含定向边界框（Oriented Bounding Boxes）的 `OBB` 对象。
`speed`	`dict`	字典，包含预处理、推理、后处理速度（以毫秒为单位）。
`names`	`dict`	字典，包含类别名称映射。
`path`	`str`	图像文件路径。

1.9.1.2 Results 方法

方法名	返回类型	描述
`update()`	`None`	更新 `boxes`、`masks` 和 `probs` 属性。
`cpu()`	`Results`	返回将所有张量移到 CPU 的 `Results` 对象副本。
`numpy()`	`Results`	返回将所有张量转换为 NumPy 数组的 `Results` 对象副本。
`cuda()`	`Results`	返回将所有张量移到 GPU 的 `Results` 对象副本。
`to()`	`Results`	返回将张量移动到指定设备的 `Results` 对象副本。
`new()`	`Results`	返回一个新的 `Results` 对象，包含相同图像、路径和名称。
`plot()`	`numpy.ndarray`	绘制检测结果，返回标注的图像（NumPy 数组）。
`show()`	`None`	显示标注结果。
`save()`	`None`	将标注结果保存到文件。

方法名	返回类型	描述
`verbose()`	`str`	返回每个任务的日志字符串。
`save_txt()`	`None`	将预测结果保存为文本文件，包含 `[class, x_center, y_center, width, height, confidence]`。
`save_crop()`	`None`	保存裁剪的检测目标到指定目录。
`tojson()`	`str`	将结果对象转换为 JSON 格式字符串。

1.9.2 附加对象类型

1.9.2.1 Boxes（检测框）

`Boxes` 对象包含检测边界框，可以进行索引、操作和格式转换。

方法/属性	类型	描述
`cpu()`	方法	将对象移动到 CPU。
`numpy()`	方法	将对象转换为 NumPy 数组。
`cuda()`	方法	将对象移动到 CUDA。
`to()`	方法	将对象移动到指定设备。
`xyxy`	属性（`torch.Tensor`）	返回框的 `xyxy` 格式。
`conf`	属性（`torch.Tensor`）	返回框的置信度值。
`cls`	属性（`torch.Tensor`）	返回框的类别值。

1.9.2.2 Masks（检测掩码）

`Masks` 对象包含实例分割掩码。

方法/属性	类型	描述
`cpu()`	方法	将掩码张量移动到 CPU。
`numpy()`	方法	将掩码张量转换为 NumPy 数组。
`cuda()`	方法	将掩码张量移动到 GPU。
`to()`	方法	将掩码张量移动到指定设备。
`xyn`	属性（`torch.Tensor`）	返回归一化的掩码段。
`xy`	属性（`torch.Tensor`）	返回以像素为单位的掩码段。

1.9.2.3 Keypoints（关键点）

`Keypoints` 对象包含检测的关键点。

方法/属性	类型	描述
`cpu()`	方法	将关键点张量移动到 CPU。
`numpy()`	方法	将关键点张量转换为 NumPy 数组。
`cuda()`	方法	将关键点张量移动到 GPU。
`to()`	方法	将关键点张量移动到指定设备。
`xyn`	属性（`torch.Tensor`）	返回归一化的关键点。
`xy`	属性（`torch.Tensor`）	返回以像素为单位的关键点。

1.9.3 结果可视化

1.9.3.1 使用 `plot()` 方法

`plot()` 方法将检测结果（如边界框、掩码、关键点）绘制在原始图像上，并返回标注后的图像。

示例：

```
from PIL import Image
from ultralytics import YOLO

model = YOLO("yolo11n.pt")
results = model("bus.jpg")

for r in results:
    annotated_img = r.plot()
    annotated_img_pil = Image.fromarray(annotated_img[..., ::-1])  # 转换为 RGB
    annotated_img_pil.show()
```

1.9.3.2 参数说明

参数	类型	默认值	描述
`conf`	`bool`	`True`	是否显示置信度分数。
`line_width`	`float`	`None`	边界框的线宽，若为 `None`，则根据图像大小自动调整。
`labels`	`bool`	`True`	是否显示类别标签。
`show`	`bool`	`False`	是否直接显示标注后的图像。
`save`	`bool`	`False`	是否将标注图像保存为文件。

1.9.4. 视频流推理示例

使用 OpenCV 处理视频并对每帧运行 YOLO 推理：

```
import cv2
from ultralytics import YOLO

model = YOLO("yolo11n.pt")
cap = cv2.VideoCapture("path/to/video.mp4")

while cap.isOpened():
    success, frame = cap.read()
    if not success:
        break

    results = model(frame)
    annotated_frame = results[0].plot()
    cv2.imshow("YOLO Inference", annotated_frame)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

cap.release()
cv2.destroyAllWindows()
```

此脚本会逐帧推理，并在窗口中显示带标注的帧。

1.10 总结

以上涵盖了 YOLO11 推理结果的主要功能和使用方式。如有问题请查阅官方原文 https://docs.ultralytics.com/modes/predict/