# c1.YOLOv8 代码部署

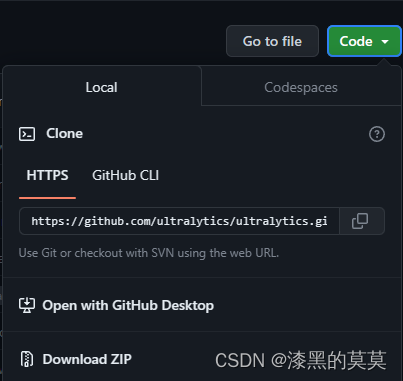
漆黑的莫莫 已于 2024-11-26 19:59:06 修改

原文链接：<https://blog.csdn.net/qq_45831414/article/details/134699926>

本文详细指导如何从GitHub获取YOLOv8代码，配置环境，安装依赖，进行代码测试，设置保存路径，以及如何为自己的数据集做准备，包括训练、验证和预测。还解决了WindowsGPU训练时可能出现的问题。

## **一、获取代码**

[YOLOv8官方GitHub网址(v8.0.4)](https://github.com/ultralytics/ultralytics/tree/v8.0.4)：<https://github.com/ultralytics/ultralytics/tree/v8.0.4>



获取YOLOv8代码压缩包。

二、虚拟环境配置

这个就不写了，装个Anaconda，网上教程很多

三、PyCharm安装与配置（可选）

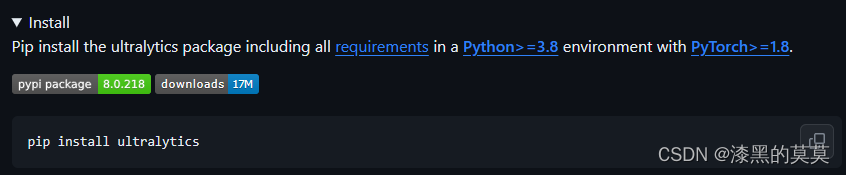
这个也不写了，直接用命令行也行，或者使用其他IDE，网上教程很多

## 四、下载ultralytics依赖库

这里默认代码压缩包已经打开，并且PyCharm也已经装好并打开了。

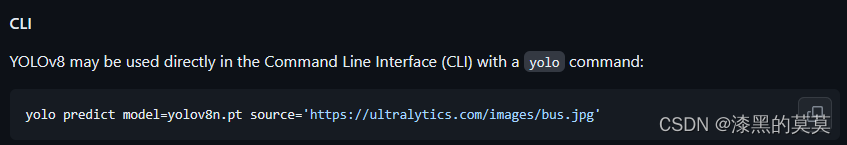
此时的终端已经打开，并且路径在YOLOv8的文件夹下。

输入YOLOv8官方代码里的REDAME.md里的第一步（代码部署都应该先看一下作者写的README.md文件，相当于代码的使用说明书）



pip install ultralytics

## **五、测试YOLOv8安装的成功与否**



这是YOLOv8作者的使用在网络图片的测试语句

yolo predict model=yolov8n.pt source='https://ultralytics.com/images/bus.jpg'

这是使用本地图片的测试语句

yolo task=detect mode=predict model=yolov8n.pt conf=0.25 source='ultralytics/assets/bus.jpg'

只要能正确得出运行结果，就说明YOLOv8安装成功了。  
而这个运行结果为什么会保存在ultralytics\runs\detect\train这个路径下，则会在下一节讲述。

## 六、YOLOv8的保存路径

YOLOv8的保存路径与YOLOv5不同，是使用settings.yaml文件来设定的。

这个文件放在这个路径下

C:\Users\你的用户名\AppData\Roaming\Ultralytics

详情参考这个博客 [YOLOv8数据集存放位置]( https://blog.csdn.net/weixin\_74319382/article/details/133325970 )

之所以会谈到这个问题，是因为YOLOv8默认的文件路径相比YOLOv5有所变动。

所以在后续Python文件调用YOLOv8的代码时，所用的资源文件建议用绝对路径。

## 七、为使用YOLOv8来训练自己的数据集做一些准备工作

首先需要数据集，也就是train(训练集)、val(验证集)、test(测试集)。

这个就不多说了，网络上也有很多教程。

主要讲所需的另一份文件dataset.yaml

这里最好用绝对路径，不然会报错，相对路径也不是不可以，就是没那个耐心调整了。

train: 训练集所在的绝对路径

# train images (relative to 'path')

# 例如D:/yolov8/ultralytics/datasets/train/images

val: 验证集所在的绝对路径

# val images (relative to 'path')

# 例如D:/yolov8/ultralytics/datasets/val/images

test: 测试集所在的绝对路径

# test images (optional)

# 例如D:/yolov8/ultralytics/datasets/test/images

# number of classes

nc: 数据集种类的数量

# Classes

names: [数据集的种类的名称]

## 八、使用Python文件来训练自己的数据集

YOLOv8的Python语句用法详见 [YOLOv8官方Python文档]( https://docs.ultralytics.com/usage/python/ )

将以下代码写入train.py文件

from ultralytics import YOLO

import os

os.environ["KMP\_DUPLICATE\_LIB\_OK"] = "TRUE"

# Load a model

#model = YOLO("yolov8n.yaml") # build a new model from scratch

model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)

# Use the model

model.train(data="./dataset.yaml", epochs=3) # train the model

下面开始数据集的训练，运行以下代码

(此处应当在你的YOLOv8项目的根路径下，且已经打开对应的虚拟环境)

python train.py

## 九、使用GPU训练数据集时报错（仅限WIN系统）

如果直接使用CPU来训练数据集时，不会报错，但是当使用GPU来训练数据集时，则会报错。

此错误具体为：

RuntimeError:

An attempt has been made to start a new process before the

current process has finished its bootstrapping phase.

错误详情参考这篇文章

解决RuntimeError: An attempt has been made to start a new process before…办法:

https://zhuanlan.zhihu.com/p/553407062

解决方法也很简单，在YOLOv8的项目里搜索num\_workers

将num\_workers=nw改为num\_workers=0

这行代码的具体位置是

你的YOLOv8项目根目录\ultralytics\data\build.py

在这个py文件里搜索num\_workers=nw即可。

之后再次训练数据集即可正常运行。

## 十、使用Python文件训练、验证、测试自己的数据集

**训练加验证加测试**

#训练加验证加测试

from ultralytics import YOLO

# Load a model

#model = YOLO("yolov8n.yaml") # build a new model from scratch

model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)

# Use the model

model.train(data="./dataset.yaml", epochs=200) # train the model

# Export the model to ONNX format

success = model.export(format='onnx')

# Evaluate the model's performance on the validation set

model.val()

# Define path to directory containing images and videos for inference

source = r'./ultralytics/datasets/test/images/'

# Run inference on the source

results = model.predict(source, save=True) # generator of Results objects

**为验证而训练**

#为验证而训练

from ultralytics import YOLO

# Load a model

#model = YOLO("yolov8n.yaml") # build a new model from scratch

model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)

# Use the model

model.train(data="./dataset.yaml", epochs=3) # train the model

# Export the model to ONNX format

success = model.export(format='onnx')

**在训练的基础上验证**

#在训练的基础上验证

from ultralytics import YOLO

model = YOLO(r'ultralytics\runs\detect\train30\weights\best.pt') #非通用

#model = YOLO(r'ultralytics\runs\detect\train30\weights\best.onnx') #通用

model.val()

**在训练的基础上预测**

#在训练的基础上预测

from ultralytics import YOLO

# Load a model

model = YOLO(r'ultralytics\runs\detect\train30\weights\best.pt') #非通用

#model = YOLO(r'ultralytics\runs\detect\train30\weights\best.onnx') #通用

# Define path to directory containing images and videos for inference

source = r'./ultralytics/datasets/test/images'

# Run inference on the source

results = model.predict(source, save=True) # generator of Results objects

**独自训练**

#独自训练

from ultralytics import YOLO

# Load a model

#model = YOLO("yolov8n.yaml") # build a new model from scratch

model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)

# Use the model

model.train(data="./dataset.yaml", epochs=3) # train the model

# Export the model to ONNX format

success = model.export(format='onnx')

**独自验证**

#独自验证

from ultralytics import YOLO

# Load a model

model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)

# Evaluate the model's performance on the validation set

model.val(data="./dataset.yaml")

**独自预测**

#独自预测

from ultralytics import YOLO

# Load a model

model = YOLO("yolov8n.pt") # load a pretrained model (recommended for training)

# Define path to directory containing images and videos for inference

source = r'./ultralytics/datasets/test/images'

# Run inference on the source

results = model.predict(source, save=True) # generator of Results objects

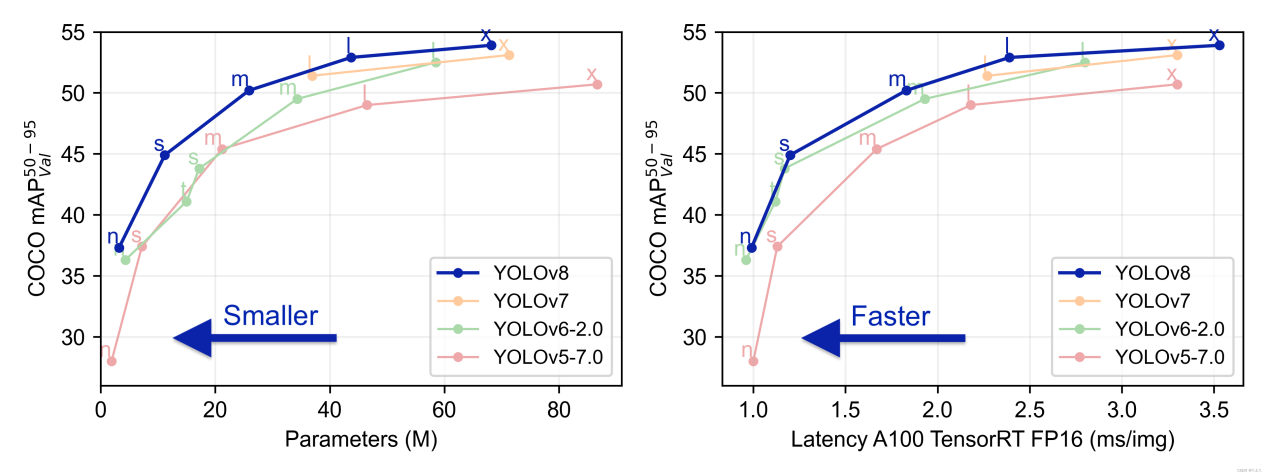
# c2.目标检测——YOLOv8（十四）

[Y.Z.Y.](https://blog.csdn.net/qq_42823043" \o "Y.Z.Y." \t "https://blog.csdn.net/qq_42823043/article/details/_blank) 于 2023-04-24 11:54:24 发布

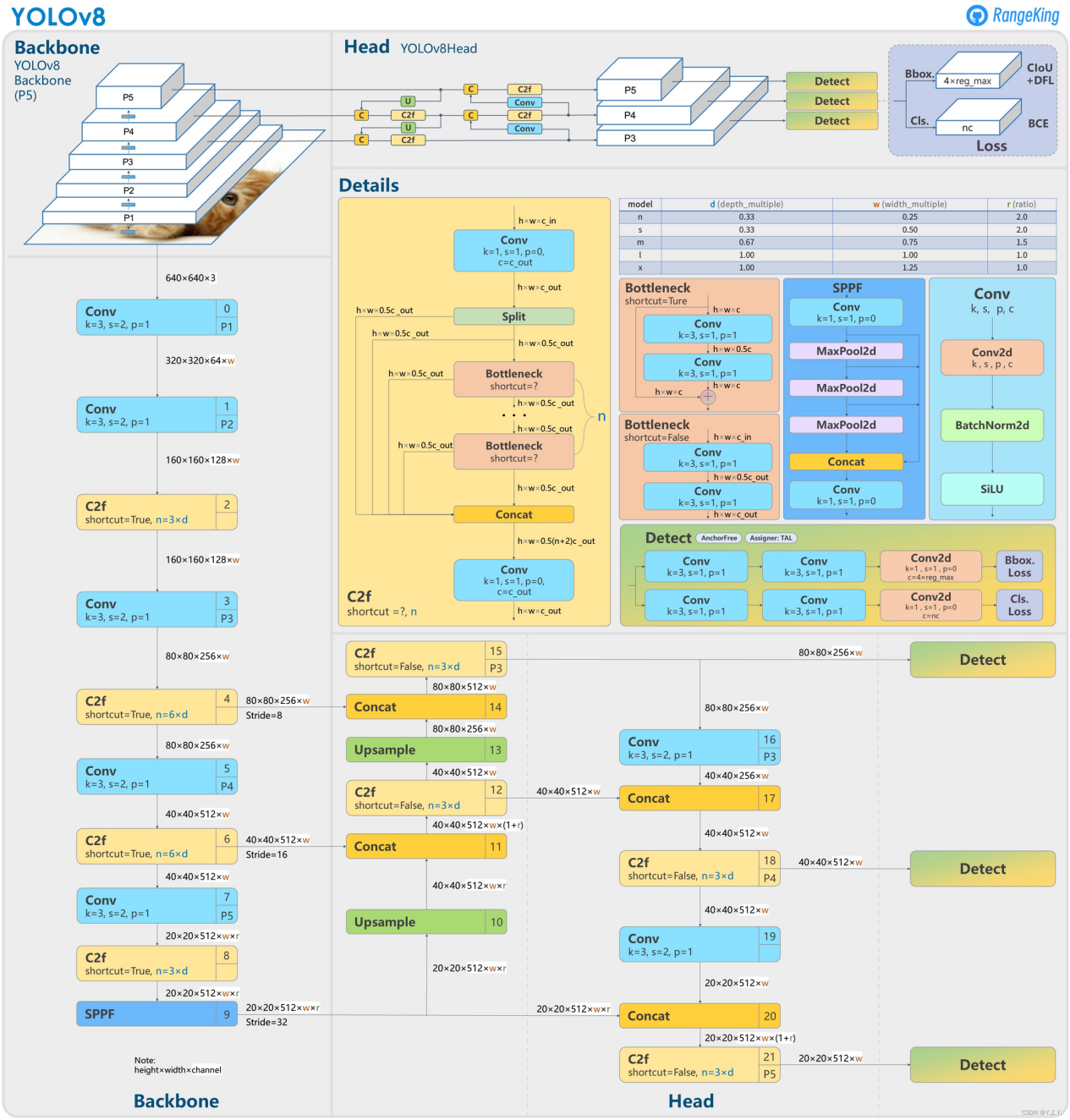
简介：

YOLOv8 是 ultralytics 公司在 2023 年 1月 10 号开源的 YOLOv5 的下一个重大更新版本，目前支持图像分类、物体检测和实例分割任务。不过 ultralytics 并没有直接将开源库命名为 YOLOv8，而是直接使用 ultralytics 这个词，原因是 ultralytics 将这个库定位为算法框架，而非某一个特定算法，一个主要特点是可扩展性。

Github地址：https://github.com/ultralytics/ultralytics



### **网络结构**



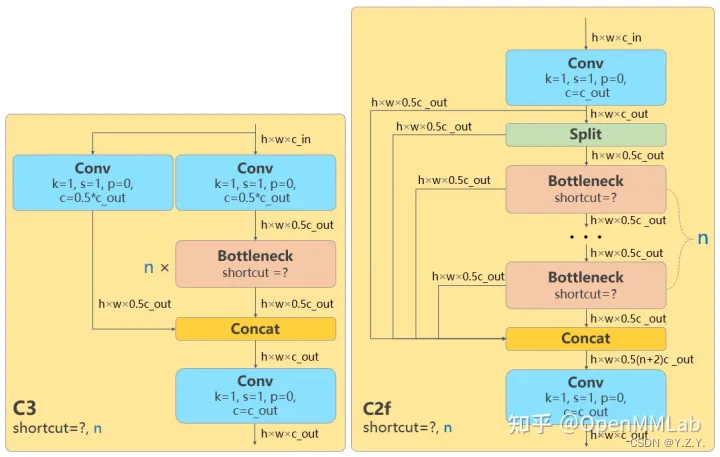
### **核心原理详解：**

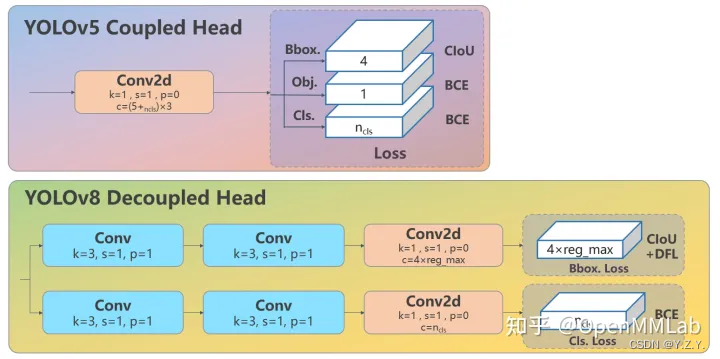
1、Backbone：

使用的依旧是CSP的思想，不过YOLOv5中的C3模块被替换成了梯度流更丰富的C2f模块，为了轻量化也缩减了骨干网络中最大stage的blocks数，同时不同缩放因子N/S/M/L/X的模型不再是共用一套模型参数，M/L/X大模型还缩减了最后一个stage的输出通道数，进一步减少参数量和计算量。

2、Neck ：

毫无疑问YOLOv8依旧使用了PAN的思想，不过通过对比YOLOv5与YOLOv8的结构图可以看到，YOLOv8将YOLOv5中PAN-FPN上采样阶段中的卷积结构删除了，同时也将C3模块替换为了C2f模块。



1. ****Head ：****  
   换成了目前主流的Decoupled-Head解耦头结构，将分类分支和定位分支分离，缓解了分类和回归任务的内在冲突。  
   同时也从 Anchor-Based 换成了 Anchor-Free。  
   

4、Loss：

采用了TAL(Task Alignment Learning)动态匹配，并引入了DFL(Distribution Focal Loss)结合CIoU Loss做回归分支的损失函数，使得分类和回归任务之间具有较高的一致性。

TaskAlignedAssigner 的匹配策略简单总结为： 根据分类与回归的分数加权的分数选择正样本。

5、训练策略：

训练的数据增强部分引入了 YOLOX 中的最后 10 epoch 关闭 Mosiac 增强的操作，可以有效地提升精度。

总结

回顾整个虎年，堪称YOLO内卷元年，美团发布YOLOv6、Alexey发布YOLOv7、百度发布PP-YOLOE+、阿里发布DAMO-YOLO、OpenMMLab发布RTMDet。同时这些系列模型也在不断更新迭代。

YOLOv8 主要参考了最近提出的诸如 YOLOX、YOLOv6、YOLOv7 和 PPYOLOE 等算法的相关设计，本身的创新点不多，偏向工程实践，主推的还是 ultralytics 这个框架本身。

# c3.手把手教你用YOLOv8训练自己的数据集以及YOLOv8的多任务使用

CV开发者都爱看的 [小白学视觉](javascript:void(0);) 2024年11月21日 10:34 中国香港

记录直接在YOLOv8的官方仓库上直接配置和训练yolov5的全过程。

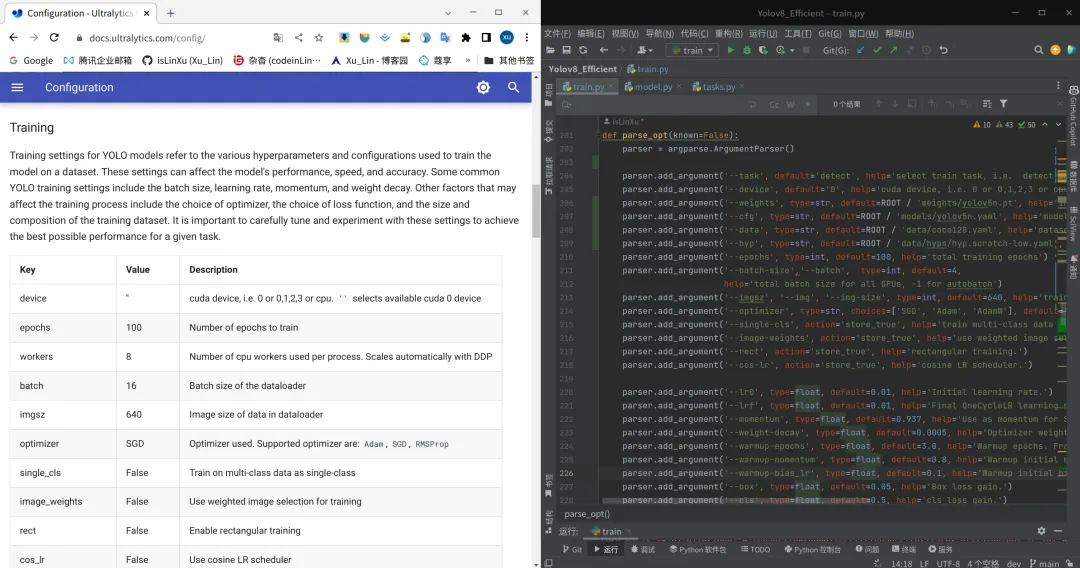
## 1、YOLOv8\_Efficient的介绍

* Github地址:https://github.com/isLinXu/YOLOv8\_Efficient

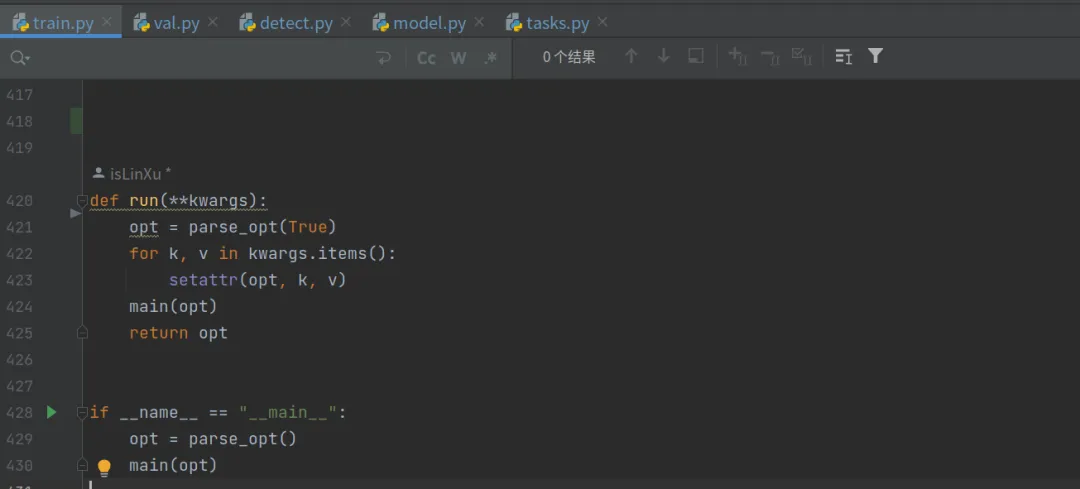
本项目基于ultralytics及yolov5等进行综合参考，致力于让yolo系列的更加高效和易用。

目前主要做了以下的工作：

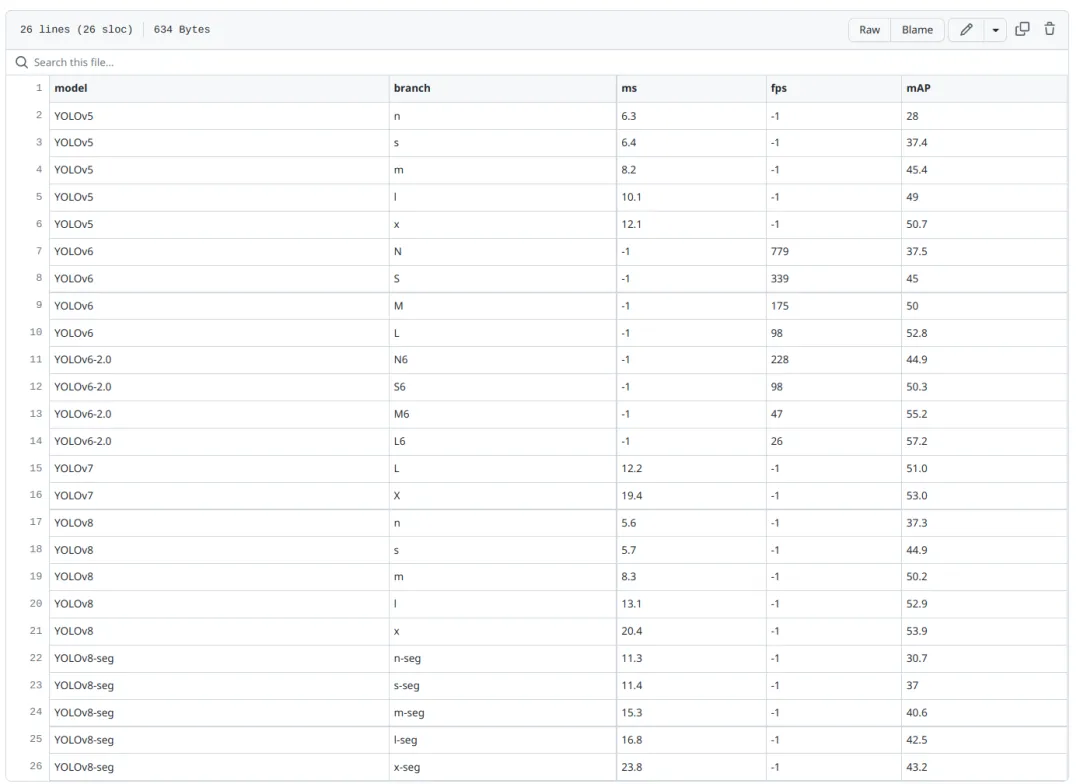
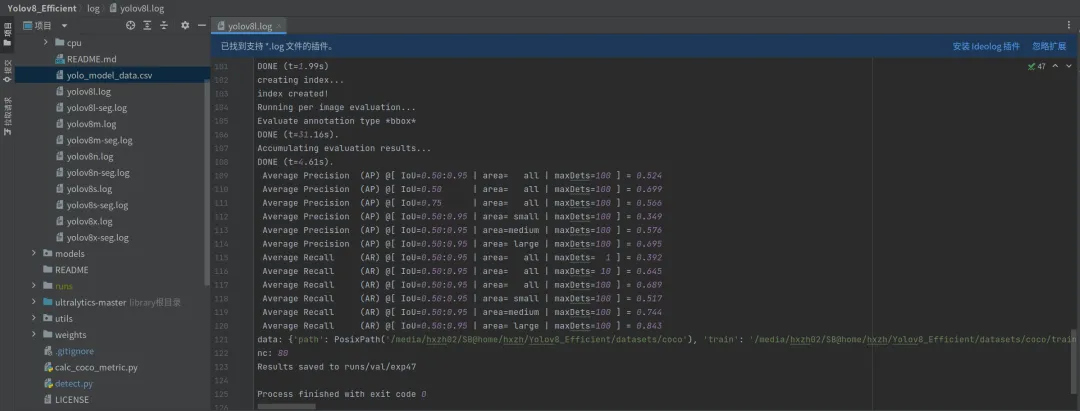
* 参考https://docs.ultralytics.com/config/中的Configuration参数，分别针对train.py、detect.py、val.py等做了相应参数的配置对齐。



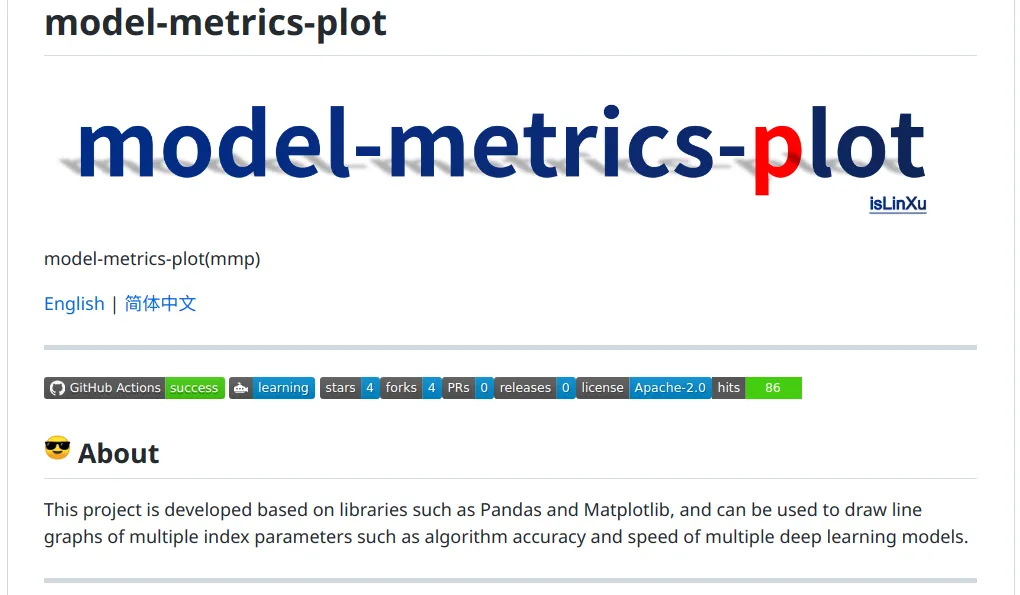
* 结合yolov5的使用习惯以及代码结构做了兼容和优化。



* 通过在coco数据集上在自己的机器上进行验证和计算的权重的指标参数，实验记录存放在https://github.com/isLinXu/YOLOv8\_Efficient/tree/main/log.实验数据记录在:



* 根据计算出来的结果绘制了相应的指标参数对比图，这个绘图程序也开源在https://github.com/isLinXu/model-metrics-plot中。

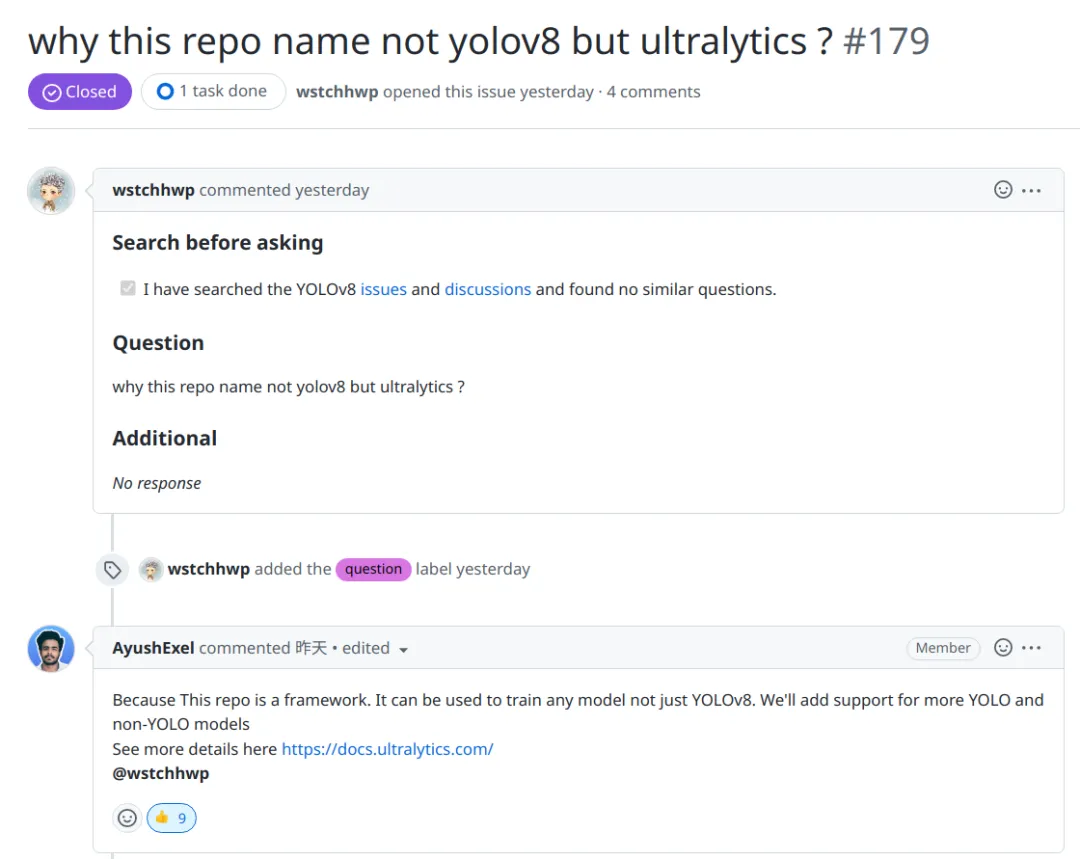


* 融合其他更多网络模型结构进行集成整合和配置，正在进行中...

## 2、关于ultralytics的名字

为什么这个仓库取名为ultralytics，而不是yolov8，结合这个issue，笔者认为主要有以下几个方面的原因：

* 1.因为ultralytics团队希望将这个项目设计和建成一个集合分类，检测，分割等视觉任务的集成训练推理框架，而不仅仅只是yolov8。后续可能会有更多更全的网络模型会集成进来。
* 2.因为pypi.org上的第三方已经把yolov6,yolov7,yolov8等名字给取了，pip install名称的规则是不允许有重复名的。



issue链接：https://github.com/ultralytics/ultralytics/issues/179

## 3、关于自定义配置模型训练

结合上面的讨论，自然而然会有这个想法，既然ultralytics要建一个集成训练框架，那么能否直接在ultralytics仓库上直接配置和训练yolov5呢，笔者做了下面一系列的尝试:

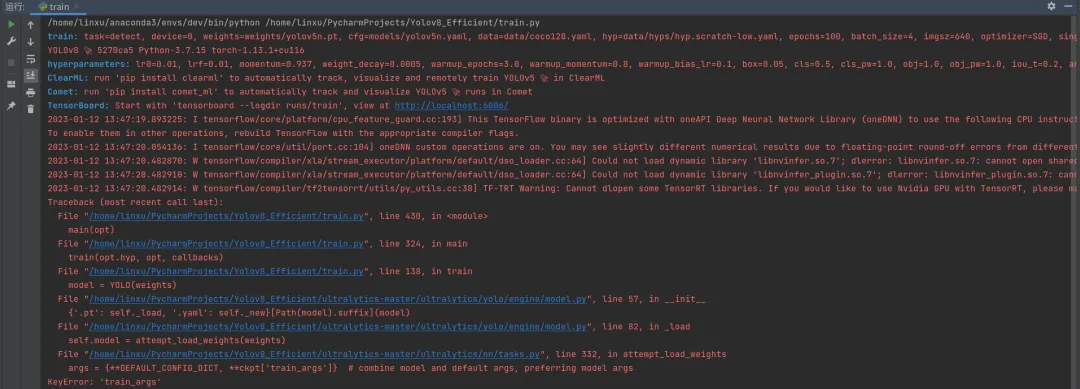
* 在models中加入相应的.yaml文件和yolov5沿用的模块，如common.py、experimental.py、google\_utils.py
* 在models/common.py中，加入了yolov5所需的网络结构

  class C3(nn.Module):    
    *# CSP Bottleneck with 3 convolutions*  
    def \_\_init\_\_(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  *# ch\_in, ch\_out, number, shortcut, groups, expansion*  
        super().\_\_init\_\_()    
        c\_ = int(c2 \* e)  *# hidden channels*  
        self.cv1 = Conv(c1, c\_, 1, 1)    
        self.cv2 = Conv(c1, c\_, 1, 1)    
        self.cv3 = Conv(2 \* c\_, c2, 1)  *# optional act=FReLU(c2)*  
        self.m = nn.Sequential(\*(Bottleneck(c\_, c\_, shortcut, g, e=1.0) for \_ in range(n)))    
    
    def forward(self, x):    
        return self.cv3(torch.cat((self.m(self.cv1(x)), self.cv2(x)), 1))

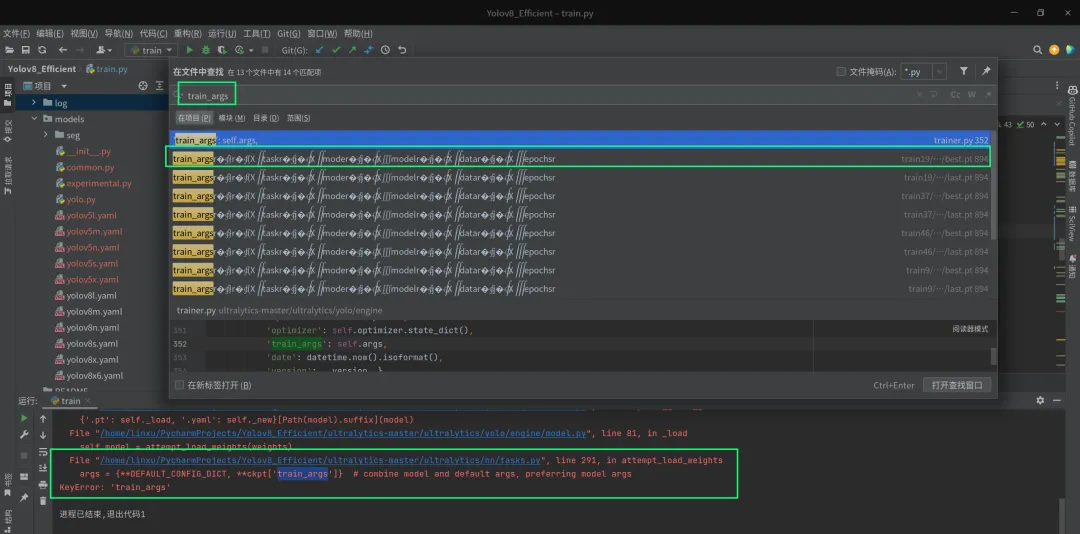
* 在运行时加入模块用于测试

最后一通操作下来，已经可以根据yolov5s.yaml去读取网络结构了，但是在跑的时候还是报错。

报错信息如下：



于是针对"train\_args"做了一个全局搜索，发现了下面的结果：



可以看到，之前训练出来的v8的权重内包含了"train\_args"的信息。顺着程序运行的流程，相应地发现了yolo/engine/model中的"\_\_init\_\_(self)"函数，

    def \_\_init\_\_(self, model='yolov8n.yaml', type="v8") -> None:    
        """    
        Initializes the YOLO object.    
    
        Args:    
            model (str, Path): model to load or create    
            type (str): Type/version of models to use. Defaults to "v8".    
        """    
        self.type = type    
        self.ModelClass = None  *# model class*  
        self.TrainerClass = None  *# trainer class*  
        self.ValidatorClass = None  *# validator class*  
        self.PredictorClass = None  *# predictor class*  
        self.model = None  *# model object*  
        self.trainer = None  *# trainer object*  
        self.task = None  *# task type*  
        self.ckpt = None  *# if loaded from \*.pt*  
        self.ckpt\_path = None    
        self.cfg = None  *# if loaded from \*.yaml*  
        self.overrides = {}  *# overrides for trainer object*  
        self.init\_disabled = False  *# disable model initialization*  
    
        *# Load or create new YOLO model*  
        {'.pt': self.\_load, '.yaml': self.\_new}[Path(model).suffix](model)

读取模型和配置是在"\_\_init\_\_"的最后一行:

*# Load or create new YOLO model*  
{'.pt': self.\_load, '.yaml': self.\_new}[Path(model).suffix](model)

而def \_load(self, weights: str):中实际读取模型权重的实现是self.model = attempt\_load\_weights(weights)。可以看到，相比于yolov5，v8读取权重的函数attempt\_load\_weights，多了下面这行

args = {\*\*DEFAULT\_CONFIG\_DICT, \*\*ckpt['train\_args']}  *# combine model and default args, preferring model args*

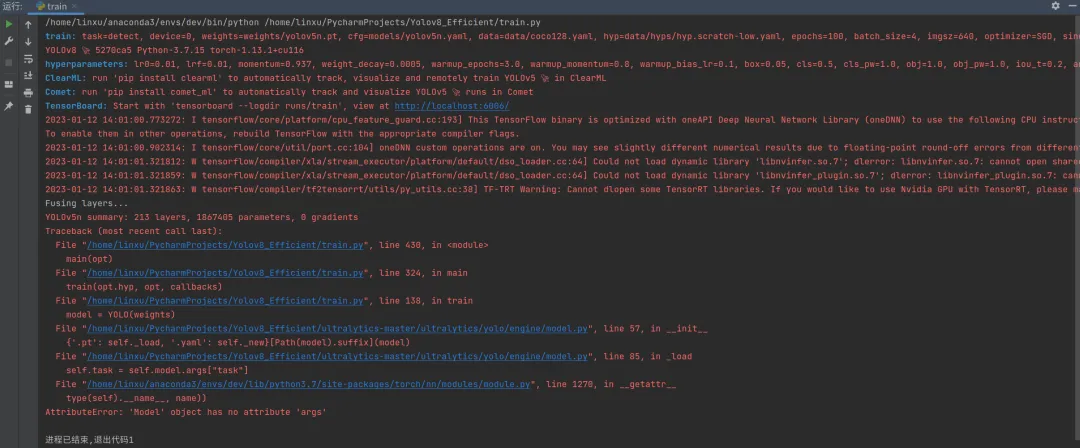
那么，能否直接将v5的项目中，将相应的函数补充过来给v8做适配呢，自然是可以的，当笔者将model.py的\_load函数中这行代码：

self.model = attempt\_load\_weights(weights)

替换为下面这行时：

self.model = attempt\_load(weights)

重新运行了一遍，发现又出现了下面的问题:



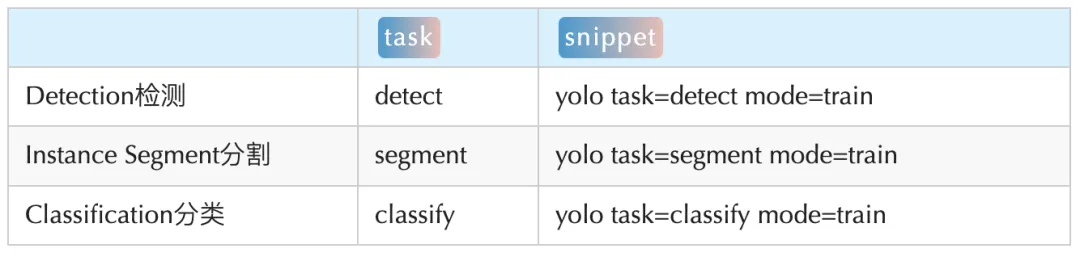
错误信息为AttributeError: 'Model' object has no attribute 'args'，既然是Model定义和配置上的问题，那么就没有再往下修改的必要了，还是等官方团队的更新和修改吧，等等党永远不亏。

## 4、关于v8的多任务使用

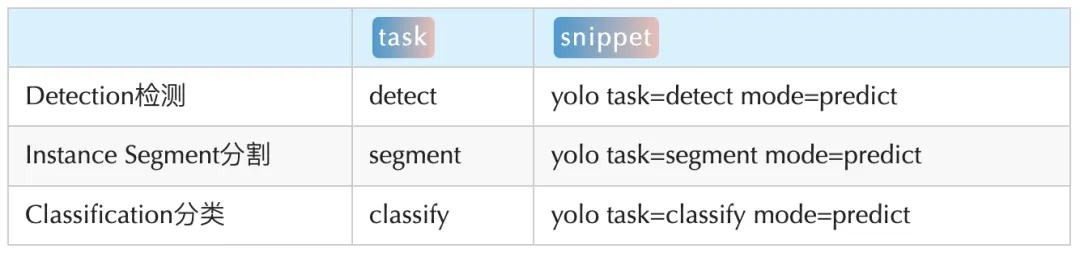
根据官方的文档介绍，还有对代码的分析，目前v8项目是支持检测、分类和分割的。设定是通过"task"进行区分任务，又通过mode来设置是训练还是检测的模式，如下使用：

yolo task=detect    mode=train    model=yolov8n.yaml    epochs=1 ...    
            ...           ...           ...    
          segment        predict        yolov8n-seg.pt    
          classify        val           yolov8n-cls.pt

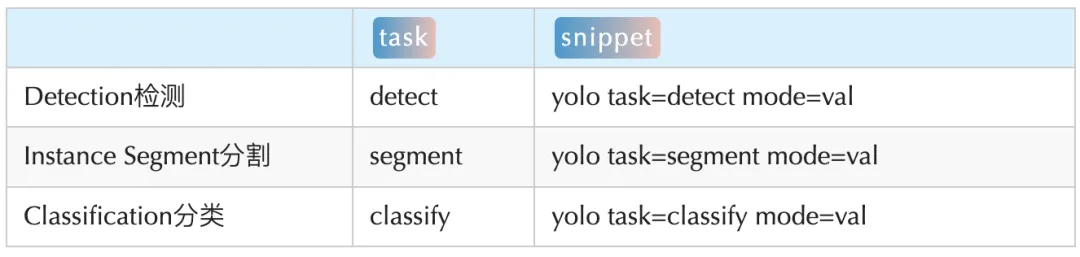
### **4.1、训练**



### **4.2、预测**



### **4.3、验证**



* !关于这三个任务，YOLOv8\_Efficient项目后续会分别设置相应的模块用于执行，目前正在更新中。

## 5、附件

### 5.1、YOLOv8读取权重

def attempt\_load\_weights(weights, device=None, inplace=True, fuse=False):    
    *# Loads an ensemble of models weights=[a,b,c] or a single model weights=[a] or weights=a*  
    from ultralytics.yolo.utils.downloads import attempt\_download    
    
    model = Ensemble()    
    for w in weights if isinstance(weights, list) else [weights]:    
        ckpt = torch.load(attempt\_download(w), map\_location='cpu')  *# load*  
        args = {\*\*DEFAULT\_CONFIG\_DICT, \*\*ckpt['train\_args']}  *# combine model and default args, preferring model args*  
        ckpt = (ckpt.get('ema') or ckpt['model']).to(device).float()  *# FP32 model*  
    
        ...

### **5.2、YOLOv5读取权重**

def attempt\_load(weights, device=None, inplace=True, fuse=True):    
    *# Loads an ensemble of models weights=[a,b,c] or a single model weights=[a] or weights=a*  
    from models.yolo import Detect, Model    
    
    model = Ensemble()    
    for w in weights if isinstance(weights, list) else [weights]:    
        ckpt = torch.load(attempt\_download(w), map\_location='cpu')  *# load*  
        ckpt = (ckpt.get('ema') or ckpt['model']).to(device).float()  *# FP32 model*  
        ...

**参考**

[1].https://github.com/isLinXu/YOLOv8\_Efficient.  
[2].https://github.com/isLinXu/model-metrics-plot.

# c4.yolov8训练自己的数据集以及应用

qq\_74368591 已于 2024-10-10 14:17:36 修改

原文链接：https://blog.csdn.net/qq\_74368591/article/details/142756914

## **环境配置**

### **前文**

github获取yolov8

github链接：<https://github.com/ultralytics/ultralytics.git>

环境配置 ：pip install ultralytics

### **CUDA配置**

如果没有安装英伟达驱动你需要安装nvidia驱动建议最新版

https://www.nvidia.cn/geforce/drivers/

安装好驱动后需要安装一个工具CUDA TOOLKIT

https://developer.nvidia.com/cuda-toolkit

需要用到CUDA训练所以需要下载一个CUDA的另一个工具cuDNN

这里具体引用别人的方法

[Cuda和cuDNN安装教程(超级详细)-CSDN博客](

https://blog.csdn.net/jhsignal/article/details/111401628

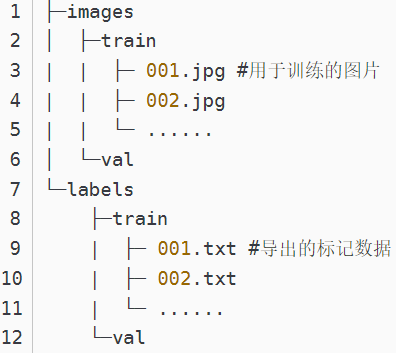
)

## **数据集的准备**

这里我们需要创建dataset文件夹，yolov8.yaml和dataset.yaml

其中dataset文件夹下面需要再建立2个文件一个是images和labels

dataset文件夹资源分布

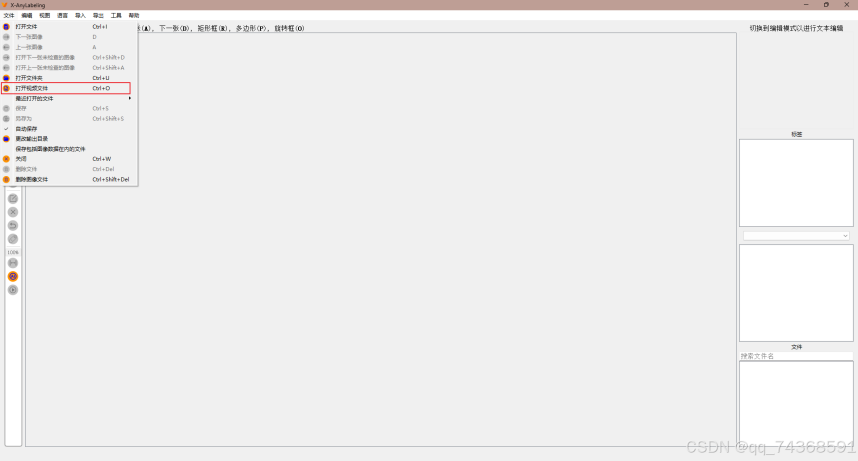


### **数据标记**

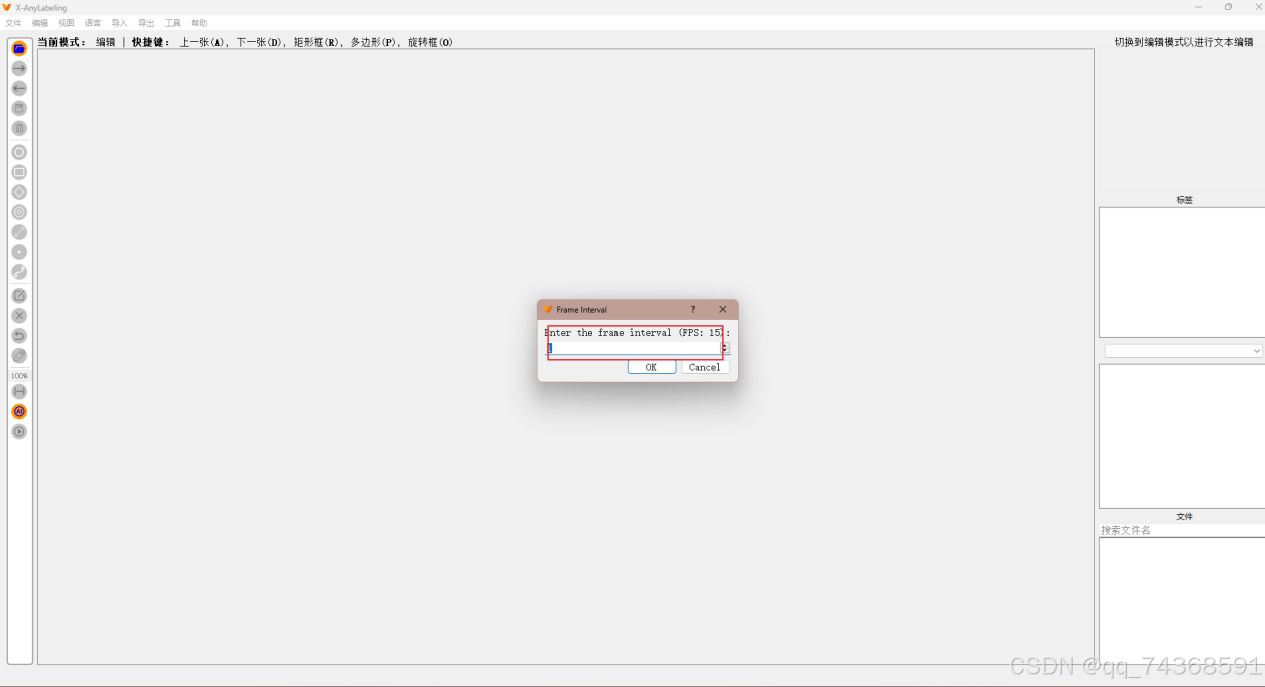
这里运用的是X-AnyLabeling ： <https://github.com/CVHub520/X-AnyLabeling>

建议是git到pycharm里面运行

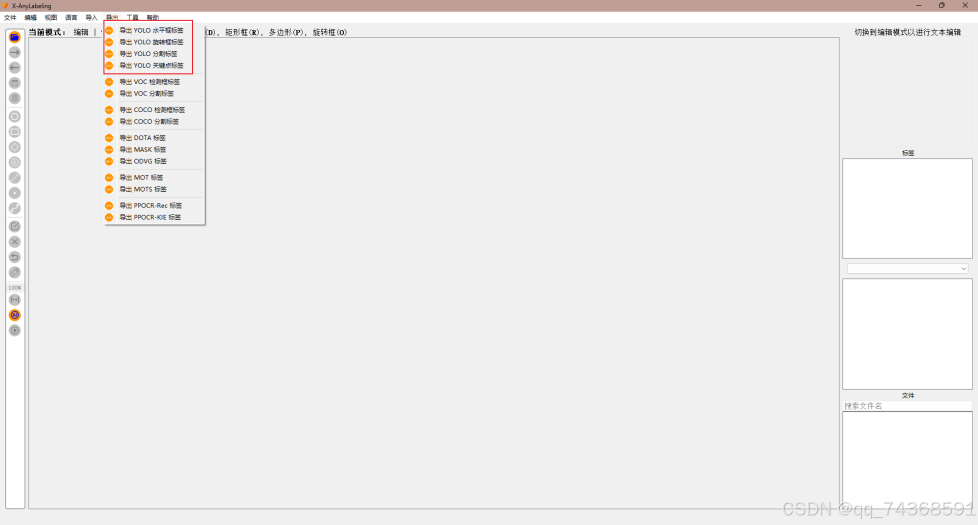
命令行输入： python anylabeling/app.py



打开文件，选择视频文件

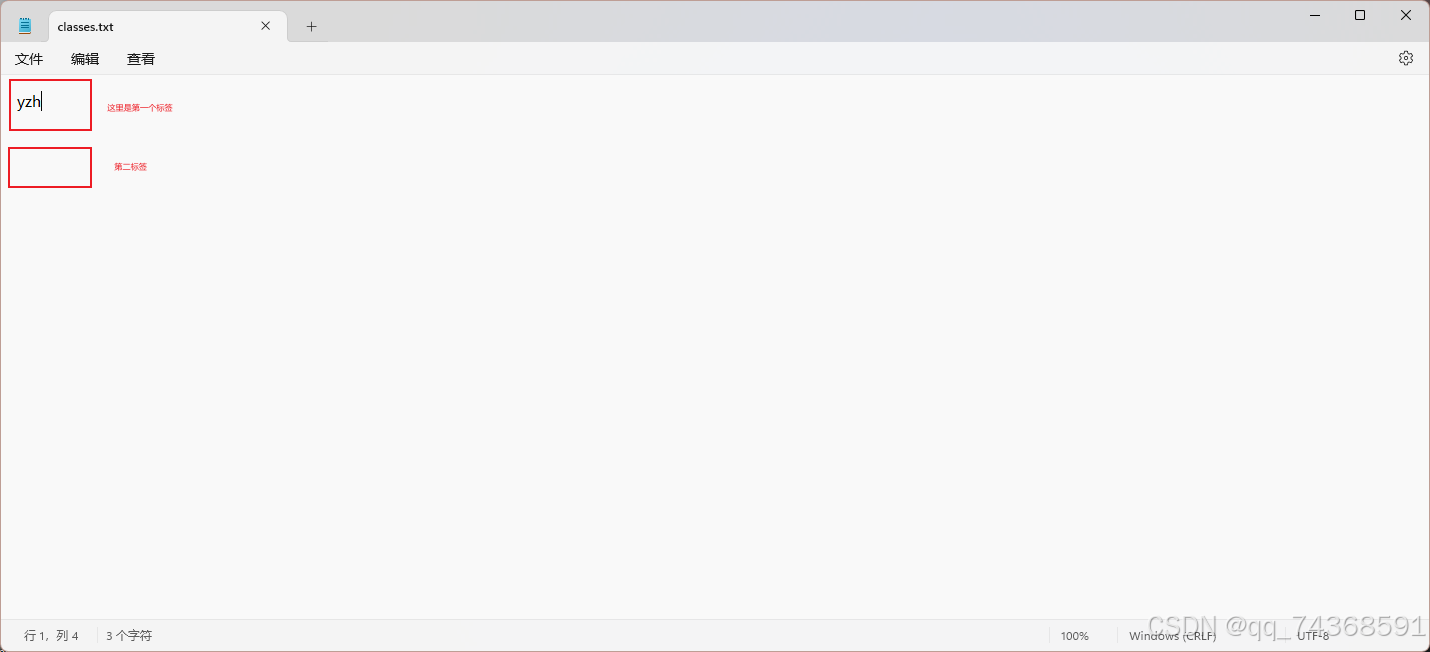


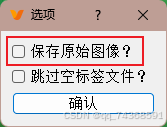
框的部分是选择获取的频率，建议越大越好，越大图片越清晰



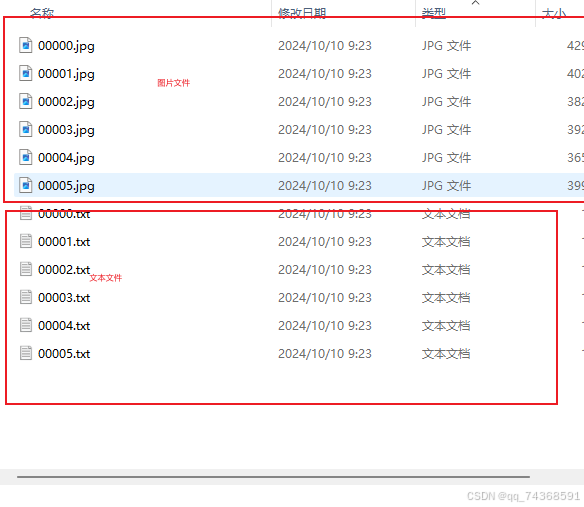
框完后需要创建一个txt文件里面放置标签

按标记的顺序发放至置





软件会自动生成一个labels文件夹，里面包含图片文件和文本文件



图片文件复制到images里面的train里面文本文件放到val文件里面

再把images文件里面的train文件和val文件复制到labels文件里面

### **配置yaml文件**

dataset.yaml文件、# 这里的路径是相对与运行目录(一般为train.py程序)的路径，如不清楚如何填写，建议填写绝对路径，参考如下：

# train: C:/Users/your\_username/PycharmProjects/yolov5/source/images/train

# val: C:/Users/your\_username/PycharmProjects/yolov5/source/images/val

train: ./source/images/train

val: ./source/images/val

# nc为种类数量

nc: 1

# 名称，classes.txt，X-AnyLabeling中的群组编号顺序对应

names:

0: 'A meng'

# 1: 'other'

# 2: ......

## **训练**

先下载你需要的预训练模型： <https://github.com/ultralytics/assets/releases>

在pycharm命令行输入

yolo task=detect mode=train model=yolov8.pt data=dataset.yaml epochs=1000 batch=16

以上参数解释如下：

task：选择任务类型，可选['detect', 'segment', 'classify', 'init']

mode: 选择是训练、验证还是预测的任务蕾西 可选['train', 'val', 'predict']

model: 选择yolov8不同的模型配置文件，可选yolov8s.pt、yolov8m.pt、yolov8l.pt、yolov8x.pt

data: 选择生成的数据集配置文件

epochs：指的就是训练次数,训练次数过多会过拟合，同时过多也会导致训练时间增加

batch：差不多是通道的意思，一次性加载图片的数量如果过大会导致电脑死机黑屏

## **使用训练好的模型**

训练完成得到.pt文件这文件就是模型文件

import cv2 as cv

from ultralytics import YOLO

import time

class yzh\_detect:

class \_\_results\_\_:

def \_\_init\_\_(self):

self.name = []

self.x = []

self.y = []

self.confidence = []

self.image = None

def \_\_init\_\_(self, model\_path):

# 加载模型

self.model = YOLO(model\_path)

print(f"Model loaded from: {model\_path}")

self.is\_detecting = False

def detect(self, image):

'''

检测图像中的物体

:param image: 输入图像

:return: 结果类结构

result.name: 物体名称列表

result.x: 物体中心点x坐标列表

result.y: 物体中心点y坐标列表

result.confidence: 物体置信度列表

result.image: 检测后的图像

'''

if self.is\_detecting:

return None

self.is\_detecting = True

results = self.model(image, augment=True)

# 存储检测结果的列表

result = self.\_\_results\_\_()

# 遍历检测结果

try:

for \*xyxy, conf, cls in results[0].boxes.data:

label = f'{self.model.names[int(cls)]} {conf:.2f}'

# 画出矩形框

cv.rectangle(image, (int(xyxy[0]), int(xyxy[1])), (int(xyxy[2]), int(xyxy[3])), (0, 0, 255), 2)

cv.putText(image, label, (int(xyxy[0]), int(xyxy[1]) - 10), cv.FONT\_HERSHEY\_SIMPLEX, 1.0, (0, 0, 255),

2)

# 计算中心点坐标

center\_x = int((xyxy[0] + xyxy[2]) / 2)

center\_y = int((xyxy[1] + xyxy[3]) / 2)

# 画出中心点

cv.circle(image, (center\_x, center\_y), 5, (255, 0, 0), -1)

# 存储中心点坐标,物体名称,置信度和图像

result.name.append(self.model.names[int(cls)])

result.x.append(center\_x)

result.y.append(center\_y)

result.confidence.append(float(conf))

result.image = image

except Exception as e:

print("未检测到物体或发生错误:", e)

self.is\_detecting = False

return result

if \_\_name\_\_ == "\_\_main\_\_":

# 加载自定义训练的YOLOv8模型

model\_path = 'best.pt'

detector = yzh\_detect(model\_path)

# 打开摄像头

cap = cv.VideoCapture(1) ##打开摄像头

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

# 检测物体

results = detector.detect(frame)

# 显示结果图像

if results and results.image is not None:

cv.imshow('YOLOv8 Real-time Object Detection', results.image)

# 退出条件

if cv.waitKey(1) & 0xFF == ord('q'):

break

# 清理工作

cap.release()

cv.destroyAllWindows()

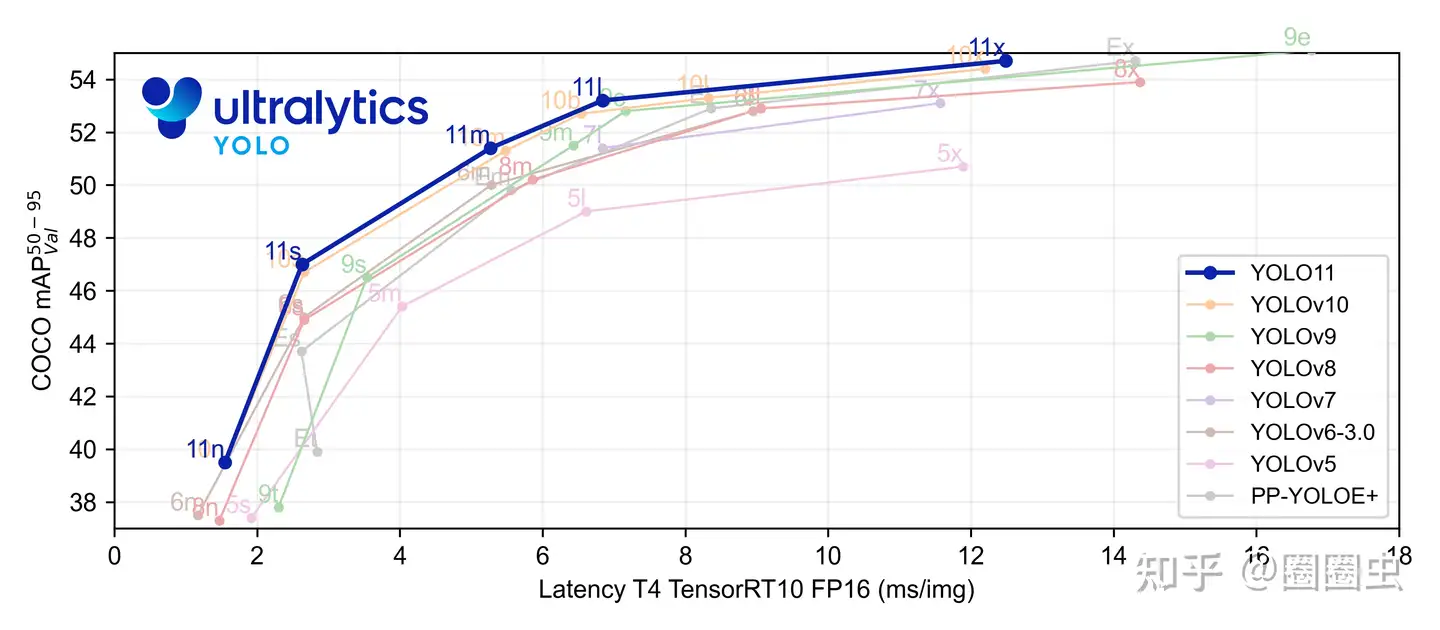
# C5.如何评价Ultralytics出的YOLOv11？

87 人赞同了该回答

## **#0背景**

[Ultralytics](https://zhida.zhihu.com/search?content_id=691925464&content_type=Answer&match_order=1&q=Ultralytics&zhida_source=entity" \t "https://www.zhihu.com/question/694696741/answer/_blank) 的官方今天开源了 YOLO11（Discover YOLO11, the latest advancement in [state-of-the-art](https://zhida.zhihu.com/search?content_id=691925464&content_type=Answer&match_order=1&q=state-of-the-art&zhida_source=entity" \t "https://www.zhihu.com/question/694696741/answer/_blank) object detection, offering unmatched accuracy and efficiency for diverse computer vision tasks.），是的它叫做 YOLO11 不是 YOLOv11……

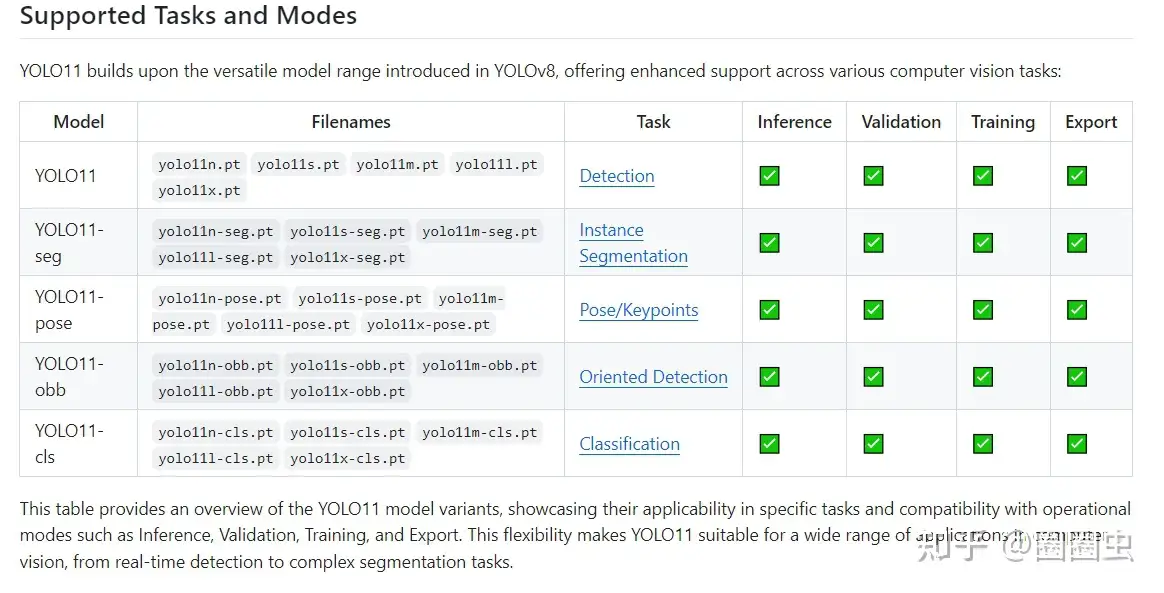
[ultralytics​github.com/ultralytics/ultralytics](https://link.zhihu.com/?target=https://github.com/ultralytics/ultralytics" \t "https://www.zhihu.com/question/694696741/answer/_blank)



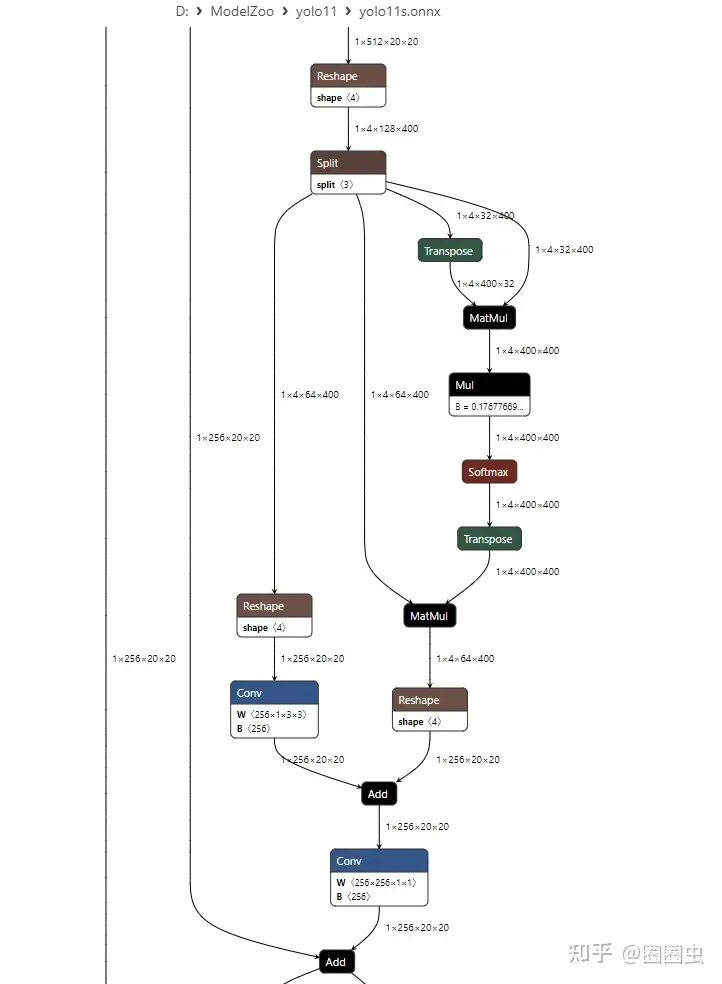
## **#1.YOLO11**

Ultralytics 的上一个官方版本是 2023年1月发布的 [YOLOv8](https://zhida.zhihu.com/search?content_id=691925464&content_type=Answer&match_order=1&q=YOLOv8&zhida_source=entity" \t "https://www.zhihu.com/question/694696741/answer/_blank) 系列，已经过去了1年多了。那么 YOLO11 作为最新官方版本都有哪些更新呢？

* 强化特征提取：YOLO11 采用了改进的后端和颈部架构，增强了特征提取能力，从而实现更精确的物体检测和更复杂的任务执行；
* 优化效率与速度：YOLO11 引入了经过改进的架构设计和优化的训练流程，实现了更快的处理速度，并保持了准确性和性能之间的最佳平衡；
* 更精准、更少参数：随着模型设计的进步，YOLO11m 在 [COCO 数据集](https://zhida.zhihu.com/search?content_id=691925464&content_type=Answer&match_order=1&q=COCO+%E6%95%B0%E6%8D%AE%E9%9B%86&zhida_source=entity" \t "https://www.zhihu.com/question/694696741/answer/_blank)上的平均精确度（mAP）更高，同时使用了比 YOLOv8m 少 22% 的参数，在不牺牲精度的情况下提高了计算效率；
* 跨环境适应性：YOLO11 可以在各种环境中无缝部署，包括边缘设备、云平台以及支持 NVIDIA GPU 的系统，确保最大限度的灵活性（这句话非常认可，因为在 AX650N 上直接用 YOLOv8 的 sample 就能运行）；
* YOLO11 支持多种任务：无论是目标检测、实例分割、图像分类、姿态估计还是定向对象检测（OBB），YOLO11 的设计旨在满足各种计算机视觉挑战。（本文只介绍目标检测任务的部署）



新增了 C2PSA 模块用于特征增强，其实就是一个小的 Self-Attention



## **#2.ONNX 模型获得**

### **#2.1环境准备**

默认已经基于 [miniconda](https://zhida.zhihu.com/search?content_id=691925464&content_type=Answer&match_order=1&q=miniconda&zhida_source=entity" \t "https://www.zhihu.com/question/694696741/answer/_blank) 搭建 ultralytics repo 必要的 python 环境。同时准备好导出 ONNX 模型必要的第三方依赖。

pip install onnx

pip install onnxruntime

pip install onnxsim

完成 ultralytics 项目安装。

pip install ultralytics

### **#2.2导出 ONNX**

下面的 python 脚本用于 下载、运行、导出 YOLO11s 的 ONNX 模型

from ultralytics import YOLO

# Load a model

model = YOLO("yolo11s.pt")

model.info()

# Use the model

results = model("ssd\_horse.jpg")

# Save the results

results[0].save("yolo11s-result.jpg")

# Export to onnx with simplify

model.export(format='onnx', simplify=True)

执行过程如下

YOLO11s summary: 319 layers, 9,458,752 parameters, 0 gradients, 21.7 GFLOPs

image 1/1 /home/qtang/ultralytics/ssd\_horse.jpg: 480x640 3 persons, 1 car, 1 truck, 1 bench, 1 dog, 1 horse, 1 chair, 94.4ms

Speed: 4.4ms preprocess, 94.4ms inference, 394.4ms postprocess per image at shape (1, 3, 480, 640)

Ultralytics 8.3.0 Python-3.9.18 torch-2.2.0+cu121 CPU (AMD Ryzen Threadripper 3970X 32-Core Processor)

PyTorch: starting from 'yolo11s.pt' with input shape (1, 3, 640, 640) BCHW and output shape(s) (1, 84, 8400) (18.4 MB)

ONNX: starting export with onnx 1.15.0 opset 17...

ONNX: slimming with onnxslim 0.1.34...

ONNX: export success ✅ 1.2s, saved as 'yolo11s.onnx' (36.3 MB)

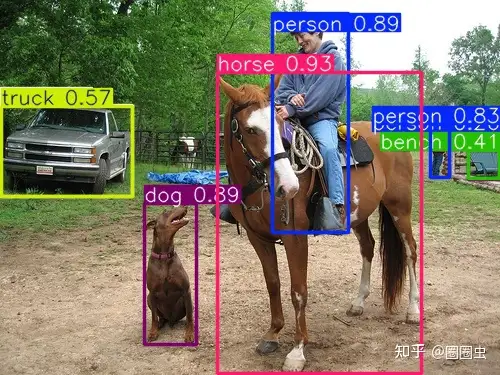
Export complete (1.6s)

Results saved to /home/qtang/ultralytics

Predict: yolo predict task=detect model=yolo11s.onnx imgsz=640

Validate: yolo val task=detect model=yolo11s.onnx imgsz=640 data=/usr/src/ultralytics/ultralytics/cfg/datasets/coco.yaml

Visualize: https://netron.app



执行完成后将获得 yolo11s.onnx 模型以及使用测试图片 ssd\_horse.jpg 的推理计算结果。

## **#3.模型转换**

### **#3.1ONNX 模型优化**

在使用 pulsar2 工具链转换之前，先对前一步获得的 yolo11s.onnx 模型进行必要的计算图优化，便于提高模型部署效率。

**import** onnx

input\_path **=** "yolo11s.onnx"output\_path **=** "yolo11s-cut.onnx"input\_names **=** ["images"]output\_names **=** ["/model.23/Concat\_output\_0", "/model.23/Concat\_1\_output\_0", "/model.23/Concat\_2\_output\_0"]

onnx**.**utils**.**extract\_model(input\_path, output\_path, input\_names, output\_names)

执行完成后将获得 yolo11s-cut.onnx 模型

### **#3.2Pulsar2 编译**

默认已经搭建好了基于 Docker 的 [Pulsar2](https://link.zhihu.com/?target=https://pan.baidu.com/s/1FftjDGfh06L6zccWqa0K-A?pwd=ns6s" \t "https://www.zhihu.com/question/694696741/answer/_blank) 工具链使用环境。为了提高学习效率，我们已经在网盘中提供了所有的材料

|  |  |
| --- | --- |
| 文件名 | 描述 |
| ax\_yolo11 | YOLO11 DEMO 程序，可在 AX650N 上运行 |
| yolo11\_config.json | pulsar2 build 依赖的[配置文件](https://zhida.zhihu.com/search?content_id=691925464&content_type=Answer&match_order=1&q=%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6&zhida_source=entity" \t "https://www.zhihu.com/question/694696741/answer/_blank) |
| yolo11s-cut.onnx | 已导出的 ONNX 模型 |
| cut-onnx.py | onnx yolo11 后处理裁剪脚本 |
| ssd\_horse.jpg | 测试图片 |
| yolo11s.axmodel | Pulsar2 build 生成的 NPU 模型文件 axmodel |

转换命令如下

pulsar2 build --input yolo11s-cut.onnx --config yolo11\_config.json --output\_dir ouput --target\_hardware AX650 --npu\_mode NPU3 --output\_name yolo11s.axmodel

转换完成之后，将在 ./output 路径下生成 yolo11s.axmodel 文件，该文件用于最终上板运行。

## **#4.上板示例**

默认已完成 AX650N 开发板测试环境搭建，能通过 SSH 登录进控制台。

ax\_yolo11 请参考 [ax-samples](https://link.zhihu.com/?target=https://github.com/AXERA-TECH/ax-samples" \t "https://www.zhihu.com/question/694696741/answer/_blank) 项目编译获得。执行结果如下

/opt/test # ./ax\_yolo11 -m yolo11s.axmodel -i ssd\_horse.jpg -r 10

--------------------------------------

model file : yolo11s.axmodel

image file : ssd\_horse.jpg

img\_h, img\_w : 640 640

--------------------------------------

Engine creating handle is done.

Engine creating context is done.

Engine get io info is done.

Engine alloc io is done.

Engine push input is done.

--------------------------------------

post process cost time:4.04 ms

--------------------------------------

Repeat 10 times, avg time 3.20 ms, max\_time 3.21 ms, min\_time 3.19 ms

--------------------------------------

detection num: 6

17: 96%, [ 216, 74, 421, 373], horse

0: 91%, [ 274, 14, 349, 231], person

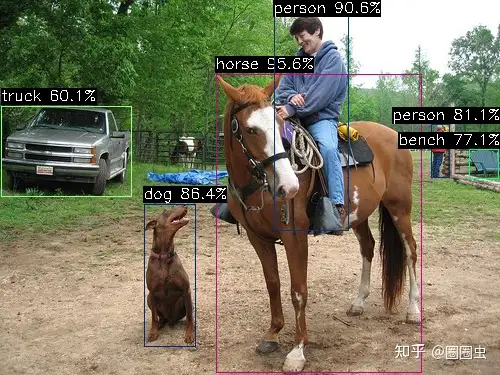
16: 86%, [ 144, 204, 196, 347], dog

0: 81%, [ 431, 124, 450, 178], person

13: 77%, [ 469, 149, 499, 179], bench

7: 60%, [ 1, 106, 132, 197], truck

--------------------------------------



### **#4.1性能统计**

基于 AX650N 平台，18TOPs@Int8

|  |  |  |  |
| --- | --- | --- | --- |
| Models | 参数量(M) | 计算量(GFLOPs) | 帧率(FPS) |
| YOLO11n | 2.6 | 6.5 | 653 |
| YOLO11s | 9.4 | 21.5 | 312 |
| YOLO11m | 20.1 | 68.0 | 114 |
| YOLO11l | 25.3 | 86.9 | 88 |
| YOLO11x | 59.9 | 194.9 | 40 |

本篇文章会根据用户反馈不断更新和完善，毕竟基于 YOLO11 的部署流程跟 YOLOv8 完全一样，说不定哪里手贱就抄错了……

* 本文相关文件均可通过 [百度网盘](https://link.zhihu.com/?target=https://pan.baidu.com/s/1ewX5H9riYeD4CDXBRal_FQ?pwd=95xn" \t "https://www.zhihu.com/question/694696741/answer/_blank) 获取
* 本文部署 YOLO11 的方案同样适用于 Ultralytics 的 YOLOv8、[YOLOv10](https://zhida.zhihu.com/search?content_id=691925464&content_type=Answer&match_order=1&q=YOLOv10&zhida_source=entity" \t "https://www.zhihu.com/question/694696741/answer/_blank)

## **#5.Acknowledge**

* [三木君：速通一下YOLOv11的推理和ONNX转换](https://zhuanlan.zhihu.com/p/769076635" \t "https://www.zhihu.com/question/694696741/answer/_blank)
* [编辑于 2024-10-22 14:41](https://www.zhihu.com/question/694696741/answer/3861567276" \t "https://www.zhihu.com/question/694696741/answer/_blank)

# C6.YOLOV11 训练自己的数据集及ONNX/tensorRT模型部署

**[Ctrl CV](https://www.zhihu.com/people/mldl-55" \t "https://zhuanlan.zhihu.com/p/_blank)** keep learning 编辑于 2024-10-16 15:33・IP 属地江苏

https://zhuanlan.zhihu.com/p/848129092

YOLOV11 code: [https://github.com/ultralytics/ultralytics](https://link.zhihu.com/?target=https://github.com/ultralytics/ultralytics" \t "https://zhuanlan.zhihu.com/p/_blank)

## 一、CBC**[数据集](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=%E6%95%B0%E6%8D%AE%E9%9B%86&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)**

CBC数据集地址如下：

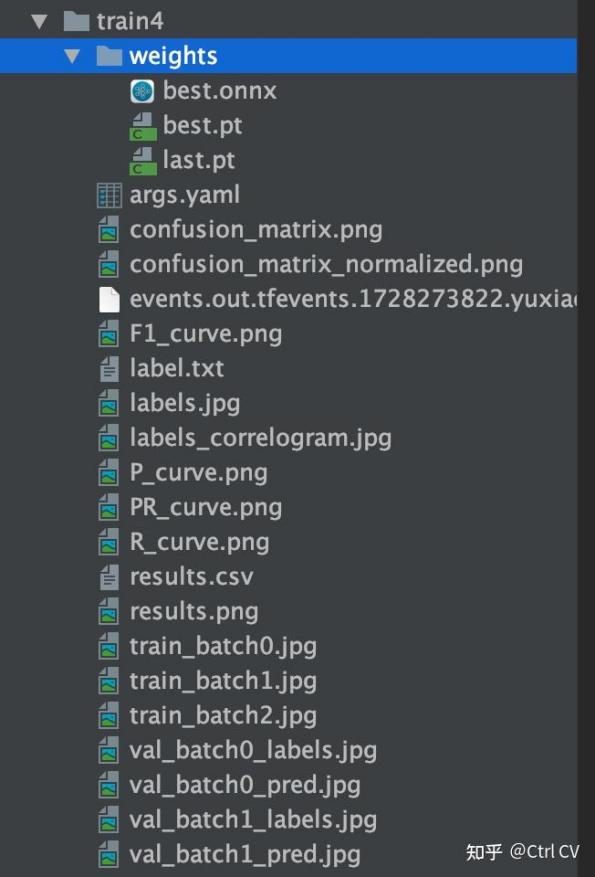
https://github.com/MahmudulAlam/Complete-Blood-Cell-Count-Dataset

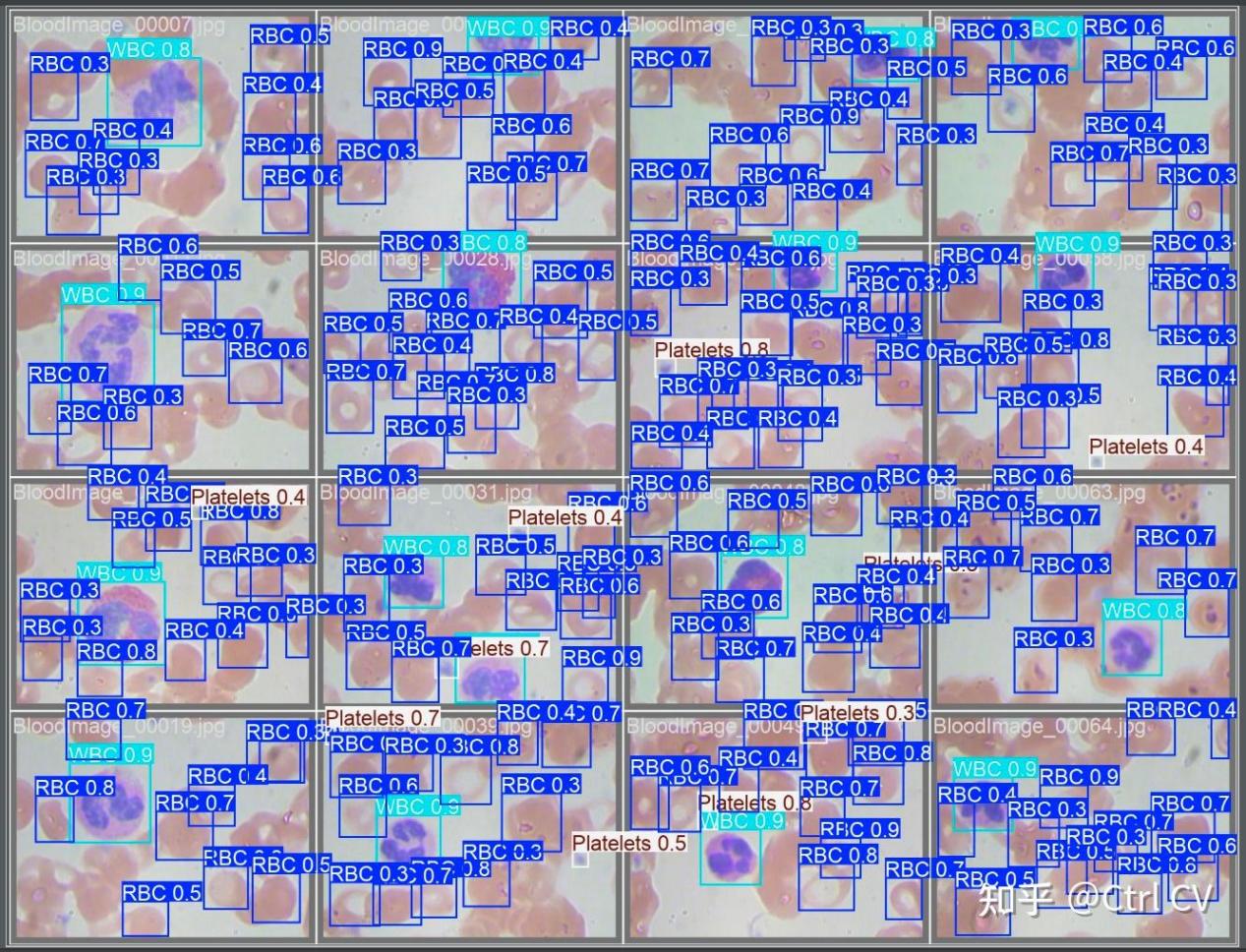


CBC数据集包含 360 张红血细胞图像及其注释文件，分为[训练集](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=%E8%AE%AD%E7%BB%83%E9%9B%86&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)与验证集。训练文件夹包含 300 张带有注释的图像。测试和验证文件夹都包含 60 张带有注释的图像。我们对原始数据集进行了一些修改以准备此 CBC 数据集，并将数据集分成三部分。在360张涂片图像中，首先使用300张带注释的血细胞图像作为训练集，然后将其余60张带有注释的图像用作测试集。

## 二、[模型训练](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=%E6%A8%A1%E5%9E%8B%E8%AE%AD%E7%BB%83&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)

yolo train model=yolo11n.pt data=cbc\_dataset.yaml epochs=25 imgsz=640 batch=4

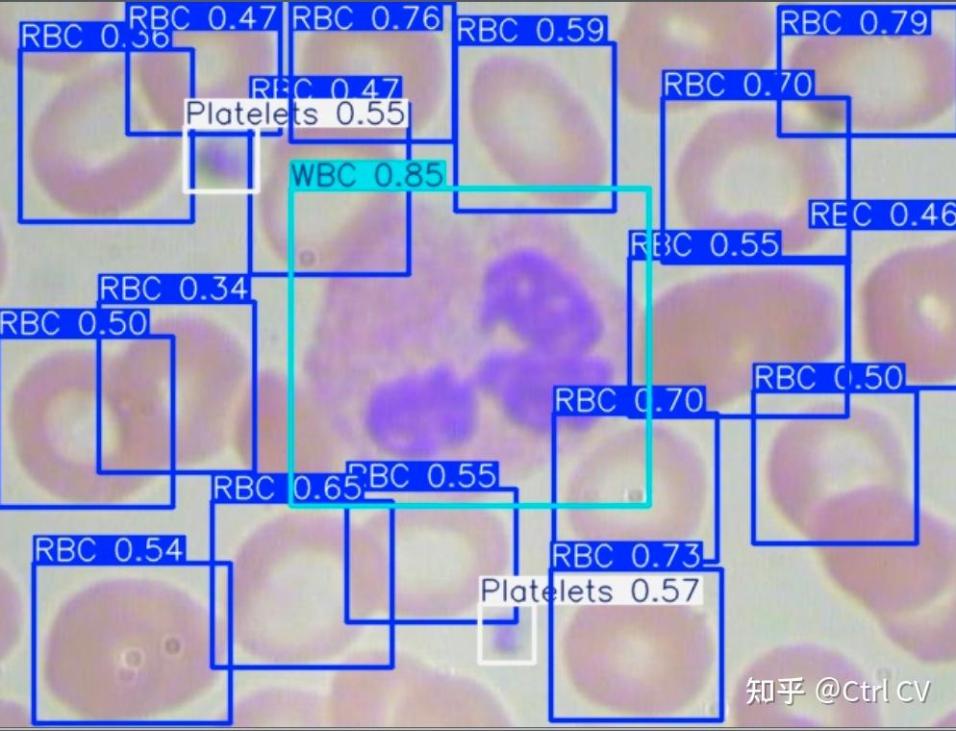




## 三、模型导出与测试

yolo export model=cbc\_11.pt format=onnx

yolo predict model=cbc\_11.pt source=./cbc



## 四、[部署推理](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=%E9%83%A8%E7%BD%B2%E6%8E%A8%E7%90%86&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)

转成ONNX格式文件以后，基于OpenVINO-Python部署推理，相关代码如下

class\_list = load\_classes()

colors = [(255, 255, 0), (0, 255, 0), (0, 255, 255), (255, 0, 0)]

ie = Core()

for device in ie.available\_devices:

print(device)

# Read IR

model = ie.read\_model(model="yolov11-cbc.onnx")

compiled\_model = ie.compile\_model(model=model, device\_name="CPU")

output\_layer = compiled\_model.output(0)

## xywhr

frame = cv.imread("D:/wh860.jpg")

# frame = cv.imread("wh300.jpg")

# frame = cv.imread("obb\_01.jpeg")

bgr = format\_[yolov8](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=yolov8&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)(frame)

img\_h, img\_w, img\_c = bgr.shape

start = time.time()

image = cv.dnn.blobFromImage(bgr, 1 / 255.0, (1024, 1024), swapRB=True, crop=False)

res = compiled\_model([image])[output\_layer] # 1x25x8400

rows = np.squeeze(res, 0).T

boxes, confidences, angles, class\_ids = post\_process(rows)

indexes = cv.dnn.NMSBoxes(boxes, confidences, 0.25, 0.45)

M = np.zeros((2, 3), dtype=np.float32)

for index in indexes:

box = boxes[index]

#d1 = -angles[index]

color = colors[int(class\_ids[index]) % len(colors)]

rr = int((box[2] + box[3])/4)

cv.circle(frame, (box[0]+int(box[2]/2), box[1]+int(box[3]/2)), rr-4, color, 2)

cv.putText(frame, class\_list[class\_ids[index]], (box[0] + int(box[2] / 2), box[1] + int(box[3] / 2)), cv.FONT\_HERSHEY\_SIMPLEX, .5, (0, 0, 0))

cv.putText(frame, "Red Blood Cell@2024", (20, 45), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

#pts = [(box[0], box[1]), (box[0]+box[2], box[1]), (box[0]+box[2], box[1]+box[3]), (box[0], box[1]+box[3])]

#rrt\_pts = get\_rotate\_point(pts, M, d1, box)

#cv.drawContours(frame, [np.asarray(rrt\_pts).astype(np.int32)], 0, (255, 0, 255), 2)

#cv.rectangle(frame, (box[0], box[1] - 20), (box[0] + box[2], box[1]), color, -1)

#cv.putText(frame, class\_list[class\_ids[index]], (box[0], box[1]-8), cv.FONT\_HERSHEY\_SIMPLEX, .5, (255, 255, 255))

end = time.time()

inf\_end = end - start

fps = 1 / inf\_end

fps\_label = "FPS: %.2f" % fps

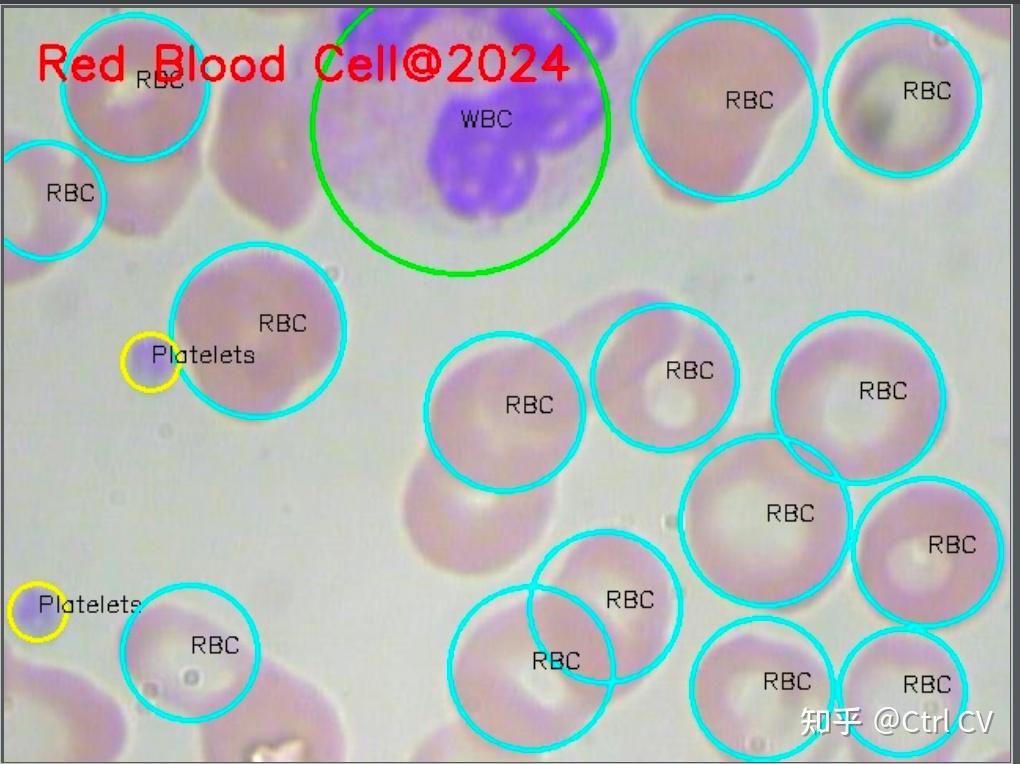
cv.putText(frame, fps\_label, (20, 45), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 0, 255), 2)

cv.imshow("YOLOv11+OpenVINO2024 Object Detection", frame)

cv.imwrite("cbc\_result.jpg", frame)

cv.waitKey(0)

cv.destroyAllWindows()



### ONNX-Python部署推理

import argparse

import cv2

import numpy as np

import onnxruntime as ort

import os

class YOLOv11:

"""YOLOv11 object detection model class for handling inference and visualization."""

def \_\_init\_\_(self, onnx\_model, confidence\_thres, iou\_thres):

"""

Initialize the YOLOv11 model for object detection.

Args:

onnx\_model (str): Path to the ONNX model.

confidence\_thres (float): Confidence threshold for filtering detections.

iou\_thres (float): IoU threshold for non-maximum [suppression](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=suppression&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank) (NMS).

"""

# Load the ONNX model and specify the providers (CUDA and CPU)

self.onnx\_model = onnx\_model

self.session = ort.InferenceSession(self.onnx\_model,

providers=["CUDAExecutionProvider", "CPUExecutionProvider"])

self.confidence\_thres = confidence\_thres

self.[iou\_thres](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=3&q=iou_thres&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank) = iou\_thres

self.model\_inputs = self.session.get\_inputs()

# Store the shape of the input for later use

self.input\_width = self.[model\_inputs](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=2&q=model_inputs&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)[0].shape[2]

self.input\_height = self.model\_inputs[0].shape[3]

# Load the class names from the COCO dataset (80 classes)

self.classes = ['RBC', 'WBC', 'Platelets']

# self.classes = ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat',

# 'traffic light',

# 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep',

# 'cow',

# 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase',

# 'frisbee',

# 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard',

# '[surfboard](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=surfboard&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)',

# 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana',

# 'apple',

# 'sandwich', 'orange', '[broccoli](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=broccoli&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair',

# 'couch',

# 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard',

# 'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase',

# 'scissors', 'teddy bear', 'hair drier', 'toothbrush']

# Generate a color palette for the classes (random colors for each class)

self.color\_palette = np.random.uniform(0, 255, size=(len(self.classes), 3))

def draw\_detections(self, img, box, score, class\_id):

"""

Draws bounding boxes and labels on the input image based on the detected objects.

Args:

img: The input image to draw detections on.

box: Detected bounding box.

score: Corresponding detection score.

class\_id: Class ID for the detected object.

Returns:

None

"""

# Extract the coordinates of the bounding box

x1, y1, w, h = box

# Retrieve the color for the class ID

color = self.color\_palette[class\_id]

# Draw the bounding box on the image

cv2.rectangle(img, (int(x1), int(y1)), (int(x1 + w), int(y1 + h)), color, 2)

# Create the label text with class name and score

label = f"{self.classes[class\_id]}: {score:.2f}"

# Calculate the [dimensions](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=dimensions&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank) of the label text

(label\_width, label\_height), \_ = cv2.getTextSize(label, cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, 1)

# Calculate the position of the label text

label\_x = x1

label\_y = y1 - 10 if y1 - 10 > label\_height else y1 + 10

# Draw a filled rectangle as the background for the label text

cv2.rectangle(

img, (label\_x, label\_y - label\_height), (label\_x + label\_width, label\_y + label\_height), color, cv2.FILLED

)

# Draw the label text on the image

cv2.putText(img, label, (label\_x, label\_y), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE\_AA)

def [preprocess](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=preprocess&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)(self, image):

"""

Preprocess the input image before inference.

Args:

image: The input image to be preprocessed.

Returns:

Preprocessed image data in the expected format for the model.

"""

# Convert the image color space from BGR to RGB

img = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Resize the image to match the input shape expected by the model

img = cv2.resize(img, (self.input\_width, self.input\_height))

# Normalize the image data by dividing by 255.0 to bring pixel values between 0 and 1

image\_data = np.array(img) / 255.0

# Transpose the image to have the channel dimension as the first dimension

image\_data = np.transpose(image\_data, (2, 0, 1)) # Channel first

# Expand the dimensions of the image data to match the expected input shape (batch size 1)

image\_data = np.expand\_dims(image\_data, axis=0).astype(np.float32)

# Return the preprocessed image data

return image\_data

def postprocess(self, input\_image, output):

"""

Post-process the model's output to get final bounding boxes, scores, and class IDs.

Args:

input\_image: The original input image.

output: The model's raw output.

Returns:

The input image with bounding boxes and labels drawn.

"""

# Transpose and squeeze the output to match the expected shape

outputs = np.transpose(np.squeeze(output[0]))

# Get the number of rows in the outputs array (number of detections)

rows = outputs.shape[0]

# Lists to store the bounding boxes, scores, and class IDs of the detections

boxes = []

scores = []

class\_ids = []

# Calculate the scaling factors for the bounding box coordinates

img\_height, img\_width = input\_image.shape[:2]

x\_factor = img\_width / self.input\_width

y\_factor = img\_height / self.input\_height

# Iterate over each row in the outputs array

for i in range(rows):

# Extract the class scores from the current row

classes\_scores = outputs[i][4:]

# Find the maximum score among the class scores

max\_score = np.amax(classes\_scores)

# If the maximum score is above the confidence threshold

if max\_score >= self.confidence\_thres:

# Get the class ID with the highest score

class\_id = np.[argmax](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=argmax&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)(classes\_scores)

# Extract the bounding box coordinates from the current row

x, y, w, h = outputs[i][0], outputs[i][1], outputs[i][2], outputs[i][3]

# Calculate the scaled coordinates of the bounding box

left = int((x - w / 2) \* x\_factor)

top = int((y - h / 2) \* y\_factor)

width = int(w \* x\_factor)

height = int(h \* y\_factor)

# Add the class ID, score, and box coordinates to the respective lists

class\_ids.append(class\_id)

scores.append(max\_score)

boxes.append([left, top, width, height])

# Apply non-maximum suppression to filter out overlapping bounding boxes

indices = cv2.dnn.NMSBoxes(boxes, scores, self.confidence\_thres, self.iou\_thres)

for i in indices:

# Get the box, score, and class ID corresponding to the index

box = boxes[i]

score = scores[i]

class\_id = class\_ids[i]

# Draw the bounding box and label on the input image

self.draw\_detections(input\_image, box, score, class\_id)

# Return the image with detections drawn

return input\_image

def main(self, input\_image):

"""

Run the full pipeline: preprocess, inference, postprocess.

Args:

input\_image: The image to run detection on.

Returns:

The processed image with detections.

"""

# Preprocess the image

img\_data = self.preprocess(input\_image)

# Run inference on the preprocessed image

outputs = self.session.run(None, {self.model\_inputs[0].name: img\_data})

# Postprocess the output and return the final image

return self.postprocess(input\_image, outputs)

if \_\_name\_\_ == "\_\_main\_\_":

# Create an argument parser to handle command-line arguments

parser = [argparse](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=2&q=argparse&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank).ArgumentParser()

parser.add\_argument("--model", type=str, default="./runs/detect/train4/weights/best.onnx", help="Input your ONNX model.")

parser.add\_argument("--image", type=str, default="CBC\_data/Testing/images/BloodImage\_00339.jpg", help="Path to your image file.")

parser.add\_argument("--conf-thres", type=float, default=0.25, help="Confidence threshold.")

parser.add\_argument("--[iou-thres](https://zhida.zhihu.com/search?content_id=248937470&content_type=Article&match_order=1&q=iou-thres&zhida_source=entity" \t "https://zhuanlan.zhihu.com/p/_blank)", type=float, default=0.5, help="IoU threshold for NMS.")

parser.add\_argument("--output", type=str, default="output.jpg", help="Path to save the output image.")

args = parser.parse\_args()

# Initialize the YOLOv11 model with the ONNX model and thresholds

model = YOLOv11(args.model, args.conf\_thres, args.iou\_thres)

# Load the input image

image = cv2.imread(args.image)

# Run object detection on the input image

output\_image = model.main(image)

# Display the processed image

cv2.imshow("YOLOv11", output\_image)

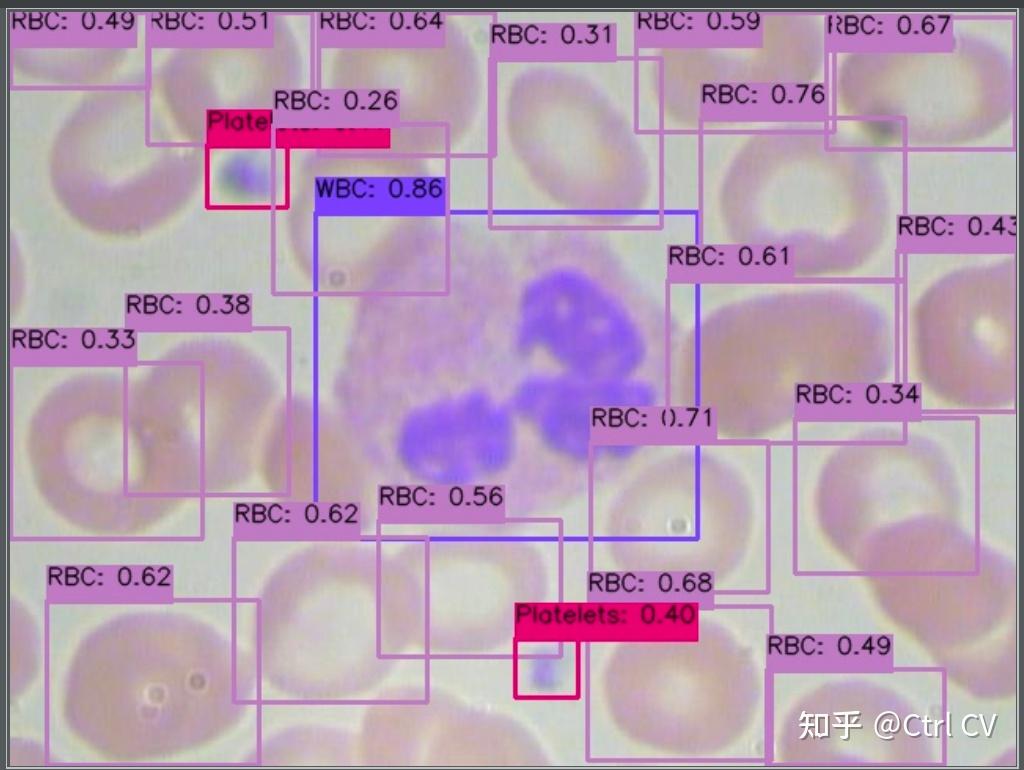
# Save the processed image

cv2.imwrite(args.output, output\_image)

# Check for key press to exit

cv2.waitKey(0)

cv2.destroyAllWindows()



### TensorRT部署推理

from ultralytics import YOLO

from PIL import Image

# Load the YOLOv11 model

model = YOLO("best.pt")

# Export the model to TensorRT format

model.export(format="engine") # creates 'best.engine'

# Load the exported TensorRT model

tensorrt\_model = YOLO("best.engine")

# Run inference

results = tensorrt\_model("BloodImage\_00339.jpg")

# Visualize the results

for i, r in enumerate(results):

# Plot results image

im\_bgr = r.plot() # BGR-order numpy array

im\_rgb = Image.fromarray(im\_bgr[..., ::-1]) # RGB-order PIL image

# Show results to screen (in supported environments)

r.show()

# Save results to disk

r.save(filename=f"results{i}.jpg")

# C7.pyproject.toml 如何用

pyproject.toml是 Python 项目中的一个重要配置文件，用于定义项目的构建系统、依赖关系和其他元数据等，以下是其常见的使用方法：

## #1.基本结构与作用

### #1.1构建系统配置：

**[build-system]**

requires = ["setuptools>=61.0", "wheel"]

build-backend = "setuptools.build\_meta"

其中，requires指定了构建项目所需的依赖包及其版本，build-backend指定了用于构建项目的后端工具，如setuptools、poetry、hatchling等。

### #1.2项目元数据配置：

**[project]**

name = "my\_project"

version = "0.1.0"

description = "A short description of my project"

authors = [

{name = "Your Name", email = "your\_email@example.com"},

]

license = "MIT"

requires-python = ">=3.6"

这里定义了项目的基本信息，如项目名称、版本号、描述、作者、许可证以及所需的 Python 版本等。

### #1.3依赖管理配置：

**[project]**

dependencies = [

"numpy",

"pandas",

"torch>=1.10.0",

]

**[project.optional-dependencies]**

dev = [

"pytest",

"flake8",

"black"

]

gui = ["pyqt5"]

dependencies列出了项目运行时的依赖包，optional-dependencies则可以定义可选依赖，如开发依赖、特定功能依赖等。可以通过pip install your-project-name[dev]的方式安装指定的可选依赖。

## #2.使用步骤

**#(1)创建文件**：在项目根目录下创建一个名为pyproject.toml的文件。如果使用某些构建**工具，如poetry**，可以通过poetry init命令自动创建。

**#(2)配置构建系统**：根据项目所使用的构建后端，在[build-system]部分配置相应的requires和build-backend。例如，如果使用setuptools，可以按照上述示例进行配置。

**#(3)填写项目元数据**：在[project]部分填写项目的名称、版本、描述、作者、许可证等信息。

**#(4)管理依赖关系**：在[project]的dependencies和optional-dependencies中列出项目的依赖包。可以手动添加，也可以使用构建工具提供的命令来添加，如poetry add命令。

**#(5)配置其他工具**：如果需要，可以在[tool]部分配置其他工具的特定设置，如代码格式化工具black、代码检查工具ruff等的配置。

**#(6)安装依赖和构建项目**：如果使用pip安装项目依赖，可以在项目根目录下运行pip install.。如果使用poetry，则运行poetry install。构建项目时，构建后端会根据pyproject.toml中的配置进行构建。