# A1 yolov5face

## A1.1 用github下载代码，用venv结合vscode跑起来

### Content：

A1.1.1 如何在有墙的情况下，安装并使用（拉仓库，代码同步与提交）github客户端

A1.1.2 venv结合vscode用法

A1.1.3第一次运行test.py需要的一些调整

A1.1.4数据集Dataset和.pt的路径调整，以及对应需要为test.py改哪些东西

### A1.1.1 墙的问题

以下操作，在有墙的时候，会非常不稳定：需要翻墙（当你可以翻墙时），或使用Watt Toolkit勾选github加速

1. 下载Github客户端安装软件GitHubDesktopSetup-x64.exe
2. Github客户端登陆
3. GIthub同步代码
4. 运行库代码是，自带的下载.pt和coco 数据集操作

### A1.1.2 venv结合vscode用法

请参考教程：[傻瓜级Venv+VsCode调试(Release和Debug两种模式)配置教程（含常见问题解决办法）\_vscode venv-CSDN博客](https://blog.csdn.net/weixin_42941859/article/details/140243028)

截止到2025年1月6日：按照以上教程有可能会出现，不能debug的问题（尤其迁移了环境之后）。解决的办法或许有使用vscode自带的venv，具体方法为ctrl+shift+p，选择Create Virtual Environment

### A1.1.3 仓库，requirements.txt和第一次运行

使用的仓库路径为 <https://github.com/deepcam-cn/yolov5-face>

问题：该仓库并没有包含requirements.txt以方便使用pip install直接配置环境。

解决：鉴于本仓库明显是基于yolov5的，所以从yolov5的官网（https://github.com/ultralytics/yolov5）下载了requirements.txt使用。

结果：基本是可以适配的，

问题a&解决：numpy.int需要全部替换成python自带的int

问题b&解决：common.py需要import warnings

问题c：

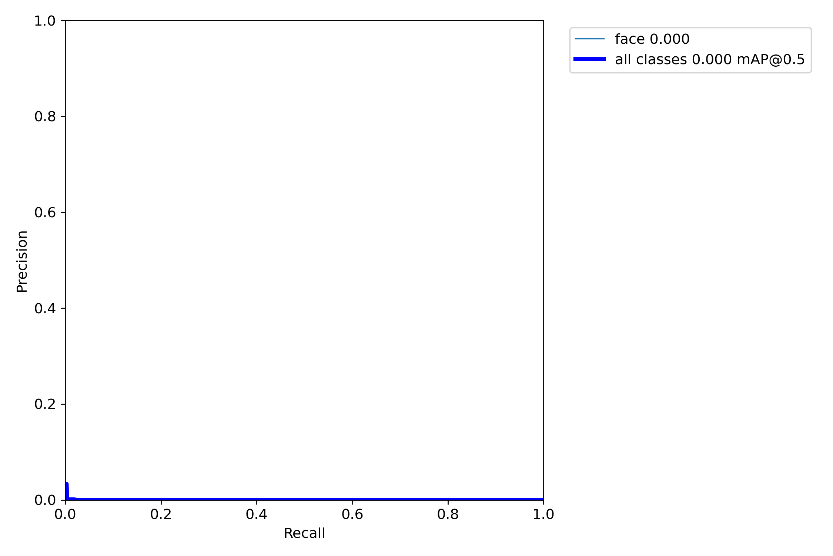
分析：根据仓库的issues 204（），此问题是由于test.py默认下载并运行的‘yolov5s.pt’

    parser.**add\_argument**('--weights', nargs='+', type=**str**, default='yolov5s.pt', help='model.pt path(s)')

解决：手动下载，在test.py将上面的default=‘yolov5s.py’换成default='yolov5n-face.pt'

Link: <https://pan.baidu.com/s/1UgiKwzFq5NXI2y-Zui1kiA> pwd: s5ow  
Link: <https://pan.baidu.com/s/1xsYns6cyB84aPDgXB7sNDQ> pwd: lw9j  
Link: <https://pan.baidu.com/s/1fyzLxZYx7Ja1_PCIWRhxbw> Link: eq0q  
Link: <https://pan.baidu.com/s/1oePvd2K6R4-gT0g7EERmdQ> pwd: jmtk  
Link: <https://pan.baidu.com/s/11l4qSEgA2-c7e8lpRt8iFw> pwd: 0mq7

问题d：运行出来的不正确PR数据集不太正确，如下图所示。



分析：这估计是因为coco数据集是多类型目标检测，以至于单目标网络的结果画不出来。

结果：暂时别管了

### A1.1.4 调整Dataset和.pt的目录

将Coco128数据集放到DataSet文件夹下。此时需要做两个同步的修改：

1. data/coco128.yaml，里面存着Coco128文件夹的绝对路径。
2. 删除\DataSet\coco128\labels\train2017.cache，里面缓存着Coco128的各张图的绝对路径。删掉之后，代码运行时就会再次自动产生，从而更新路径。

将pretrain过的yolov5n-0.5.pt、yolov5n-face.pt等文件收拢到\weights\Pretrained。

则test.py中的如下代码中的路径，也需要跟着修改：

parser.**add\_argument**('--weights', nargs='+', type=**str**, default='weights/Pretrained/yolov5n-face.pt', help='model.pt path(s)')

也需要跟着修改

## A1.2 WiderFace数据集

### Content：

链接：[WIDER FACE: A Face Detection Benchmark](http://shuoyang1213.me/WIDERFACE/)

截至到20250107，WiderFace的下载共有5项，分别是Training Images, Validation Images, Test Images，Face annotation和Example and formats of the submissions

### A1.2.1 Training Images, Validation Images, Test Images

三个zip中，分别有12880(train) + 16097(test) + 3226(val)张jpg图像

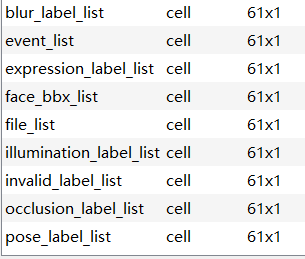
### A1.2.2 Face annotation: wider\_face\_val.mat/\_gt.txt

共有7个文件，除了readme.txt以外，以下先分析wider\_face\_val.mat

这个.mat文件得用matlab打开，没有matlab也可以用octave打开，指令如下：

load('C:\Users\xiaoyaopan\Downloads\wider\_face\_split\wider\_face\_val.mat')

使用此指令，得到两个变量：



这里的61对应的是WIDER\_val/images中的61个文件夹。

file\_list从这61个文件夹中，挑选对应的图像，例如file\_list[0][0]的值是0\_Parade\_marchingband\_1\_465，对应了WIDER\_val/images/0—Parade/0\_Parade\_marchingband\_1\_465.jpg。则blur\_label\_list[0][0]是这张图片中，各个脸的模糊程度，face\_bbx\_list[0][0]是各个脸的bbox。具体每个的含义，可参考wider\_face\_val.mat同路径下的readme.txt。

以这个顺序排列的ground truth, 则列在了wider\_face\_val\_bbx\_gt.txt中，也就是上面这个mat整理出了简约的形式。

### A1.2.3 Face annotation: wider\_face\_test\_filelist.txt

A1.2.2解释了，训练集train和验证集val的label们是如何排列在.mat和.txt文件中的。

而测试集test略有不同：概括地说，wider\_face\_test.mat相比wider\_face\_val.mat，以及wider\_face\_test\_filelist.txt相比wider\_face\_val.txt，删减只剩了文件路径和场景。

因为test集是用来做Benchmark的，不能用来训练，所以不提供ground truth.

### A1.2.4 retinaface\_gt\_v1.1.zip

在执行A1.2.5 Wider数据集转成Yolo5face格式之前，还需要下载retinaface\_gt\_v1.1.zip

链接：

<https://drive.google.com/file/d/1tU_IjyOwGQfGNUvZGwWWM4SwxKp2PUQ8/view?usp=sharing>

因为，Yolo5face产生的结果是‘框+5个关键点’，所以要用retinaface的输出作为gt。

### A1.2.5 Wider数据集转成Yolo5face格式

Yolo5face的官网是这样描述的：



这段代码的作用，是把WiderFace的数据集，转换成yolo格式

也就是每张图配一个txt



每个txt中，每一行是一张脸，14个float，对应的意义是：

框的中心点(x,y,w,h) + 5个关键点(x,y)

### A1.2.6 CPU训练：指令

在运行这个指令之前，和A1.1.4一样，需要调整data/widerface.yam 测试集的路径，train该路径应该设置为0\_Parade\_marchingband\_1\_5.txt所在的路径（如果A1.2.5结果正确的话，应该存在这个.txt），val路径依次类推。

python train.py --data data/widerface.yaml --cfg models/yolov5n-0.5.yaml --weights weights/Pretrained/yolov5n-0.5.pt --device cpu --workers 1 --epoch 1 --img-size 128

相比github官网上的指令：--data data/widerface.yaml --cfg models/yolov5s.yaml --weights 'pretrained models'，有如下修改

1. --weights weights/Pretrained/yolov5s-face.pt，因为A1.1.4中，模型.pt的文件位置调整了；同时因为A1.1.3中的问题c，需要把yolov5s.pt换成yolov5s-face.pt。
2. --device cpu 要设置成cpu，否则默认会调用GPU。
3. --workers 1 这个代表并行训练数量，默认是4，可能会把电脑内存塞满，因为每个worker都要载入一个batch

### A1.2.7 CPU训练：结果分析

问题a：文件夹exp/12下的results.txt是一组数据，但没有第一行数据是什么。

问题b：文件夹exp/12下的results.txt的第一列，从/249突变成/119，看不懂。

问题c：文件夹exp/12下的results.png，横轴是什么东西？纵轴是什么？为什么有阶跃？

TODO以上一系列问题，暂时列为TODO，与其调试yolo5face代码，不如调试yolo5代码。

# A2 小土堆B站Pytorch教程

https://www.bilibili.com/video/BV1hE411t7RN/?spm\_id\_from=333.337.search-card.all.click

## P1：Pytorch环境的配置及安装

0432：用anaconda做venv

0932：装什么样的Pytorch（尤其是和Cuda配合）

1026：Nvidia驱动查看与更新

1234：cudaToolKit和pytorch的bz2安装

1405：torch.cuda.is\_abailable()

## P2：IDE

略

## P2.5：Q&A torch.cuda.is\_available返回false

RT，需要时参考

后半截的内容是，conda安装慢怎么办，参考A1.1节的内容理应能够解决。

## P3 Python学习中的两大法宝函数dir和help

从0415开始看即可，是实际的例子

## P4 PyCharm以及Jupyter的使用及对比

暂略

## P5 PyTorch加载数据初认识

# B1 yolo5 quick start

源码官网 [ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite](https://github.com/ultralytics/yolov5)

<https://github.com/ultralytics/yolov5> [B1A]

## B1.1 Install

根据A1.1.1~A1.1.3的方法，安装yolo5

## B1.2 First Try (建议放弃)

[B1A]中建议的使用Yolo5的First try是以下这个：



问题：在用torch.hub.load和img=时，会有各种网络的问题，建议直接使用下一节B1.3

## B1.3 更简单的First Try

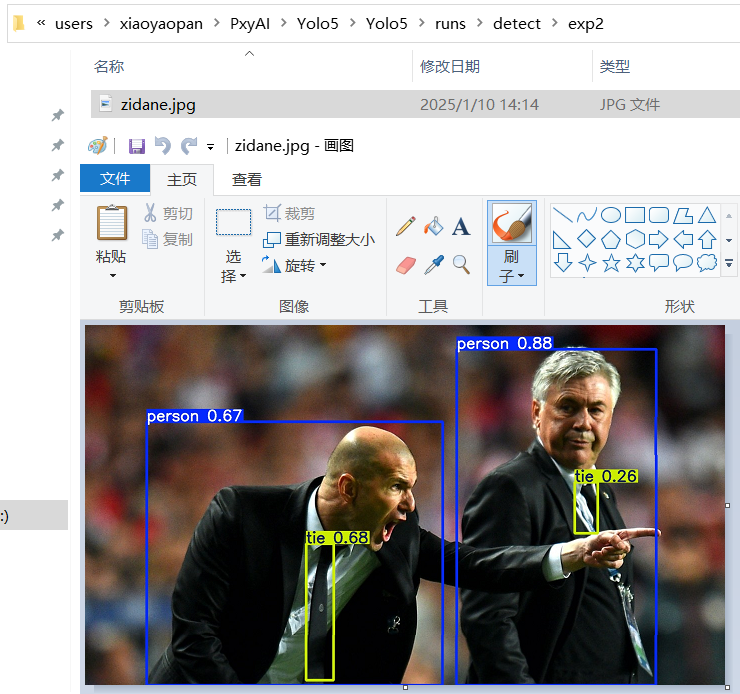
（在激活vevn之后）进入clone Yolo5的目录，使用以下指令：

python detect.py --weights./yolov5s.pt --source ./data/images/zidane.jpg

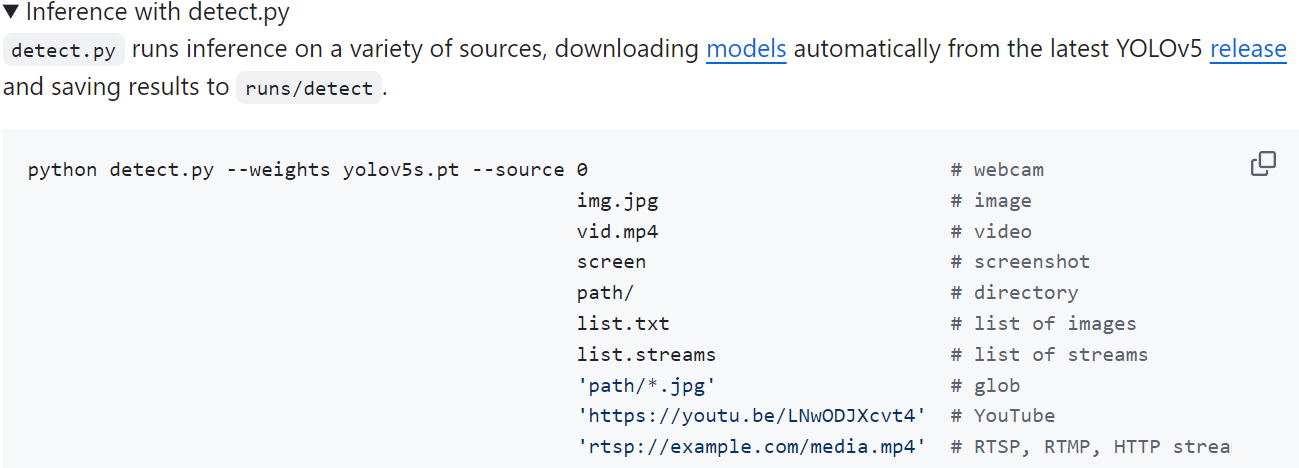
cmd成功界面如下：



出图界面如下：



以上测试，对应的是[B1A]的这一段：用到的detect.py文件的cmd 参数可以打开文件看，很容易理解。



## B1.4 用CPU跑Yolo5的coco128数据集

### 指令和指令解析：

（管理员）D:\users\xiaoyaopan\PxyAI\Yolo5\Yolo5>python train.py --data coco.yaml --epochs 1 --weights '' --cfg yolov5n.yaml --batch-size 32

指令解析：

基本等同于A1.2.6的作法，由于此时尚未安装Pytorch的Nvidia版本，所以不需要单独指定--device cpu

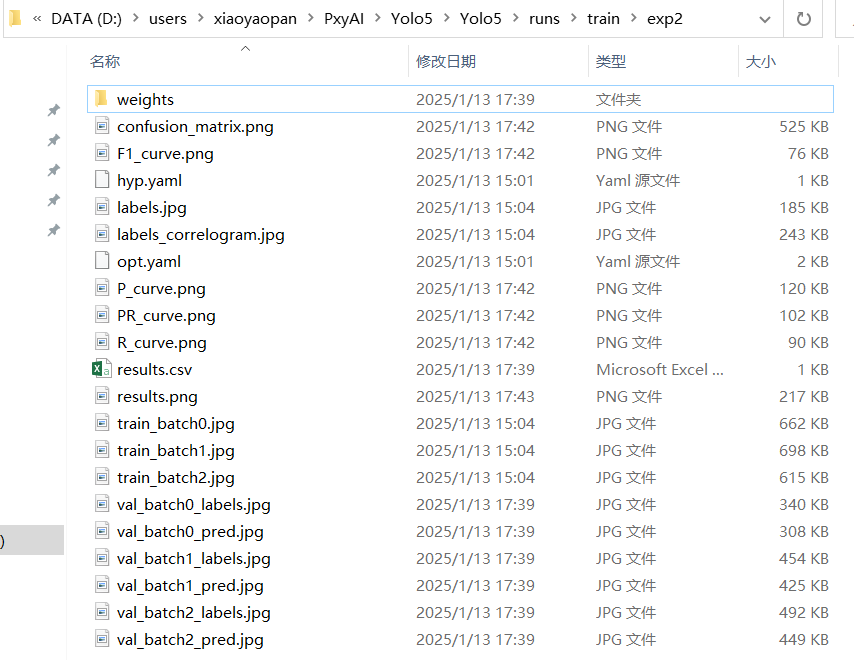
### 运行结果：

1. 下载coco128的数据集以及解压缩（尤其解压缩）非常慢，请耐心等待，并没有死机。

2. 运行完毕后cmd窗口如下：

TODO

3. 新产生了如下目录D:\users\xiaoyaopan\PxyAI\Yolo5\Yolo5\runs\train\exp2，也就是本次训练的产出，以下逐个分析：



### 结果分析：weights

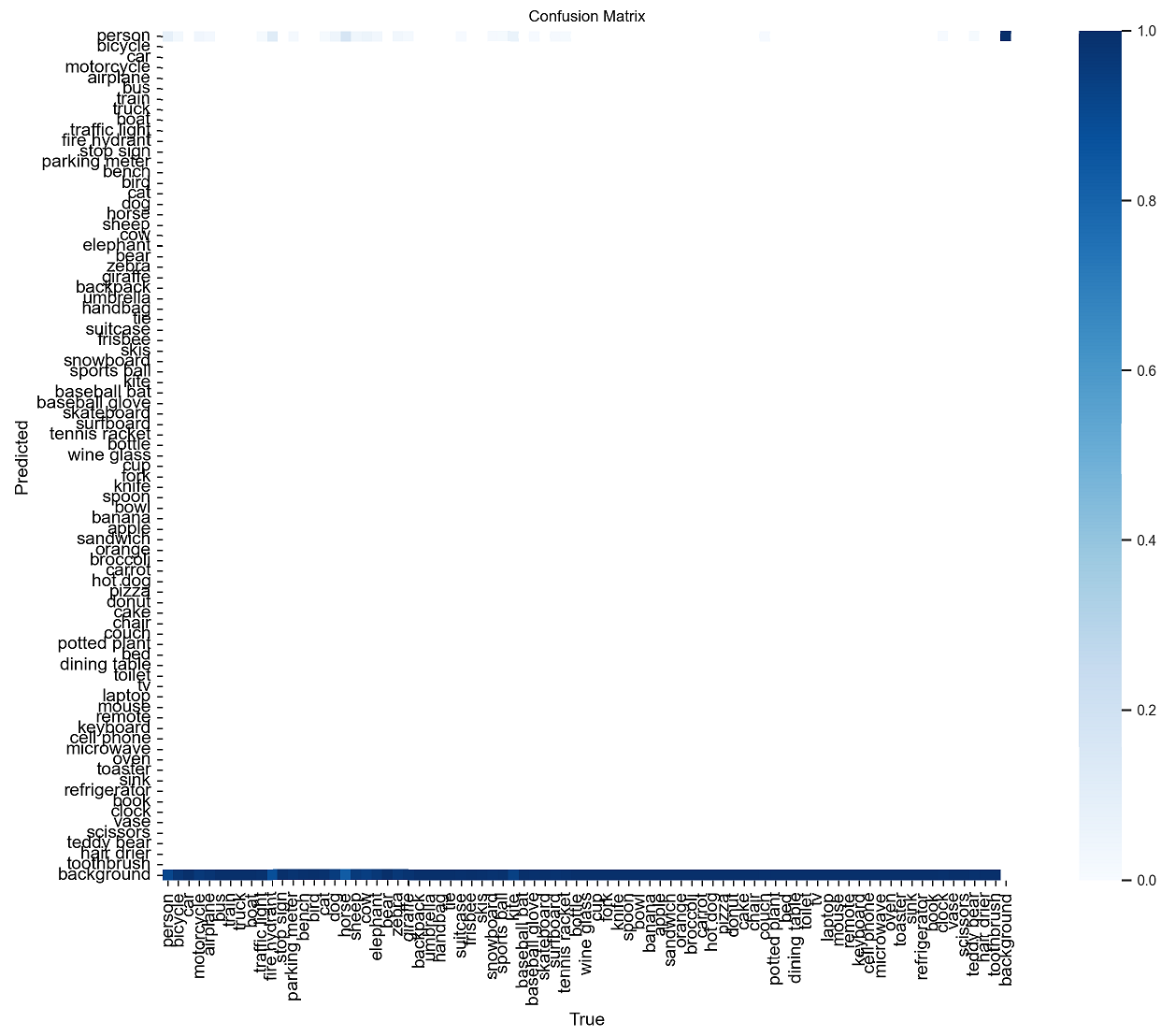
weights/best.pt 大小：4,021,032字节

weights/last.pt 大小：4,021,032字节

显然是同一个，毕竟只训练了一个epoch。

没有初始版本.pt可以比较，因为是从头开始训练的。

### 结果分析：confusion\_matrix.png 以及 normalized



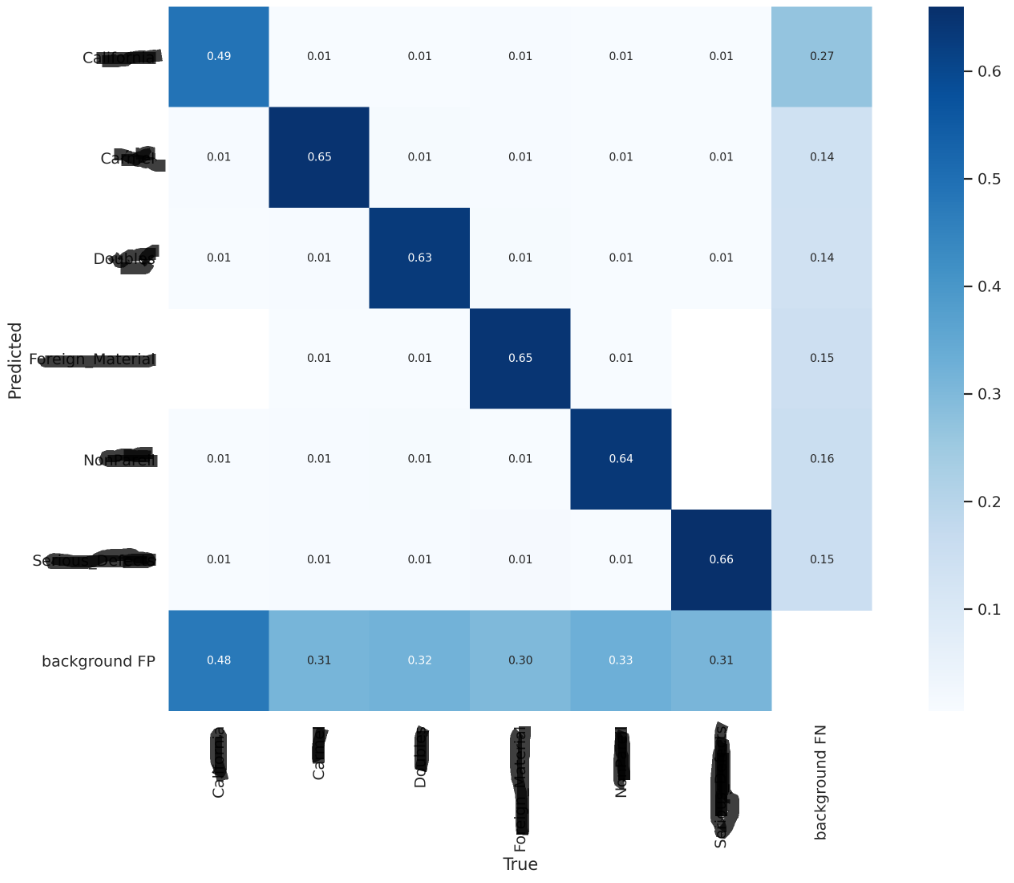
横坐标：True：指的是应该检测出，图像在这里有一个属于当前横坐标的框。

纵坐标：Predicted：指的是实际检测出，图像在这里有一个属于当前纵坐标的框。

为什么纵坐标Predicted为background的那一行颜色很深（比较接近1）：因为有大量的该检测出框，没有被检测出来，从而被归于了background。（原因是coco128数据集人的数量很多，在只有一次epoch的情况下还能有点拟合效果，其他的类别都欠拟合/欠训练了，详见“结果分析：labels.jpg”）。

为什么纵坐标为Predicted为person，横坐标True为background的那一格颜色很深（很接近1）：因为有大量的背景被检测成了人脸（或者说人类的框预测得大致对，但不准，所以和标准答案IoU之后，认为框出来的不能算人，还是得算background）

看一个理想中训练的不错的confusion\_matrix的样子：



事实上，以上两张图都是confusion matrix normalized，因为都以行方向**或**列方向归一化了。

当按行方向归一化时：预测为某个类的结果，其真实类分别是百分之多少。

当按列方向归一化时：真实值为某个类的结果，其预测的各结果类分别是百分之多少。

为什么没有按整个confusion matrix的2维方向进行归一化，因为这样的话，就会受到各类别标签数量不同的影响了。

### 结果分析：F1\_curve.png

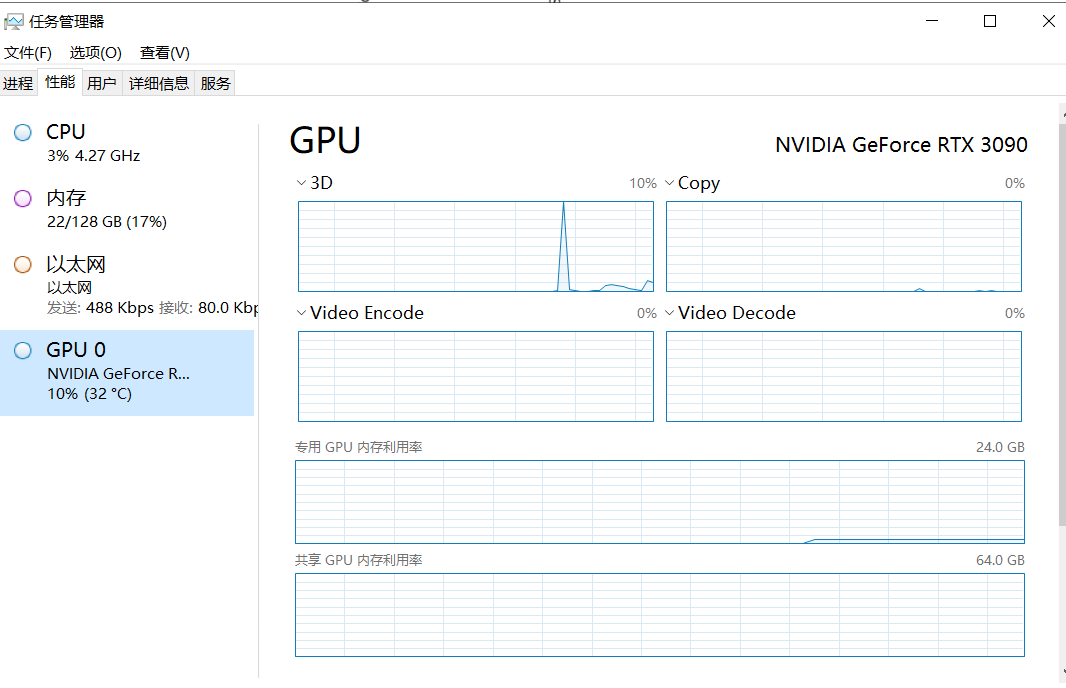
剩余结果分析，详见[B3.2 TrainCoco8-pose：结果分析](#_B3.2_TrainCoco8-pose：结果分析)

### 结果分析：hyp.yaml（炼丹超参数，暂略）

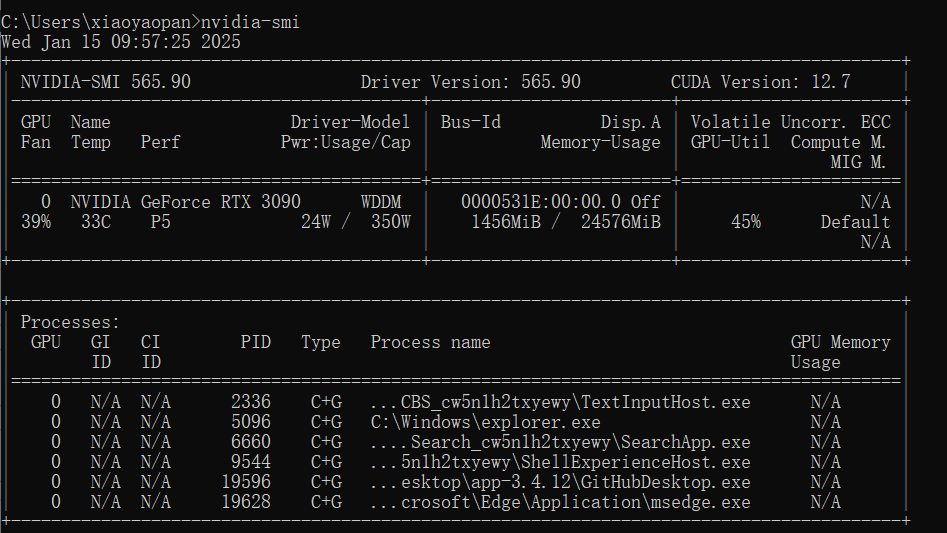
## B1.5 用GPU跑Yolo5的coco128

### Nvidia版本Pytorch的安装

当前显卡参数：



当前驱动版本：cmd窗口输入nvidia-msi



指令：

pip uninstall torchvision （不运行这一行会遇到的问题，详见[B1.5a1问题&解决](#_B1.5a1问题&解决：)）

pip install torch==2.4.1 torchvision==0.19.1 torchaudio==2.4.1 --index-url https://download.pytorch.org/whl/cu124

指令哪里来：<https://pytorch.org/get-started/previous-versions/>

为什么要用previous-version，因为本章中安装的python3.8不支持最新版的pytorch

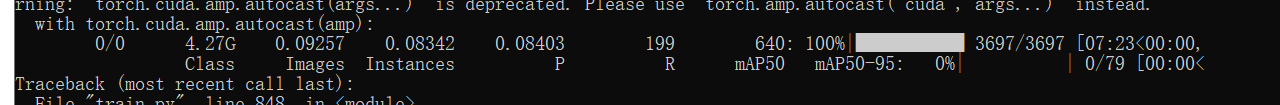
指令的效果：

使用这个指令，会自动让pip卸载掉cpu版本的pytorch，然后安装Nvidia版本的pytorch。

### 训练指令：

（管理员）D:\users\xiaoyaopan\PxyAI\Yolo5\Yolo5>python train.py --data coco.yaml --epochs 1 --weights '' --cfg yolov5n.yaml --batch-size 32

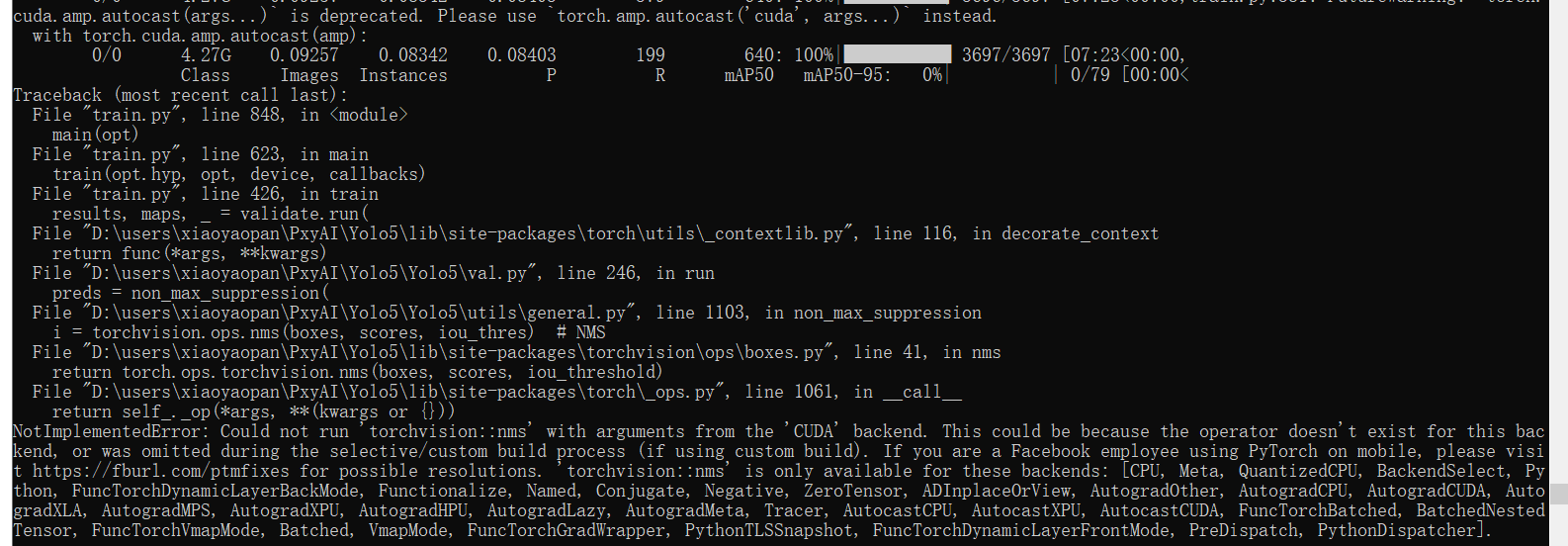
### 时间节省效果：



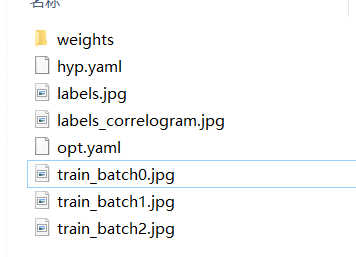
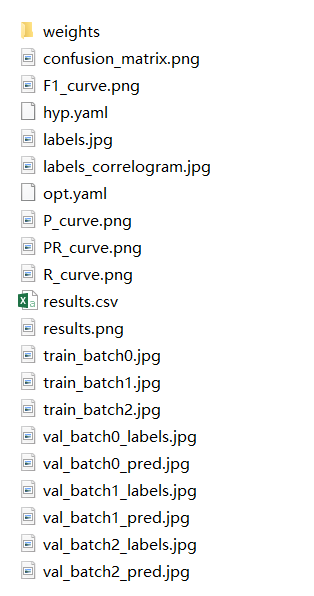
速度确实快多了，只用了7分23秒；相比之下，CPU版本的需要使用2~3个小时。

### B1.5a1问题&解决：

问题：



Torchvision.ops.nms调用有问题，导致val集没能跑，看不到验证集的验证结果

本次正常训练完，但val没能跑到的，exp文件夹下的内容（左图）

正常测试完成后的exp文件夹下内容（右图）

原因：

torch从CPU版本换成Nvidia版本后，torchvision没有跟着换（本应是跟着torch自动换的）

解决：

先卸载掉cpu版本的torchvision，在安装用上面的pip install torch指令，把合适版本的torchvision装回来

pip uninstall torchvision

pip install torch==2.4.1 torchvision==0.19.1 torchaudio==2.4.1 --index-url <https://download.pytorch.org/whl/cu124>

### 训练输出：



训练成功

## B1.6 断点续炼

### 训练指令：

（管理员）D:\users\xiaoyaopan\PxyAI\Yolo5\Yolo5>python train.py --data coco.yaml --epochs 10 --weights ”runs/train/Coco128\_Gpu001/weights/best.pt” --cfg yolov5n.yaml --batch-size 32

用时1.3小时（3697/3697）

将上面的结果重命名成 Coco128\_Gpu011，再用如下命令续炼

（管理员）D:\users\xiaoyaopan\PxyAI\Yolo5\Yolo5>python train.py --data coco.yaml --epochs 10 --weights ”runs/train/Coco128\_Gpu011/weights/best.pt” --cfg yolov5n.yaml --batch-size 64

用时1.28小时（1849/1849）

（管理员）D:\users\xiaoyaopan\PxyAI\Yolo5\Yolo5>python train.py --data coco.yaml --epochs 10 --weights ”runs/train/Coco128\_Gpu021/weights/best.pt” --cfg yolov5n.yaml --batch-size 128

用时1.28小时（1849/1849）

# B2 Yolo11

作为开始，先在Yolo11源代码上，复现[B3 Yolo11 OnlyUse](#_B3_Yolo11_OnlyUse)的基本操作，并学习Yolo11源代码。

## B2.0 环境配置

git clone Yolo11 代码（名字叫ultralytics）

注意，不要另外安装 pip install ultralytics，如果install了，请uninstall调它。

因为ultralytics库就是用Yolo11代码整合出来的，我们是来学习源代码的。

## B2.1

### B2.1.1 源代码跑通predict

用源代码，实现了[B3.1.1](#_B3.1.1_官方代码跑通predict)用调用ultralytics库实现的功能。

## B2.2 coco8-pose

coco8-pose数据集是coco-pose的一个最小子集，用来测试流程是否正确。

[COCO8-pose -Ultralytics YOLO 文档](https://docs.ultralytics.com/zh/datasets/pose/coco8-pose/#dataset-yaml) <https://docs.ultralytics.com/zh/datasets/pose/coco8-pose/#dataset-yaml>

### 运行代码：

用vscode打开clone下yolo11代码的文件夹

from **ultralytics**.**models** import **YOLO**

if \_\_name\_\_ == '\_\_main\_\_':

    model = **YOLO**(model='./ultralytics/weights/yolo11n-pose.pt')

    model.**train**(data="./ultralytics/cfg/datasets/coco8-pose.yaml", epochs=1, imgsz=320)

### 自动下载与路径不可控：

当D:\users\xiaoyaopan\PxyAI\Yolo11\ultralytics\ultralytics\cfg\datasets\coco8-pose.yaml的Download没有被注释的时候，会下载coco8-pose数据集，但路径比较奇怪，可以详见运行日志。

比较方便的解决办法是，下载完一次之后，把下载的内容移动到自己需要的地方（记得删除 .cache文件），然后注释掉coco8-pose.yaml中download的内容。那么路径就可控了。

## B2.3 coco8-2-pose

将coco8-pose中visible参数去除，

TODO：记录不visible的landmark该怎么办，其实我不知道yolo11会怎么处理

## B2.4 Lapa8

Lapa8转Yolo格式：详见github代码/datasets/Lapa/utilLapa8yolo11.py

训练能跑通（✔）

但pose误差毫不下降，不收敛（TODO1），

同时，import ultralytics和git clone ultralytics存在一些代码/功能/配置文件的重复（TODO2）

所以直接开启Yolo11OnlyUse，彻底从只import ultralytics使用，不care它代码的角度，来尽可能学习yolo。

# B3 Yolo11 OnlyUse

## B3.1 预测

用官方代码跑通predict、显示结果并研究其返回。

### B3.1.1 官方代码跑通predict

用源代码跑通请见：[B2.1.1](#_B2.1.1_源代码跑通predict)

运行如下代码：01\_BasicRunPos/01\_PredictBus.py

官方代码链接：https://docs.ultralytics.com/tasks/pose/#predict

from **ultralytics** import **YOLO**

import **cv2**

*# -----official code from https://docs.ultralytics.com/tasks/pose/#predict ----- #*

*# Load a model*

model = **YOLO**("yolo11n-pose.pt")  *# load an official model*

*# Predict with the model*

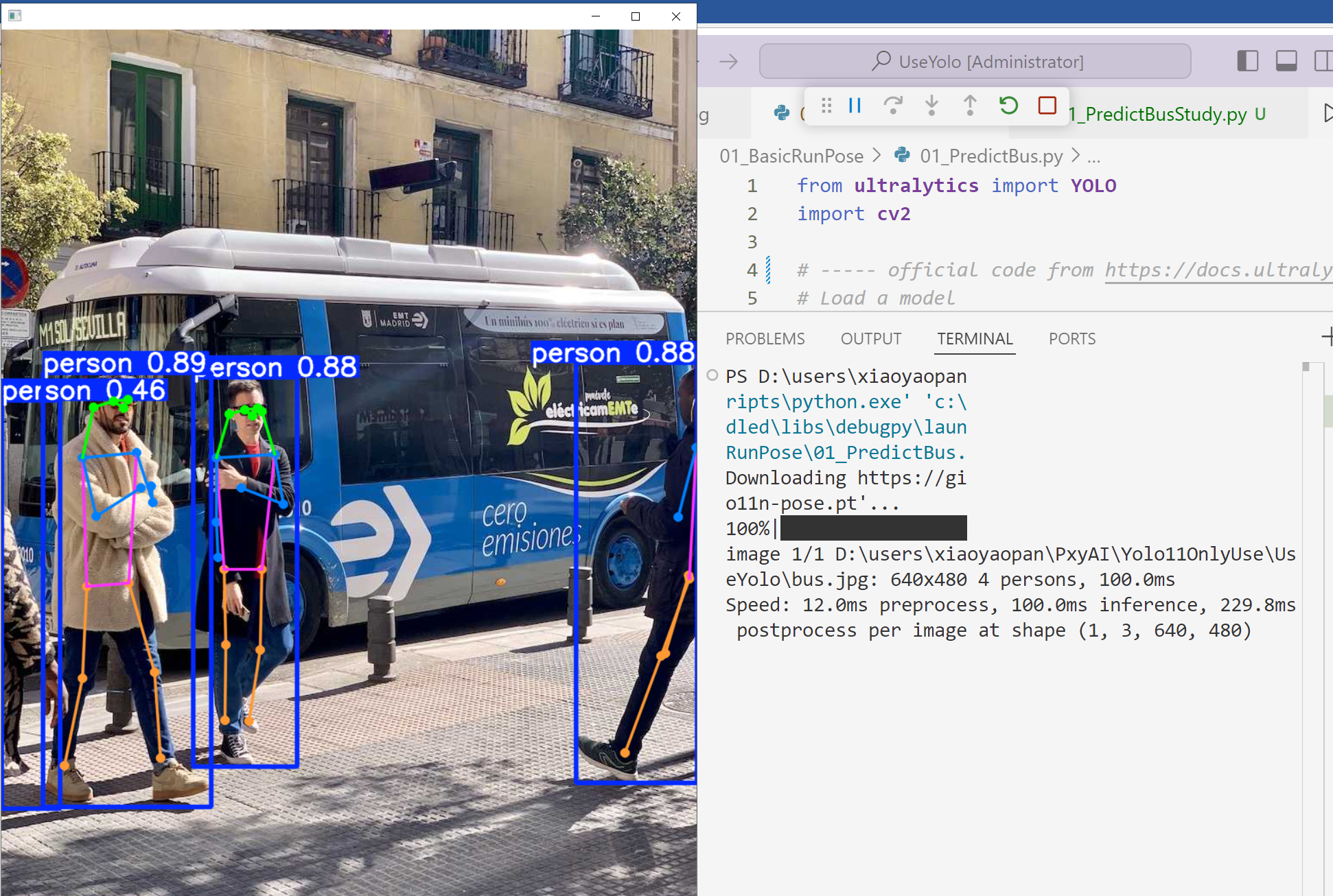
results = model("https://ultralytics.com/images/bus.jpg")  *# predict on an image*

img = results[0].plot()

**cv2**.**imshow**("", img)

**cv2**.**waitKey**(0)

结果如下：



### B3.1.2 官方predict代码分析

目的：分析B3.1.1中的代码，各个输出都是什么东西。

运行如下代码：01\_BasicRunPos/01\_PredictBusStudy.py

Q: 能否代码不要下载神经网络.pt和运行图片（参考）

A: 可以，将对应代码改成下面这个样子就行，也就是不再默认（裸）路径就好。

*# model = YOLO("yolo11n-pose.pt")  # load an official model*

**model** = **YOLO**("weights/yolo11n-pose.pt")  *# load an local model*

*# results = model("https://ultralytics.com/images/bus.jpg")  # predict an image after download*

results = **model**("01\_BasicRunPose/01\_bus.jpg")  *# predict a local image*

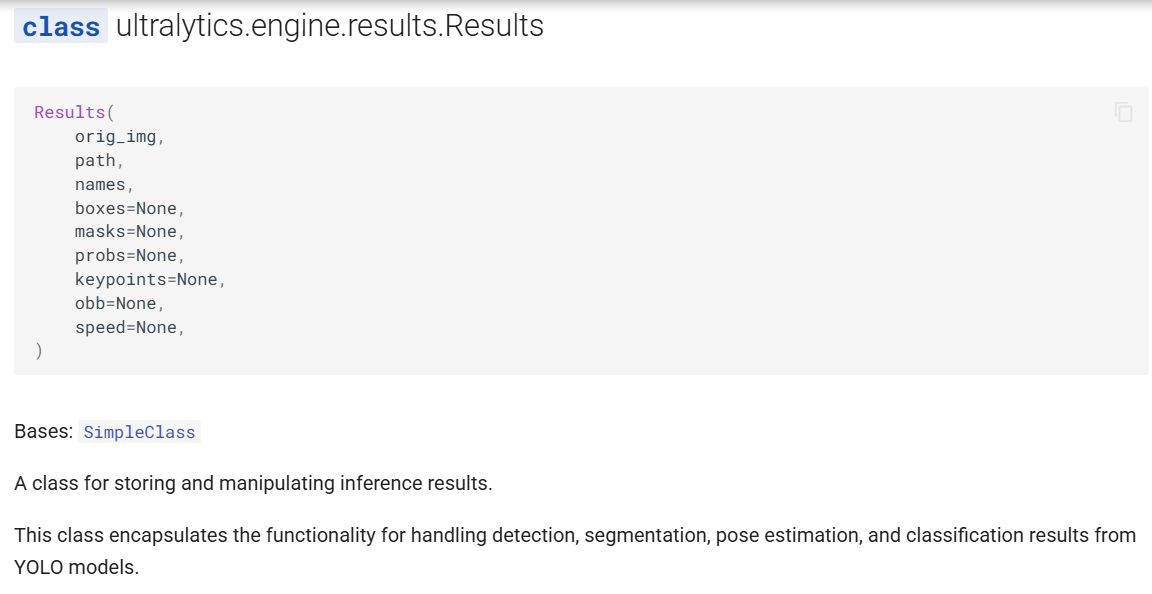
Q: **YOLO**("yolo11n-pose.pt")返回的是什么

A: 是一个ultralytics基于nn.Module开发的一个派生类。nn.Module是Pytorch的一个很重要的类，常用的作用可以问deepseek如下问题，这里不做展开。

“nn.Module 是pytorch的一个累吗？这个类常用的属性和接口是什么。”

Q: results是什么

A: 是一个list，里面的内容通过print(type([results[0]]))结合百度，可以搜索到下图

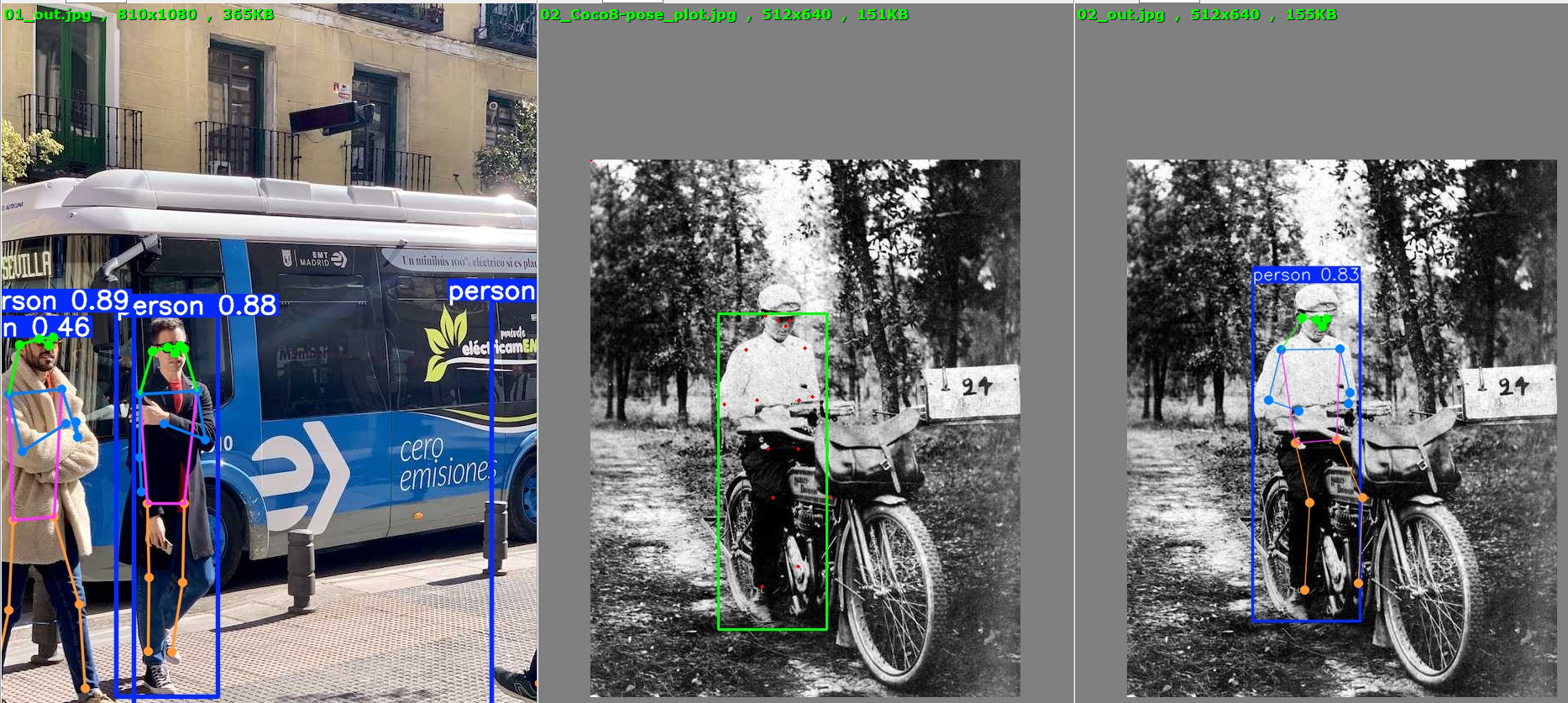


参考网址

https://docs.ultralytics.com/reference/engine/results/#ultralytics.engine.results.Results

### B3.1.3 Predict Coco-8pose

目的：确定B3.1.1的神经网络.pt跑的就是Coco8-pose的数据格式



上图中：

左图是B3.1.1代码的输出。

中图是数据集D:\PxyAI\DataSet\coco8-pose下的plot.py（是我自己写的）的输出

右图是拿B3.1.1代码，输入换成中图的输入，得到的结果。

结论：yolo11n-pose.pt和coco-pose数据集是配套的。

## B3.2 TrainCoco8-pose：结果分析

运行03\_TrainCoco8-pose.py中的iMode=1的代码，分析对应输出目录中的结果。

### B3.2.1 BoxF1\_curve

由于是从，预训练过的weights/yolo11n-pose.pt中，迁移学习来的，所以输出的结果上来就表现的还可以。

#### F1-Score的定义分析

F1-Score是Precision和Recall的（归一化后的）调和平均数。

调和平均数的定义如下：

其中，2是用来归一化的，因为precision(p)和recall(r)的取值范围都是[0,1]，最理想情况下，两者都是1，则此时F1-Score取最高得分=1。

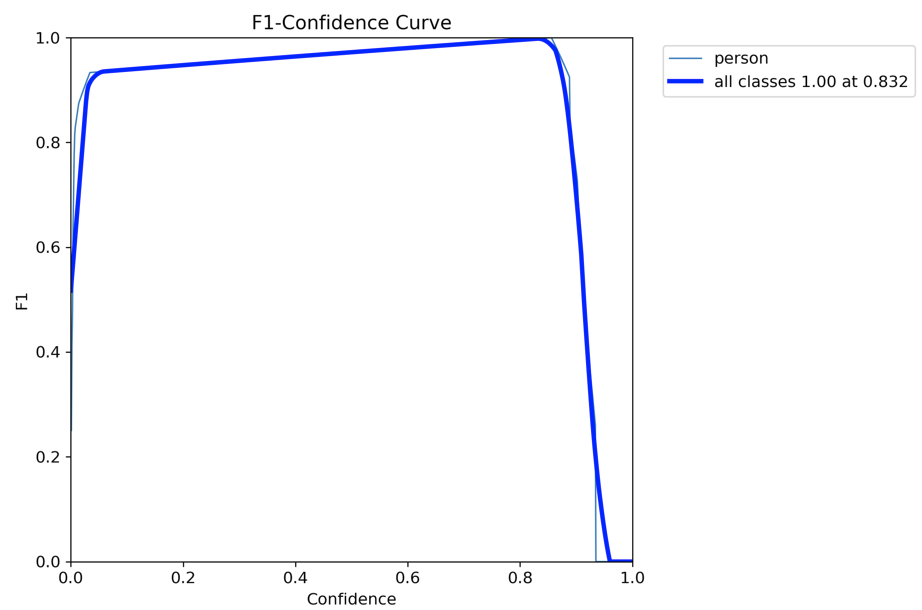
将上式稍作变形，可以更好地理解

可以看到：

相比算数平均数：p和r的任意一个接近0，都会很快的将F1score拉到零，用意是，不能过度忽略p和r中的任意一个。

相比几何平均数：调和平均数在几何平均数的下面除了一个算数平均数，用意是，用较大的值作为分母，将较小的值拉低分数的效果，进一步放大。

#### 则对结果图的分析：



图B3.2.1 BoxF1\_curve

当Confidence阈值设置很小时，输出框密密麻麻，则r必定很高（基本接近1，因为密密麻麻的输出框基本总能覆盖），但p接近0(但不会是0，这是因为IoU=0.5的原因)。所以F1-Confidence曲线从一个0~1之间的值开始。

随着Confidence阈值增大，误检的框渐渐消失（p开始接近1），正确检测的框得以尽可能保留（r保持在接近1不怎么掉），（对于这个网络结构和训练程度，Coco8的人体检测任务比较简单，所以是全部得以保留，则r始终保持在=1）。所以F1-Confidence曲线在0.832处，达到峰值1（对于较难的任务或者不够的训练，这个是到不了1的，详情可参考B3.3节）。

随着Confidence阈值继续增大，所有输出框都被过滤掉，r开始下降直到0，所以F1-Confidence的终点就是0。

#### F1-β Score

就是在F1的基础上，将p和r做一个β加权，表示更侧重p或者r，用意是比如（宁可错杀三千，不可放过一个）这样的极端任务。

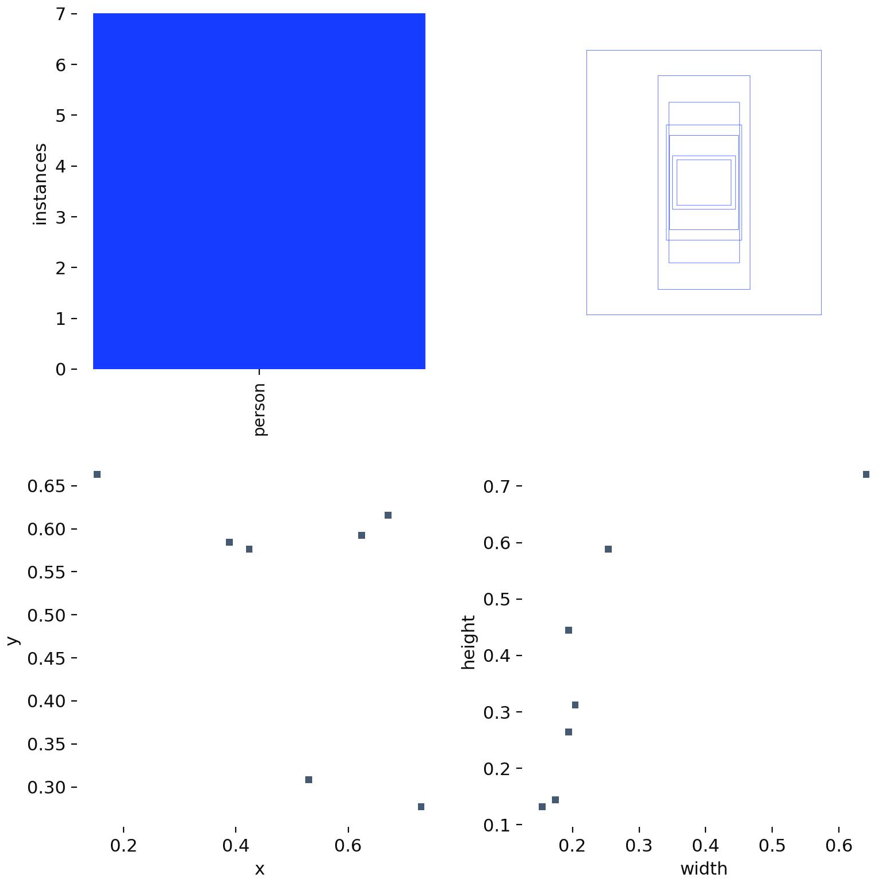
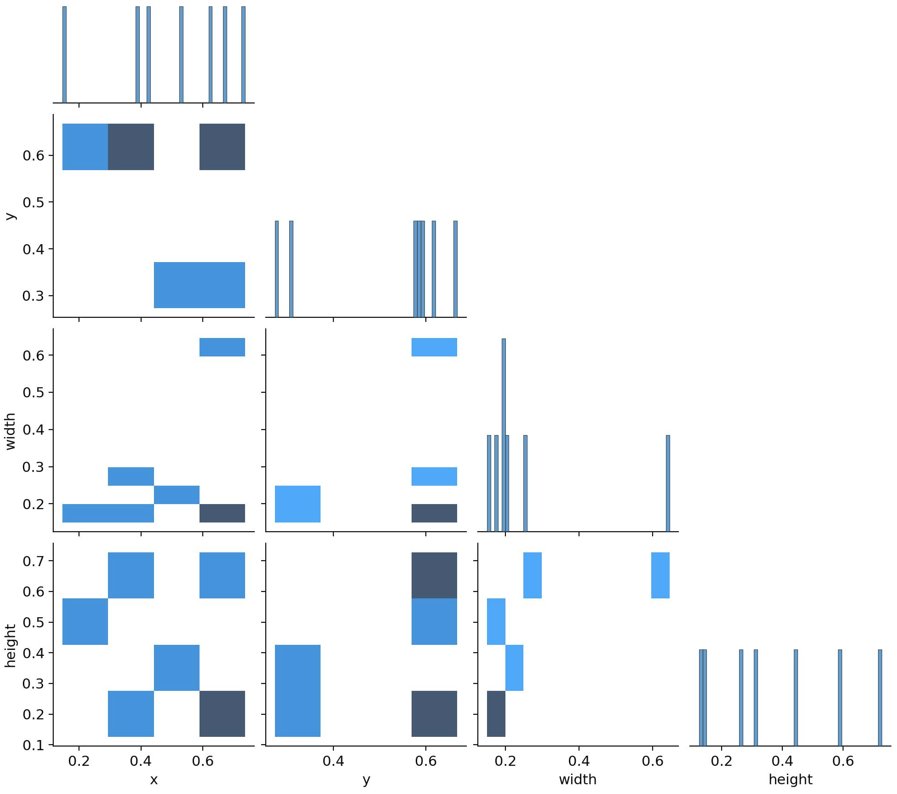
### B3.2.2 BoxPcurve, Rcurve和PRcurve

就是将B3.2.1的F1曲线拆开，各个维度再画了几张图。

### B3.2.3 Confusion matrix (normalized)

详见[结果分析：confusion\_matrix.png 以及 normalized](#_结果分析：confusion_matrix.png_以及_normal)

### B3.2.4 labels 和 labels\_correlogram

图：labels 图：labels\_correlogram

图labels中的x~y图，就对应了图labels\_correlogram中的图x~y；

图labels中的右上角的框框图，就对应了图labels\_ correlogram中的图width~height。

所以，labels\_correlogram其实是补充了图labels中缺失的x,y与width, height的关系。

### B3.2.5 Pose的Precision，Recall和Confidence的关系

Pose，也就是关键点的Precision，目前还没有完全弄清楚。以下列出已有的知识和尚不清楚的问题。

1. 判断某个关键点是否被回收（Recall），应该有一个搜索范围ε，当推理输出的点和label点距离小于ε，则认为被回收。这似乎是唯一的办法，因为推理输出的点的位置，肯定会有偏移。
2. TODO：按照框大小IoU取0.5的做法类比，ε应该有一个默认统一值，但这个值是多少呢？没查到
3. TODO：Confidence是点的置信度吗？还是和b中的ε有关系。

### B3.3 迁移学习微调(pretrain)vs随机初始化权重(scratch)

对于Lapa数据集，当我们载入预训练权重，或者训练权重时

对比同样epoch=200时，pretrained=true 和 false的差别，以及缺省值

B3.2.3 epoch次数

对比epoch=200和

目的：学习训练，包括？？训练和？？训练，观察训练指标的走法

# L1 Win64上VsCode调试C++代码

在学习本章之前，请务必先看看L2的结论，再决定是否需要走下去。

## L1.1 VsCode安装C/C++插件

### L1.1.1 C/C++插件的作用

就像调试Python代码时，需要安装Python 插件一样，那么C/C++插件是用来干嘛的呢？

对deepseek的回答挑重点总结如下：

代码补全与智能提示：提供基于上下文的关键字、函数、类、变量等代码补全。

代码导航：跳转到定义：查找引用：大纲视图：

代码调试：支持使用 GDB 或 LLDB 调试器进行调试。

代码格式化：集成 Clang-Format 等工具，支持自动格式化代码。

### L1.1.2 IntelliSense实现代码补全、智能提示、代码导航

代码补全、智能提示、代码导航是通过IntelliSense实现的（读作Intelligence Sense， 微软开发的，和Intel没关系）

### L1.1.3 GDB或LLVM实现代码调试

代码调试：GDB：Gnu Debugger（适合Linux） 和 LLVM Debugger（适合Windows）

### L1.1.4 Clang-Format实现统一代码风格化

Clang-Format是一个很厉害，用配置文件，统一C++代码风格的插件，被整合在来C/C++插件里

## L1.2、VsCode安装C/C++ Extension Pack

装了再说吧，DeepSeek告诉我是用来服务多个C/C++项目协作用的。

## L1.3、安装GCC和Mingw-w64

什么是GCC

GCC是GNU Compiler Collection，多种语言各自编译器的Collection

什么是Mingw-w64

基于GCC，添加了Windows特有库的工具

什么是MSYS2

MSYS2 是一个Windows上的开发环境，提供了一个类 Unix 的 shell 和工具集，使得开发者可以在 Windows 上使用类 Unix 的开发工具。

什么是pacman

MSYS2 提供了一个包管理器（pacman），可以方便地安装和管理 MinGW-w64 工具链

Cmake和GCC是什么关系？  
CMake 是构建工具，负责生成构建文件。 MinGW 提供编译器，用于编译代码。可以认为是CMake确定了怎么编译，然后用MinGW中的编译器来编译。

MSYS2和Cmake是什么关系？

MSYS2 中的 MINGW64 和 MINGW32 可以各自安装独立的 CMake，但这些 CMake 本质上是一样的，只是配置为使用不同的编译器。

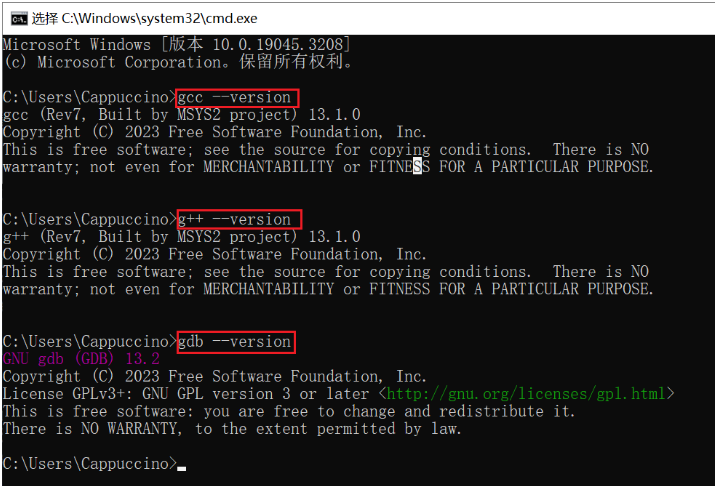
微软自带了一个编译器对吗？

应该说是windows平台的官方编译器，尤其MS Build下的MSVS（cl.exe），是一款C/C++编译器，可以和VisualStudio无缝衔接。

可以直接安装Mingw-w64，也可以通过MSYS2安装Mingw-w64

还有设置Path路径啥的

总之，cmd下能弄出来以下这三个，就行了



## L1.4、三大编译器gcc、Clang、MsBuild的冲突

我的提问：MS Build编译器、Gcc编译器、Clang编译器编译出来的C++库，互相调用时，会出现兼容性问题吗？

DeepSeek的回答：

是的，\*\*MS Build（MSVC）\*\*、\*\*GCC\*\* 和 \*\*Clang\*\* 编译出来的 C++ 库在互相调用时，可能会出现兼容性问题。这些问题主要源于以下几个方面：

---

### 1. \*\*C++ ABI（应用二进制接口）\*\*

- \*\*问题\*\*：

- 不同编译器使用的 C++ ABI 可能不同，导致二进制兼容性问题。

- 例如，MSVC 和 GCC/Clang 的 C++ ABI 不兼容，主要是因为它们对名称修饰（name mangling）、异常处理、RTTI（运行时类型信息）等的实现方式不同。

- \*\*解决方案\*\*：

- 使用 \*\*C 接口\*\*（`extern "C"`）来暴露库的 API，因为 C ABI 是标准化的，兼容性更好。

- 如果必须使用 C++ 接口，确保所有库使用相同的编译器（如全部使用 MSVC 或全部使用 GCC/Clang）。

---

### 2. \*\*标准库实现\*\*

- \*\*问题\*\*：

- MSVC 使用 Microsoft 的 C++ 标准库（MSVCRT），而 GCC 和 Clang 通常使用 libstdc++ 或 libc++。

- 这些标准库的实现方式不同，可能导致链接或运行时错误。

- \*\*解决方案\*\*：

- 确保所有库使用相同的标准库实现。

- 如果必须混合使用，可以通过动态链接库（DLL 或共享库）隔离标准库的依赖。

---

### 3. \*\*名称修饰（Name Mangling）\*\*

- \*\*问题\*\*：

- 不同编译器对 C++ 函数的名称修饰规则不同，导致链接时找不到符号。

- 例如，MSVC 和 GCC/Clang 的名称修饰规则完全不同。

- \*\*解决方案\*\*：

- 使用 `extern "C"` 禁用名称修饰，或者确保所有库使用相同的编译器。

---

### 4. \*\*异常处理\*\*

- \*\*问题\*\*：

- MSVC 和 GCC/Clang 的异常处理机制不同，可能导致跨编译器调用时异常无法正确传递。

- \*\*解决方案\*\*：

- 避免跨编译器传递异常，或者在接口中使用错误码代替异常。

---

### 5. \*\*运行时库\*\*

- \*\*问题\*\*：

- MSVC 和 GCC/Clang 使用不同的运行时库（如 MSVCRT 和 glibc），可能导致链接或运行时冲突。

- \*\*解决方案\*\*：

- 确保所有库使用相同的运行时库，或者通过动态链接库隔离运行时依赖。

---

### 6. \*\*结构体对齐（Alignment）\*\*

- \*\*问题\*\*：

- 不同编译器对结构体的默认对齐方式可能不同，导致内存布局不一致。

- \*\*解决方案\*\*：

- 使用 `#pragma pack` 或 `alignas` 显式指定结构体的对齐方式。

---

### 7. \*\*动态链接库（DLL/共享库）\*\*

- \*\*问题\*\*：

- 不同编译器生成的动态链接库可能不兼容。

- \*\*解决方案\*\*：

- 使用 C 接口（`extern "C"`）导出动态链接库的符号。

- 确保动态链接库和调用方使用相同的运行时库。

---

### 8. \*\*具体场景的解决方案\*\*

#### \*\*场景 1：MSVC 调用 GCC/Clang 编译的库\*\*

- 使用 `extern "C"` 导出库的 API。

- 确保 GCC/Clang 编译时使用与 MSVC 兼容的运行时库（如 `-static-libgcc` 和 `-static-libstdc++`）。

- 使用动态链接库（DLL）隔离运行时依赖。

#### \*\*场景 2：GCC 调用 MSVC 编译的库\*\*

- 使用 `extern "C"` 导出库的 API。

- 确保 MSVC 编译时使用与 GCC 兼容的运行时库（如 `/MT` 或 `/MD`）。

- 使用动态链接库（DLL）隔离运行时依赖。

#### \*\*场景 3：GCC 调用 Clang 编译的库\*\*

- GCC 和 Clang 的 ABI 兼容性较好，但仍需确保使用相同的标准库（如 libstdc++）。

- 如果使用 libc++，需要显式链接并确保运行时一致。

---

### 9. \*\*总结\*\*

- \*\*MSVC、GCC 和 Clang 编译的库在互相调用时，可能会出现兼容性问题\*\*，主要涉及 C++ ABI、标准库、名称修饰、异常处理和运行时库等方面。

- \*\*最佳实践\*\*：

- 使用 `extern "C"` 暴露库的 API。

- 确保所有库使用相同的编译器和运行时库。

- 使用动态链接库隔离运行时依赖。

- 显式指定结构体对齐方式。

如果你必须混合使用不同编译器编译的库，建议通过动态链接库（DLL 或共享库）和 C 接口来降低兼容性风险。

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

我的提问：我发现，gcc和clang编译出的C++库之间也会有冲突，这是为什么？

是的，\*\*GCC\*\* 和 \*\*Clang\*\* 编译的 C++ 库之间也可能出现冲突，尽管它们的兼容性比 MSVC 更好。以下是可能导致冲突的原因以及解决方法：

---

### 1. \*\*C++ ABI（应用二进制接口）\*\*

- \*\*问题\*\*：

- GCC 和 Clang 虽然都遵循 Itanium C++ ABI（在 Linux 和 macOS 上），但在某些细节上可能仍有差异。

- 例如，名称修饰（name mangling）、异常处理、RTTI（运行时类型信息）等的实现可能略有不同。

- \*\*解决方案\*\*：

- 确保 GCC 和 Clang 使用相同的标准库（如 libstdc++）。

- 如果使用 Clang 的 libc++，需要显式链接并确保运行时一致。

---

### 2. \*\*标准库实现\*\*

- \*\*问题\*\*：

- GCC 默认使用 libstdc++，而 Clang 默认使用 libc++（在 macOS 上）或 libstdc++（在 Linux 上）。

- 如果 GCC 和 Clang 使用不同的标准库实现，可能导致链接或运行时错误。

- \*\*解决方案\*\*：

- 确保 GCC 和 Clang 使用相同的标准库（如 libstdc++）。

- 如果必须使用 libc++，需要显式链接并确保运行时一致。

---

### 3. \*\*名称修饰（Name Mangling）\*\*

- \*\*问题\*\*：

- GCC 和 Clang 的名称修饰规则虽然基本一致，但在某些情况下（如模板实例化）可能略有不同。

- \*\*解决方案\*\*：

- 使用 `extern "C"` 禁用名称修饰，或者确保所有库使用相同的编译器。

---

### 4. \*\*异常处理\*\*

- \*\*问题\*\*：

- GCC 和 Clang 的异常处理机制虽然基本一致，但在某些情况下（如跨模块传递异常）可能存在问题。

- \*\*解决方案\*\*：

- 避免跨模块传递异常，或者在接口中使用错误码代替异常。

---

### 5. \*\*运行时库\*\*

- \*\*问题\*\*：

- GCC 和 Clang 使用不同的运行时库（如 libgcc 和 libunwind），可能导致链接或运行时冲突。

- \*\*解决方案\*\*：

- 确保 GCC 和 Clang 使用相同的运行时库（如 `-static-libgcc` 和 `-static-libstdc++`）。

---

### 6. \*\*编译器版本\*\*

- \*\*问题\*\*：

- 不同版本的 GCC 和 Clang 可能对 C++ 标准的实现有所不同，导致兼容性问题。

- \*\*解决方案\*\*：

- 确保 GCC 和 Clang 使用相同的 C++ 标准（如 `-std=c++17`）。

- 尽量使用相同版本的编译器。

---

### 7. \*\*具体场景的解决方案\*\*

#### \*\*场景 1：GCC 调用 Clang 编译的库\*\*

- 确保 Clang 使用 libstdc++ 而不是 libc++：

```bash

clang++ -stdlib=libstdc++ -o mylib.so -shared mylib.cpp

```

- 使用 `extern "C"` 导出库的 API。

- 确保 GCC 和 Clang 使用相同的运行时库（如 `-static-libgcc` 和 `-static-libstdc++`）。

#### \*\*场景 2：Clang 调用 GCC 编译的库\*\*

- 确保 GCC 编译时使用与 Clang 兼容的标准库（如 libstdc++）。

- 使用 `extern "C"` 导出库的 API。

- 确保 GCC 和 Clang 使用相同的运行时库。

---

### 8. \*\*总结\*\*

- \*\*GCC 和 Clang 编译的 C++ 库之间也可能出现冲突\*\*，主要涉及 C++ ABI、标准库、名称修饰、异常处理和运行时库等方面。

- \*\*最佳实践\*\*：

- 使用 `extern "C"` 暴露库的 API。

- 确保 GCC 和 Clang 使用相同的标准库（如 libstdc++）。

- 确保 GCC 和 Clang 使用相同的运行时库。

- 避免跨模块传递异常。

如果你必须混合使用 GCC 和 Clang 编译的库，建议通过动态链接库（共享库）和 C 接口来降低兼容性风险。

## L1.5、VsCode编译使用C++的重点

在**.vscode**文件下，

**tasks.json**是编译配置文件

"-I", "D:/users/xiaoyaopan/sdai\_face\_demo\_v10", // include路径

"-L", "D:/users/xiaoyaopan/ /opencv/out/win64/", // 寻找库路径

"-l", "opencv\_world451.lib", // link lib using windows way

"-lopencv\_world451", // link lib using unix way / or Gcc in windows for .a

**launch.json**是调试配置文件，所以配合着tasks.json设置：  
 "preLaunchTask" // 需要先根据tasks.json编译

"program" // 需要运行编译出的文件  
 “miDebuggerPath” // 需要和tasks.json设置的编译器配套

**c\_cpp\_properties.json**是给intelliSense用的，配合IDE让你看代码的，对编译没用。（很多网上的教程说，这里设置东西可以影响tasks.json的编译，比如includelu路径，但我始终没成功过）

# L2 Win64上VsCode调试C++AI推理代码

存在问题如下：

NCNN喜欢用MSbuild编译、OpenCV喜欢用Gcc编译、Tensorflow喜欢用Clang编译（或者说windows下就不好编译），于是由于L1.4的原因，很难兼容，单价太大。

结论：还是在Linux/Ubuntu上编，或者用VS2022吧。