基于RPI和OpenCV的多场景流量智能检测系统

Multi-scenario traffic intelligent detection system based on RPI and OpenCV

摘要

随着人工智能技术以及深度学习目标检测算法的发展，针对如今建立智慧城市的需求，本项目聚焦多类目标流量测算问题，完善系统架构，减轻了数据处理压力，加快数据计算和传输速度。该智能检测系统使用树莓派4B搭载摄像头，使用YOLOv5框架训练模型，搭建OpenCV处理图像目标检测架构实现如核酸检测等不同场景下的目标识别，并将相关数据上传云端服务器，通过PC端对识别数据进行进一步的处理，并将其可视化结果显示在网页端以供用户实时访问。

关键词： 树莓派　深度学习　YOLOv5目标识别　OpenCV图像处理 云服务器　Web开发

Abstract

With the development of artificial intelligence technology and deep learning target detection algorithm, the project focuses on the measurement of multi-target traffic, improves the system architecture, reduces the pressure of data processing, and speeds up the data calculation and transmission speed in view of the need to build a smart city. The intelligent detection system uses the camera of Raspberry Pi 4B, uses the YOLOv5 framework training model, builds the OpenCV processing image target detection architecture to realize target recognition in different scenarios such as nucleic acid detection, and uploads the relevant data to the cloud server, further processes the identification data through the PC, and displays the visualization results on the web for real-time access by users.

目录

一、项目背景与概述

新型冠状病毒肺炎(COVID-19)，是近百年人类遭遇的影响范围最广，规模最大的全球性大流行病。2020 年 1 月 30 日世界卫生组织宣布，已经将新型冠状病毒疫情列为国际关注的突发公共卫生事件。现如今，经过政府，社会各界的不懈努力，新冠肺炎疫情防控工作由应急状态转变为常态化防控，而在人群中开展新冠病毒核酸检测是落实防控措施中一项重要措施，有助于及早发现感染者以及密切接触者以及公共人群，对早诊早治、疫情防控和复工复产都具有重要意义。在排查感染者时，核酸检测至关重要，因此目前在全国多个城市已经实行了居民须保持核酸有效时间48小时的政策，并在城市内建立了多个核酸检测点。到核酸检测点进行核算检测已经成为了人民生活中不可或缺的一环。

在日常化核酸检测中，我们经常发现有些时间段是核酸检测高峰时段，这时核酸检测点人满为患；而有些时间核酸检测点却空无一人。由此，我们想到如果可以将各个核酸检测点的实时核酸检测人数在网络上发布，并提供一些时间规划的建议，这样人们可以更好地安排每个人自己核酸检测的时间，从而节约时间，提高核酸检测效率；对于社会来说，可以有效地整合社会资源，可以更好地引导人群，提高医疗资源的利用率，并且减少了大量人群聚集的场景，也可以减少病毒传播的风险。

在居民方面，通过实时动态更新数据，有利于合理安排核酸采样时间，提交整体效率。了解实时核酸采样人数，不仅有利于资源供给与分配，更有助于疫情防控指挥部的宏观调控。进一步，将每个核酸采样点的数据上升到城市云端服务器，依托大数据挖掘技术，建立相关平台，实现精准调控，确保本土疫情扩散最小化，，切实将人民生命安全放在第一位，有力保障经济社会秩序全面恢复，从而减少人力物力财力的开支。

我们本次的项目——基于 RPI 和 OpenCV 的多场景流量智能检测系统，就是以核酸检测的场景为出发点。同时我们意识到在许多场景下，人流量的检测将会起到非常重要的作用。例如景区中，有效地检测人流量对于景区管理、人流量的引导可以起到非常显著的效果；又比如食堂餐厅中，人流量的检测不仅可以评测此食堂餐厅的上座率，从而推断此食堂餐厅受顾客喜爱的程度，还可以对每个窗口进行细致的人流检测，从而发现最受欢迎的菜品，对食堂餐厅提升服务水平、改善菜品质量有着重要意义；还有在车站或者人流量较大的场景中，对人流量的检测可以更好地提供路线规划的建议，有效节约时间，并且提高公共资源的利用率;在城市道路路口，对于车辆和非机动车多目标检测对于城市交通管理有着重要意义，并且填补了市面上同时检测多目标产品的空白。此次项目适用的场景还有很多，此处不再一一列举。

总而言之，本次项目基于数字图像处理、边缘-云端、人工智能等多项技术，从日常生活中入手提高了系统之间的相互联系，使各项技术通信感知一体化，提高了信息的处理效率。本项目的应用场景广泛，可布置在核酸检测点、商圈、道路，公共交通枢纽、方舱医院、隔离场所等等设施中，通过云端整合经由边缘处理过的各类数据，立体、全面地把握各个检测目标流量动态。这对于建立智慧城市，完善公共管理，建立自动化管理系统有着重要意义。基于此，本项目具有广阔的应用前景和深刻的研究意义。

二、项目特色与相关项目调查

随着社会经济以及人工智能技术的发展，各类场景下的流量检测已经成为当下布局“智慧城市”其中热点问题之一，本项目所提出的系统属于“AI-未来技术”主题中“智能测量”领域，但是根据我们调查的相关专利结果来看，所提出能同时适用于多场景的流量检测系统很少，为此，该项目的提出能够很好地弥补这一空白。

具体调查情况如下，在此以其中两项和本系统较为相近的专利进行分析：

根据“一种路口人流量与车流量的实时检测装置及方法”（专利号：CN114530039A）[1]提出的路口人流量与车流量的实时检测装置中，通过使用激光雷达对路口等待的行人与路上的车辆进行检测扫描，收集完行人与车辆的数据后将数据输入算法模块中进行计算，再将计算后的数据输出到数据后台。与之相比，本系统避免使用昂贵的激光雷达，仅需通过树莓派接入摄像头接入结合YOLOv5等AI算法同样可以很好地实现实时人流量与车流量的检测，不仅可以减少成本，同时还能对于机动车与非机动车进行区分，从而也能更好地将流量进行分类统计。

在“一种智慧城市用人流量或车流量检测仪”（专利号：CN10896175A）[2]中使用的智慧城市用检测仪提出利用图像识别技术，通过摄像头对拍摄区域的人流量或车流量进行数据统计的一种检测仪，利用该检测仪可以在规定时间内统计出经过该检测仪的车辆数量，或人流数量，使得数据的统计操作更加便捷。在与该专利中提出的检测仪器对比，本项目提出搭载摄像头的树莓派多场景检测系统占用体积很小，极大地提高便携性与实用性。并且得益于以上性质，也使得该系统避免像上述专利提及只能固定于某一特定区域进行检测。

在相关文献的调查结果中，其中由张桂杰,郭泽旸,韩睿等人提出的基于YOLOV5的人车流量监测系统[3]与本项目中所提出的多场景流量智能检测系统有非常多的共通之处。在该文献中提出了一套基于YOLOv5目标检测技术,使用Java Server Page与ECharts相结合进行Web服务器端的开发,将数据存储在MySQL数据库中的人车流量监测系统，实现对行人流量及车辆流量的检测，并利用Python技术对检测结果进行可视化,直观展示各区域人流量信息,从而给人们的出行提供实时、动态的参考,从而更好地规划出行目的地及出行路线,减少等待时间,避免大量人车聚集。由马永杰，宋晓凤提出的基于YOLO和嵌入式系统的车流量检测[4]中提出了基于深度学习YOLO(You Only Look Once)的车流量检测算法。用YOLOv2检测道路上移动的目标，对检出目标中的车辆目标进行识别与筛选，设置感兴趣区域，在车辆目标经过感兴趣区域时计数，并用相关滤波器跟踪车辆，避免车辆重复计数。

与以上文献所提出的检测系统对比，本项目中提出基于树莓派4B嵌入式Linux开发，搭载摄像头进行实时监控，将所捕获到的画面信息利用目标检测算法YOLOv5结合跨平台计算机视觉库OpenCV目标检测架构实现多场景下的目标识别，可以实现更加轻量级的嵌入式架构，利于智慧城市的大范围部署，并且相较于上述方案成本大大减小。

本项目通过软硬件结合，采用多种数据处理技术，可以实现多场景、多目标的检测系统部署。其特点为灵活、轻量，将云端与边缘结合，提升了系统处理数据的速度与效率，大大提高数据处理资源的利用率，并且成本较低，操作简便，部署便利，边缘和云端的分层处理结构更有利于整个检测系统的管理和维护，且更便于系统的更新和优化。

图9显示的是最后处理结果。如图10所示，在树莓派中采用传统的YOLOv5和OpenCV处理图片的方法，处理一张640*480的图片用时需要17s左右的时间，在这种状态下实时测量并显示画面较为困难。

于是对方案进行改进，将模型进行简化优化之后，采用每间隔一段时间，例如10s拍摄照片并将此照片进行处理识别，通过深度学习算法后输出相应的结果，例如此刻摄像头拍摄照片范围内核酸检测场地内的人数或者道路上机动车/非机动车/行人的人数等。

测算出结果后，通过python程序调用socket库将数据传输到网站上，进行实时更新。具体原理会在网页原理部分详细介绍。

2. 进行优化

若想要进行处理速度优化，并且希望实现实时画面检测功能，此改进方案考虑简化YOLOv5模型，使用轻量级YOLOv5-lite框架训练模型，保证画面实时性，并将实时画面数据传输给云端。

通过上一部分的测试已经证明，使用实时画面检测并通过YOLOv5模型对每一帧画面进行检测，由于每帧图像处理速度较慢，则帧率（FPS）较小，其表现为画面会表现的十分卡顿，无法完成画面实时检测并显示的目标。

为解决这一问题，本系统首先采用轻量级YOLOv5-lite框架，但速度仍然较慢，故在树莓派配置runtimeonnx环境，试使用简化后的模型文件进行图片处理以及目标检测。OpenCV不适合用于搭建模型，通常使用其他框架训练模型。ONNX 作为通用的模型描述格式被众多框架支持，故这里使用 ONNX 作为模型保存格式。配置好环境后将 YOLOv5官方.pt格式模型文件转化成为.onnx格式模型文件，并通过.onnx格式模型进行推理处理图像，从而加快处理速度。目标是能使帧率控制在可以接受的程度，并可以实现实时检测。若处理速率还是较慢，可能是官方提供的模型检测目标过多导致网络模型较为复杂，大大增加了处理时间。所以考虑自己通过训练集训练单个或较少目标的模型，加快处理速度，减少处理时间，提高帧率，完成实时检测的功能。

如上图所示；在核酸检测场景下，两折线图分别表示两个核酸测量地点各段时间内核酸测量人数，每2个小时为一段记录该段时间内的核酸测量总人数。右上角可以将图片从折线图转换为柱状图，并有保存键可以保存图片下方则是实时人数，用于显示当前个站点核酸人数。其中数据会每隔20秒刷新一次，以便用户获得实时数据。

与传统核酸点通过实时传输视频的方式相比，该套智能测量系统通过WebSocket将树莓派中所采集和处理的数据直接传递给服务器中并展现在网页中出来，各个时间段的数据也会被保存在服务器中以供获取和调用。通过这种方法能减少了网络负载与流量负担，数字也更能直观体现出各个场景的具体情况，并且通过记录各个时间段的人数，可以更方便用户做出合理的时间安排避免时间的浪费。

十二、实现功能

1. 摄像头监控视频图像对预设目标捕获识别与流量统计
2. 多场景模式切换，如核酸检测点（检测人流量），交通道路（检测车流量）等
3. 检测数据上传服务器与显示数据图表
4. 通过网页访问与查看实时流量数据
5. 红外遥控控制与LED模式切换显示

附录一：源代码及程序清单

test_video.py(用于实时视频检测)

```python
import cv2
import numpy as np
import onnxruntime as ort
import time
import random


def plot_one_box(x, img, color=None, label=None, line_thickness=None):
    """
    description: Plots one bounding box on image img,
                 this function comes from YOLOv5 project.
    param:
        x:      a box likes [x1,y1,x2,y2]
        img:    a OpenCV image object
        color:  color to draw rectangle, such as (0,255,0)
        label:  str
        line_thickness: int
    return:
        no return
    """
    tl = (
            line_thickness or round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1
    )  # line/font thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
```

```
    cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_AA)
    if label:
        tf = max(tl - 1, 1)  # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(img, c1, c2, color, -1, cv2.LINE_AA)  # filLED
        cv2.putText(
            img,
            label,
            (c1[0], c1[1] - 2),
            0,
            tl / 3,
            [225, 255, 255],
            thickness=tf,
            lineType=cv2.LINE_AA,
        )


def _make_grid(nx, ny):
    xv, yv = np.meshgrid(np.arange(ny), np.arange(nx))
    return np.stack((xv, yv), 2).reshape((-1, 2)).astype(np.float32)


def cal_outputs(outs, nl, na, model_w, model_h, anchor_grid, stride):
    row_ind = 0
    grid = [np.zeros(1)] * nl
    for i in range(nl):
        h, w = int(model_w / stride[i]), int(model_h / stride[i])
        length = int(na * h * w)
        if grid[i].shape[2:4] != (h, w):
            grid[i] = _make_grid(w, h)

        outs[row_ind:row_ind + length, 0:2] = (outs[row_ind:row_ind + length, 0:2] * 2. - 0.5 + np.tile(
            grid[i], (na, 1))) * int(stride[i])
        outs[row_ind:row_ind + length, 2:4] = (outs[row_ind:row_ind + length, 2:4] * 2) ** 2 * np.repeat(
            anchor_grid[i], h * w, axis=0)
        row_ind += length
    return outs


def post_process_OpenCV(outputs, model_h, model_w, img_h, img_w, thred_nms, thred_cond):
    conf = outputs[:, 4].tolist()
    c_x = outputs[:, 0] / model_w * img_w
    c_y = outputs[:, 1] / model_h * img_h
    w = outputs[:, 2] / model_w * img_w
    h = outputs[:, 2] / model_h * img_h
```

```
    p_cls = outputs[:, 5:]
    if len(p_cls.shape) == 1:
        p_cls = np.expand_dims(p_cls, 1)
    cls_id = np.argmax(p_cls, axis=1)

    p_x1 = np.expand_dims(c_x - w / 2, -1)
    p_y1 = np.expand_dims(c_y - h / 2, -1)
    p_x2 = np.expand_dims(c_x + w / 2, -1)
    p_y2 = np.expand_dims(c_y + h / 2, -1)
    areas = np.concatenate((p_x1, p_y1, p_x2, p_y2), axis=-1)

    areas = areas.tolist()
    ids = cv2.dnn.NMSBoxes(areas, conf, thred_cond, thred_nms)
    if len(ids) > 0:
        return np.array(areas)[ids], np.array(conf)[ids], cls_id[ids]
    else:
        return [], [], []


def infer_img(img0, net, model_h, model_w, nl, na, stride, anchor_grid, thred_nms=0.4, thred_cond=0.5):
    # 图像预处理
    img = cv2.resize(img0, [model_w, model_h], interpolation=cv2.INTER_AREA)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32) / 255.0
    blob = np.expand_dims(np.transpose(img, (2, 0, 1)), axis=0)

    # 模型推理
    outs = net.run(None, {net.get_inputs()[0].name: blob})[0].squeeze(axis=0)

    # 输出坐标矫正
    outs = cal_outputs(outs, nl, na, model_w, model_h, anchor_grid, stride)

    # 检测框计算
    img_h, img_w, _ = np.shape(img0)
    boxes, confs, ids = post_process_OpenCV(outs, model_h, model_w, img_h, img_w, thred_nms, thred_cond)

    return boxes, confs, ids


if __name__ == "__main__":
    # 模型加载
    model_pb_path = "YOLOv5s.onnx"
    so = ort.SessionOptions()
    net = ort.InferenceSession(model_pb_path, so)

    # 标签字典
dic_labels = {0: 'person',
```

```
            1: "bicycle",
            2: "car",
            3: "motorbike",
            4: "aeroplane",
            5: "bus",
            6: "train",
            7: "truck",
            8: "boat",
            9: "traffic light",
            10: "fire hydrant",
            11: "stop sign",
            12: "parking meter",
            13: "bench",
            14: "bird",
            15: "cat",
            16: "dog",
            17: "horse",
            18: "sheep",
            19: "cow",
            20: "elephant",
            21: "bear",
            22: "zebra",
            23: "giraffe",
            24: "backpack",
            25: "umbrella",
            26: "handbag",
            27: "tie",
            28: "suitcase",
            29: "frisbee",
            30: "skis",
            31: "snowboard",
            32: "sports ball",
            33: "kite",
            34: "baseball bat",
            35: "baseball glove",
            36: "skateboard",
            37: "surfboard",
            38: "tennis racket",
            39: "bottle",
            40: "wine glass",
            41: "cup",
            42: "fork",
            43: "knife",
            44: "spoon",
            45: "bowl",
            46: "banana",
            47: "apple",
            48: "sandwich",
            49: "orange",
```

```
                50: "broccoli",
                51: "carrot",
                52: "hot dog",
                53: "pizza",
                54: "donut",
                55: "cake",
                56: "chair",
                57: "sofa",
                58: "pottedplant",
                59: "bed",
                60: "diningtable",
                61: "toilet",
                62: "tvmonitor",
                63: "laptop",
                64: "mouse",
                65: "remote",
                66: "keyboard",
                67: "cell phone",
                68: "microwave",
                69: "oven",
                70: "toaster",
                71: "sink",
                72: "refrigerator",
                73: "book",
                74: "clock",
                75: "vase",
                76: "scissors",
                77: "teddy bear",
                78: "hair drier",
                79: "toothbrush", }

# 模型参数
model_h = 640
model_w = 640
nl = 3
na = 3
stride = [8., 16., 32.]
anchors = [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198
    , 373, 326]]
anchor_grid = np.asarray(anchors, dtype=np.float32).reshape(nl, -1, 2)

video = 0
cap = cv2.VideoCapture(video)
flag_det = False
while True:
    success, img0 = cap.read()
    if success:

        if flag_det:
```

```
            t1 = time.time()
            det_boxes, scores, ids = infer_img(img0, net, model_h, model_w, nl, na, stride, anchor_grid,
                                               thred_nms=0.4, thred_cond=0.5)
            t2 = time.time()

            print(scores)
            for id in ids:
                print(dic_labels[id], end=" ")

            for box, score, id in zip(det_boxes, scores, ids):

                label = '%s:%.2f' % (dic_labels[id], score)

                plot_one_box(box.astype(np.int16), img0, color=(255, 0, 0), label=label, line_thickness=None)

            str_FPS = "FPS: %.2f" % (1. / (t2 - t1))

            cv2.putText(img0, str_FPS, (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 3)

        cv2.imshow("video", img0)
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):

        break
    elif key & 0xFF == ord('s'):
        flag_det = not flag_det
        print(flag_det)

cap.release()
```

test_one_img.py(用于捕获图片检测)

```
import cv2
import numpy as np

import onnxruntime as ort
import math
import time

def plot_one_box(x, img, color=None, label=None, line_thickness=None):
    """
    description: Plots one bounding box on image img,
                 this function comes from YOLOv5 project.
    param:
        x:      a box likes [x1,y1,x2,y2]
        img:    a OpenCV image object
        color:  color to draw rectangle, such as (0,255,0)
```

```
        label:  str
        line_thickness: int
    return:
        no return
    """
    tl = (
        line_thickness or round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1
    )  # line/font thickness
    color = color or [random.randint(0, 255) for _ in range(3)]
    c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
    cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_AA)
    if label:
        tf = max(tl - 1, 1)  # font thickness
        t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
        c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
        cv2.rectangle(img, c1, c2, color, -1, cv2.LINE_AA)  # filLED
        cv2.putText(
            img,
            label,
            (c1[0], c1[1] - 2),
            0,
            tl / 3,
            [225, 255, 255],
            thickness=tf,
            lineType=cv2.LINE_AA,
        )

def _make_grid( nx, ny):
        xv, yv = np.meshgrid(np.arange(ny), np.arange(nx))
        return np.stack((xv, yv), 2).reshape((-1, 2)).astype(np.float32)

def cal_outputs(outs,nl,na,model_w,model_h,anchor_grid,stride):

    row_ind = 0
    grid = [np.zeros(1)] * nl
    for i in range(nl):
        h, w = int(model_w/ stride[i]), int(model_h / stride[i])
        length = int(na * h * w)
        if grid[i].shape[2:4] != (h, w):
            grid[i] = _make_grid(w, h)

        outs[row_ind:row_ind + length, 0:2] = (outs[row_ind:row_ind + length, 0:2] * 2. - 0.5 + np.tile(
            grid[i], (na, 1))) * int(stride[i])
        outs[row_ind:row_ind + length, 2:4] = (outs[row_ind:row_ind + length, 2:4] * 2) ** 2 * np.repeat(
            anchor_grid[i], h * w, axis=0)
        row_ind += length
    return outs
```

```
63.
64.
65.
66. def post_process_OpenCV(outputs,model_h,model_w,img_h,img_w,thred_nms,thred_cond):
67.     conf = outputs[:,4].tolist()
68.     c_x = outputs[:,0]/model_w*img_w
69.     c_y = outputs[:,1]/model_h*img_h
70.     w  = outputs[:,2]/model_w*img_w
71.     h  = outputs[:,2]/model_h*img_h
72.     p_cls = outputs[:,5:]
73.     if len(p_cls.shape)==1:
74.         p_cls = np.expand_dims(p_cls,1)
75.     cls_id = np.argmax(p_cls,axis=1)
76.
77.     p_x1 = np.expand_dims(c_x-w/2,-1)
78.     p_y1 = np.expand_dims(c_y-h/2,-1)
79.     p_x2 = np.expand_dims(c_x+w/2,-1)
80.     p_y2 = np.expand_dims(c_y+h/2,-1)
81.     areas = np.concatenate((p_x1,p_y1,p_x2,p_y2),axis=-1)
82.
83.     areas = areas.tolist()
84.     ids = cv2.dnn.NMSBoxes(areas,conf,thred_cond,thred_nms)
85.     return  np.array(areas)[ids],np.array(conf)[ids],cls_id[ids]
86.
87. def infer_img(img0,net,model_h,model_w,nl,na,stride,anchor_grid,thred_nms=0.4,thred
    _cond=0.5):
88.     # 图像预处理
89.     img = cv2.resize(img0, [model_w,model_h], interpolation=cv2.INTER_AREA)
90.     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
91.     img = img.astype(np.float32) / 255.0
92.     blob = np.expand_dims(np.transpose(img, (2, 0, 1)), axis=0)
93.
94.     # 模型推理
95.     outs = net.run(None, {net.get_inputs()[0].name: blob})[0].squeeze(axis=0)
96.
97.     # 输出坐标矫正
98.     outs = cal_outputs(outs,nl,na,model_w,model_h,anchor_grid,stride)
99.
100.     # 检测框计算
101.     img_h,img_w,_ = np.shape(img0)
102.     boxes,confs,ids = post_process_OpenCV(outs,model_h,model_w,img_h,img_w,thred_n
    ms,thred_cond)
103.
104.     return  boxes,confs,ids
105.
106.
107.
108.
109. if __name__ == "__main__":
```

```
110.
111.     # 模型加载
112.     model_pb_path = "YOLOv5s.onnx"
113.     so = ort.SessionOptions()
114.     net = ort.InferenceSession(model_pb_path, so)
115.
116.     # 标签字典
117.     dic_labels= {0:'LED',
118.             1:'buzzer',
119.             2:'teeth'}
120.
121.     # 模型参数
122.     model_h = 320
123.     model_w = 320
124.     nl = 3
125.     na = 3
126.     stride=[8.,16.,32.]
127.     anchors = [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156,
     198, 373, 326]]
128.     anchor_grid = np.asarray(anchors, dtype=np.float32).reshape(nl, -1, 2)
129.
130.     # 进行推理
131.     img0 = cv2.imread('3.jpg')
132.     t1 = time.time()
133.     det_boxes,scores,ids = infer_img(img0,net,model_h,model_w,nl,na,stride,anchor_
    grid,thred_nms=0.4,thred_cond=0.5)
134.     t2 = time.time()
135.     print("%.2f"%(t2-t1))
136.     # 结果绘图
137.     for box,score,id in zip(det_boxes,scores,ids):
138.         label = '%s:%.2f'%(dic_labels[id],score)
139.
140.         plot_one_box(box.astype(np.int), img0, color=(255,0,0), label=label, line_
    thickness=None)
141.     cv2.imshow('img',img0)
142.
143.     cv2.waitKey(0)
144.
145.     # img = cv2.resize(img0, [320,320], interpolation=cv2.INTER_AREA)
146.
147.     # img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
148.     # img = img.astype(np.float32) / 255.0
149.     # blob = np.expand_dims(np.transpose(img, (2, 0, 1)), axis=0)
150.
151.     # outs = net.run(None, {net.get_inputs()[0].name: blob})[0].squeeze(axis=0)
152.
153.     # nl = 3
154.     # na = 3
155.     # stride=[8.,16.,32.]
```

```
    # anchors = [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]]
    # anchor_grid = np.asarray(anchors, dtype=np.float32).reshape(nl, -1, 2)
    # model_w = 320
    # model_h = 320
    # outs = cal_outputs(outs,nl,na,model_w,model_h,anchor_grid,stride)

    # print(outs)
    # boxes,confs,ids = post_process_OpenCV(outs,model_h,model_w,img_h=480,img_w=640,thred_nms=0.4,thred_cond=0.5)
    # print(boxes)
```

附录二 作品操作说明

将摄像头固定在所需测量的场景中，根据需要选择合适的场景模式，在树莓派上运行程序后即可通过访问网页端获取处理后的可视化结果。

附录三 资料与文献参考

[1] 浙江梧斯源通信科技股份有限公司. 一种路口人流量与车流量的实时检测装置及方法:CN202210098688.6[P]. 2022-05-24.

[2] 陕西普顿信息科技有限责任公司. 一种智慧城市用人流量或车流量检测仪:CN201820915124.6[P]. 2019-01-01.

[3] 张桂杰,郭泽旸,韩睿,等. 基于YOLOv5的人车流量监测系统设计与实现[J]. 吉林师范大学学报（自然科学版）,2021,42(4):118-126. DOI:10.16862/j.cnki.issn1674-3873.2021.04.021.

[4]马永杰,宋晓凤. 基于YOLO和嵌入式系统的车流量检测[J]. 液晶与显示,2019,34(6):613-618. DOI:10.3788/YJYXS20193406.0613.

原创性声明

本参赛团队郑重声明：所呈交的参赛作品《基于RPI和OpenCV的多场景流量智能检测系统》，是本参赛团队在教师的指导下，独立进行研究工作所取得的真实成果。除文中已注明引用的内容外，参赛作品中不含任何其他个人或集体已经发表或撰写过的作品成果。对本人参赛作品的创作做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：_____

日期：_____