1. yolov5数据集的训练笔记

[原文]

[视频]

项目的克隆和安装依赖

1. 项目克隆[链接]

├── data：主要是存放一些超参数的配置文件（这些文件（yaml文件）是用来配置训练集和测试集还有验证集的路径的，其中还包括目标检测的种类数和种类的名称）；还有一些官方提供测试的图片。如果是训练自己的数据集的话，那么就需要修改其中的yaml文件。但是自己的数据集不建议放在这个路径下面，而是建议把数据集放到yolov5项目的同级目录下面。

├── models：里面主要是一些网络构建的配置文件和函数，其中包含了该项目的四个不同的版本，分别为是s、m、l、x。从名字就可以看出，这几个版本的大小。他们的检测测度分别都是从快到慢，但是精确度分别是从低到高。这就是所谓的鱼和熊掌不可兼得。如果训练自己的数据集的话，就需要修改这里面相对应的yaml文件来训练自己模型。

├── utils：存放的是工具类的函数，里面有loss函数，metrics函数，plots函数等等。

├── weights：放置训练好的权重参数。

├── detect.py：利用训练好的权重参数进行目标检测，可以进行图像、视频和摄像头的检测。

├── train.py：训练自己的数据集的函数。

├── test.py：测试训练的结果的函数。

├── requirements.txt：这是一个文本文件，里面写着使用yolov5项目的环境依赖包的一些版本，可以利用该文本导入相应版本的包。

2. 安装依赖

```
pip install -r requirements.txt -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

3.制作数据集标签

1. labelimg介绍

Labelimg是一款开源的数据标注工具，可以标注三种格式。

1 VOC标签格式，保存为xml文件。

2 yolo标签格式，保存为txt文件。

3 createML标签格式，保存为json格式。

2.安装Labelimg

```
pip install labelimg -i https://pypi.tuna.tsinghua.edu.cn/simple
```

[链接] [Labelimg步骤]

4. VOC标签格式转yolo格式并数据集和验证集划分

改两个地方
1. classes=["face"]
2. TRAIN_RATIO = 80
特别要注意的是，classes里面必须正确填写xml里面已经标注好的类，要不然生成的txt的文件是不对的。
TRAIN_RATIO是训练集和验证集的比例，当等于80的时候，说明划分80%给训练集，20%给验证集。

```
import xml.etree.ElementTree as ET
import pickle
import os
from os import listdir, getcwd
from os.path import join
import random
from shutil import copyfile

classes = ["hat", "person"]
#classes=["ball"]

TRAIN_RATIO = 80

def clear_hidden_files(path):
    dir_list = os.listdir(path)
    for i in dir_list:
        abspath = os.path.join(os.path.abspath(path), i)
        if os.path.isfile(abspath):
            if i.startswith("._"):
                os.remove(abspath)
        else:
            clear_hidden_files(abspath)

def convert(size, box):
```

```
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0
    y = (box[2] + box[3])/2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)

def convert_annotation(image_id):
    in_file = open('VOCdevkit/VOC2007/Annotations/%s.xml' %image_id)
    out_file = open('VOCdevkit/VOC2007/YOLOLabels/%s.txt' %image_id, 'w')
    tree=ET.parse(in_file)
    root = tree.getroot()
    size = root.find('size')
    w = int(size.find('width').text)
    h = int(size.find('height').text)

    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult) == 1:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (float(xmlbox.find('xmin').text),
float(xmlbox.find('xmax').text), float(xmlbox.find('ymin').text),
float(xmlbox.find('ymax').text))
        bb = convert((w,h), b)
        out_file.write(str(cls_id) + " " + " ".join([str(a) for a in bb]) +
'\n')
    in_file.close()
    out_file.close()

wd = os.getcwd()
wd = os.getcwd()
data_base_dir = os.path.join(wd, "VOCdevkit/")
if not os.path.isdir(data_base_dir):
    os.mkdir(data_base_dir)
work_sapce_dir = os.path.join(data_base_dir, "VOC2007/")
```

```
if not os.path.isdir(work_sapce_dir):
    os.mkdir(work_sapce_dir)
annotation_dir = os.path.join(work_sapce_dir, "Annotations/")
if not os.path.isdir(annotation_dir):
        os.mkdir(annotation_dir)
clear_hidden_files(annotation_dir)
image_dir = os.path.join(work_sapce_dir, "JPEGImages/")
if not os.path.isdir(image_dir):
        os.mkdir(image_dir)
clear_hidden_files(image_dir)
yolo_labels_dir = os.path.join(work_sapce_dir, "YOLOLabels/")
if not os.path.isdir(yolo_labels_dir):
        os.mkdir(yolo_labels_dir)
clear_hidden_files(yolo_labels_dir)
yolov5_images_dir = os.path.join(data_base_dir, "images/")
if not os.path.isdir(yolov5_images_dir):
        os.mkdir(yolov5_images_dir)
clear_hidden_files(yolov5_images_dir)
yolov5_labels_dir = os.path.join(data_base_dir, "labels/")
if not os.path.isdir(yolov5_labels_dir):
        os.mkdir(yolov5_labels_dir)
clear_hidden_files(yolov5_labels_dir)
yolov5_images_train_dir = os.path.join(yolov5_images_dir, "train/")
if not os.path.isdir(yolov5_images_train_dir):
        os.mkdir(yolov5_images_train_dir)
clear_hidden_files(yolov5_images_train_dir)
yolov5_images_test_dir = os.path.join(yolov5_images_dir, "val/")
if not os.path.isdir(yolov5_images_test_dir):
        os.mkdir(yolov5_images_test_dir)
clear_hidden_files(yolov5_images_test_dir)
yolov5_labels_train_dir = os.path.join(yolov5_labels_dir, "train/")
if not os.path.isdir(yolov5_labels_train_dir):
        os.mkdir(yolov5_labels_train_dir)
clear_hidden_files(yolov5_labels_train_dir)
yolov5_labels_test_dir = os.path.join(yolov5_labels_dir, "val/")
if not os.path.isdir(yolov5_labels_test_dir):
        os.mkdir(yolov5_labels_test_dir)
clear_hidden_files(yolov5_labels_test_dir)

train_file = open(os.path.join(wd, "yolov5_train.txt"), 'w')
test_file = open(os.path.join(wd, "yolov5_val.txt"), 'w')
train_file.close()
test_file.close()
```

```
train_file = open(os.path.join(wd, "yolov5_train.txt"), 'a')
test_file = open(os.path.join(wd, "yolov5_val.txt"), 'a')
list_imgs = os.listdir(image_dir) # list image files
prob = random.randint(1, 100)
print("Probability: %d" % prob)
for i in range(0,len(list_imgs)):
    path = os.path.join(image_dir,list_imgs[i])
    if os.path.isfile(path):
        image_path = image_dir + list_imgs[i]
        voc_path = list_imgs[i]
        (nameWithoutExtention, extention) =
os.path.splitext(os.path.basename(image_path))
        (voc_nameWithoutExtention, voc_extention) =
os.path.splitext(os.path.basename(voc_path))
        annotation_name = nameWithoutExtention + '.xml'
        annotation_path = os.path.join(annotation_dir, annotation_name)
        label_name = nameWithoutExtention + '.txt'
        label_path = os.path.join(yolo_labels_dir, label_name)
    prob = random.randint(1, 100)
    print("Probability: %d" % prob)
    if(prob < TRAIN_RATIO): # train dataset
        if os.path.exists(annotation_path):
            train_file.write(image_path + '\n')
            convert_annotation(nameWithoutExtention) # convert label
            copyfile(image_path, yolov5_images_train_dir + voc_path)
            copyfile(label_path, yolov5_labels_train_dir + label_name)
    else: # test dataset
        if os.path.exists(annotation_path):
            test_file.write(image_path + '\n')
            convert_annotation(nameWithoutExtention) # convert label
            copyfile(image_path, yolov5_images_test_dir + voc_path)
            copyfile(label_path, yolov5_labels_test_dir + label_name)
train_file.close()
test_file.close()
```

文件格式

1. 新建VOCdevkit文件夹

```
|——VOCdevkit
    |——voc2yolo.py
```

2. 运行voc2yolo.py，产生空文件夹

```
|——VOCdevkit
        |——VOCdevkit
                |——images
                        |——train
                        |——val
                |——labels
                        |——train
                        |——val
                |——VOC2007
                        |——Annotation
                        |——JEPGImage
        |——voc2yolo.py
        |——yolov5_train.txt
        |——yolov5_val.txt
```

3. 将打好VOC标签的Annotation和JEPGImage文件复制到VOCdevkit/VOC2007/文件下
4. 运行voc2yolo.py产生

```
|——VOCdevkit
```

5.获得预训练权重

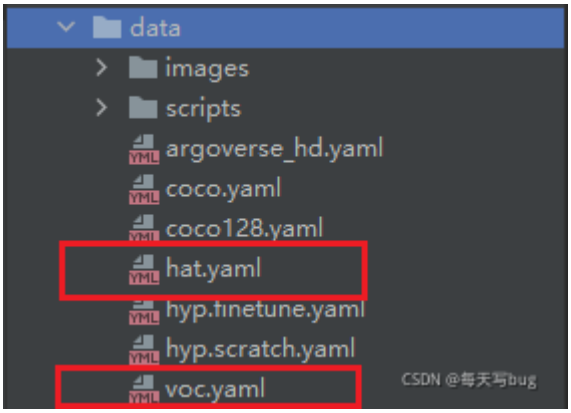
一般为了缩短网络的训练时间，并达到更好的精度，我们一般加载预训练权重进行网络的训练。而yolov5的5.0版本给我们提供了几个预训练权重，我们可以对应我们不同的需求选择不同的版本的预训练权重。通过如下的图可以获得权重的名字和大小信息，可以预料的到，预训练权重越大，训练出来的精度就会相对来说越高，但是其检测的速度就会越慢。预训练权重可以通过这个网址进行下载，本次训练自己的数据集用的预训练权重为yolov5s.pt。

6.训练自己的模型

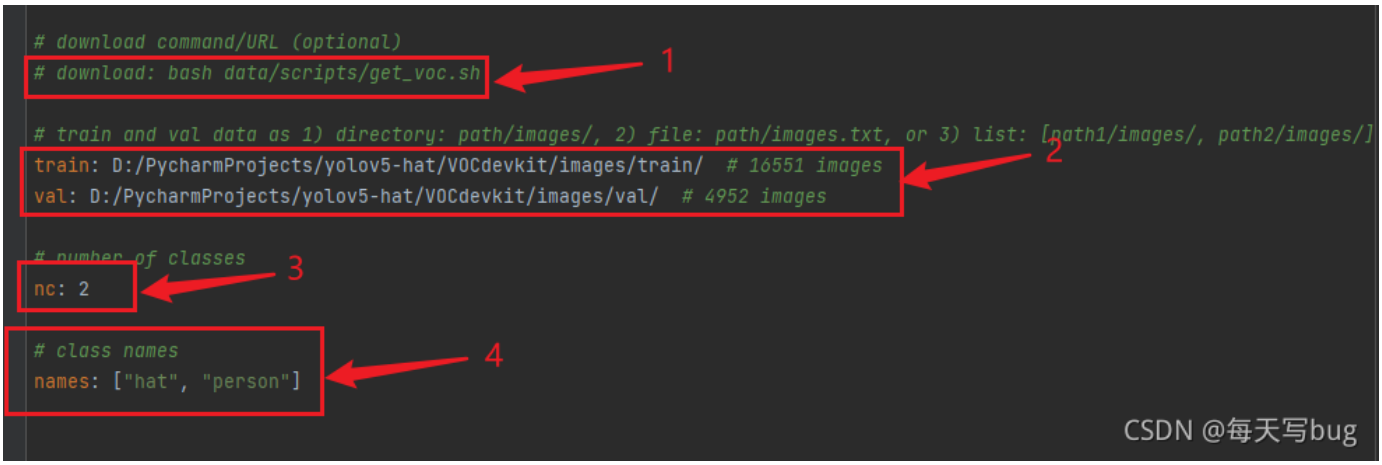
1修改数据配置文件

预训练模型和数据集都准备好了，就可以开始训练自己的yolov5目标检测模型了，训练目标检测模型需要修改两个yaml文件中的参数。一个是data目录下的相应的yaml文件，一个是model目录文件下的相应的yaml文件。

修改data目录下的相应的yaml文件。找到目录下的voc.yaml文件，将该文件复制一份，将复制的文件重命名，最好和项目相关，这样方便后面操作。我这里修改为hat.yaml。该项目是对安全帽的识别。

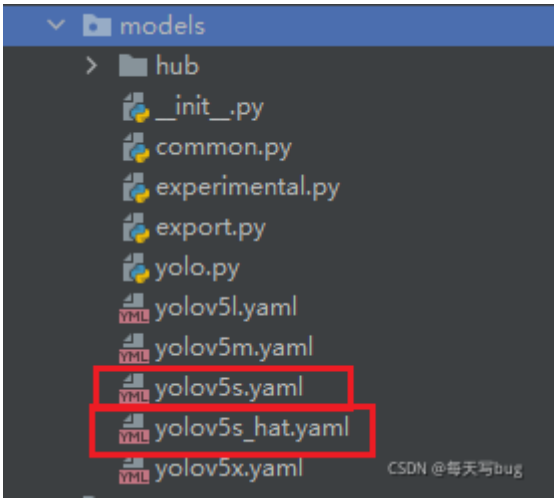


打开这个文件夹修改其中的参数，首先将箭头1中的那一行代码注释掉（我已经注释掉了），如果不注释这行代码训练的时候会报错；箭头2中需要将训练和测试的数据集的路径填上（最好要填绝对路径，有时候由目录结构的问题会莫名奇妙的报错）；箭头3中需要检测的类别数，我这里是识别安全帽和人，所以这里填写2；最后箭头4中填写需要识别的类别的名字（必须是英文，否则会乱码识别不出来）。到这里和data目录下的yaml文件就修改好了。

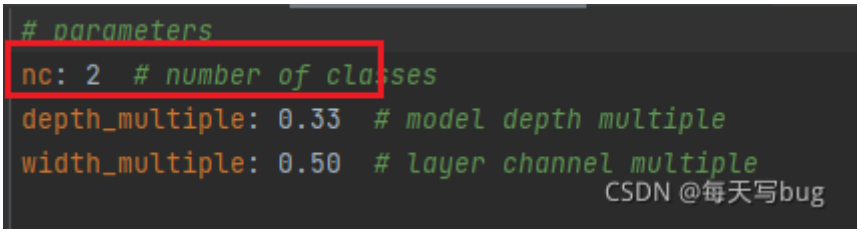


修改模型配置文件

由于该项目使用的是yolov5s.pt这个预训练权重，所以要使用models目录下的yolov5s.yaml文件中的相应参数（因为不同的预训练权重对应着不同的网络层数，所以用错预训练权重会报错）。同上修改data目录下的yaml文件一样，我们最好将yolov5s.yaml文件复制一份，然后将其重命名，我将其重命名为yolov5_hat.yaml。



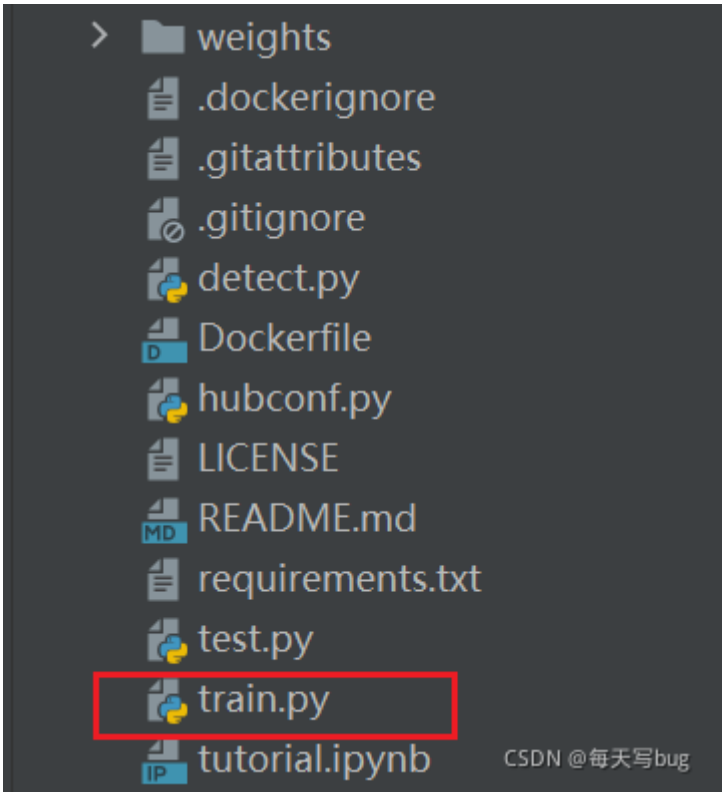
打开yolov5_hat.yaml文件只需要修改如图中的数字就好了，这里是识别两个类别。



至此，相应的配置参数就修改好了。

3训练自己的模型启用tensorbord查看参数

如果上面的数据集和两个yaml文件的参数都修改好了的话，就可以开始yolov5的训练了。首先我们找到train.py这个py文件。



然后找到主函数的入口，这里面有模型的主要参数。模型的主要参数解析如下所示。

```
if __name__ == '__main__':
"""
    opt模型主要参数解析:
    --weights: 初始化的权重文件的路径地址
    --cfg: 模型yaml文件的路径地址
    --data: 数据yaml文件的路径地址
    --hyp: 超参数文件路径地址
    --epochs: 训练轮次
    --batch-size: 喂入批次文件的多少
    --img-size: 输入图片尺寸
    --rect:是否采用矩形训练，默认False
    --resume:接着打断训练上次的结果接着训练
```

```
    --nosave:不保存模型，默认False
    --notest:不进行test，默认False
    --noautoanchor:不自动调整anchor，默认False
    --evolve:是否进行超参数进化，默认False
    --bucket:谷歌云盘bucket，一般不会用到
    --cache-images:是否提前缓存图片到内存，以加快训练速度，默认False
    --image-weights: 使用加权图像选择进行训练
    --device:训练的设备，cpu；0(表示一个gpu设备cuda:0)；0,1,2,3(多个gpu设备)
    --multi-scale:是否进行多尺度训练，默认False
    --single-cls:数据集是否只有一个类别，默认False
    --adam:是否使用adam优化器
    --sync-bn:是否使用跨卡同步BN,在DDP模式使用
    --local_rank: DDP参数，请勿修改
    --workers: 最大工作核心数
    --project:训练模型的保存位置
    --name: 模型保存的目录名称
    --exist-ok: 模型目录是否存在，不存在就创建
"""
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolov5s.pt',
help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml
path')
    parser.add_argument('--data', type=str, default='data/coco128.yaml',
help='data.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyp.scratch.yaml',
help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-size', type=int, default=16, help='total
batch size for all GPUs')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640,
640], help='[train, test] image sizes')
    parser.add_argument('--rect', action='store_true', help='rectangular
training')
    parser.add_argument('--resume', nargs='?', const=True, default=False,
help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save
final checkpoint')
    parser.add_argument('--notest', action='store_true', help='only test
final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable
autoanchor check')
    parser.add_argument('--evolve', action='store_true', help='evolve
```

```
hyperparameters')
    parser.add_argument('--bucket', type=str, default='', help='gsutil
bucket')
    parser.add_argument('--cache-images', action='store_true', help='cache
images for faster training')
    parser.add_argument('--image-weights', action='store_true', help='use
weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary
img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train
multi-class data as single-class')
    parser.add_argument('--adam', action='store_true', help='use
torch.optim.Adam() optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use
SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP
parameter, do not modify')
    parser.add_argument('--workers', type=int, default=8, help='maximum
number of dataloader workers')
    parser.add_argument('--project', default='runs/train', help='save to
project/name')
    parser.add_argument('--entity', default=None, help='W&B entity')
    parser.add_argument('--name', default='exp', help='save to
project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad
dataloader')
    parser.add_argument('--linear-lr', action='store_true', help='linear
LR')
    parser.add_argument('--label-smoothing', type=float, default=0.0,
help='Label smoothing epsilon')
    parser.add_argument('--upload_dataset', action='store_true',
help='Upload dataset as W&B artifact table')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='Set
bounding-box image logging interval for W&B')
    parser.add_argument('--save_period', type=int, default=-1, help='Log
model after every "save_period" epoch')
    parser.add_argument('--artifact_alias', type=str, default="latest",
help='version of dataset artifact to be used')
    opt = parser.parse_args()
```

训练自己的模型需要修改如下几个参数就可以训练了。首先将weights权重的路径填写到对应的参数里面，然后将修好好的models模型的yolov5s.yaml文件路径填写到相应的参数里面，最后将data数据的hat.yaml文件路径填写到相对于的参数里面。这几个参数就必须要修改的参数。

```
    parser.add_argument('--weights', type=str, default='weights/yolov5s.pt',
help='initial weights path')
    parser.add_argument('--cfg', type=str,
default='models/yolov5s_hat.yaml', help='model.yaml path')
    parser.add_argument('--data', type=str, default='data/hat.yaml',
help='data.yaml path')
```

还有几个需要根据自己的需求来更改的参数：

首先是模型的训练轮次，这里是训练的300轮。

```
parser.add_argument('--epochs', type=int, default=300)
```

其次是输入图片的数量和工作的核心数，这里每个人的电脑都不一样，所以这里每个人和自己的电脑的性能来。这里可以根据我的电脑的配置做参考，我的电脑是拯救者R9000,3060版本的显卡，cpu的核心数是8核。我的电脑按默认的参数输入图片数量为16，工作核心为8的话就会出现GPU显存溢出的报错。报错信息如下：

```
RuntimeError: CUDA out of memory. Tried to allocate 26.00 MiB (GPU 0; 6.00 GiB total capacity; 4.26 GiB already allocated;
```


这里就要调小这两个参数了，每个人的电脑配置不一样，所以可以根据自己的电脑配置来修改参数。

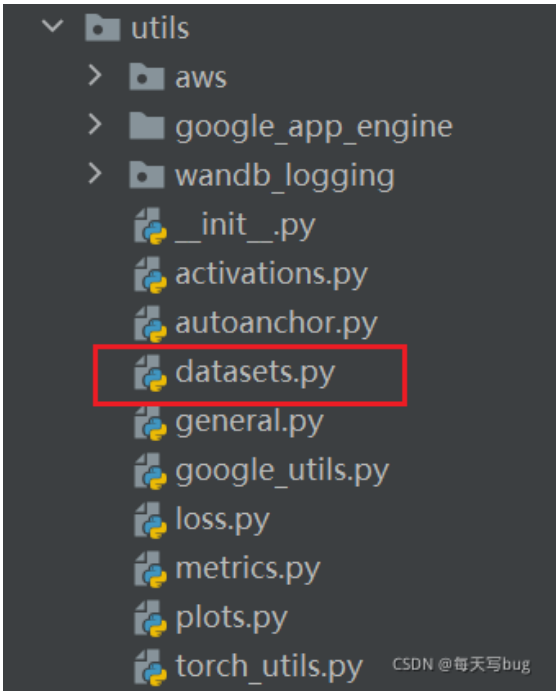
```
parser.add_argument('--batch-size', type=int, default=8, help='total batch
size for all GPUs')
```

```
parser.add_argument('--workers', type=int, default=8, help='maximum number
of dataloader workers')
```

以上都设置好了就可以训练了。但是pycharm的用户可能会出现如下的报错。这是说明虚拟内存不够了。

```
OSError: [WinError 1455] 页面文件太小，无法完成操作。 Error loading "D:\code\Anaconda3-2021.05-Windows-x86_64\er
```


可以根据如下的操作来修改，在utils路径下找到datasets.py这个文件，将里面的第81行里面的参数nw改完0就可以了。




```
79      dataloader = loader(dataset,
80                          batch_size=batch_size,
81                          num_workers=0,
82                          sampler=sampler,
83                          pin_memory=True,
84                          collate_fn=LoadImagesAndLabels.c
85      return dataloader, dataset
```


至此，就可以运行train.py函数训练自己的模型了。

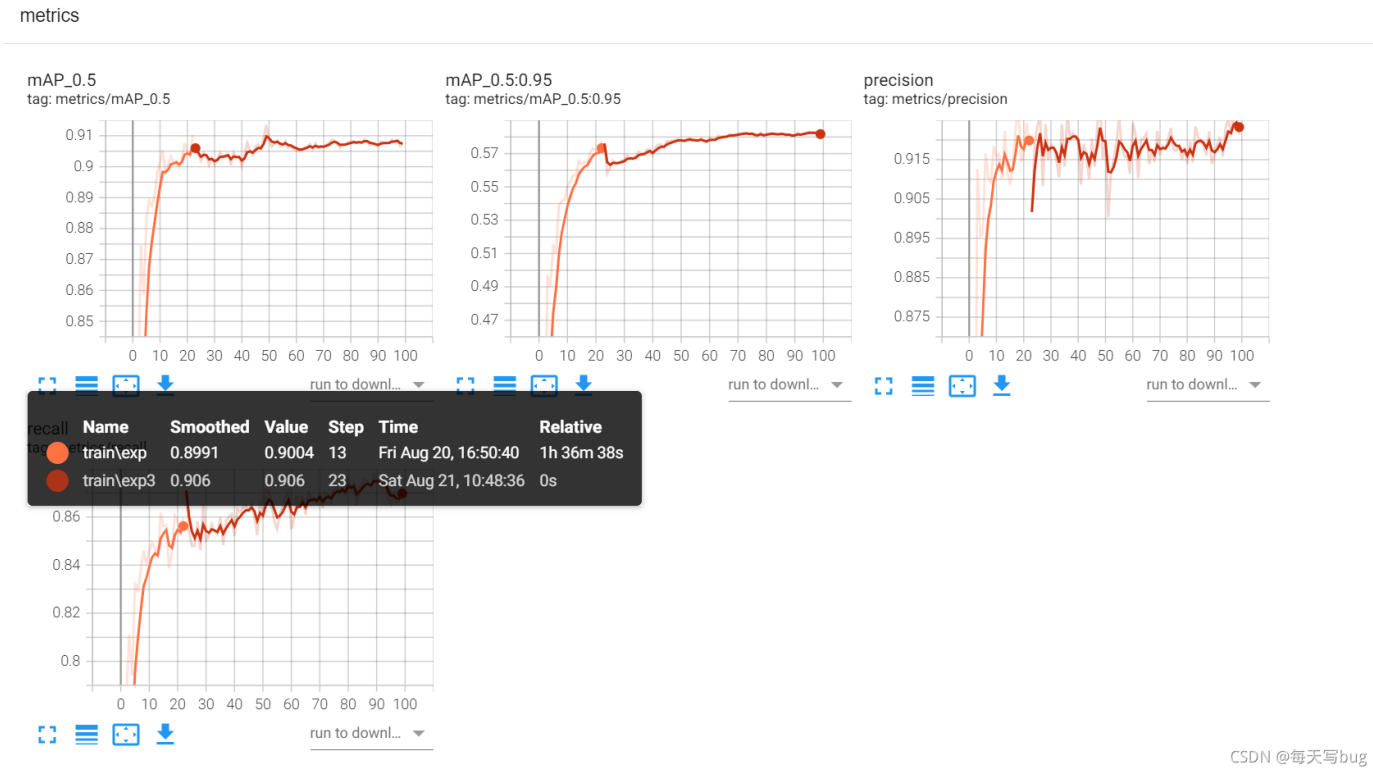
4. 启用tensorbord查看参数

yolov5里面有写好的tensorbord函数，可以运行命令就可以调用tensorbord，然后查看tensorbord了。首先打开pycharm的命令控制终端，输入如下命令，就会出现一个网址地址，将那行网址复制下来到浏览器打开就可以看到训练的过程了

```
tensorboard --logdir=runs/train
```

```
PS D:\PycharmProjects\yolov5.5_example\yolov5-hat> tensorboard --logdir=runs/train
TensorFlow installation not found - running with reduced feature set.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.6.0 at http://localhost:6006/ (Press CTRL+C to quit)
```


如下图所示，这是已经训练了100轮了。

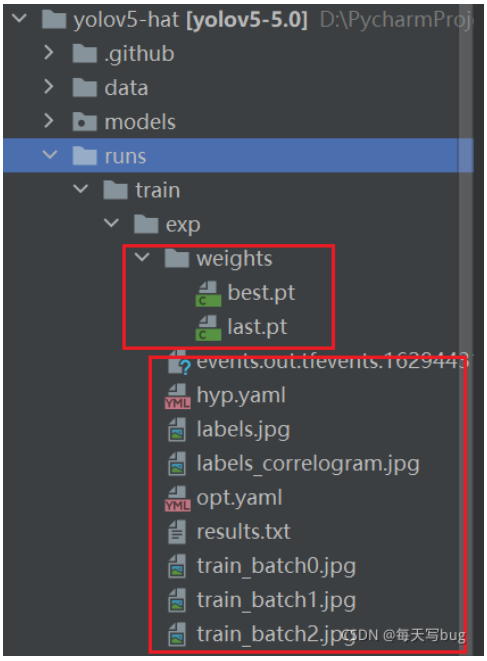


如果模型已经训练好了，但是我们还想用tensorbord查看此模型的训练过程，就需要输入如下的命令。就可以看到模型的训练结果了。

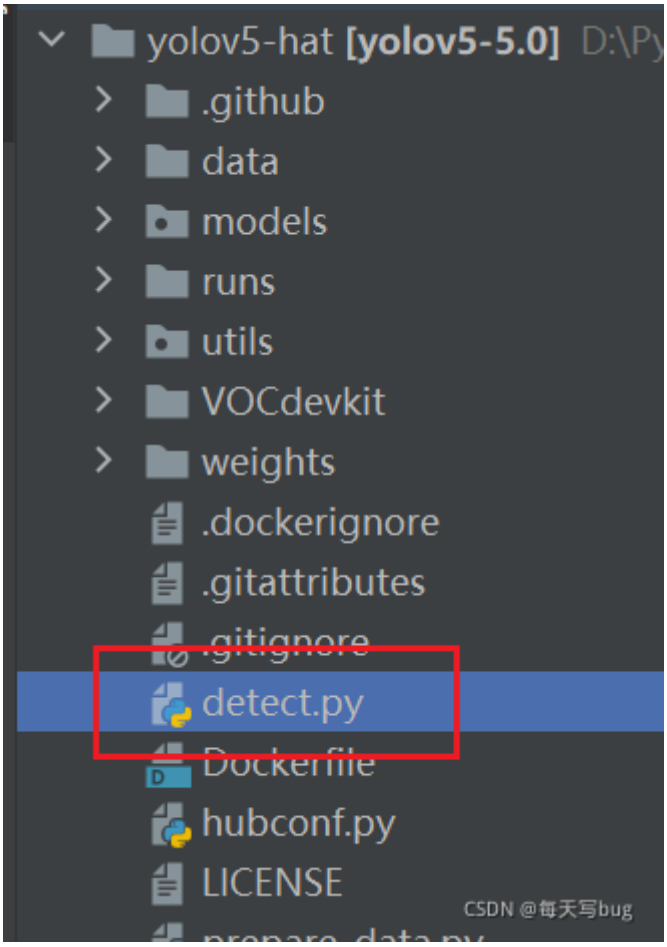
```
tensorboard --logdir=runs
```

4 推理测试

等到数据训练好了以后，就会在主目录下产生一个run文件夹，在run/train/exp/weights目录下会产生两个权重文件，一个是最后一轮的权重文件，一个是最好的权重文件，一会我们就要利用这个最好的权重文件来做推理测试。除此以外还会产生一些验证文件的图片等一些文件。



找到主目录下的detect.py文件，打开该文件。



然后找到主函数的入口，这里面有模型的主要参数。模型的主要参数解析如下所示。

```
f __name__ == '__main__':
"""
--weights:权重的路径地址
--source:测试数据，可以是图片/视频路径，也可以是'0'(电脑自带摄像头),也可以是rtsp等视频流
--output:网络预测之后的图片/视频的保存路径
--img-size:网络输入图片大小
--conf-thres:置信度阈值
--iou-thres:做nms的iou阈值
--device:是用GPU还是CPU做推理
--view-img:是否展示预测之后的图片/视频，默认False
--save-txt:是否将预测的框坐标以txt文件形式保存，默认False
--classes:设置只保留某一部分类别，形如0或者0 2 3
--agnostic-nms:进行nms是否也去除不同类别之间的框，默认False
--augment:推理的时候进行多尺度，翻转等操作(TTA)推理
--update:如果为True，则对所有模型进行strip_optimizer操作，去除pt文件中的优化器等信息，默认为False
--project: 推理的结果保存在runs/detect目录下
--name: 结果保存的文件夹名称
"""
    parser = argparse.ArgumentParser()
```

```
    parser.add_argument('--weights', nargs='+', type=str,
default='yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='data/images',
help='source')  # file/folder, 0 for webcam
    parser.add_argument('--img-size', type=int, default=640, help='inference
size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.25,
help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='IOU
threshold for NMS')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or
0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='display
results')
    parser.add_argument('--save-txt', action='store_true', help='save
results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save
confidences in --save-txt labels')
    parser.add_argument('--nosave', action='store_true', help='do not save
images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by
class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-
agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented
inference')
    parser.add_argument('--update', action='store_true', help='update all
models')
    parser.add_argument('--project', default='runs/detect', help='save
results to project/name')
    parser.add_argument('--name', default='exp', help='save results to
project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing
project/name ok, do not increment')
    opt = parser.parse_args()
```

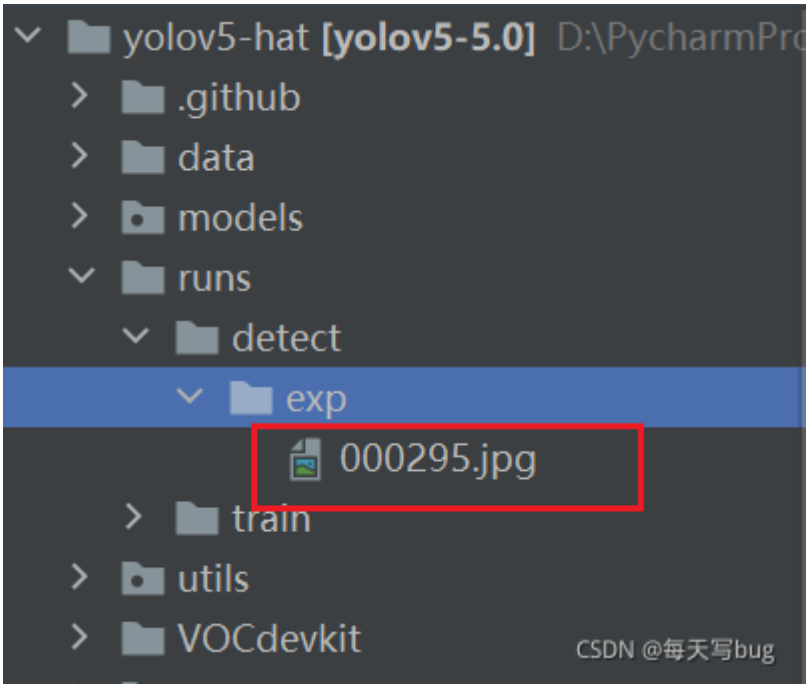
这里需要将刚刚训练好的最好的权重传入到推理函数中去。然后就可以对图像视频进行推理了。

```
parser.add_argument('--weights', nargs='+', type=str,
default='runs/train/exp/weights/best.pt', help='model.pt path(s)')
```

对图片进行测试推理，将如下参数修改成图片的路径，然后运行detect.py就可以进行测试了。

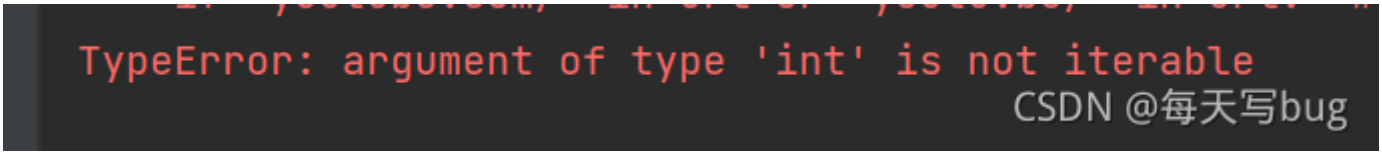
```
 parser.add_argument('--source', type=str, default='000295.jpg',
help='source')
```

推理测试结束以后，在run下面会生成一个detect目录，推理结果会保存在exp目录下。如图所示。

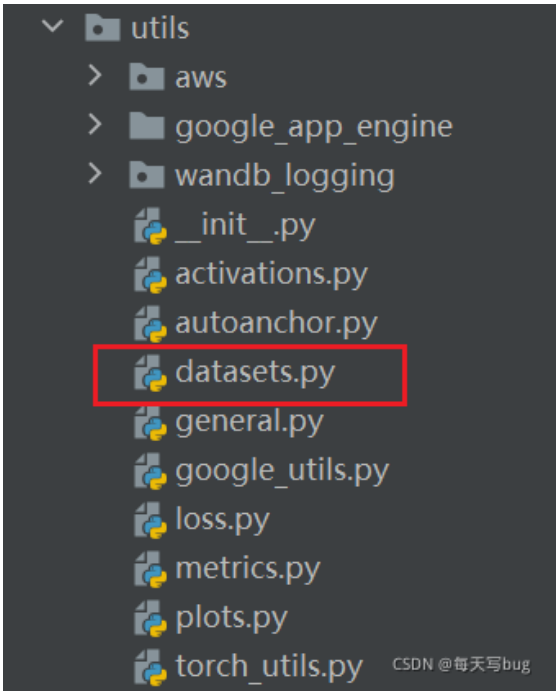


图片的推理结果如下所示。效果还是很不错的。

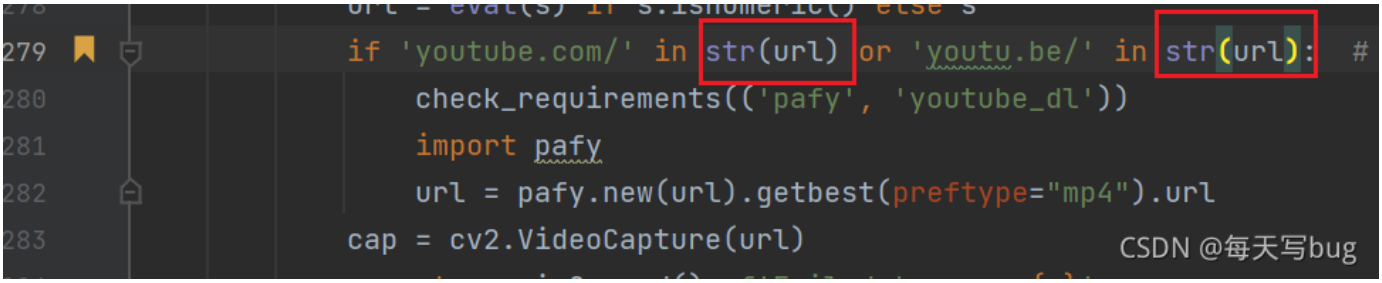
对视频进行测试，和如上的图片的测试是一样的，只不过是将图片的路径改为视频的路径而已。利用摄像头进行测试只需将路径改写为0就好了。但是好像还是会报错，这一点卡了我很久。报错如下。



解决方法：首先找到datasets.py这个py文件。



打开文件，找到第279行代码，给两个url参数加上str就可以了，如图所示，就可以完美运行电脑的摄像头了。



至此yolov5训练自己的模型就完全搞定了。