

# EC-python (elective course)

## 代码总览

```
# YOLOv5 by Ultralytics, GPL-3.0 license

"""
Run inference on images, videos, directories, streams, etc.
Usage:
    $ python path/to/detect.py --weights yolov5s.pt --source 0

The commands which can be use:
    --source 0                # webcam
    img.jpg                   # image
    vid.mp4                   # video
    path/                     # directory
    path/*.jpg                # glob
    https://youtu.be/Zgi9g1ksQHc' # YouTube
    'rtsp://example.com/media.mp4' # RTSP, RTMP, HTTP stream
"""

'''===== 一、导入包 ====='''

'''===== 1.导入安装好的 python 库 ====='''

import argparse             # 解析命令行参数的库
import os                   # 与操作系统进行交互的文件库 包含文件路径操作与解析
import sys                  # sys模块包含了与python解释器和它的环境有关的函数。
from pathlib import Path     # Path能够更加方便得对字符串路径进行处理

import cv2                  # sys模块包含了与python解释器和它的环境有关的函数。
import torch                # pytorch 深度学习库
import torch.backends.cudnn as cudnn
                             #让内置的cudnn的 auto-tuner 自动寻找最适合当前配置的高效算法

'''===== 2.获取当前文件的绝对路径 ====='''
FILE = Path(__file__).resolve() # __file__指的是当前文件(即
detect.py)                      #FILE最终保存着当前文件的绝对路
                                #径,比如D://yolov5/detect.py

ROOT = FILE.parents[0]          # YOLOv5 root directory ROOT
                                #保存着当前项目的父目录,比如 D://yolov5

if str(ROOT) not in sys.path:   # sys.path即当前python环境可以
    运行的路径                  #假如当前项目不在该路径中,就无法
                                #运行其中的模块,所以需要加载路径

    sys.path.append(str(ROOT))   # add ROOT to PATH 把ROOT添加
    到运行路径上
```

```

ROOT = Path(os.path.relpath(ROOT, Path.cwd())) # relative ROOT设置为相对路径

'''===== 3.加载自定义模块 ====='''
from models.common import DetectMultiBackend
from utils.datasets import IMG_FORMATS, VID_FORMATS, LoadImages, LoadStreams
from utils.general import (LOGGER, check_file, check_img_size, check_imshow,
                           check_requirements, colorstr,
                           increment_path, non_max_suppression, print_args,
                           scale_coords, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, time_sync

'''===== 二、run 函数—传入参数 ====='''

'''===== 1.载入参数 ====='''
@torch.no_grad() # 该标注使得方法中所有计算得出的tensor的requires_grad都自动设置为
False，也就是说不进行梯度的计算(当然也就没办法反向传播了)， 节约显存和算
def run(weights=ROOT / 'yolov5s.pt', # model.pt path(s) 事先训练完成的权重文
件，比如yolov5s.pt,默认 weights/，假如使用官方训练好的文件（比如yolov5s），则会自动
下载
        source=ROOT / 'data/images', # file/dir/URL/glob, 0 for webcam 预测时
的输入数据，可以是文件/路径/URL/glob，输入是0的话调用摄像头作为输入，默认
data/images/
        # data=ROOT / 'data/coco128.yaml', # dataset.yaml path, data文件路径，
包括类别/图片/标签等信息
        imgsz=(640, 640), # inference size (pixels) 预测时的放缩后图片大小(因
为YOLO算法需要预先放缩图片)，两个值分别是height, width。默认640*640
        conf_thres=0.25, # confidence threshold 置信度阈值，高于此值的
bounding_box才会被保留。默认0.25，用在nms中
        iou_thres=0.45, # NMS IOU threshold IOU阈值,高于此值的bounding_box才会
被保留。默认0.45，用在nms中
        max_det=1000, # maximum detections per image 一张图片上检测的最大目标数
量，用在nms中
        device='', # cuda device, i.e. 0 or 0,1,2,3 or cpu 所使用的GPU编号，如
果使用CPU就写cpu
        view_img=False, # show results 是否展示预测之后的图片或视频，默认False
        save_txt=False, # save results to *.txt 是否将预测的框坐标以txt文件形式
保存，默认False，使用--save-txt 在路径runs/detect/exp*/labels/*.txt下生成每张图片
预测的txt文件
        save_conf=False, # save confidences in --save-txt labels 是否将结果中
的置信度保存在txt文件中，默认False
        save_crop=False, # save cropped prediction boxes 是否保存裁剪后的预测
框，默认为False，使用--save-crop 在runs/detect/exp*/crop/剪切类别文件夹/ 路径下会
保存每个接下来的目标
        nosave=False, # do not save images/videos 不保存图片、视频，要保存图片，
不设置--nosave 在runs/detect/exp*/会出现预测的结果
        classes=None, # filter by class: --class 0, or --class 0 2 3 过滤指定
类的预测结果
        agnostic_nms=False, # class-agnostic NMS 进行NMS去除不同类别之间的框，
默认False
        augment=False, # augmented inference TTA测试时增强/多尺度预测，可以提分
        visualize=False, # visualize features 是否可视化网络层输出特征
        update=False, # update all models 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=ROOT / 'runs/detect', # save results to project/name 预测结果
保存的路径

```

```

    name='exp', # save results to project/name 结果保存文件夹的命名前缀
    exist_ok=False, # existing project/name ok, do not increment True: 推
理结果覆盖之前的结果 False: 推理结果新建文件夹保存,文件夹名递增
    line_thickness=3, # bounding box thickness (pixels) 绘制Bounding_box的
线宽度
    hide_labels=False, # hide labels 若为True: 隐藏标签
    hide_conf=False, # hide confidences 若为True: 隐藏置信度
    half=False, # use FP16 half-precision inference 是否使用半精度推理 (节
约显存)
    dnn=False, # use OpenCV DNN for ONNX inference 是否使用OpenCV DNN预测
):

    '''===== 2.初始化配置 ====='''
    # 输入的路径变为字符串
    source = str(source)
    # 是否保存图片 and txt文件, 如果nosave(传入的参数)为false且source的结尾不是txt则保
存图片
    save_img = not nosave and not source.endswith('.txt') # save inference
images
    # 判断source是不是视频/图像文件路径
    # Path()提取文件名。suffix: 最后一个组件的文件扩展名。若source
是"D://YOLOv5/data/1.jpg", 则Path(source).suffix是".jpg",
Path(source).suffix[1:]是"jpg"
    # 而IMG_FORMATS 和 VID_FORMATS两个变量保存的是所有的视频和图片的格式后缀。
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    # 判断source是否是链接
    # .lower()转化成小写 .upper()转化成大写 .title()首字符转化成大写, 其余为小写,
.startswith('http://')返回True or False
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://',
'https://'))
    # 判断是source是否是摄像头
    # .isnumeric()是否是由数字组成, 返回True or False
    webcam = source.isnumeric() or source.endswith('.txt') or (is_url and not
is_file)
    if is_url and is_file:
        # 返回文件。如果source是一个指向图片/视频的链接,则下载输入数据
        source = check_file(source) # download

    '''===== 3.保存结果 ====='''
    # Directories
    # save_dir是保存运行结果的文件夹名, 是通过递增的方式来命名的。第一次运行时路径
是"runs\detect\exp", 第二次运行时路径是"runs\detect\exp1"
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)
    # 根据前面生成的路径创建文件夹
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True,
exist_ok=True) # make dir

    '''===== 4.加载模型 ====='''
    # Load model 加载模型
    # 获取设备 CPU/CUDA
    device = select_device(device)
    # DetectMultiBackend定义在models.common模块中, 是我们要加载的网络, 其中
weights参数就是输入时指定的权重文件 (比如yolov5s.pt)
    model = DetectMultiBackend(weights, device=device, dnn=dnn)
    stride, names, pt, jit, onnx = model.stride, model.names, model.pt,
model.jit, model.onnx
    ...

```

**stride**：推理时所用到的步长，默认为32，大步长适合于大目标，小步长适合于小目标

**names**：保存推理结果名的列表，比如默认模型的值是['person', 'bicycle', 'car', ...]

**pt**：加载的是否是pytorch模型（也就是pt格式的文件）

**jit**：当某段代码即将第一次被执行时进行编译，因而叫“即时编译”

**onnx**：利用Pytorch我们可以将model.pt转化为model.onnx格式的权重，在这里onnx充当一个后缀名称，

**model.onnx**就代表ONNX格式的权重文件，这个权重文件不仅包含了权重值，也包含了神经网络的网络流动信息以及每一层网络的输入输出信息和一些其他的辅助信息。

```
'''
# 确保输入图片的尺寸imgsz能整除stride=32 如果不能则调整为能被整除并返回
imgsz = check_img_size(imgsz, s=stride) # check image size

# Half
# 如果不是CPU，使用半精度(图片半精度/模型半精度)
half &= pt and device.type != 'cpu' # half precision only supported by
PyTorch on CUDA
if pt:
    model.model.half() if half else model.model.float()

'''===== 5.加载数据 ====='''
# Dataloader
# 通过不同的输入源来设置不同的数据加载方式
if webcam: # 使用摄像头作为输入
    view_img = check_imshow() # 检测cv2.imshow()方法是否可以执行，不能执行则
抛出异常
    cudnn.benchmark = True # set True to speed up constant image size
inference 该设置可以加速预测
    dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt
and not jit)# 加载输入数据流
'''
    source：输入数据源；image_size 图片识别前被放缩的大小；stride：识别时的步
长，
    auto的作用可以看utils.augmentations.letterbox方法，它决定了是否需要将图片
填充为正方形，如果auto=True则不需要
'''
    bs = len(dataset) # batch_size 批大小
else: # 直接从source文件下读取图片
    dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt
and not jit)
    bs = 1 # batch_size
# 保存视频的路径
vid_path, vid_writer = [None] * bs, [None] * bs # 前者是视频路径,后者是一个
cv2.VideoWriter对象

'''===== 6.推理部分 ====='''
# Run inference
if pt and device.type != 'cpu':
    # 使用空白图片（零矩阵）预先用GPU跑一遍预测流程，可以加速预测
    model(torch.zeros(1, 3,
*imgsz).to(device).type_as(next(model.model.parameters())))) # warmup
    dt, seen = [0.0, 0.0, 0.0], 0
'''
    dt：存储每一步骤的耗时
    seen：计数功能，已经处理完了多少帧图片
'''
```

```

# 去遍历图片，进行计数，
for path, im, im0s, vid_cap, s in dataset:
    ...

    在dataset中，每次迭代的返回值是self.sources, img, img0, None, ''
    path：文件路径（即source）
    im：resize后的图片（经过了放缩操作）
    im0s：原始图片
    vid_cap=None
    s：图片的基本信息，比如路径，大小
    ...

    # ===以下部分是做预处理===#
    t1 = time_sync() # 获取当前时间
    im = torch.from_numpy(im).to(device) # 将图片放到指定设备(如GPU)上识别。
#torch.size=[3,640,480]
    im = im.half() if half else im.float() # uint8 to fp16/32 # 把输入从整
    型转化为半精度/全精度浮点数。
    im /= 255 # 0 - 255 to 0.0 - 1.0 归一化，所有像素点除以255
    if len(im.shape) == 3:
        im = im[None] # expand for batch dim 添加一个第0维。缺少batch这个尺
    寸，所以将它扩充一下，变成[1, 3,640,480]
    t2 = time_sync() # 获取当前时间
    dt[0] += t2 - t1 # 记录该阶段耗时

    # Inference
    # 可视化文件路径。如果为True则保留推理过程中的特征图，保存在runs文件夹中
    visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if
visualize else False
    # 推理结果，pred保存的是所有的bound_box的信息，
    pred = model(im, augment=augment, visualize=visualize) #模型预测出来的
    所有检测框，torch.size=[1,18900,85]
    t3 = time_sync()
    dt[1] += t3 - t2

    # NMS
    # 执行非极大值抑制，返回值为过滤后的预测框
    pred = non_max_suppression(pred, conf_thres, iou_thres, classes,
agnostic_nms, max_det=max_det)
    ...

    pred：网络的输出结果
    conf_thres：置信度阈值
    iou_thres：iou阈值
    classes：是否只保留特定的类别 默认为None
    agnostic_nms：进行nms是否也去除不同类别之间的框
    max_det：检测框结果的最大数量 默认1000
    ...

    # 预测+NMS的时间
    dt[2] += time_sync() - t3

    # Second-stage classifier (optional) 设置第二次分类，默认不使用
    # pred = utils.general.apply_classifier(pred, classifier_model, im,
im0s)

    # Process predictions
    # 把所有的检测框画到原图中
    for i, det in enumerate(pred): # per image 每次迭代处理一张图片
        ...
        i：每个batch的信息

```

```

det:表示5个检测框的信息
'''
seen += 1 #seen是一个计数的功能
if webcam: # batch_size >= 1
    # 如果输入源是webcam则batch_size>=1 取出dataset中的一张图片
    p, im0, frame = path[i], im0s[i].copy(), dataset.count
    s += f'{i}: ' # s后面拼接一个字符串i
else:
    p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame',
0)
'''
    大部分我们一般都是从LoadImages流读取本都文件中的照片或者视频 所以
batch_size=1
    p: 当前图片/视频的绝对路径 如 F:\yolo_v5\yolov5-
U\data\images\bus.jpg
    s: 输出信息 初始为 ''
    im0: 原始图片 letterbox + pad 之前的图片
    frame: 视频流,此次取的是第几张图片
'''
# 当前路径yolov5/data/images/
p = Path(p) # to Path
# 图片/视频的保存路径save_path 如 runs\detect\exp8\fire.jpg
save_path = str(save_dir / p.name) # im.jpg
# 设置保存框坐标的txt文件路径, 每张图片对应一个框坐标信息
txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode
== 'image' else f'_{frame}') # im.txt
# 设置输出图片信息。图片shape (w, h)
s += '%gx%g ' % im.shape[2:] # print string
# 得到原图的宽和高
gn = torch.tensor(im0.shape)[[1, 0, 1, 0]] # normalization gain
whwh
# 保存截图。如果save_crop的值为true, 则将检测到的bounding_box单独保存
成一张图片。
imc = im0.copy() if save_crop else im0 # for save_crop
# 得到一个绘图类, 类中预先存储了原图、线条宽度、类名
annotator = Annotator(im0, line_width=line_thickness,
example=str(names))

# 判断有没有框
if len(det):
    # Rescale boxes from img_size to im0 size
    # 将预测信息映射到原图
    # 将标注的bounding_box大小调整为和原图一致 (因为训练时原图经过了放
缩) 此时坐标格式为xyxy
    det[:, :4] = scale_coords(im.shape[2:], det[:, :4],
im0.shape).round() #scale_coords: 坐标映射功能

    # Print results
    # 打印检测到的类别数量
    for c in det[:, -1].unique():
        n = (det[:, -1] == c).sum() # detections per class
        s += f"{n} {names[int(c)]}{s' * (n > 1)}, " # add to
string

    # Write results
    # 保存预测结果: txt/图片画框/crop-image
    for *xyxy, conf, cls in reversed(det):

```

```

        # 将每个图片的预测信息分别存入save_dir/labels下的xxx.txt中 每
行: class_id + score + xywh
        if save_txt: # Write to file 保存txt文件
            # 将xyxy(左上角+右下角)格式转为xywh(中心点+宽长)格式, 并归
一化, 转化为列表再保存
            xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
gn).view(-1).tolist() # normalized xywh
            # line的形式是: "类别 x y w h" 若save_conf为true, 则
line的形式是: "类别 x y w h 置信度"
            line = (cls, *xywh, conf) if save_conf else (cls,
*xywh) # label format
            with open(txt_path + '.txt', 'a') as f:
                # 写入对应的文件夹里, 路径默认
为"runs\detect\exp*\labels"
                f.write((' %g ' * len(line)).rstrip() % line +
'\n')

        # 在原图上画框+将预测到的目标剪切出来保存成图片, 保存在
save_dir/crops下, 在原图像画图或者保存结果
        if save_img or save_crop or view_img: # Add bbox to image
            c = int(cls) # integer class # 类别标号
            label = None if hide_labels else (names[c] if
hide_conf else f'{names[c]} {conf:.2f}') # 类别名
            annotator.box_label(xyxy, label, color=colors(c,
True)) # 绘制边框

            # 在原图上画框+将预测到的目标剪切出来保存成图片, 保存在
save_dir/crops下 (单独保存)
            if save_crop:
                save_one_box(xyxy, imc, file=save_dir / 'crops' /
names[c] / f'{p.stem}.jpg', BGR=True)

        # Print time (inference-only)
        # 打印耗时
        LOGGER.info(f'{s}Done. ({t3 - t2:.3f}s)')

        # Stream results
        # 如果设置展示, 则show图片 / 视频
        im0 = annotator.result() # im0是绘制好的图片
        # 显示图片
        if view_img:
            cv2.imshow(str(p), im0)
            cv2.waitKey(1) # 暂停 1 millisecond

        # Save results (image with detections)
        # 设置保存图片/视频
        if save_img: # 如果save_img为true, 则保存绘制完的图片
            if dataset.mode == 'image': # 如果是图片, 则保存
                cv2.imwrite(save_path, im0)
            else: # 'video' or 'stream' 如果是视频或者"流"
                if vid_path[i] != save_path: # new video vid_path[i] !=
save_path, 说明这张图片属于一段新的视频, 需要重新创建视频文件
                    vid_path[i] = save_path
                # 以下的部分是保存视频文件
                if isinstance(vid_writer[i], cv2.VideoWriter):
                    vid_writer[i].release() # release previous video
writer

                if vid_cap: # video

```



```

        fps = vid_cap.get(cv2.CAP_PROP_FPS) # 视频帧速率
FPS
        w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH)) #
获取视频帧宽度
        h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT)) #
获取视频帧高度

    else: # stream
        fps, w, h = 30, im0.shape[1], im0.shape[0]
        save_path += '.mp4'
        vid_writer[i] = cv2.VideoWriter(save_path,
cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
        vid_writer[i].write(im0)

'''===== 7.在终端里打印出运行的结果 ====='''
# Print results
t = tuple(x / seen * 1E3 for x in dt) # speeds per image 平均每张图片所耗费
时间
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per
image at shape {(1, 3, *imgsz)}' % t)
    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to
{save_dir / 'labels'}" if save_txt else '' # 标签保存的路径
        LOGGER.info(f"Results saved to {colorstr('bold', save_dir)}{s}")
    if update:
        strip_optimizer(weights) # update model (to fix SourceChangeWarning)

'''===== 三、Parse_opt()用来设置输入参数的子函数
====='''
def parse_opt():
    """
    weights: 训练的权重路径,可以使用自己训练的权重,也可以使用官网提供的权重
    默认官网的权重
yolov5s.pt(yolov5n.pt/yolov5s.pt/yolov5m.pt/yolov5l.pt/yolov5x.pt/区别在于网络
的宽度和深度以此增加)
    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,默认data/images
    data: 配置数据文件路径,包括image/label/classes等信息,训练自己的文件,需要
作相应更改,可以不用管如果设置了只显示个别类别即使用了--classes = 0 或二者1, 2, 3
等,则需要设置该文件,数字和类别相对应才能只检测某一个类
    imgsz: 网络输入图片大小,默认的大小是640
    conf-thres: 置信度阈值,默认为0.25
    iou-thres: 做nms的iou阈值,默认为0.45
    max-det: 保留的最大检测框数量,每张图片中检测目标的个数最多为1000类
    device: 设置设备CPU/CUDA,可以不用设置
    view-img: 是否展示预测之后的图片/视频,默认False, --view-img 电脑界面出现图
片或者视频检测结果
    save-txt: 是否将预测的框坐标以txt文件形式保存,默认False,使用--save-txt 在
路径runs/detect/exp*/labels/*.txt下生成每张图片预测的txt文件
    save-conf: 是否将置信度conf也保存到txt中,默认False
    save-crop: 是否保存裁剪预测框图片,默认为False,使用--save-crop 在
runs/detect/exp*/crop/剪切类别文件夹/ 路径下会保存每个接下来的目标
    nosave: 不保存图片、视频,要保存图片,不设置--nosave 在runs/detect/exp*/会出
现预测的结果
    classes: 设置只保留某一部分类别,形如0或者0 2 3,使用--classes = n, 则在路径
runs/detect/exp*/下保存图片为n所对应的类别,此时需要设置data
    agnostic-nms: 进行NMS去除不同类别之间的框,默认False
    augment: TTA测试时增强/多尺度预测

```



```

visualize: 是否可视化网络层输出特征
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: 保存测试日志的文件夹路径
name: 保存测试日志文件夹的名字, 所以最终是保存在project/name中
exist_ok: 是否重新创建日志文件, False时重新创建文件
line-thickness: 画框的线条粗细
hide-labels: 可视化时隐藏预测类别
hide-conf: 可视化时隐藏置信度
half: 是否使用F16精度推理, 半精度提高检测速度
dnn: 用OpenCV DNN预测
"""

parser = argparse.ArgumentParser()
parser.add_argument('--weights', nargs='+', type=str, default=ROOT /
' yolov5s.pt', help='model path(s)')
parser.add_argument('--source', type=str, default=ROOT / 'data/images',
help='file/dir/URL/glob, 0 for webcam')
parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int,
default=[640], help='inference size h,w')
parser.add_argument('--conf-thres', type=float, default=0.5,
help='confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU
threshold')
parser.add_argument('--max-det', type=int, default=1000, help='maximum
detections per image')
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='show
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('--save-crop', action='store_true', help='save cropped
prediction boxes')
parser.add_argument('--nosave', action='store_true', help='do not save
images/videos')
parser.add_argument('--classes', nargs='+', type=int, help='filter by
class: --classes 0, or --classes 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('--visualize', action='store_true', help='visualize
features')
parser.add_argument('--update', action='store_true', help='update all
models')
parser.add_argument('--project', default=ROOT / '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')
parser.add_argument('--line-thickness', default=3, type=int,
help='bounding box thickness (pixels)')
parser.add_argument('--hide-labels', default=False, action='store_true',
help='hide labels')

```

```

    parser.add_argument('--hide-conf', default=False, action='store_true',
                        help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-
precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for
ONNX inference')

    opt = parser.parse_args() # 扩充维度
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1 # expand
    print_args(FILE.stem, opt) # 打印所有参数信息
    return opt

'''===== 四、设置main函数 ====='''
def main(opt):
    # 检查环境/打印参数,主要是requirement.txt的包是否安装,用彩色显示设置的参数
    check_requirements(exclude=('tensorboard', 'thop'))
    # 执行run()函数
    run(**vars(opt))

# 命令使用
# python detect.py --weights runs/train/exp_yolov5s/weights/best.pt --source
data/images/fishman.jpg # webcam
if __name__ == "__main__":
    opt = parse_opt() # 解析参数
    main(opt) # 执行主函数

```

##### 代码分析

'''===== 一.导入安装好的 python 库 =====''' import argparse # 解析命令行参数的库 import os # 与操作系统进行交互的文件库 包含文件路径操作与解析 import sys # sys模块包含了与python解释器和它的环境有关的函数。 from pathlib import Path # Path能够更加方便得对字符串路径进行处理

import cv2 # sys模块包含了与python解释器和它的环境有关的函数。 import torch #pytorch深度学习库 import torch.backends.cudnn as cudnn #让内置的cudnn的 auto-tuner 自动寻找最适合当前配置的高效算法,来达到优化运行效率的问题

**argparse:** 它是一个用于命令选项与参数解析的模块,通过在程序中定义好我们需要的参数, argparse将会从sys.argv中解析出这些参数,并自动生成帮助和使用信息 **os:** 它提供了多种操作系统的接口。通过os模块提供的操作系统接口,我们可以对操作系统里文件、终端、进程等进行操作 **sys:** 它是与python解释器交互的一个接口,该模块提供对解释器使用或维护的一些变量的访问和获取,它提供了许多函数和变量来处理 Python 运行时环境的不同部分 **pathlib:** 这个库提供了一种面向对象的方式来与文件系统交互,可以让代码更简洁、更易读 **torch:** 这是主要的Pytorch库。它提供了构建、训练和评估神经网络的工具 **torch.backends.cudnn:** 它提供了一个接口,用于使用cuDNN库,在 NVIDIA GPU上高效地进行深度学习。cudnn模块是一个Pytorch库的扩展

'''===== 二.获取当前文件的绝对路径 =====''' FILE = Path(file).resolve() # \_\_file\_\_指的是当前文件(即detect.py),FILE最终保存着当前文件的绝对路径,比如D://yolov5/detect.py ROOT = FILE.parents[0] # YOLOv5 root directory ROOT保存着当前项目的父目录,比如 D://yolov5 if str(ROOT) not in

**sys.path:** # sys.path即当前python环境可以运行的路径,假如当前项目不在该路径中,就无法运行其中的模块,所以需要加载路径 **sys.path.append(str(ROOT))** # add ROOT to PATH 把ROOT添加到运行路径上 **ROOT = Path(os.path.relpath(ROOT, Path.cwd()))** # relative ROOT 设置为相对路径

这一部分的主要作用有两个: 将当前项目添加到系统路径上,以使得项目中的模块可以调用。 将当前项目的相对路径保存在ROOT中,便于寻找项目中的文件。

```
'''===== 三.加载自定义模块 =====''' from models.common import
DetectMultiBackend from utils.datasets import IMG_FORMATS, VID_FORMATS,
LoadImages, LoadStreams from utils.general import (LOGGER, check_file,
check_img_size, check_imshow, check_requirements, colorstr, increment_path,
non_max_suppression, print_args, scale_coords, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box from
utils.torch_utils import select_device, time_sync
```

**models.common.py:** 这个文件定义了一些通用的函数和类,比如图像的处理、非极大值抑制等等。

**utils.dataloaders.py:** 这个文件定义了两个类, **LoadImages**和**LoadStreams**,它们可以加载图像或视频帧,并对它们进行一些预处理,以便进行物体检测或识别。 **utils.general.py:** 这个文件定义了一些常用的工具函数,比如检查文件是否存在、检查图像大小是否符合要求、打印命令行参数等等。

**utils.plots.py:** 这个文件定义了**Annotator**类,可以在图像上绘制矩形框和标注信息。

**utils.torch\_utils.py:** 这个文件定义了一些与PyTorch有关的工具函数,比如选择设备、同步时间等等。

这些都是用户自定义的库,由于上一步已经把路径加载上了,所以现在可以导入,这个顺序不可以调换。具体来说,代码从如下几个文件中导入了部分函数和类: 通过导入这些模块,可以更方便地进行目标检测的相关任务,并且减少了代码的复杂度和冗余。

```
'''===== 二、设置main函数 =====''' def main(opt): # 检查环境/打印参
数,主要是requirement.txt的包是否安装,用彩色显示设置的参数
check_requirements(exclude=('tensorboard', 'thop')) # 执行run()函数
run(**vars(opt))
```

## 命令使用

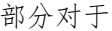
---

```
python detect.py --weights
runs/train/exp_yolov5s/weights/best.pt
--source data/images/fishman.jpg #
webcam
```

---

```
if name == "main": opt = parse_opt() # 解析参数 main(opt) # 执行主函数
```

**if name == 'main':** 的作用: 一个python文件通常有两种使用方法,第一是作为脚本直接执行,第二是 **import**到其他的python脚本中被调用(模块重用)执行。因此 **if name == "main":** 的作用就是控制这两种情况执行代码的过程,在 **if name == "main":** 下的代码只有在第一种情况下(即文件作为脚本直接执行)才会被执行,而**import**到其他脚本中是不会被执行的。

`check_requirements (exclude= ('tensorboard', 'thop'))` : 检查程序所需的依赖项是否已安装。  
`run (**vars (opt))` : 将 `opt` 变量的属性和属性值作为关键字参数传递给 `run ()` 函数。  
`opt= parse_opt ()` : 解析命令行传进的参数。该段代码分为三部分, 第一部分定义了一些可以传导的参数类型, 第二部分对于部分进行了额外的判断 (640\*640), 第三部分打印所有参数信息, `opt`变量存储所有的参数信息, 并返回。  
`main (opt)` : 执行命令行参数。该段代码分为两部分, 第一部分首先完成对于requirements.txt的检查, 检测这些依赖包有没有安装; 第二部分, 将`opt`变量参数传入, 执行`run`函数。

'''===== 三、 Parse\_opt() 用来设置输入参数的子函数 =====''' def parse\_opt():

```

parser = argparse.ArgumentParser()
parser.add_argument('--weights', nargs='+', type=str, default=ROOT /
' yolov5s.pt', help='model path(s)')
parser.add_argument('--source', type=str, default=ROOT / 'data/images',
help='file/dir/URL/glob, 0 for webcam')
parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int,
default=[640], help='inference size h,w')
parser.add_argument('--conf-thres', type=float, default=0.5, help='confidence
threshold')
parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU
threshold')
parser.add_argument('--max-det', type=int, default=1000, help='maximum
detections per image')
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='show 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('--save-crop', action='store_true', help='save cropped
prediction boxes')
parser.add_argument('--nosave', action='store_true', help='do not save
images/videos')
parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --
classes 0, or --classes 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('--visualize', action='store_true', help='visualize
features')
parser.add_argument('--update', action='store_true', help='update all models')
parser.add_argument('--project', default=ROOT / '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')
parser.add_argument('--line-thickness', default=3, type=int, help='bounding box
thickness (pixels)')

```

```

parser.add_argument('--hide-labels', default=False, action='store_true',
help='hide labels')
parser.add_argument('--hide-conf', default=False, action='store_true',
help='hide confidences')
parser.add_argument('--half', action='store_true', help='use FP16 half-precision
inference')
parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX
inference')

opt = parser.parse_args() # 扩充维度
opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1 # expand
print_args(FILE.stem, opt) # 打印所有参数信息
return opt

```

**--weights:** 训练的权重路径，可以使用自己训练的权重，也可以使用官网提供的权重。默认官网的权重 `yolov5s.pt` (`yolov5n.pt/yolov5s.pt/yolov5m.pt/yolov5l.pt/yolov5x.pt`/区别在于网络的宽度和深度以此增加) **--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，默认 `data/images` **--data:** 配置数据文件路径，包括 `image/label/classes` 等信息，训练自己的文件，需要作相应更改，可以不用管 **--imgsz:** 预测时网络输入图片的尺寸，默认值为 `[640]` **--conf-thres:** 置信度阈值，默认为 `0.50` **--iou-thres:** 非极大抑制时的 IoU 阈值，默认为 `0.45` **--max-det:** 保留的最大检测框数量，每张图片中检测目标的个数最多为 `1000` 类 **--device:** 使用的设备，可以是 `cuda` 设备的 ID (例如 `0, 0, 1, 2, 3`) 或者是 `'cpu'`，默认为 `'0'` **--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`，使用 `--save-txt` 在 `runs/detect/exp*/labels/*.txt` 下生成每张图片预测的 `txt` 文件 **--save-conf:** 是否保存检测结果的置信度到 `txt` 文件，默认为 `False` **--save-crop:** 是否保存裁剪预测框图片，默认为 `False`，使用 `-save-crop` 在 `runs/detect/exp/crop/` 剪切类别文件夹/路径下会保存每个接下来的目标 **--nosave:** 不保存图片、视频，要保存图片，不设置 `--nosave` 在 `runs/detect/exp/` 会出现预测的结果 **--classes:** 仅检测指定类别，默认为 `None` **--agnostic-nms:** 是否使用类别不敏感的非极大抑制 (即不考虑类别信息)，默认为 `False` **--augment:** 是否使用数据增强进行推理，默认为 `False` **--visualize:** 是否可视化特征图，默认为 `False` **--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:** 结果保存的项目目录路径，默认为 `'R00T/runs/detect'` **--name:** 结果保存的子目录名称，默认为 `'exp'` **--exist-ok:** 是否覆盖已有结果，默认为 `False` **--line-thickness:** 画 `bounding box` 时的线条宽度，默认为 `3` **--hide-labels:** 是否隐藏标签信息，默认为 `False` **--hide-conf:** 是否隐藏置信度信息，默认为 `False` **--half:** 是否使用 `FP16` 半精度进行推理，默认为 `False` **--dnn:** 是否使用 `OpenCV DNN` 进行 `ONNX` 推理，默认为 `False`

```

'''===== 1.载入参数 =====''' @torch.no_grad() # 该标注使得方法中所有
计算得出的tensor的requires_grad都自动设置为False，也就算不进行梯度的计算(当然也就没办法反向传播了)，节约显存和算
def run(weights=R00T / 'yolov5s.pt', # model.pt
path(s) 事先训练完成的权重文件，比如yolov5s.pt,默认 weights/, 假如使用官方训练好的文件
(比如yolov5s),则会自动下载 source=R00T / 'data/images', # file/dir/URL/glob, 0
for webcam 预测时的输入数据，可以是文件/路径/URL/glob, 输入是0的话调用摄像头作为输入，默认data/images/ # data=R00T / 'data/coco128.yaml', # dataset.yaml path, data文件
路径，包括类别/图片/标签等信息 imgsz=(640, 640), # inference size (pixels) 预测时的
放缩后图片大小(因为YOLO算法需要预先放缩图片)，两个值分别是height, width。默认640640
conf_thres=0.25, # confidence threshold 置信度阈值，高于此值的bounding_box才会被保

```

留。默认0.25, 用在nms中 `iou_thres=0.45`, # NMS IOU threshold IOU阈值, 高于此值的 `bounding_box` 才会被保留。默认0.45, 用在nms中 `max_det=1000`, # maximum detections per image 一张图片上检测的最大目标数量, 用在nms中 `device=''`, # cuda device, i.e. 0 or 0,1,2,3 or cpu 所使用的GPU编号, 如果使用CPU就写`cpu` `view_img=False`, # show results 是否展示预测之后的图片或视频, 默认False `save_txt=False`, # save results to .txt 是否将预测的框坐标以txt文件形式保存, 默认False, 使用`--save-txt` 在路径 `runs/detect/exp/labels/.txt`下生成每张图片预测的txt文件 `save_conf=False`, # save confidences in `--save-txt labels` 是否将结果中的置信度保存在txt文件中, 默认False `save_crop=False`, # save cropped prediction boxes 是否保存裁剪后的预测框, 默认为False, 使用`--save-crop` 在`runs/detect/exp*/crop/`剪切类别文件夹/ 路径下会保存每个接下来的目标 `nosave=False`, # do not save images/videos 不保存图片、视频, 要保存图片, 不设置`--nosave` 在`runs/detect/exp*/`会出现预测的结果 `classes=None`, # filter by class: `--class 0`, or `--class 0 2 3` 过滤指定类的预测结果 `agnostic_nms=False`, # class-agnostic NMS 进行NMS去除不同类别之间的框, 默认False `augment=False`, # augmented inference TTA测试时增强/多尺度预测, 可以提分 `visualize=False`, # visualize features 是否可视化网络层输出特征 `update=False`, # update all models 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=R00T / 'runs/detect'`, # save results to project/name 预测结果保存的路径 `name='exp'`, # save results to project/name 结果保存文件夹的命名前缀 `exist_ok=False`, # existing project/name ok, do not increment True: 推理结果覆盖之前的结果 False: 推理结果新建文件夹保存, 文件夹名递增 `line_thickness=3`, # bounding box thickness (pixels) 绘制 Bounding\_box的线宽度 `hide_labels=False`, # hide labels 若为True: 隐藏标签 `hide_conf=False`, # hide confidences 若为True: 隐藏置信度 `half=False`, # use FP16 half-precision inference 是否使用半精度推理(节约显存) `dnn=False`, # use OpenCV DNN for ONNX inference 是否使用OpenCV DNN预测):

```
'''===== 2.初始化配置 =====''' # 输入的路径变为字符串 source =
str(source) # 是否保存图片 and txt文件, 如果nosave(传入的参数)为false且source的结尾不是
txt则保存图片 save_img = not nosave and not source.endswith('.txt') # save
inference images # 判断source是不是视频/图像文件路径 # Path()提取文件名。suffix: 最
最后一个组件的文件扩展名。若source是"D://Y0L0v5/data/1.jpg", 则Path(source).suffix
是".jpg", Path(source).suffix[1:]是"jpg" # 而IMG_FORMATS 和 VID_FORMATS两个变量
保存的是所有的视频和图片的格式后缀。is_file = Path(source).suffix[1:] in
(IMG_FORMATS + VID_FORMATS) # 判断source是否是链接 # .lower()转化成小写 .upper()
转化成大写 .title()首字符转化成大写, 其余为小写, .startswith('http://')返回True or
False is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://',
'https://')) # 判断是source是否是摄像头 # .isnumeric()是否是由数字组成, 返回True or
False webcam = source.isnumeric() or source.endswith('.txt') or (is_url and
not is_file) if is_url and is_file: # 返回文件。如果source是一个指向图片/视频的链
接, 则下载输入数据 source = check_file(source) # download
```

这段代码主要用于处理输入来源。定义了一些布尔值区分输入是图片、视频、网络流还是摄像头。首先将source转换为字符串类型, 然后判断是否需要保存输出结果。如果nosave和source的后缀不是.txt, 则会保存输出结果。接着根据source的类型, 确定输入数据的类型:

```
'''===== 三.保存结果 =====''' # Directories # save_dir是保存运行结
果的文件夹名, 是通过递增的方式来命名的。第一次运行时路径是"runs\detect\exp", 第二次运行时
```



```

路径是“runs\detect\exp1” save_dir = increment_path(Path(project) / name,
exist_ok=exist_ok) # 根据前面生成的路径创建文件夹 (save_dir / 'labels' if save_txt
else save_dir).mkdir(parents=True, exist_ok=True) # make dir

```

这段代码主要是用于创建保存输出结果的目录。创建一个新的文件夹exp（在runs文件夹下）来保存运行的结果。首先代码中的project指run函数中的project，对应的是runs/detect的目录，name对应run函数中的"name-exp"，然后进行拼接操作。使用ncrement\_path函数来确保目录不存在，如果存在，则在名称后面添加递增的数字。然后判断save\_txt 是否为true，save\_txt 在run 函数以及parse\_opt（）函数中都有相应操作，如果传入save\_txt，新建"labels"文件夹存储结果 这个过程中，如果目录已经存在，而exist\_ok为False，那么会抛出一个异常，指示目录已存在。如果exist\_ok为True，则不会抛出异常，而是直接使用已经存在的目录。

```

'''===== 四.加载模型 =====''' # Load model 加载模型 # 获取设备
CPU/CUDA device = select_device(device) # DetectMultiBackend定义在
models.common模块中，是我们要加载的网络，其中weights参数就是输入时指定的权重文件（比如
yolov5s.pt） model = DetectMultiBackend(weights, device=device, dnn=dnn)
stride, names, pt, jit, onnx = model.stride, model.names, model.pt,
model.jit, model.onnx ''' stride: 推理时所用到的步长，默认为32， 大步长适合于大目标，
小步长适合于小目标 names: 保存推理结果名的列表，比如默认模型的值是['person', 'bicycle',
'car', ...] pt: 加载的是否是pytorch模型（也就是pt格式的文件） jit: 当某段代码即将第一次
被执行时进行编译，因而叫“即时编译” onnx: 利用Pytorch我们可以将model.pt转化为model.onnx
格式的权重，在这里onnx充当一个后缀名称， model.onnx就代表ONNX格式的权重文件，这个权重文件
不仅包含了权重值，也包含了神经网络的网络流动信息以及每一层网络的输入输出信息和一些其他的辅助
信息。 ''' # 确保输入图片的尺寸imgsz能整除stride=32 如果不能则调整为能被整除并返回
imgsz = check_img_size(imgsz, s=stride) # check image size

```

```

# Half
# 如果不是CPU · 使用半精度(图片半精度/模型半精度)
half &= pt and device.type != 'cpu' # half precision only supported by PyTorch
on CUDA
if pt:
    model.model.half() if half else model.model.float()

```

这段代码主要是用于选择设备、初始化模型和检查图像大小。首先调用select\_device函数选择设备，如果device为空，则使用默认设备。然后使用DetectMultiBackend类来初始化模型，其中weights 指模型的权重路径 device 指设备 dnn 指是否使用OpenCV DNN data 指数据集配置文件的路径 fp16指是否使用半精度浮点数进行推理 接着从模型中获取stride、names和pt等参数，其中stride 指下采样率 names 指模型预测的类别名称 pt 是Pytorch模型对象 最后调用check\_img\_size函数检查图像大小是否符合要求，如果不符合则进行调整。

```

'''===== 五.加载数据 =====''' # Dataloader # 通过不同的输入源来设置不
同的数据加载方式 if webcam: # 使用摄像头作为输入 view_img = check_imshow() # 检测
cv2.imshow()方法是否可以执行，不能执行则抛出异常 cudnn.benchmark = True # set True
to speed up constant image size inference 该设置可以加速预测 dataset =
LoadStreams(source, img_size=imgsz, stride=stride, auto=pt and not jit)# 加载
输入数据流 ''' source: 输入数据源; image_size 图片识别前被放缩的大小; stride: 识别时的
步长， auto的作用可以看utils.augmentations.letterbox方法，它决定了是否需要将图片填充为

```

```

正方形，如果auto=True则不需要 ''' bs = len(dataset) # batch_size 批大小 else: # 直
接从source文件下读取图片 dataset = LoadImages(source, img_size=imgsz,
stride=stride, auto=pt and not jit) bs = 1 # batch_size # 保存视频的路径
vid_path, vid_writer = [None] * bs, [None] * bs # 前者是视频路径,后者是一个
cv2.VideoWriter对象

```

这段代码是根据输入的source参数来判断是否是通过webcam摄像头捕捉视频流如果是，则使用LoadStreams 加载视频流 否则，使用LoadImages加载图像 如果是webcam模式，则设置cudnn.benchmark= True以加速常量图像大小的推理。bs表示batch\_size（批量大小），这里是1或视频流中的帧数。vid\_path和vid\_writer分别是视频路径和视频编写器，初始化为长度为batch\_size的空列表。

```

'''===== 六.推理部分 ====='''

```

推理部分是整个算法的核心部分。通过for循环对加载的数据进行遍历，一帧一帧地推理，进行NMS非极大值抑制、绘制bounding box、预测类别。

```

'''===== 七.在终端里打印出运行的结果 =====''' # Print results t =
tuple(x / seen * 1E3 for x in dt) # speeds per image 平均每张图片所耗费时间
LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per
image at shape {(1, 3, imgsz)}' % t) if save_txt or save_img: s =
f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir /
'labels'}" if save_txt else '' # 标签保存的路径 LOGGER.info(f"Results saved to
{colorstr('bold', save_dir)}{s}") if update: strip_optimizer(weights) #
update model (to fix SourceChangeWarning)

```

这部分代码用于打印结果，记录了一些总共的耗时，以及信息保存。 输出结果包括每张图片的预处理、推理和NMS时间，以及结果保存的路径。 如果update为True，则将模型更新，以修复SourceChangeWarning。

```

=====
=====

```