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)  
 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()