

华中科技大学

课程实验报告

课程名称: AI 派第二轮面试

专业班级 计卓 2201 班

学 号 U202212313

姓 名 赖永烨

指导教师

报告日期 2023 年 8 月 21 日

计算机科学与技术学院

目 录

1	尝试 YOLO 工程实战复现中的困难与成果	2
1.1	YOLOv5	2
1.2	YOLOv8 v6 v7	3
1.3	YOLOv4、v3	5
2	YOLO 论文学习笔记	6
2.1	YOLO 原论文	6
2.2	YOLOv3v4	6
2.3	YOLOv5v8	7
2.4	YOLOv6v7	7
3	YOLO 库的制作	8

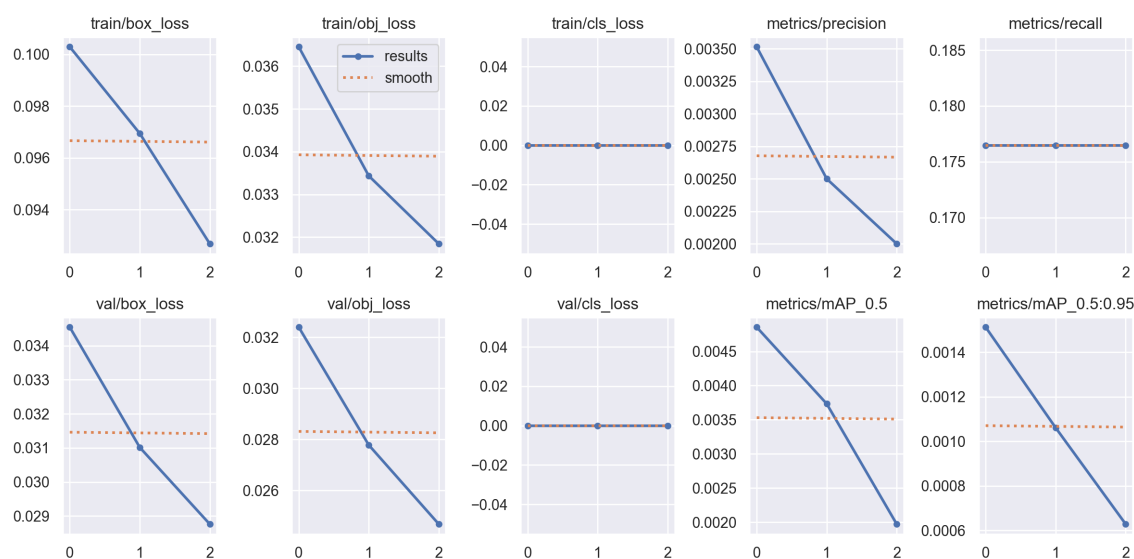
1 尝试 YOLO 工程实战复现中的困难与成果

我确实没有太理解题目的意思，于是我拿到这个题目的第一想法就是：把各个版本的 YOLO 下载下来，然后尝试去学习如何使用 YOLO。在以下的所有叙述中我运用的都是在 <https://universe.roboflow.com/tennistracker-dogbm/tennis-court-detection/dataset/9> 网站上寻找到的数据集资源。不使用 COCO 数据集的原因是这个数据集实在是太大了，我的设备大概率无法支撑，而我选用的数据集是一个仅有一百多张图片的简单数据集。

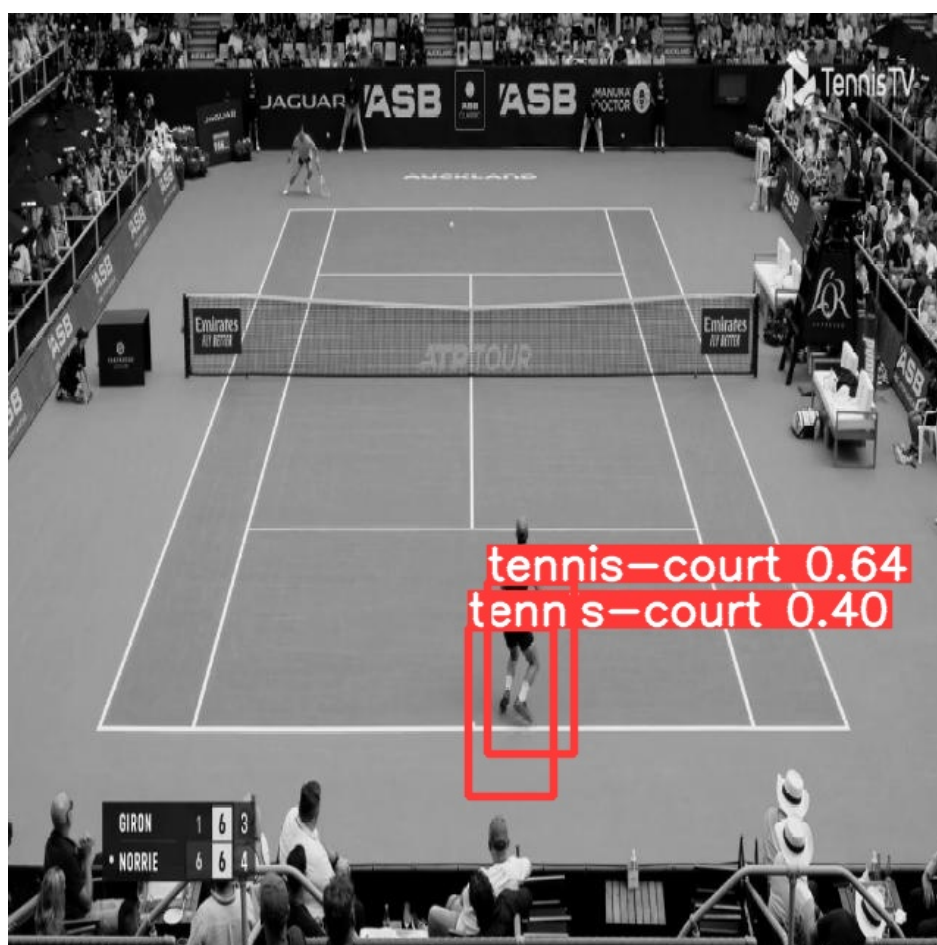
这一章节我会侧重介绍一下我学习过程的经历和困难。总体来说我由于硬件原因没能很好的完成学习任务，但是大体掌握了 YOLO 的使用方法。

1.1 YOLOv5

YOLOv5 的进展较为顺利。根据 ultralytics 提供的官方教程，在花费了一定的时间配置环境之后，我很快便成功跑出了第一份数据集。输出的 results 如下：



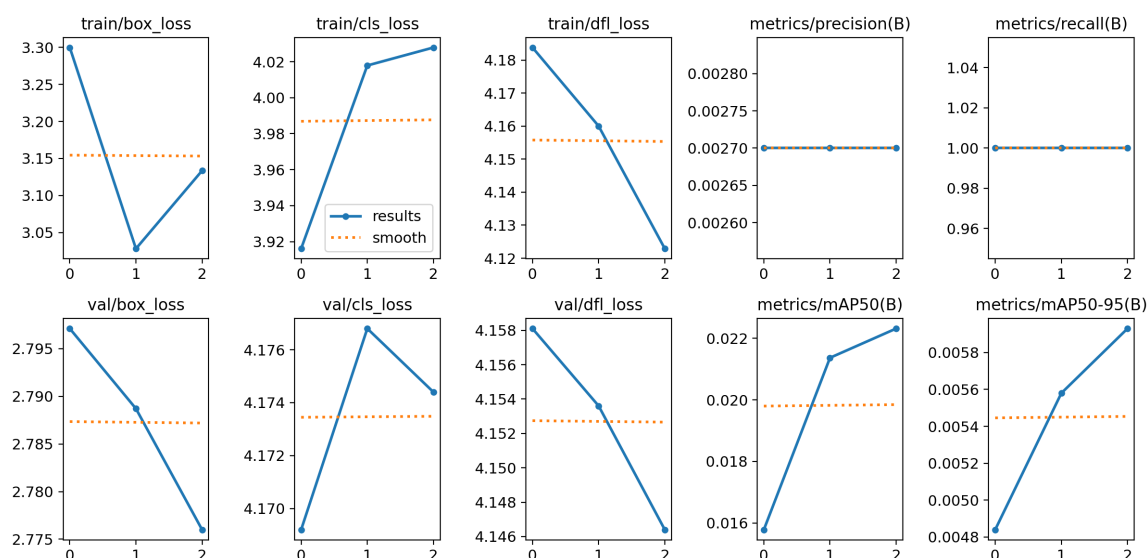
这份图片我不太理解它的意思，主要是不了解这些数据是偏大、偏小还是正好。不过从跑出来的结果来看还是不太可以的。



以上用的是 YOLOv5n 模型跑出来的结果，因为由于网络带宽等原因，我无法运行更庞大的模型。

1.2 YOLOv8 v6 v7

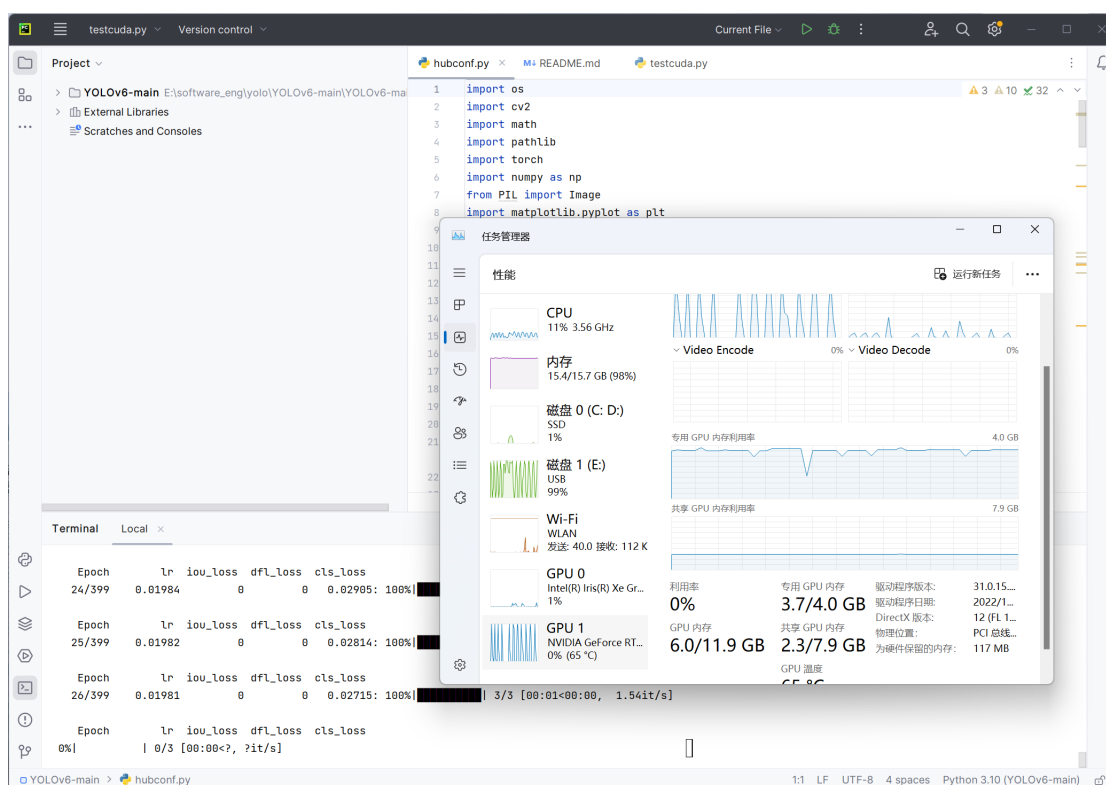
在运行 YOLOv8 的时候我觉得还没遇到什么困难，只不过 YOLOv8 需要我自己写一个脚本文件来运行，相对花费了更多的时间。比起 YOLOv5，YOLOv8 封装的更像一个库，我在库里面调用 YOLO 来制作模型与预测图片。但是我发现我跑一个 epoch 需要大约 15s，而默认的是需要跑 100 个 epoch 左右。我认为让我的电脑跑 30 分钟的 YOLO 不是一个明智的选择。因为我的电脑此时基本上是满负荷运转，很难说会发生什么不测。于是我减少了 epoch 重新训练，但是结果非常差劲，基本无法产生 boxes。下一页的图是此次运行的 result，感觉上 loss 变大了不少。



接下来的内容主要讲述我遇到的困难，我的结果可以直接查看下一页

这次经历让我开始尝试使用 GPU 加速。我先去了解了 CUDA 与 cudnn，并且下载、安装。第一次安装了 12.2 的 CUDA，安装报错无法正常安装。在这里我卡了许久才知道要查看自己电脑的显卡情况来决定 CUDA 的版本。尔后安装了 12.0 适配版本的 CUDA，但是依旧报错，在这里我按照大部分教程的方法取消勾选了相关内容仍无济于事，我于是自己删掉了一些项目，最后奇迹般的成功下载并且在 Windows 的 cmd 上面成功运行了。但是由于我使用的是 pycharm 环境，我还需要在 pycharm 上面配置 CUDA，而且在 GPT 和网上各平台给的一些教程中，不同版本的 pycharm 情况不同，在我找到适合我的版本的 pycharm 教程之后，按照教程做了一定调试之后依旧没有反应。此时我又去寻找帮助，发现 pycharm 下载的 torch 是 CPU 的不是 cu 的，我于是又重新下载了 2.0.1+cu118 的 torch 库，可是依旧无法正常运行。最后由于各种原因我的电脑直接蓝屏死机，我也搁置了此事，决定放弃使用 GPU 加速。

但是，YOLOv6、v7 都需要 GPU，官网的示例代码必须要求 GPU，甚至 v6 在代码里内置了对 cuda 的 assert，这回是不搞出来不行了。然而很神奇的是，我在没有做任何操作的情况下，仅仅只是蓝屏死机了之后重启了电脑，CUDA 就配置完成了。于是我采用 GPU 加速试图配置 YOLOv6、v7，但是我发现一次训练依旧需要 20 分钟左右。如果我放任它们训练，不到十分钟我的内存就会爆炸导致无法继续进行。



由于硬件原因，我放弃了对 v6v7v8 的复现尝试，但是至少我学会了该算法的使用。

1.3 YOLOv4、v3

由于前面对 GPU 加速的尝试花费的大量时间（一周左右），我于是没有对 v4 进行比较深入的研究。毕竟 v4 的 windows 环境配置较为困难与复杂，我放弃了对 v4 的复现。

2 YOLO 论文学习笔记

2.1 YOLO 原论文

YOLO 的原论文展示了 YOLO 的基本优势和基础算法，但是好像没有展示细致的算法细节。总的来说，YOLO 就是把一张图片拆分成若干的小块 ($7*7$)，然后我猜测是根据像素点的分布来找到这一块图片和对应的标签的联系，从而快速地构造模型与产出预测。YOLO 的损失函数也有一些特点。一个是运用了两个不同的系数来放大坐标误差、减小类别误差，我虽然不理解为什么要这么做，但是猜测是这样子在实际训练中炼丹效果更好。另一个是巧妙运用了平方根函数，来解决在相同的偏移量下，大目标的偏移和小目标的偏移导致的结果的差别很大的问题，真是个巧妙的做法。不过我感觉初代的 YOLO 算法 $7*7$ 的分割、每个分割预测一类这样简单粗暴的算法有点贪心，感觉很容易使预测失去准确性，比如比较小的物体或者重叠的物体。

2.2 YOLOv3v4

YOLOv3 的一大亮点是采用了交叉熵的损失函数计算方法，更加准确一些。不过我认为最重要的还是多标签分类。这使得 YOLO 算法更加贴近现实实际，应用性也高了很多。不过我在思考这会不会让 YOLO 的运行速度降低，但是根据论文给的数据，在采用了一些新型架构之后，YOLO 的速度没有降低，但是准确率提高了。

关于 YOLOv4，我找到这样一句话：“YOLOv4 = CSPDarknet53 (主干) + SPP 附加模块 (颈) + PANet 路径聚合 (颈) + YOLOv3 (头部)。”这几个模型我不是很了解，但是我认为大概意思就是保留了部分 YOLOv3 的运算，同时进行了大刀阔斧的优化。然后作者花了整整两页介绍他的“Bag of freebies”和“Bag of specials”，这些大多是一些巧妙运用的技巧，用来提升精度。然后 YOLOv4 做了很多事让 YOLO 能在 GPU 上更轻松的跑。我读论文的时候觉得这也没啥，但是去查找了相关资料后发现了他实现了从要运用工业级 GPU 跑到普通 GPU 就能跑的跨越，那确实是一个很大的进步，我也大概知道了我的 GPU 跑不动 YOLO 的原因了。

2.3 YOLOv5v8

这两个模型根本就没有论文，我于是从网上搜罗了点资料。

YOLOv5 的一个让我印象比较深的特色是自适应灰度填充，一开始我没有理解为什么这个可以增加精度，后来意识到一个物体横着和竖着会有很大的差别。其它的优化我没有太了解，但是他确实大大降低了我配置环境的难度。而且还分出来了好几个模型，可以说实用性大幅上升了。

YOLOv8 总体来说也是做了一系列的优化，并且把算法整合到了一个库里头去。

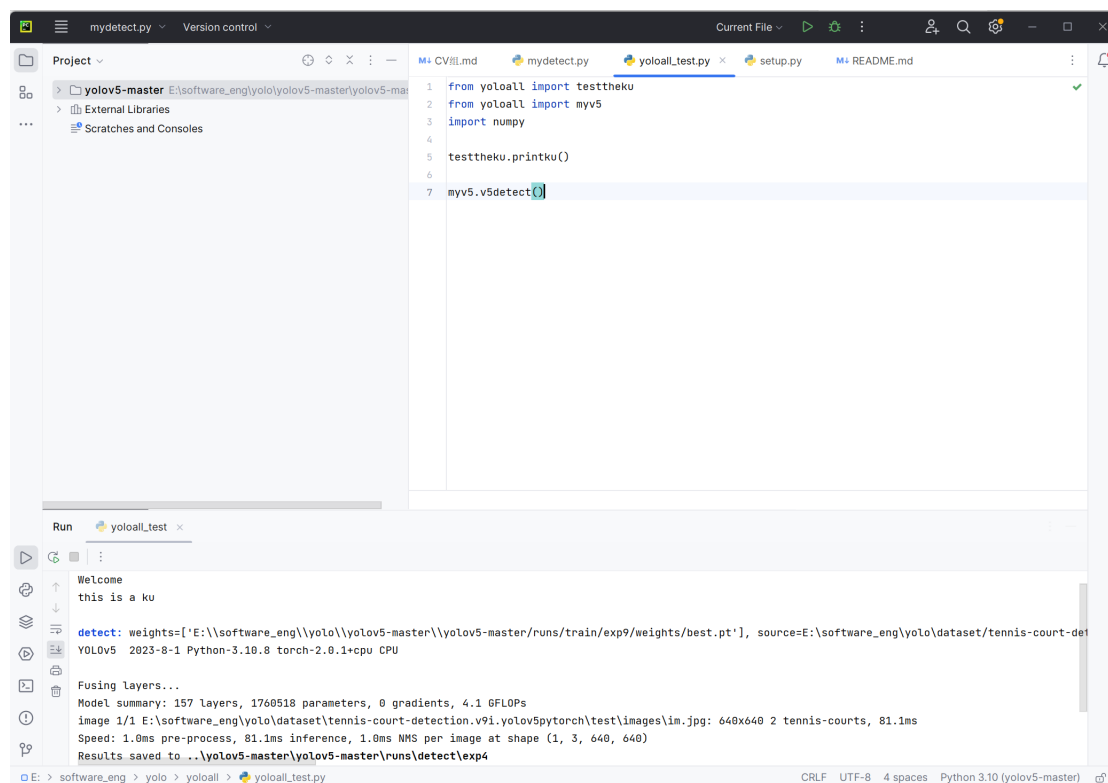
2.4 YOLOv6v7

YOLOv6 居然是美团开发的，有点惊讶。YOLOv6 感觉上是一个偏向于工业化的引擎，也难怪家用显卡不是很带得动了。YOLOv6，同样的，在网络结构方面做了一系列的优化，也采用了新的标签分配策略和损失函数。YOLOv6 还对工业化需求做了贴身改进（很符合美团公司的性质），增加了训练轮次（侧面让我本就不富裕的显卡资源雪上加霜），还提到了一个“Gray border”的概念，这我去了解了一下，大致就是加一个边框方便检测边缘物体。

YOLOv7 也是改进了一些算法结构，还特别提到了一个“trainable bag-of-freebies”来优化性能。

3 YOLO 库的制作

我的 YOLO 库可以调用 YOLOv5 v6 v7 的 train 和 detect 代码，并且可以自定义执行路径。下面是一个库的运行示例：



```
1 from yoloall import testtheke
2 from yoloall import myv5
3 import numpy
4
5 testtheke.printku()
6
7 myv5.v5detect()
```

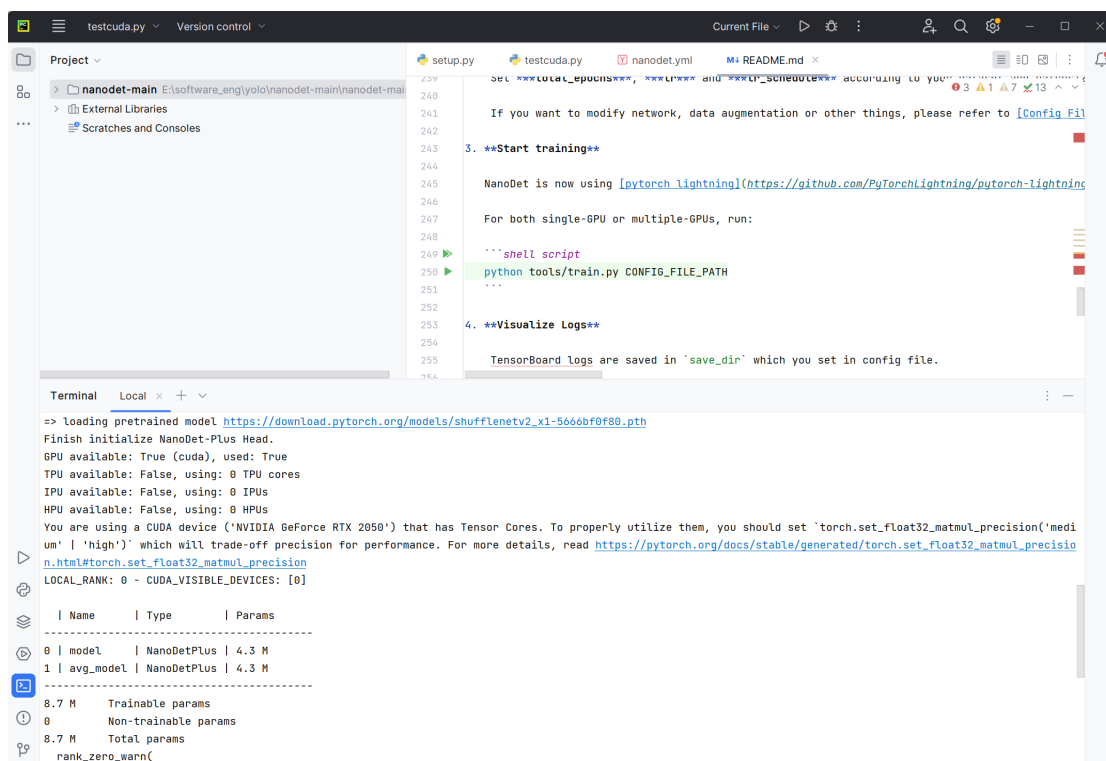
```
Welcome
this is a ku

detect: weights=['E:\\software_eng\\yolo\\yolov5-master\\yolov5-master\\runs\\train\\exp9\\weights\\best.pt'], source=E:\\software_eng\\yolo\\dataset\\tennis-court-detection.v91.yolov5pytorch\\test\\images\\im.jpg: 640x640 2 tennis-courts, 81.1ms
YOLOv5 2023-8-1 Python-3.10.8 torch-2.0.1+cpu CPU

Fusing layers...
Model summary: 157 layers, 1760518 parameters, 0 gradients, 4.1 GFLOPs
image 1/1 E:\\software_eng\\yolo\\dataset\\tennis-court-detection.v91.yolov5pytorch\\test\\images\\im.jpg: 640x640 2 tennis-courts, 81.1ms
Speed: 1.0ms pre-process, 81.1ms inference, 1.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to ..\\yolov5-master\\yolov5-master\\runs\\detect\\exp4
```

额外的目标算法我尝试了 nanodet 算法，不过在 Windows 上面运行好像会出一点问题。模型能跑是能跑，但是可视化遇到了一些麻烦。nanodet 的优势是它跑的真的很快。

华中科技大学课程实验报告



```
testcuda.py
Version control
Current File
testcuda.py
nanodet.yml
README.md
Project
nanodet-main
External Libraries
Scratches and Consoles
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
set **total_epochs**, **lr** and **lr_scheduler** according to your
If you want to modify network, data augmentation or other things, please refer to [Config File]
3. **Start training**
NanoDet is now using [pytorch lightning](https://github.com/PyTorchLightning/pytorch-lightning)
For both single-GPU or multiple-GPUs, run:
'''shell script
python tools/train.py CONFIG_FILE_PATH
'''
4. **Visualize Logs**
TensorBoard logs are saved in 'save_dir' which you set in config file.
Terminal
Local
>> loading pretrained model https://download.pytorch.org/models/shufflenetv2_x1-5666bf0f80.pth
Finish initialize NanoDet-Plus Head.
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
You are using a CUDA device ('NVIDIA GeForce RTX 2050') that has Tensor Cores. To properly utilize them, you should set 'torch.set_float32_matmul_precision('medium' | 'high')' which will trade-off precision for performance. For more details, read https://pytorch.org/docs/stable/generated/torch.set_float32_matmul_precision.html#torch.set_float32_matmul_precision
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
| Name | Type | Params
-----|-----|-----
0 | model | NanoDetPlus | 4.3 M
1 | avg_model | NanoDetPlus | 4.3 M
-----|-----|-----
8.7 M Trainable params
0 Non-trainable params
8.7 M Total params
rank_zero_warn(
```