```
import os
from collections import deque
import time
import cv2
import numpy as np
import torch
import warnings
import argparse
from person_count import tlbr_midpoint, intersect, vector_angle, get_size_with_pil, compute_color_for_labels, \
    put_text_to_cv2_img_with_pil, draw_yellow_line
from utils.datasets import LoadStreams, LoadImages
from utils.draw import draw_boxes_and_text, draw_person, draw_boxes
from utils.general import check_img_size
from person_detect_yolov5 import YoloPersonDetect
from deep_sort import build_tracker, DeepReid
from utils.parser import get_config
from utils.log import get_logger
from utils.torch_utils import select_device, load_classifier, time_synchronized
from sklearn.metrics.pairwise import cosine_similarity
from fast_reid.demo.person_bank import Reid_feature

from pycallgraph2 import PyCallGraph
from pycallgraph2.output import GraphvizOutput

def parse_args():
    parser = argparse.ArgumentParser()

    # parser.add_argument("--video_path", default='./test_video/test3', type=str) # ok

    parser.add_argument("--video_path", default='./test_video/cam1.mp4', type=str)
    parser.add_argument("--video_out_path", default='./test_video/cam2.mp4', type=str)

    # parser.add_argument("--video_path", default='./test_video/vid_in.mp4', type=str)
    # parser.add_argument("--video_out_path", default='./test_video/vid_out.mp4', type=str)

    parser.add_argument("--camera", action="store", dest="cam", type=int, default="-1")
    parser.add_argument('--device', default='cuda:0', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument("--display", default=True, help='True: show window, False: not')
    parser.add_argument("--frame_interval", type=int, default=1)
    parser.add_argument("--cpu", dest="use_cuda", action="store_false", default=True)
    # yolov5
    parser.add_argument('--weights',  nargs='+',  type=str,  default='./weights/yolov5s.pt',  help='model.pt
path(s)')
    parser.add_argument('--img-size', type=int, default=1080, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.4, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.5, help='IOU threshold for NMS')
    parser.add_argument('--classes', default=[0], 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')
    # deep_sort
    parser.add_argument("--sort", default=False, help='True: sort model or False: reid model')
    parser.add_argument("--config_deepsort", type=str, default="./configs/deep_sort.yaml")

    return parser.parse_args()

class TrafficMonitor():
    def __init__(self, cfg, args, path_in, path_out):
        self.logger = get_logger("root")
        self.args = args
        self.video_in_path = path_in
        self.video_path_out = path_out
        use_cuda = args.use_cuda and torch.cuda.is_available()
        if not use_cuda:
            warnings.warn("Running in cpu mode which maybe very slow!", UserWarning)
        self.yolo_model = YoloPersonDetect(self.args)
        self.deepsort = build_tracker(cfg, args.sort, use_cuda=use_cuda)        # Deepsort with ReID
        imgsz = check_img_size(args.img_size, s=32)  # check img_size
        self.dataset_1 = LoadImages(self.video_in_path, img_size=imgsz)        # Read video frame
        self.dataset_2 = LoadImages(self.video_path_out, img_size=imgsz)

        self.logger.info("args: ", self.args)

    # 创建目录
    def makedir(self, dir_path):
        dir_path = os.path.dirname(dir_path)  # 获取路径名，删掉文件名
        bool = os.path.exists(dir_path)  # 存在返回 True，不存在返回 False
        if bool:
            pass
        else:
            os.makedirs(dir_path)

    def demo(self):
        self.enter_cam()  # enter store
        # self.feature_extract()  # extract features of customers, who entered
        # self.exit_cam()  # exit store

    def enter_cam(self):
        idx_frame = 0
        paths = {} # 每一个 track 的行动轨迹
        last_track_id = -1
        total_track = 0
        angle = -1
        total_counter = 0
        up_count = 0
        down_count = 0
        already_counted = deque(maxlen=50)  # temporary memory for storing counted IDs
        # ------------------ 入店逻辑：截取客户的图像 & 转化成特征向量 -----------------------
        for video_path, img, ori_img, vid_cap in self.dataset_1: # 获取视频帧
```

```
            idx_frame += 1
            start_time = time_synchronized()
            # yolo detection
            bbox_xywh, cls_conf, cls_ids, xy = self.yolo_model.detect(video_path, img, ori_img, vid_cap)
            # do tracking # features: reid 模型输出 512dim 特征
            outputs, features = self.deepsort.update(bbox_xywh, cls_conf, ori_img) # TODO: 路径问题，一定要放在
test_video 下才可以
            # 1. 画黄线
            p2 = [400, 500]
            p1 = [400, 900]
            yellow_line_in = draw_yellow_line(p1, p2, ori_img)



            # 2. 统计跟踪的结果：
            #  2.1 给每一个 track 画出轨迹
            #  2.2 检查 track 是否与黄线相交
            #   2.2.1 如果 track 跨过了黄线，则判断是进入还是离开。如果是进入则提取出 ROI 并保存到 runs 目录下
            for track in outputs:
                bbox = track[:4]
                track_id = track[-1]
                midpoint_1 = tlbr_midpoint(bbox)
                origin_midpoint = (midpoint_1[0],
                                   ori_img.shape[0] - midpoint_1[1])  # get midpoint_1 respective to bottom-left
                if track_id not in paths:
                    paths[track_id] = deque(maxlen=2)  # path 保存了每个 track 的两个帧的 midpoint（运动轨迹）
                    total_track = track_id
                paths[track_id].append(midpoint_1)
                midpoint_0 = paths[track_id][0]  # 此 track 前一帧的 midpoint
                origin_previous_midpoint = (midpoint_0[0], ori_img.shape[0] - midpoint_0[1])

                if intersect(midpoint_1, midpoint_0, yellow_line_in[0], yellow_line_in[1]) \
                        and track_id not in already_counted:
                    total_counter += 1
                    last_track_id = track_id;  # 记录触线者的 ID
                    cv2.line(ori_img, yellow_line_in[0], yellow_line_in[1], (0, 0, 255), 1)  # 触碰线的情况下画
红线
                    already_counted.append(track_id)  # Set already counted for ID to true.
                    angle = vector_angle(origin_midpoint, origin_previous_midpoint)  # 计算角度，判断向上还是向下
走
                    if angle > 0:  # 进店
                        up_count += 1
                        # 进店的时候，把人物的图像抠出来
                        cv2.line(ori_img, yellow_line_in[0], yellow_line_in[1], (0, 0, 0), 1)  # 消除线条
                        ROI_person = ori_img[int(bbox[1]):int(bbox[3]), int(bbox[0]):int(bbox[2])]
                        path = str('./runs/reid_output/enter/track_id-{}.jpg'.format(track_id))
                        self.makedir(path)
                        cv2.imwrite(path, ROI_person)
                        # 打印当前的时间 & 顾客入店信息
                        current_time = int(time.time())
                        localtime = time.localtime(current_time)
```

```python
                        dt = time.strftime('%Y-%m-%d %H:%M:%S', localtime)
                        print("[Customer came👏] current customer💂: {}, "
                              "Enter time⏰ : {}".format(
                            track_id
                            , dt
                        ))
                    if angle < 0:
                        down_count += 1

                if len(paths) > 50: # TODO: 50 写到常量中
                    del paths[list(paths)[0]]

            # 4. 绘制统计信息（出入商店的人数） & 绘制检测框
            ori_img = self.print_statistics_to_frame(down_count, ori_img, total_counter, total_track, up_count)
            if last_track_id >= 0:
                ori_img = self.print_newest_info(angle, last_track_id, ori_img)
            if len(outputs) > 0:
                bbox_tlwh = []
                bbox_xyxy = outputs[:, :4]
                identities = outputs[:, -1]
                ori_im = draw_boxes_and_text(ori_img, bbox_xyxy, identities)  # 给每个 detection 画框 todo: 不需要输出
                for bb_xyxy in bbox_xyxy:
                    bbox_tlwh.append(self.deepsort._xyxy_to_tlwh(bb_xyxy))
            end_time = time_synchronized()
            # 5. 展示处理后的图像
            if self.args.display:
                cv2.imshow("test", ori_img)
                if cv2.waitKey(1) & 0xFF == 27:
                    break
            self.logger.info("Index of frame: {} / "
                             "One Image spend time: {:.03f}s, "
                             "fps: {:.03f}, "
                             "tracks : {}, "
                             "detections : {}, "
                             "features of detections: {}"
                             .format(idx_frame, end_time - start_time, 1 / (end_time - start_time)
                                     , bbox_xywh.shape[0]
                                     , len(outputs)
                                     , len(bbox_xywh)
                                     , features.shape
                                     )
                             )
        cv2.destroyAllWindows()  ## 销毁所有 opencv 显示窗口
        return idx_frame

    # 进店客户的行人特征 & 名字会存储在 'runs/query_features.npy' 和 'query/names.npy' 中
    # todo: 抽取特征和读取特征分离
    def feature_extract(self):
        reid_feature = Reid_feature() # reid model
    names = []
    embs = np.ones((1, 512), dtype=np.int)
    for image_name in os.listdir('./runs/reid_output/enter'):
        img = cv2.imread(os.path.join('./runs/reid_output/enter', image_name))
        feat = reid_feature(img)  # extract normlized feat
        pytorch_output = feat.numpy()
        embs = np.concatenate((pytorch_output, embs), axis=0)
        names.append(image_name[0:-4])  # 去除.jpg 作为顾客的名字
    names = names[::-1]
    names.append("None")
    np.save(os.path.join('./runs', 'query_features'), embs[:-1, :])
    np.save(os.path.join('./runs', 'names'), names)  # save query
    path = str('./runs/query_features.npy')
    self.makedir(path)
    query = np.load(path)
    cos_sim = cosine_similarity(embs, query)
    max_idx = np.argmax(cos_sim, axis=1)
    maximum = np.max(cos_sim, axis=1)
    max_idx[maximum < 0.6] = -1
    # store query_fratures.npy & names.npy
    self.query_feat = query
    self.names = names
    self.logger.info("Succeed extracting features for ReID.")

def exit_cam(self):
    idx_frame = 0
    results = []
    paths = {}
    last_track_id = -1
    total_track = 0
    angle = -1
    total_counter = 0
    up_count = 0
    down_count = 0
    already_counted = deque(maxlen=50)  # temporary memory for storing counted IDs
    # ------------------ 出店逻辑: 截取客户的图像 & 与入店的人做匹配 & 输出对应的 ID --------
    for video_path, img, ori_img, vid_cap in self.dataset_2:
        idx_frame += 1
        # print("[INFO] out index frame = ", idx_frame)
        start_time = time_synchronized()
        # yolo detection
        bbox_xywh, cls_conf, cls_ids, xy = self.yolo_model.detect(video_path, img, ori_img, vid_cap)
        # do tracking  # features: reid model output 512 dim features
        # outputs, features = self.deepsort_out.update(bbox_xywh, cls_conf, ori_img)
        outputs, features = self.deepsort.update(bbox_xywh, cls_conf, ori_img)

        # 1. 画黄线
        # yellow_line_out = self.draw_yellow_line_out(ori_img)
        p2 = [1500, 450]
        p1 = [1700, 1000]
```

```
            yellow_line_out = draw_yellow_line(p1, p2, ori_img)

            # 2. 统计人数
            for track in outputs:
                bbox = track[:4]
                track_id = track[-1]
                midpoint = tlbr_midpoint(bbox)
                origin_midpoint = (midpoint[0],
                                   ori_img.shape[0] - midpoint[1])  # get midpoint respective to bottom-left
                if track_id not in paths:
                    paths[track_id] = deque(maxlen=2)  # path 保存了每个 track 的最多两个帧的 midpoint
                    total_track = track_id
                paths[track_id].append(midpoint)
                previous_midpoint = paths[track_id][0]  # 此 track 前一帧的 midpoint
                origin_previous_midpoint = (previous_midpoint[0], ori_img.shape[0] - previous_midpoint[1])

                if intersect(midpoint, previous_midpoint, yellow_line_out[0], yellow_line_out[1]) \
                        and track_id not in already_counted:
                    total_counter += 1
                    last_track_id = track_id;  # 记录触线者的 ID
                    cv2.line(ori_img, yellow_line_out[0], yellow_line_out[1], (0, 0, 255), 1)  # 触碰线的情况下
画红线
                    already_counted.append(track_id)  # Set already counted for ID to true.
                    angle = vector_angle(origin_midpoint, origin_previous_midpoint)  # 计算角度，判断向上还是向下
走
                    if angle > 0: # 入店
                        up_count += 1
                    if angle < 0: # 出店
                        down_count += 1
                        # 出店的时候，把人物的图像抠出来------------- TODO: 该名称应该表示为入店时分配的 ID
                        cv2.line(ori_img, yellow_line_out[0], yellow_line_out[1], (0, 0, 0), 1)  # 消除线条
                        ROI_person = ori_img[int(bbox[1]):int(bbox[3]), int(bbox[0]):int(bbox[2])]
                        path = str('./runs/reid_output/exit/track_id-{}.jpg'.format(track_id))
                        self.makedir(path)
                        cv2.imwrite(path, ROI_person)

                if len(paths) > 50:
                    del paths[list(paths)[0]]
            # 3. 绘制人员
            person_cossim = cosine_similarity(features, self.query_feat)  # 计算 features 和 query_features 的余
弦相似度
            max_idx = np.argmax(person_cossim, axis=1)
            maximum = np.max(person_cossim, axis=1)
            max_idx[maximum < 0.6] = -1
            score = maximum
            reid_results = max_idx
            draw_person(ori_img, xy, reid_results, self.names)  # draw_person name
            # 4. 绘制统计信息
            ori_img = self.print_statistics_to_frame(down_count, ori_img, total_counter, total_track, up_count)
            if last_track_id >= 0:
```

```
            ori_img = self.print_newest_info(angle, last_track_id, ori_img)
        if len(outputs) > 0: # 只打印检测的框,
            bbox_tlwh = []
            bbox_xyxy = outputs[:, :4]
            identities = outputs[:, -1]
            ori_im = draw_boxes_and_text(ori_img, bbox_xyxy, identities)  # 给每个 detection 画框
            for bb_xyxy in bbox_xyxy:
                bbox_tlwh.append(self.deepsort._xyxy_to_tlwh(bb_xyxy))

        if self.args.display:
            cv2.imshow("Out camera", ori_img)
            if cv2.waitKey(1) & 0xFF == 27:
                break
    end_time = time_synchronized()
    self.logger.info("Index of frame: {} / "
                     "One Image spend time: {:.03f}s, "
                     "fps: {:.03f}, "
                     "tracks : {}, "
                     "detections : {}, "
                     "features of detections: {}"
                     .format(idx_frame
                             , end_time - start_time
                             , 1 / (end_time - start_time)
                             , bbox_xywh.shape[0]
                             , len(outputs)
                             , len(bbox_xywh)
                             , features.shape
                             )
                     )
    cv2.destroyAllWindows()

# *********************************************************************************************



def draw_yellow_line_in(self, ori_img):
    line = [(int(0.08 * ori_img.shape[1]), int(0.70 * ori_img.shape[0])),
            (int(0.6 * ori_img.shape[1]), int(0.45 * ori_img.shape[0]))]
    cv2.line(ori_img, line[0], line[1], (0, 255, 255), 1)
    return line

def draw_yellow_line_out(self, ori_img):
    line = [(0, int(0.42 * ori_img.shape[0])),
            (int(0.5 * ori_img.shape[1]), int(0.7 * ori_img.shape[0]))]
    cv2.line(ori_img, line[0], line[1], (0, 255, 255), 1)
    return line

def print_statistics_to_frame(self, down_count, ori_img, total_counter, total_track, up_count):
    label  =  "TOTAL:  {}  people  cross  the  yellow  line.  ({}  IN,  {}  OUT.)".format(str(total_counter),
str(up_count), str(down_count))
    t_size = get_size_with_pil(label, 15)  # 原: 25
```

```
        x1 = 20
        y1 = 850
        color = compute_color_for_labels(2)
        ori_img = put_text_to_cv2_img_with_pil(ori_img, label, (x1 + 5, y1 - t_size[1] - 2), (255, 0, 0))
        return ori_img

    def print_newest_info(self, angle, last_track_id, ori_img):
        current_time = int(time.time())
        localtime = time.localtime(current_time)
        dt = time.strftime('%Y-%m-%d %H:%M:%S', localtime)
        # ---------------------------------------
        label = "TIME: {} | Person №{} crossed yellow line. [{}]".format(dt, str(last_track_id), str("IN") if
angle >= 0 else str('OUT'))
        t_size = get_size_with_pil(label, 25)
        x1 = 20
        y1 = 900
        color = compute_color_for_labels(2)
        ori_img = put_text_to_cv2_img_with_pil(ori_img, label, (x1 + 5, y1 - t_size[1] - 2), (255, 0, 0))
        return ori_img




if __name__ == '__main__':
    # graphviz = GraphvizOutput()
    # graphviz.output_file = 'hierachy.png'
    #
    # with PyCallGraph(output=graphviz): # hierarchy graph
    # ------------------ main function -----------------
    #     main()
    # -------------------------------------------------------
    # print("[INFO] Finish output graphviz photo.")

    # initialize parameters
    args = parse_args()

    # initialize StrongSORT
    cfg = get_config()
    cfg.merge_from_file(args.config_deepsort)

    #
    monitor = TrafficMonitor(cfg, args, path_in=args.video_path, path_out=args.video_out_path)
    with torch.no_grad():
        monitor.demo()
```