成绩	（采用四级记分制）

西北大学

本科毕业论文（设计）

题目： 轻量化目标检测系统的设计与实现

学生姓名 薄劲阳

学　　号 2020115025

指导教师 卜起荣

院　　系 信息科学与技术学院

专　　业 计算机科学与技术

年　　级 2020

教务处制

二〇二四年六月

诚信声明

本人郑重声明：本人所呈交的毕业论文（设计），是在导师的指导下独立进行研究所取得的成果。毕业论文（设计）中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或在网上发表的论文。

特此声明。

论文作者签名：_____

日　期：　2024 年 5 月 5 日

摘　要

目标检测是计算机视觉领域的核心任务之一。基于神经网络的目标检测算法以其卓越的准确性在计算机视觉领域中表现突出，但它们通常具有较大的模型大小和高计算复杂度，导致在资源受限的环境下难以应用。本文研究 YOLO 系列算法，选用多个模型实现目标检测。统计各个模型的 FPS 性能。以下是本文的概述：

（1）采用了 YOLO 系列的算法来执行目标检测任务。训练的模型有 YOLOv5、YOLOv5-lite、YOLOv7。对所选择的模型进行了对比实验，得出轻量化效果最好的模型。

（2）基于 PyQt 开发轻量化目标检测系统软件，使用三种目标检测模型，实现对数据集的目标检测。

（3）将开发好的平台部署在树莓派上，通过比对不同模型在树莓派上的检测速度，对比比较模型轻量化的特点。

通过进行上述研究，我们得出来 YOLOv5-lite 的轻量化效果最好，并分析了 YOLOv5-lite 轻量化效果好的原因。最终搭建了一个部署在树莓派上的简单目标检测平台。

关键词：YOLO 目标检测 轻量化 树莓派

Abstract

Object detection plays a vital role in the domain of computer vision. Neural network-based object detection algorithms have shown excellent performance in terms of accuracy, but they often suffer from large model sizes and high computational complexity, making it difficult to apply them in resource-constrained environments. This thesis is based on the YOLO series of algorithms and employs multiple models for object detection. It analyzes the FPS performance of each model and discusses the reasons for their lightweight nature. The main content of this paper is as follows:

(1) In this thesis,we use the YOLO series of algorithms for object detection. The trained models include YOLOv5, YOLOv5-lite, and YOLOv7, which are ultimately converted into ONNX model files for model import and integration.

(2) A lightweight object detection system platform is developed using PyQt, employing three object detection models to perform object detection on a dataset.

(3) The developed platform is run on a Raspberry Pi, and the detection speeds of different models on the Raspberry Pi are compared to highlight the characteristics of model lightweightness.

Through the aforementioned research, we have concluded that YOLOv5-lite achieves the best lightweight performance, and we have analyzed the reasons behind YOLOv5-lite's excellent lightweight effect. Ultimately, we have built a simple object detection platform deployed on a Raspberry Pi.

Keywords： YOLO; Object Detection; Lightweight; Raspberry Pi

目录

1 绪论

1.1 选题背景及意义

1.1.1 目标检测的定义

目标检测是计算机视觉领域的一项核心任务。目标检测就是从图像中定位感兴趣的对象。主要有物体识别和物体定位两个任务。物体识别负责判断输入的图像中是否包含所需物体(object)，物体定位则负责表示目标物体的位置，并用外接矩形框定位[1]。

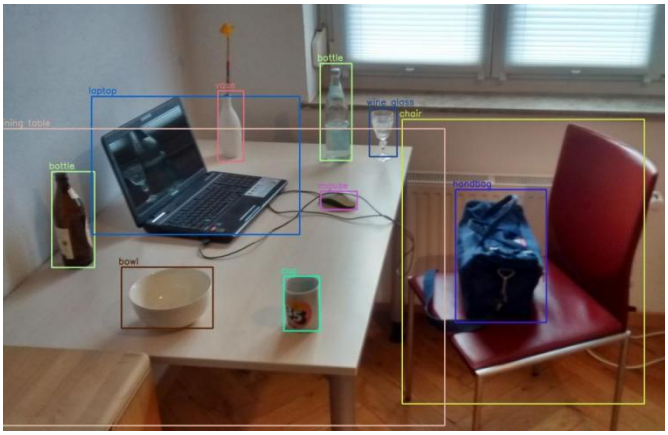


图 1-1 目标检测

1.1.2 目标检测算法的分类

目标检测算法根据特征的提取方式，分为传统目标检测算法和基于深度学习的目标检测算法[2]。

在传统目标检测算法中，特征提取通常依赖于手工设计的方法。其执行流程主要为选取感兴趣的、包含物体的区域，对该区域进行特征提取，对提取的特征进行检测分类[2]。例如方向梯度直方图 HOG(Histogram of Oriented Gradient, HOG)[3]。

深度学习驱动的目标检测算法通过卷积神经网络（CNN）实现，能够自动地从丰富的数据集中学习到特征，免除了人工设计特征提取机制的需要[24]。根据目标检测方式，分为两步式和单步式目标识别算法。两步式目标检测算法首先识别出可能包含目标的候选区域，随后利用卷积神经网络（CNN）对这些区域进行分类和边界框回归，以精确识别和定位目标物体

[26]。常见的双阶段目标检测算法有 Fast R-CNN[4]等。单步式目标检测算法在单一的网络前向传播过程中同时完成特征提取、类别概率预测和边界框位置坐标的预测[26]。典型代表有 YOLO(You Only Look Once)[5]。

1.1.3 目标检测的应用

目标检测在计算机视觉领域，有着很广泛的应用。在视频监控领域，目标检测用于实时识别和追踪视频中的行人、车辆、动物等，对于安全监控和交通管理至关重要。在医疗领域，目标检测可以帮助自动识别和监测 X 光、CT 扫描或 MRI 图像中的肿瘤、器官和其他医学特征。在安全系统和社交媒体应用中，目标检测用于快速定位和识别人脸，用于身份验证和个性化内容推荐。

1.1.4 轻量化目标检测的特点

传统的目标检测算法通常要求较大的模型大小和高计算复杂度，这会对计算和存储资源造成较大的负担。轻量化目标检测算法通过优化网络结构、减少参数数量以及采用高效的计算方式，可以大幅降低算法对资源的需求。轻量化目标检测算法如 Light-Head R-CNN[7]、Tiny-YOLO[8]等算法有效提高了轻量化目标检测算法的性能水平[2]。

1.2 国内外研究现状

1.2.1 目标检测技术现状

目标检测目前的检测算法包括 R-CNN[9]、YOLO 系列以及变种 Fast R-CNN[10]、Faster R-CNN[11]、Mask R-CNN[12]、SSD[13]。

R-CNN[9]由 Ross Girshick 等人于 2014 年提出。R-CNN[9]是一个两步式的目标检测框架，首先识别出潜在的目标区域，随后通过卷积神经网络（CNN）来提取这些区域的特征，并进行分类处理。

2015 年 Joseph Redmon[5]等人提出 YOLO 算法。YOLO（You Only Look Once）是一种单

步式目标检测算法，它能够在单一的网络前向传播过程中，高效地提取图像特征，并据此直接预测物体的类别概率及其在图像中的精确位置。目前最新的 YOLO 已经发展到了 YOLO v9[14]。

2018 年 Liu[13]等人提出了 SSD[13]算法。SSD[13]是一个单步式目标检测框架，它使用默认框(先验框)来预测目标的类别和位置。

1.2.2 轻量化目标检测技术现状

YOLO[5]、R-CNN[9]、SSD[13]系列的变种是轻量化目标检测算法的代表算法。

在 2015 年，Joseph Redmon[5]等研究者推出了著名的目标检测算法 YOLO v1，并同时开发了其轻量级版本 Tiny-YOLO v1。相较于标准版的 YOLO v1，Tiny-YOLO v1 通过将卷积层数量从 24 层精简至 9 层，实现了模型的轻量化。

2016 年，Hong 等人基于 Faster R-CNN，提出了轻量化目标检测算法 PVANet[9]。通过将 CReLU、Inception 与残差结构相结合，并参考 HyperNet 网络思想来构建特征提取网络[6]。

2018 年，Pedoeem[16]等人基于 YOLO 系列算法，提出了改进的 YOLO-LITE 算法。该算法在 CPU 上运行，其仅采用了 7 层卷积层。另外，Zhang[17]等人，提出了 MobileNet-SSD 算法。该算法使用 MobileNet v1 网络替换了 SSD 中的网络，提高了模型的目标检测速度[6]。

1.3 本文主要研究内容和组织结构

本文基于 YOLO 系列目标检测算法，进行比对实验，选择出最优模型。另外还搭建了一个目标检测软件，支持不同目标检测模型的运行。最终程序运行在树莓派上。

以下是本文的主要内容安排：

第一章绪论部分。首先定义了目标检测算法，接着对算法的不同类别进行了分类，并探讨了其在多个领域的应用潜力，以及轻量化目标检测算法的独特优势。此外，本章还综合评述了国内外在目标检测算法，尤其是轻量化目标检测算法领域的研究进展。最后，对本文的

研究内容和整体结构进行了概述。

第二章深入研究了 YOLO 目标检测算法，首先介绍了 YOLO 算法的核心流程，随后详细描述了 YOLOv5、YOLOv5-Lite 以及 YOLOv7 系列算法的网络架构设计。

第三章在第二章研究的算法基础上，开展了一系列对比实验。选用特定的数据集进行训练，并分别在计算机和树莓派平台上进行运行，检测其 FPS，并且分析了实验结果及产生该结果的背后原因。

第四章构建了目标检测系统，该系统结合第二章所提出的算法，完成了系统的总体结构设计和功能模块设计以及系统实现和系统测试，并最终部署到树莓派上，检测了不同模型的性能。

第五章对本文的研究内容进行了总结，并回顾了研究过程中遇到的问题，同时提出了未来研究的潜在改进方向。

2 YOLO 目标检测算法研究

2.1 YOLO 基本流程

YOLO 是单阶段的目标检测算法，图片输入后，通过单次前向推理过程，该方法能够同时预测出边界框的位置及其对应的类别概率。YOLO 算法的核心理念在于将 448x448 像素的图像输入神经网络，随后网络直接产生一个的三维张量。这个张量中包含了物体的定位信息和它们所属的类别信息。继而，通过执行后处理步骤，包括应用非极大值抑制（NMS）算法，从而精炼预测结果，最终确定图像中物体的具体位置和类型。流程如图 2-1 所示。

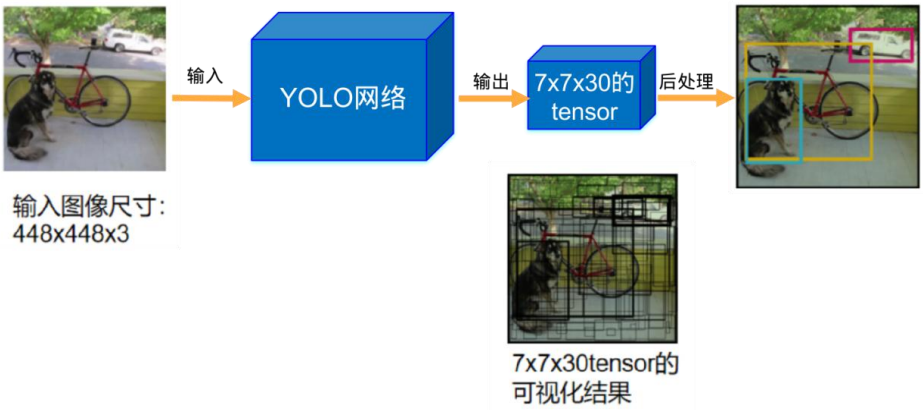


图 2-1 YOLO 目标检测流程图

算法执行的第一个步骤是将 448x448 的 RGB 三通道图像输入到 YOLO 主干网络中，得到对应的 7x7x30 tensor 张量。不同系列的 YOLO 算法，因为其不同的功能，对应不同的网络结构。具体细节见后文所示。

输入图像经过网络之后，得到了 7x7x30 的 tensor。7x7 表示输入图像被划分成 49 个单元格（grid cell），如图 2-2 所示，两个边界框（bounding box）会由一个单元格生成。训练集中会用方框标注出物体，人工标注的方框称为 ground truth。ground truth 的中心点落到那个 grid cell 中，就由该 grid cell 的两个预测框中的与 ground truth IOU 最大的那个预测框负责预测该目标。

图 2-2 grid cell

IOU 是指两个方框的交并比，即两个方框的交集所占面积与两个方框的并集所占面积的比值，用来衡量两个方框的重叠程度。计算公式见公式 2.1，可视化结果如图 2-3 所示。

$$IOU = \frac{S_{A \cap B}}{S_{A \cup B}} \tag{2.1}$$

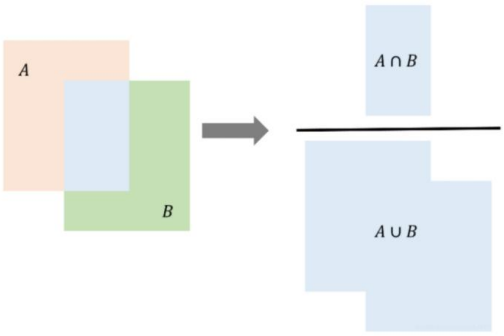


图 2-3 IOU 计算公式可视化

7x7x30 中的 30 是指 5x2+20。如上文描述，一个 grid cell 会产生两个 bounding box 预测框，一个预测框描述需要 5 个参数。分别是 x、y、w、h、c。x，y 是指 bounding box 预测框的中心坐标；w 和 h 是指该 bound box 的宽和高；c 代表置信度，表示该 bounding box 中含有目标的概率。Pr(Object) 非 0 即 1，若 ground truth 的中心点落入该 grid cell 中，则 Pr(Object)=1，否则 Pr(Object)=0。所以，综上所述，一个 grid cell 需要 5x2 个参数。20 是指不同类别的条件概率，即在该 grid cell 包含目标的条件下，该目标是某种类别的概率（VOC 数据集有 20 个类别）。

$$c = \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} \tag{2.2}$$

7x7x30 tensor 将 bounding box 预测框可视化之后，结果如图 2-4 所示。其中框线粗细表示置信度 c 的大小。为了精炼预测结果，将采用非极大值抑制（NMS）策略来消除冗余的边界框。同时，还需筛除那些置信度不高以及有重复预测的边界框。

图 2-4 预测框可视化

NMS 的具体操作流程是：首先，将 7x7x30 的张量按照 grid cell 一列一列取出。然后，使用每个预测框的置信度 c 乘以 20 个不同类别的条件概率，以此来计算对应于这 20 个类别的全概率。对于每个 grid cell，由于生成了两个预测框，因此可以得到两个 20 维的全概率向量。所以 7x7 个 gird 总共可以得到 98 个 20 维全概率向量。其构成了一个 98x20 的矩阵，矩阵的每一行代表 98 个 bounding box 对某一类别的全概率。如图 2-5 所示。

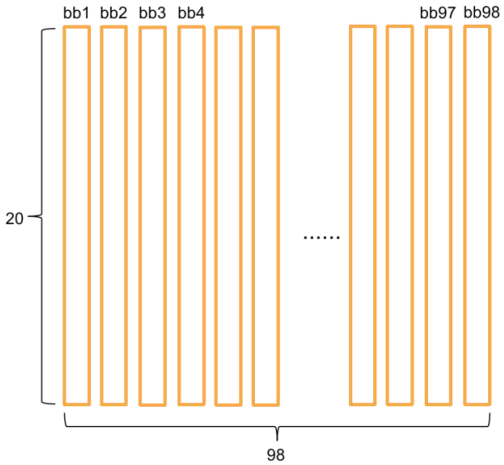


图 2-5 98x20 矩阵

对矩阵按行进行处理，首先将置信度小于某个具体阈值的数全部置为 0。然后按照从大

到小的顺序进行排序。然后从排序完的第一个非 0 元素开始，用该元素 bounding box 对应的框与剩下的非零元素的 bounding box 对应的框的 IOU 进行比较，若 IOU 大于某一个阈值（如 0.5），则把置信度小的元素置为 0。这一步是为了去除重复检测的预测框。最终处理结果如图 2-6 所示。

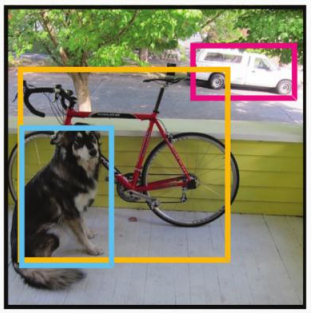


图 2-6 NMS 处理后的结果

2.2 YOLOv5、YOLOv5-lite、YOLOv7 网络结构

本篇论文主要使用的是 Yolov5s、YOLOv5-lite、YOLOv7 模型。Backbone、Neck、Head 三个部分构成了 YOLOv5s 模型。在目标检测框架中，Backbone 网络扮演着核心角色，负责从输入图像中抽取关键特征。继而，Neck 网络承担着融合多尺度特征图的任务，对这些特征进行进一步的提炼和增强。基于 Neck 网络的输出，Head 网络进行最终的物体分类，并预测物体在图像中的定位，细节如图 2-7 所示。

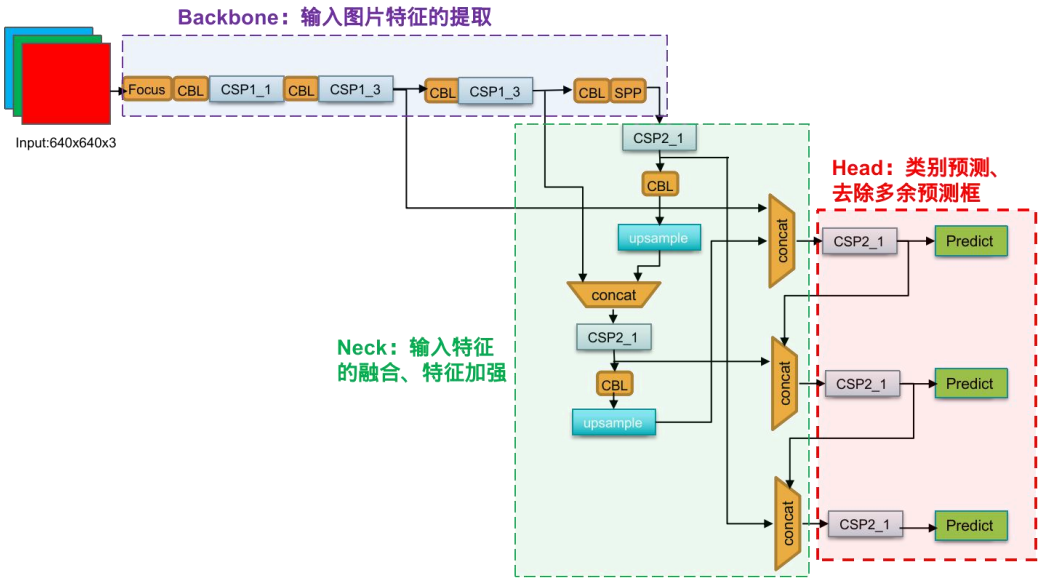


图 2-7 YOLOv5s 网络结构

Backbone 结构主要由 Focus、CBL 以及 CSP1_X 组成。Focus 模块负责对图像进行切片，将高维图像转换为低维图像。CBL 模块是由 Conv 卷积模块、BatchNorm 归一化处理模块、Leaky ReLU 激活函数模块组成。CPS1_X 由 CBL、Res Unit、Conv 构成。模块结构图如 2-8 所示。

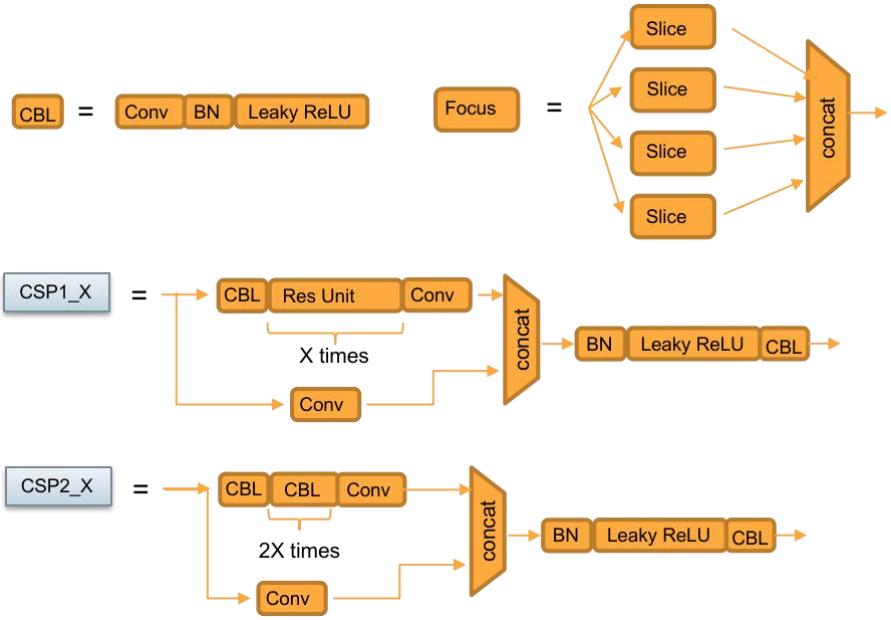


图 2-8 CBL、Focus、CSP 结构

为了获得更全面的特征信息，需要通过 Neck 结构融合不同网络层级的特征图，因为深入进行特征提取可能会导致图像的局部细节信息丢失。Neck 结构由 CSP2_X、Upsample、Concat 模块组成。Upsample 上采样模块用于增加数据的空间维度（即图像的高度和宽度）。Concat 拼接模块用于将 Backbone 结构中输出的特征图和上采样 UpSample 后的图拼接起来。

Head 根据 Neck 层处理后的特征，输出目标类别，并对候选框的位置进行修正。预测框 Bounding Box 的损失函数采用 CIoU Loss[18]，分类损失和置信度损失函数采用 BCE Loss。

YOLOv5-lite 相对于 YOLOv5s，在 Backbone 结构中去除了 Focus 层。另外在提取特征模块，使用 ShuffleNet[20]替代 CSP 模块。相比于 CSP 模块，ShuffleNet 减少了缓存的使用，加快了运行速度。另外在 Head 结构引入了通道剪枝操作。YOLOv5-lite 的网络结构如图 2-9 所示。

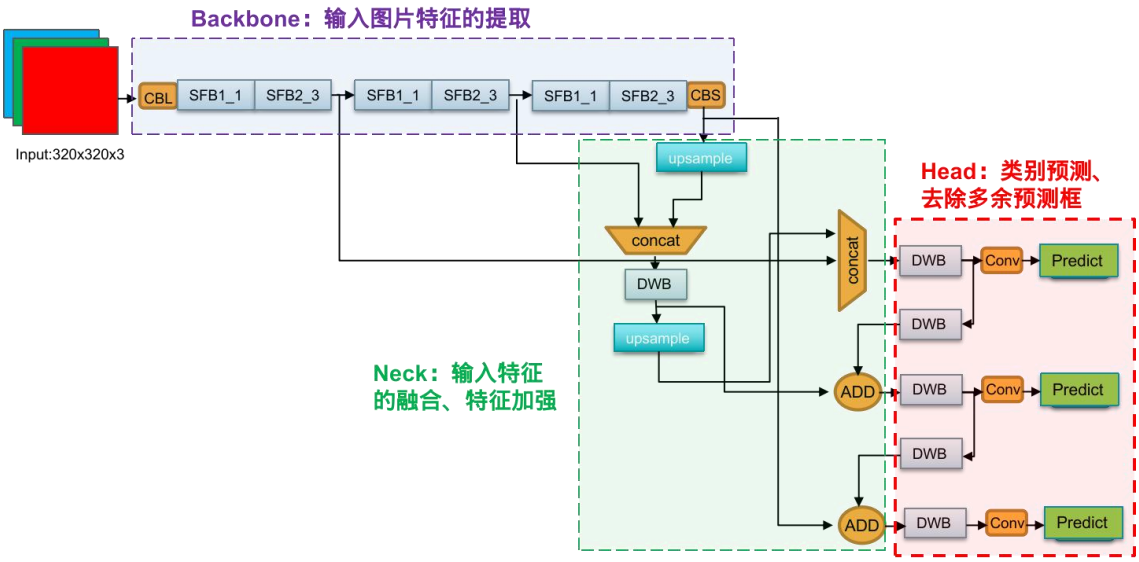


图 2-9 YOLOv5-lite 网络结构

SFB、DWB 模块如图 2-10 所示。

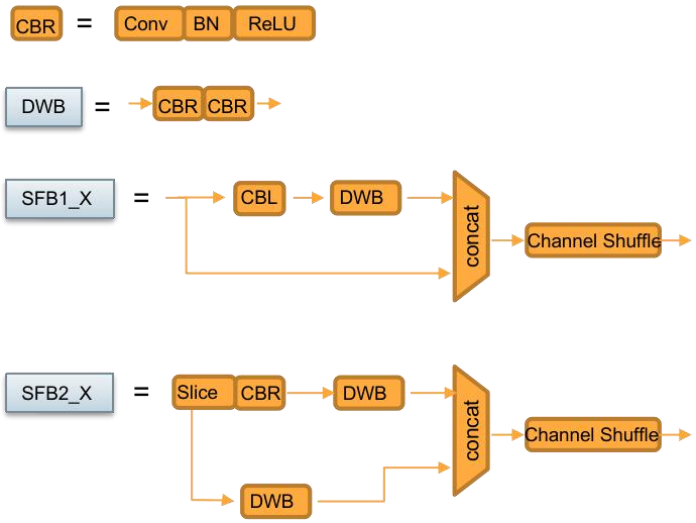


图 2-10 SFB、DWB 结构

YOLOv7 主要由三个网络构成：Backbone、Neck 和 Head，如图 2-11 所示。Backbone 网络承担着从输入图像中抽取特征的任务，生成三个不同层次的特征映射。利用这三个特征映射，Neck 网络执行特征融合，以提升特征的表达力度，并进一步深化特征的提取。随后，Head 网络依据 Neck 网络处理后的特征进行目标物体的类别判定和位置预测。相比于 YOLOv5、YOLOv5-lite，YOLOv7 的 Backbone 结构首先采用四个连续的 CBS，使用 ELAN[21]模块代替 CSP 模块。另外在 Head 结构中，使用了 ELAN-H、MP2[21]模块。

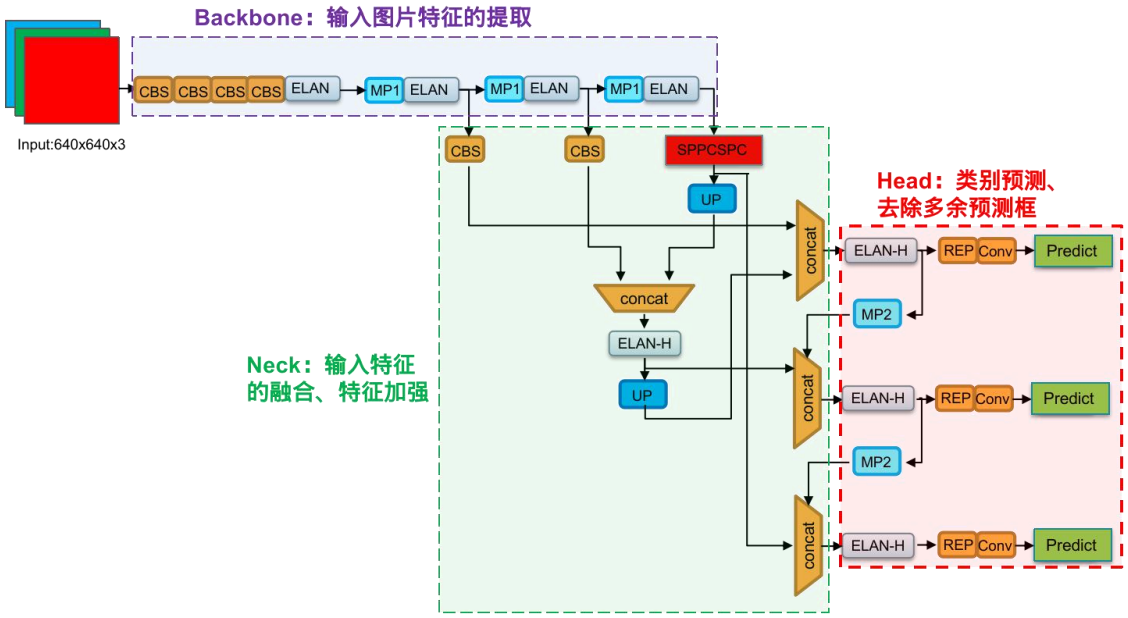


图 2-11 YOLOv7 网络结构

ELAN、MP 模块结构如下图 2-12 所示。

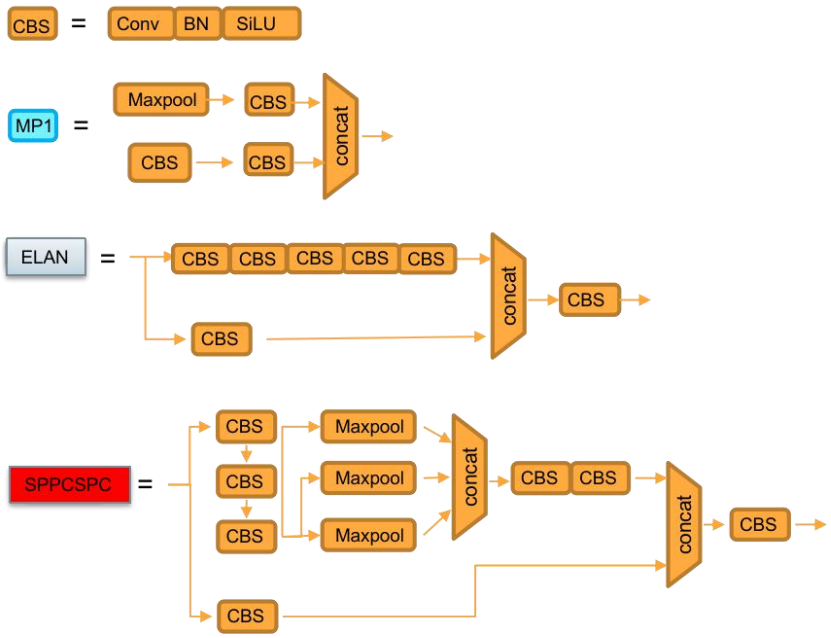


图 2-12 CBS、MP1、ELAN、SPPCSPC 结构

2.3 模型对比研究实验

2.3.1 实验环境

PC 端采用 Ubuntu 系统，使用 CPU 和 GPU 对模型进行训练。深度学习框架为 Pytorch 2.2.2、CUDA 12.3。网络训练设定批大小为 16，输入图片大小为 640x640。训练迭代次数为 10 轮。

训练好的模型分别在 PC 端和树莓派上运行，比较其检测速度。实验平台配置如表 2.1 所

示。

表 2.1 PC 端平台配置

配置环境	版本型号
操作系统	Ubuntu 22.04.4 LTS
语言	Python 3.8
CPU	AMD® Ryzen 9 7945hx
GPU	Nvidia GeForce RTX 4060
内存	16.0GB

树莓派平台配置如表 2.2 所示。

表 2.2 树莓派平台配置

配置环境	版本型号
操作系统	Raspberry Pi OS
语言	Python 3.8
CPU	BCM2711 SoC
内存	4GB

2.3.2 实验数据集

训练模型的数据集，采用的是公开数据集 Fruits detection，如图 2-13。数据集共有 8479 张照片，共有 6 种不同的水果类别。包括 Apple, Grapes, Pineapple, Orange, Banana, Watermelon。

图 2-13 Fruits detection 数据集

另外，视频输入所采用的视频是一段水果视频，如图 2-14 所示，视频长度为 58s。

图 2-14 水果视频

2.3.3 评测指标

轻量化的检测指标，本文采用每秒帧率 FPS(Frame Per Second)、参数量 Params、模型占用内存[22]三个指标进行评测。

FPS（Frames Per Second），也就是每秒传输的帧数，它描述了视频或动画中每秒展示的图片数量。对于目标检测任务而言，FPS 具体表示了模型网络在一秒钟内能够完成处理的图像帧数。FPS 的计算公式如公式 2.3 所示。FPS 越大说明检测速度越快。

$$\mathrm{FPS} = \frac{1}{\text{Processing time per frame}} \tag{2.3}$$

目标检测处理一张图片的时间，从图片处理处开始计时，到推理结束停止计时。

参数量 Params 反应了网络结构的总参数量。模型的复杂度通常与其参数的数量成正比，参数数量的增加表示模型结构的复杂化。

模型的内存占用，即模型所占用的存储空间大小，指明了该模型在计算机内存中占据的容量。

2.3.4 实验结果

分别使用三种目标检测模型，对视频文件进行检测，检测结果如下所示。

在 PC 端，YOLOv5 的检测速度 FPS 平均为 24.325。前 5 轮检测结果如表 2.3 所示。可视化结果如图 2-15 所示。

表 2.3 YOLOv5 在 PC 端的 FPS 部分测试结果表

检测轮次	FPS
第 1 轮	22.06111865
第 2 轮	23.54815963
第 3 轮	23.54815963
第 4 轮	23.54815963
第 5 轮	23.54815963
平均检测速度 FPS	24.325

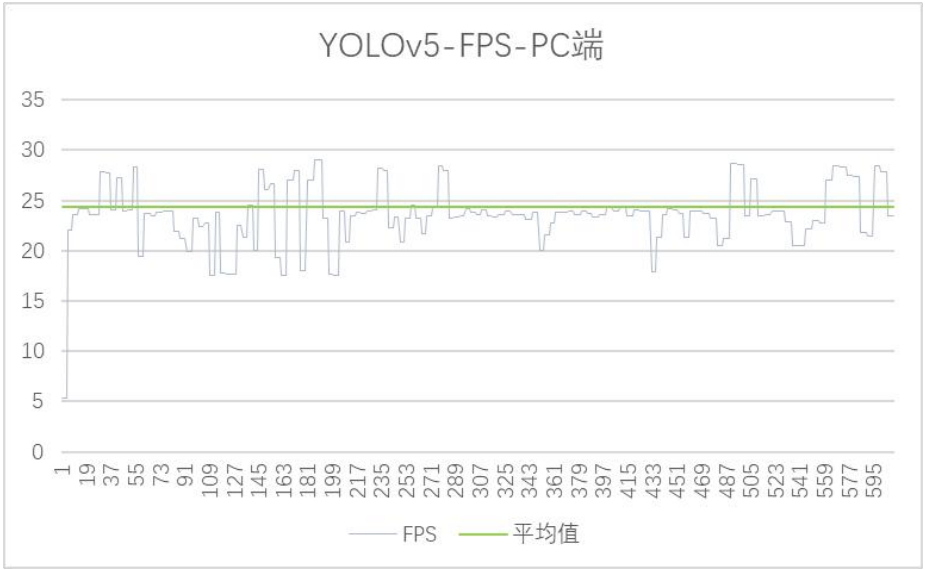


图 2-15 YOLOv5 在 PC 端的 FPS 检测结果图

YOLOv5 各层参数如下表 2.4 所示。

表 2.4 YOLOv5 各层参数

	params	module
0	3520	models.common.Conv

1	18560	models.common.Conv
2	18816	models.common.C3
3	73984	models.common.Conv
4	115712	models.common.C3
5	295424	models.common.Conv
6	625152	models.common.C3
7	1180672	models.common.Conv
8	1182720	models.common.C3
9	656896	models.common.SPPF
10	131584	models.common.Conv
11	0	torch.nn.modules.upsampling.Upsample
12	0	models.common.Concat
13	361984	models.common.C3
14	33024	models.common.Conv
15	0	torch.nn.modules.upsampling.Upsample
16	0	models.common.Concat
17	90880	models.common.C3
18	147712	models.common.Conv
19	0	models.common.Concat
20	296448	models.common.C3
21	590336	models.common.Conv
22	0	models.common.Concat
23	1182720	models.common.C3
24	229245	Detect

在 PC 端，YOLOv5-lite 的检测速度 FPS 平均值为 53.106。前 5 轮检测结果如表 2.5 所示。可视化结果如图 2-16 所示。

表 2.5 YOLOv5-lite 在 PC 端的 FPS 部分测试结果表

检测轮次	FPS
第 1 轮	57.27497918
第 2 轮	52.62284675
第 3 轮	57.07700891
第 4 轮	57.72587016
第 5 轮	60.25346569
平均检测速度 FPS	53.106

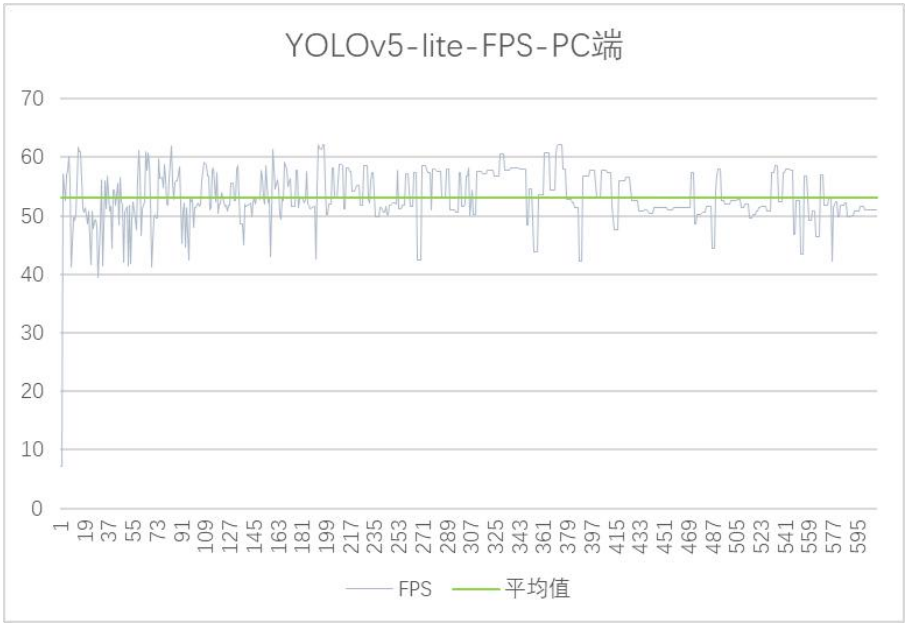


图 2-16 YOLOv5-lite 在 PC 端的 FPS 检测结果图

在树莓派上，YOLOv5-lite 的检测速度 FPS 平均为 1.102。前 5 轮检测结果如表 2.6 所示。可视化结果如图 2-17 所示。

表 2.6 YOLOv5-lite 在树莓派的 FPS 部分测试结果表

检测轮次	FPS
第 1 轮	1.069534414
第 2 轮	1.0847366
第 3 轮	1.0847366
第 4 轮	1.0847366
第 5 轮	1.0995222
平均检测速度 FPS	1.102

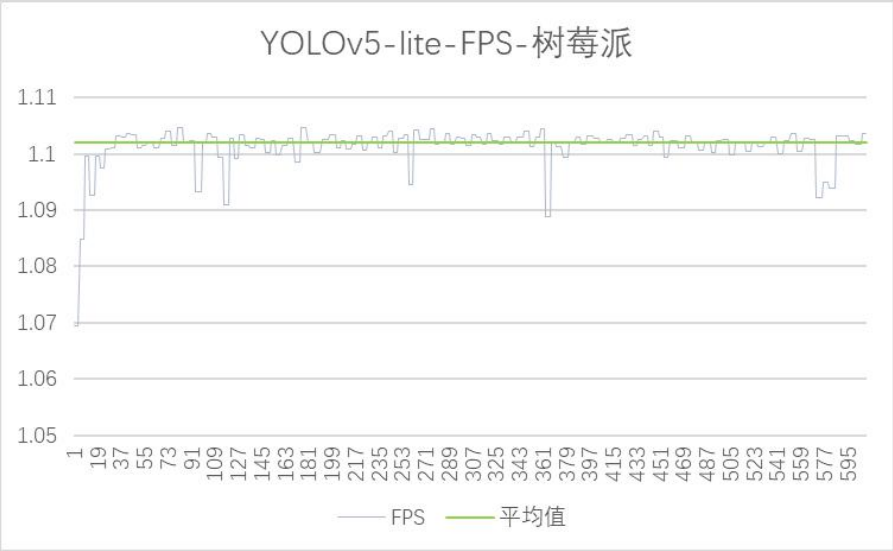


图 2-17 YOLOv5-lite 在树莓派的 FPS 检测结果图

YOLOv5-Lite 各层参数如下表 2.7 所示。

表 2.7 YOLOv5-Lite 各层参数

	params	module
0	928	models.common.conv_bn_relu_maxpool
1	8812	models.common.Shuffle_Block
2	24300	models.common.Shuffle_Block
3	44588	models.common.Shuffle_Block
4	200564	models.common.Shuffle_Block
5	167968	models.common.Shuffle_Block
6	333384	models.common.Shuffle_Block
7	59648	models.common.Conv
8	0	torch.nn.modules.upsampling.Upsample
9	0	models.common.Concat
10	104192	models.common.C3
11	8320	models.common.Conv
12	0	torch.nn.modules.upsampling.Upsample
13	0	models.common.Concat
14	26496	models.common.C3
15	36992	models.common.Conv
16	0	models.common.Concat
17	74496	models.common.C3
18	147712	models.common.Conv
19	0	models.common.Concat
20	296448	models.common.C3
21	115005	Detect

YOLOv7 的检测速度 FPS 平均值为 6.330。前 5 轮检测结果如表 2.8 所示。可视化结果如图 2-18 所示。

表 2.8 YOLOv7 在 PC 端的 FPS 部分测试结果表

检测轮次	FPS
第 1 轮	6.125816972
第 2 轮	8.167799703
第 3 轮	8.167799703
第 4 轮	8.313322551
第 5 轮	8.328195272
平均检测速度 FPS	6.330

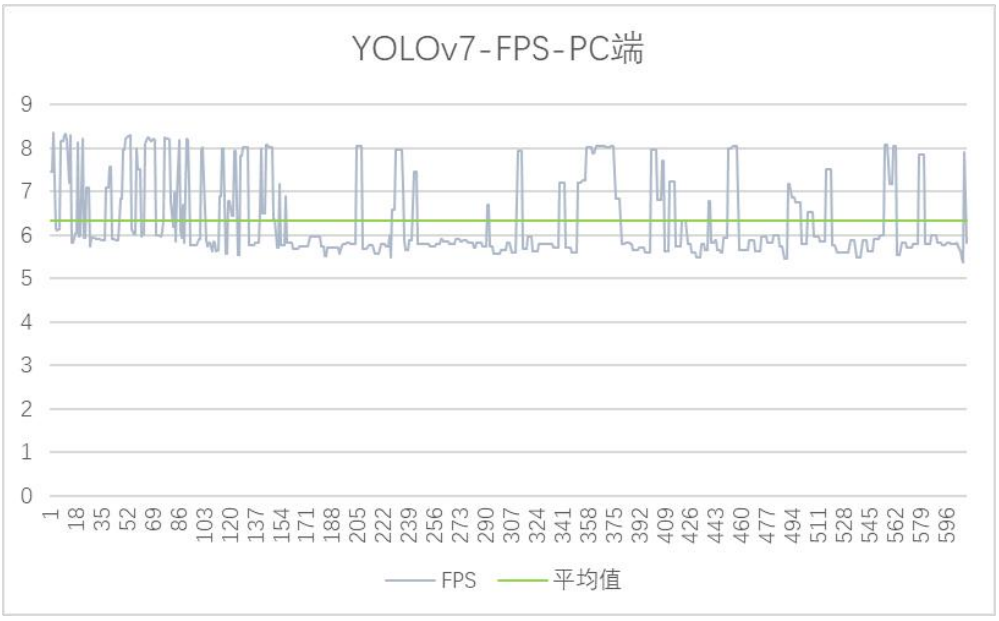


图 2-18 YOLOv7 在 PC 端的 FPS 检测结果图

在树莓派上，YOLOv7 的检测速度 FPS 平均为 0.477。前 5 轮检测结果如表 2.9 所示。可视化结果如图 2-19 所示。

表 2.9 YOLOv7 在树莓派的 FPS 部分测试结果表

检测轮次	FPS
第 1 轮	0.441360546
第 2 轮	0.472717584

第 3 轮	0.472717584
第 4 轮	0.472717584
第 5 轮	0.472717584
平均检测速度 FPS	0.477

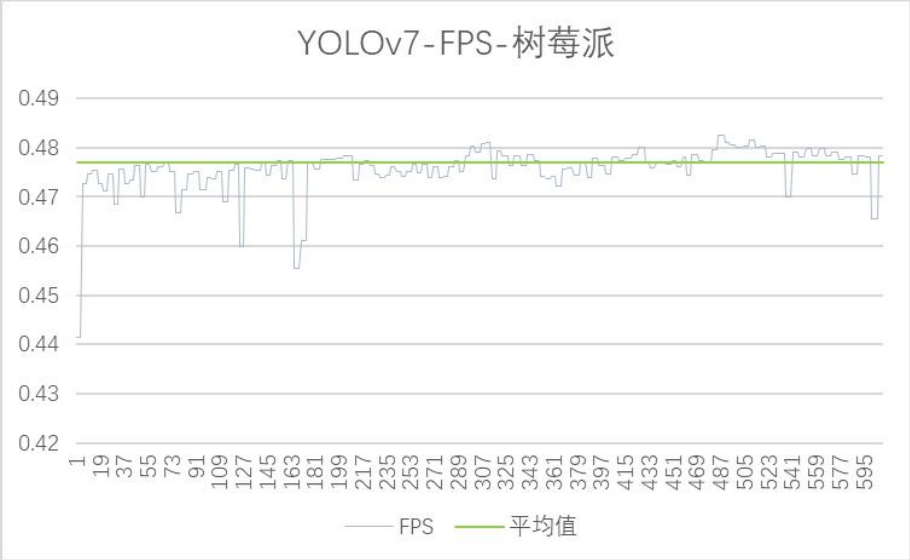


图 2-19 YOLOv7 在树莓派的 FPS 检测结果图

YOLOv7 各层参数如下表 2.10 所示。

表 2.10 YOLOv7 各层参数

	params	module		params	module
0	928	models.common.Conv	5	8320	models.common.Conv
1	18560	models.common.Conv	6	36992	models.common.Conv
2	36992	models.common.Conv	7	36992	models.common.Conv
3	73984	models.common.Conv	8	36992	models.common.Conv
4	8320	models.common.Conv	9	36992	models.common.Conv
10	0	models.common.Concat	39	525312	models.common.Conv
11	66048	models.common.Conv	40	525312	models.common.Conv
12	0	models.common.MP	41	2360320	models.common.Conv
13	33024	models.common.Conv	42	0	models.common.Concat
14	33024	models.common.Conv	43	262656	models.common.Conv
15	147712	models.common.Conv	44	262656	models.common.Conv

16	0	models.common.Concat	45	590336	models.common.Conv
17	33024	models.common.Conv	46	590336	models.common.Conv
18	33024	models.common.Conv	47	590336	models.common.Conv
19	147712	models.common.Conv	48	590336	models.common.Conv
20	147712	models.common.Conv	49	0	models.common.Concat
21	147712	models.common.Conv	50	1050624	models.common.Conv
22	147712	models.common.Conv	51	7609344	models.common.SPPCSPC
23	0	models.common.Concat	52	131584	models.common.Conv
24	263168	models.common.Conv	53	0	torch.nn.modules.upsampling.Upsample
25	0	models.common.MP	54	262656	models.common.Conv
26	131584	models.common.Conv	55	0	models.common.Concat
27	131584	models.common.Conv	56	131584	models.common.Conv
28	590336	models.common.Conv	57	131584	models.common.Conv
29	0	models.common.Concat	58	295168	models.common.Conv
30	131584	models.common.Conv	59	147712	models.common.Conv
31	131584	models.common.Conv	60	147712	models.common.Conv
32	590336	models.common.Conv	61	147712	models.common.Conv
33	590336	models.common.Conv	62	0	models.common.Concat
34	590336	models.common.Conv	63	262656	models.common.Conv
35	590336	models.common.Conv	64	33024	models.common.Conv
36	0	models.common.Concat	65	0	torch.nn.modules.upsampling.Upsample
37	1050624	models.common.Conv	66	65792	models.common.Conv
38	0	models.common.MP	67	0	models.common.Concat
68	33024	models.common.Conv	87	0	models.common.Concat
69	33024	models.common.Conv	88	262656	models.common.Conv
70	73856	models.common.Conv	89	0	models.common.MP
71	36992	models.common.Conv	90	66048	models.common.Conv
72	36992	models.common.Conv	91	66048	models.common.Conv
73	36992	models.common.Conv	92	590336	models.common.Conv

74	0	models.common.Concat	93	0	models.common.Concat
75	65792	models.common.Conv	94	525312	models.common.Conv
76	0	models.common.MP	95	525312	models.common.Conv
77	16640	models.common.Conv	96	1180160	models.common.Conv
78	16640	models.common.Conv	97	590336	models.common.Conv
79	147712	models.common.Conv	98	590336	models.common.Conv
80	0	models.common.Concat	99	590336	models.common.Conv
81	131584	models.common.Conv	100	0	models.common.Concat
82	131584	models.common.Conv	101	1049600	models.common.Conv
83	295168	models.common.Conv	102	328704	models.common.RepConv
84	147712	models.common.Conv	103	1312768	models.common.RepConv
85	147712	models.common.Conv	104	5246976	models.common.RepConv
86	147712	models.common.Conv	105	460282	IDetect

三种目标检测模型在 PC 端 FPS 对比情况如下表 2.11 所示，对比折线图如图 2-20 所示。

表 2.11 YOLOv5、YOLOv5-lite、YOLOv7 的模型在 PC 端的 FPS 对比表

目标检测模型	平均 FPS
YOLOv5	24.325
YOLOv5-lite	53.106
YOLOv7	6.330

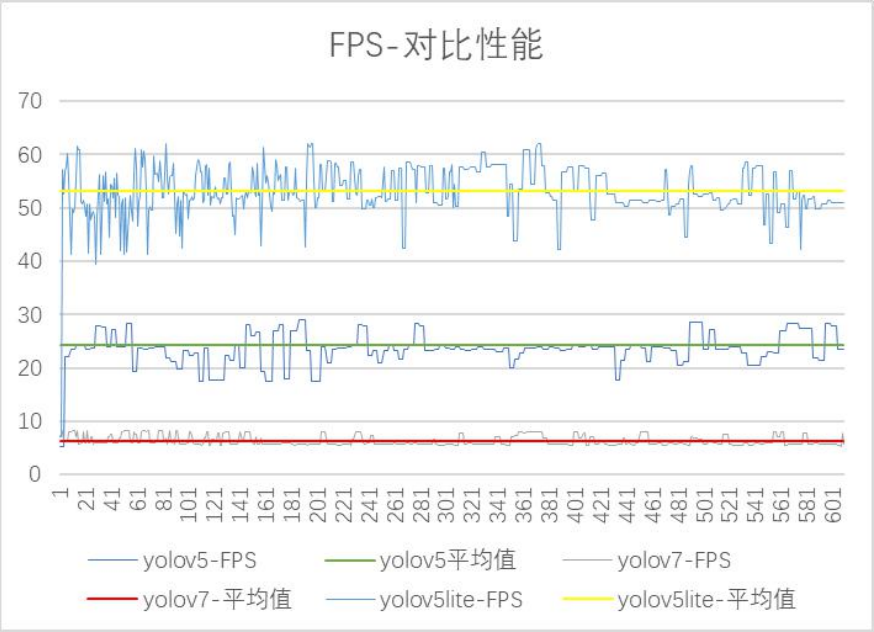


图 2-20 YOLOv5、YOLOv5-lite、YOLOv7 FPS 检测结果图(PC 端)

树莓派上的 FPS 对比数据详见表 2.12，而相应的可视化结果则展示在图 2-21 中。

表 2.12 YOLOv5、YOLOv5-lite、YOLOv7 的模型在树莓派的 FPS 对比表

目标检测模型	平均 FPS
YOLOv5	0.478
YOLOv5-lite	1.102
YOLOv7	0.477

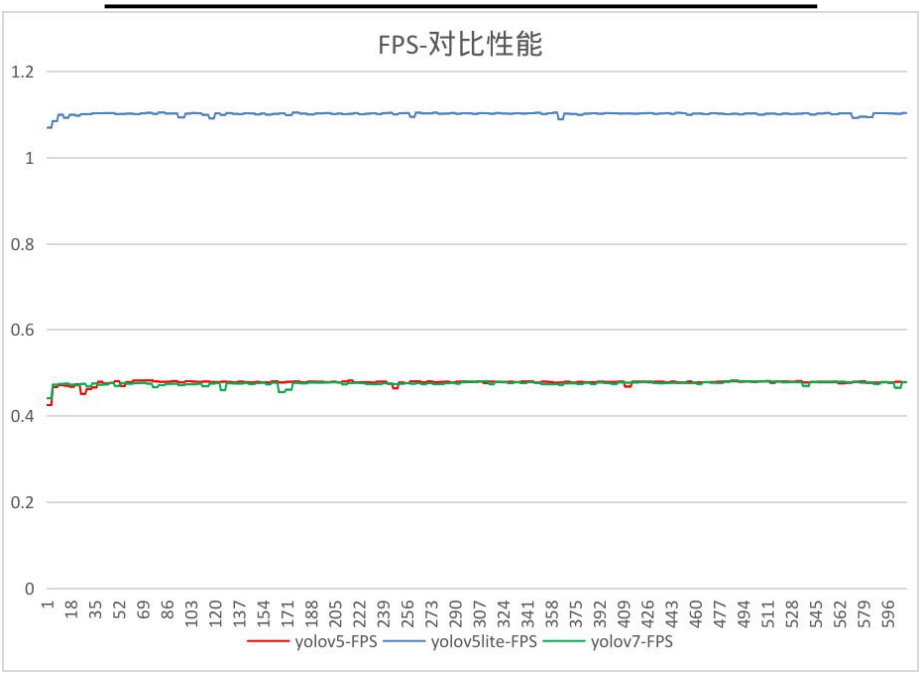


图 2-21 YOLOv5、YOLOv5-lite、YOLOv7 FPS 检测结果图(树莓派)

从模型的参数量 Params 上，YOLOv5、YOLOv5-lite、YOLOv7 的参数量 Params 对比如下表 2.13 所示。

表 2.13 YOLOv5、YOLOv5-lite、YOLOv7 的参数量对比

目标检测模型	参数量 Params
YOLOv5	7225885
YOLOv5-lite	1649853
YOLOv7	37622682

从模型占用内存大小来看，YOLOv5、YOLOv5-Lite、YOLOv7 的内存如下表 2.14 所示。

表 2.14 YOLOv5、YOLOv5-lite、YOLOv7 的模型占用内存大小对比

目标检测模型	占用内存大小
YOLOv5	28.6MB
YOLOv5-lite	6.2MB
YOLOv7	146.7MB

2.3.5 结果分析

实验结果表明，在模型轻量化方面，YOLOv5-lite 的轻量化效果要优于 YOLOv5、YOLOv7。YOLOv5 的轻量化效果次之，YOLOv7 的轻量化效果最差。

由实验结果可知，YOLOv5-Lite 相比与 YOLOv5，其 FPS 性能是 YOLOv5 的两倍。YOLOv5 的模型参数量 Params 是 YOLOv5-lite 的 5 倍。此外 YOLOv5 的模型占用内存大小为 28.6MB，而 YOLOv5-lite 为 6.2MB。YOLOv5 的模型远比 YOLOv5-lite 更加复杂。

YOLOv5 与 YOLOv7 相比较，其在 PC 端的 FPS 的平均值为 24.325，而 YOLOv7 为 6.330。在树莓派端 YOLOv5 的 FPS 的平均值与 YOLOv7 相近。因此在 PC 等资源充足大型设备上，YOLOv5 在检测速度上要优于 YOLOv7。而在小型、资源不充足的设备上二者区别不大，轻量化效果都较差。从参数量上来看，YOLOv5 的参数量 7225885 远小于 YOLOv7 的参数量

37622682。另外在模型占用内存大小方面，YOLOv5 的内存占用量为 28.6MB，相比之下，YOLOv7 的内存占用量达到 146.7MB，是 YOLOv5 占用量的五倍之多。

观察网络结构可以发现，在 YOLOv5-Lite 的版本中，Focus 层被移除了，以避免对输入图片多次采用 slice 操作，有效降低了 CPU 的计算负担也降低了缓存的使用。YOLOv5-lite 的 Backbone 结构使用 ShuffleNet 替换了 YOLOv5 的 CSP 和 YOLOv7 的 ELAN，ShuffleNet 结构如图 2-22 显示。轻化的思想是要增加运算效率,减少存储器开销。所以轻化系统的原则是要减少存储器的使用,增加网络的并行度。Shufflenet v2[23]论文中给出了轻量化设计的四大原则:同等通道大小下能够最小化内存访问量;过度使用组卷积会提高 MAC;网络过于碎片化(特别是多路)会降低并行度;不能忽略元素级操作(比如 shortcut 和 Add)[23]。

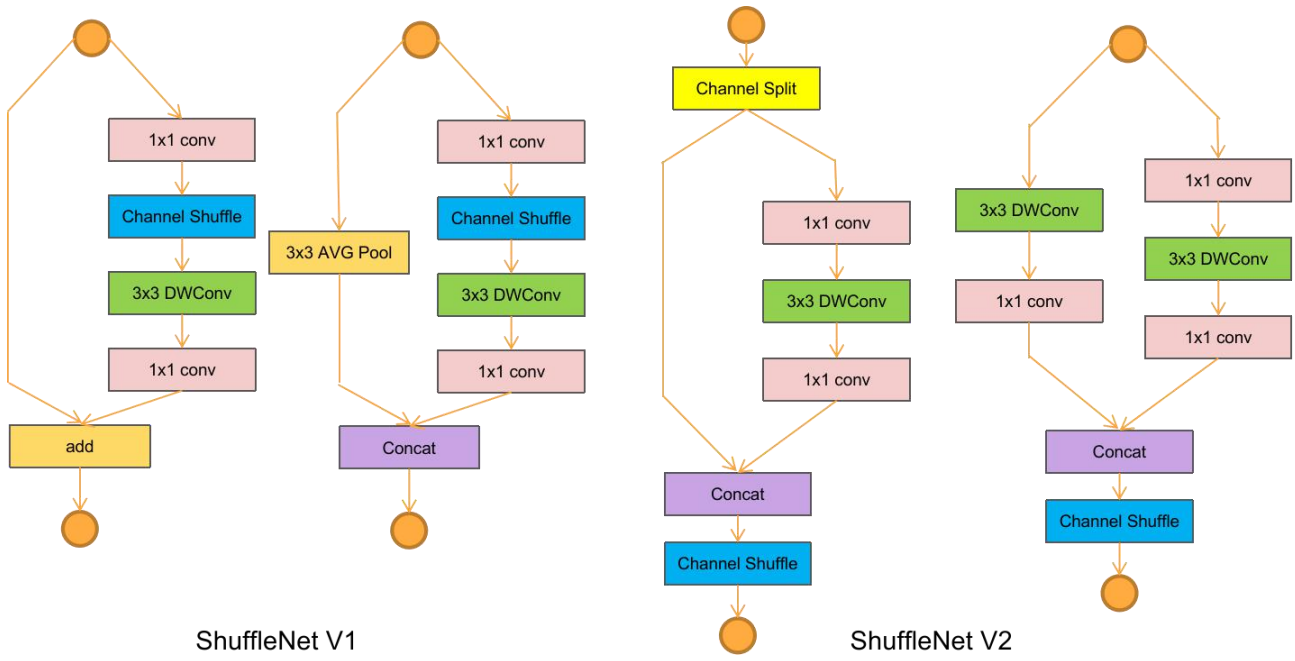


图 2-22 ShuffleNet

YOLOv5-lite 采用 ShuffleNet 作为 Backbone 的结构。ShuffleNetV2 采用了 Channel Split 通道分离操作。Channel Split 只是将通道分成两组，而不像 Slice 操作增加了通道数。这样 ShuffleNet 并没有因过量使用组卷积增加 MAC。此外最后使用 concat 操作，而没有使用 Tensor Add 操作，这也提升了运行速度。

3 轻量化目标检测系统的设计与实现

3.1 系统设计

3.1.1 总体方案

轻量化目标检测系统的工作流程如下：首先，需要在个人电脑(PC)和树莓派(Raspberry Pi)上分别设置好模型训练所需的环境。接着，利用公共数据集 Fruits detection，在 PC 上对目标检测模型进行训练。训练完成后，通过 VNC Viewer 软件，从 PC 远程访问树莓派。然后，将 PC 上训练得到的 YOLOv5、YOLOv5-lite、YOLOv7 模型，这些模型原本是以 pt 格式存储的，需要转换成 ONNX 格式，以便于在树莓派上使用。转换完成后，将这些 ONNX 格式的模型文件传输到树莓派。最后，由树莓派负责读取数据，并在 PC 端展示最终的目标检测结果。系统方案如图 3-1 所示。

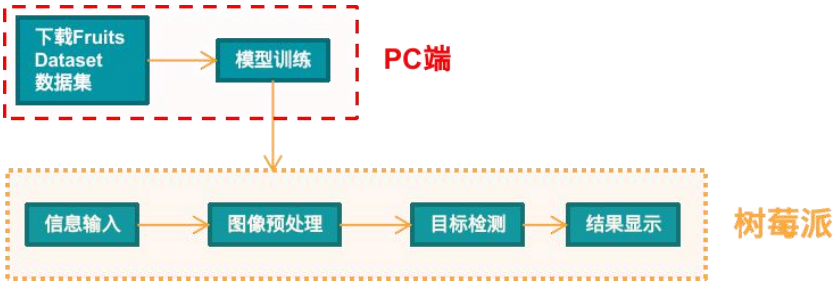


图 3-1 系统方案

3.1.2 总体功能设计

系统的整个工作流程如附图 3-2 所展示。用户通过远程方式连接到树莓派，并在启动程序后，通过弹出的对话框选择所需的 ONNX 目标检测模型（系统默认采用 YOLOv5 模型）。。导入模型后，用户可以选择图片、视频、摄像头三种导入方式。模型作用在对应的信息源上，并将检测结果在系统前端显示，最后保存检测结果到指定路径。

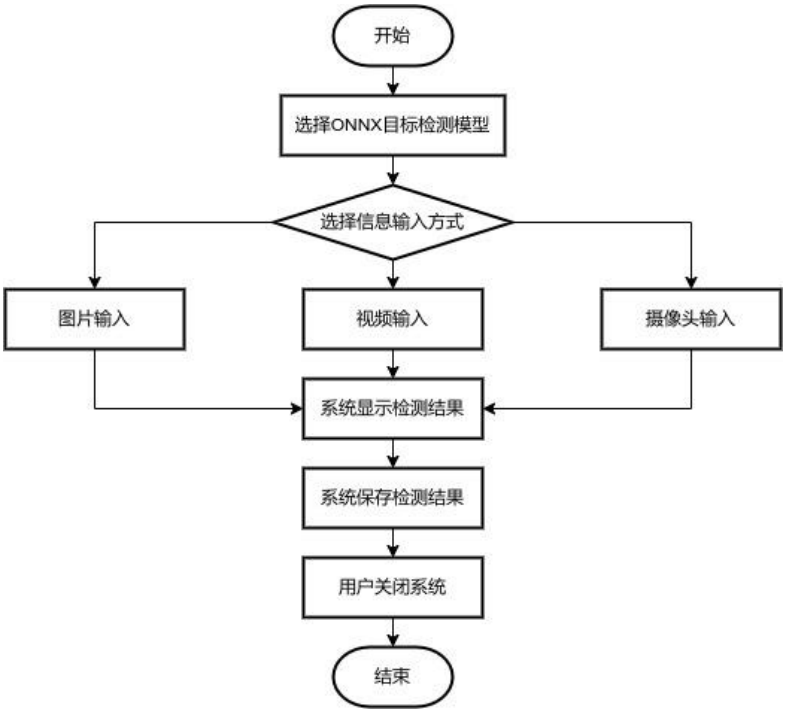


图 3-2 总体流程图

3.1.3 功能模块设计

系统的功能模块如图 3-3 所示，轻量化目标检测系统的功能模块主要为：模型选择模块、图片导入模块、视频导入模块、摄像头导入模块。

模型选择模块的功能是允许用户选择不同的 ONNX 格式目标检测模型，以便进行目标检测任务。图片输入模块则负责让用户选择不同的图片，并将这些图片送入所选模型进行目标检测，同时展示出检测结果和检测过程的性能指标 FPS（每秒传输帧数）。视频输入模块的功能是导入视频文件，利用该模型进行目标检测，不仅能够展示检测结果和 FPS（每秒帧数），还具备保存检测结果的功能。此外，摄像头导入模块通过接入的摄像头捕获实时视频流，并将检测结果实时反映在前端界面，用户也可以选择保存这些实时的检测结果。

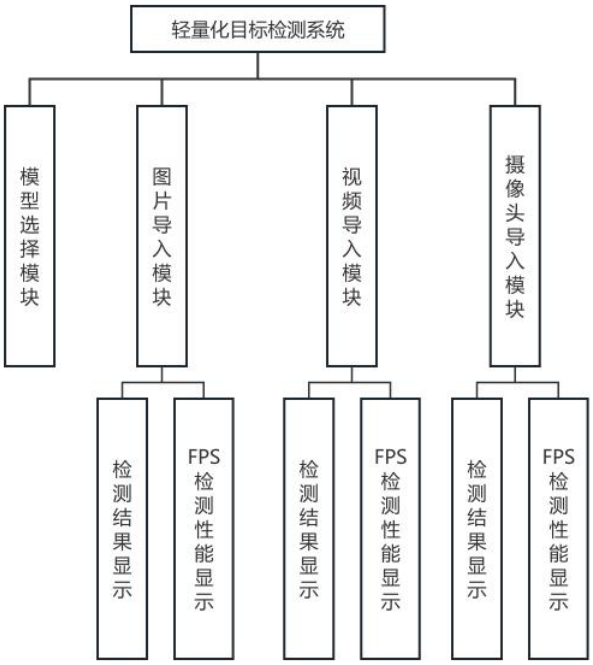


图 3-3 功能模块图

3.2 系统实现

本系统是利用 Python 语言进行开发的，其图形用户界面则是采用 PyQt 开发工具构建的。利用 YOLO 训练得到的目标检测模型，将其转换为 onnx 文件，即并运行在树莓派上。系统界面如图 3-4 所示，主要包含操作区域和检测结果可视化区域。

图 3-4 系统初始界面

用户在操作区域点击“模型选择”按钮并选择 onnx 模型文件后，界面会导入对应的模型。如图 3-5 所示。

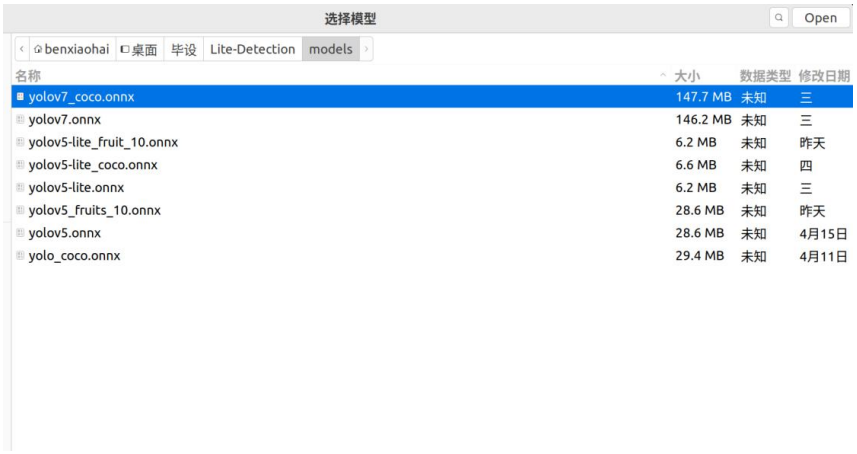


图 3-5 模型选择界面

在用户选定目标检测模型之后，接下来需要根据信息输入的类型来选择相应的操作。当用户点击“图片输入”按钮时，系统会引导用户选择用于目标检测的图片，如附图 3-6 所示。一旦用户选定了图片，系统将利用选定的模型进行推理，执行目标检测操作。检测过程结束后，系统将在图形用户界面上呈现检测成果，其中涵盖了目标物体的置信度评分及检测操作的帧速率（FPS）。如图 3-7 所示。

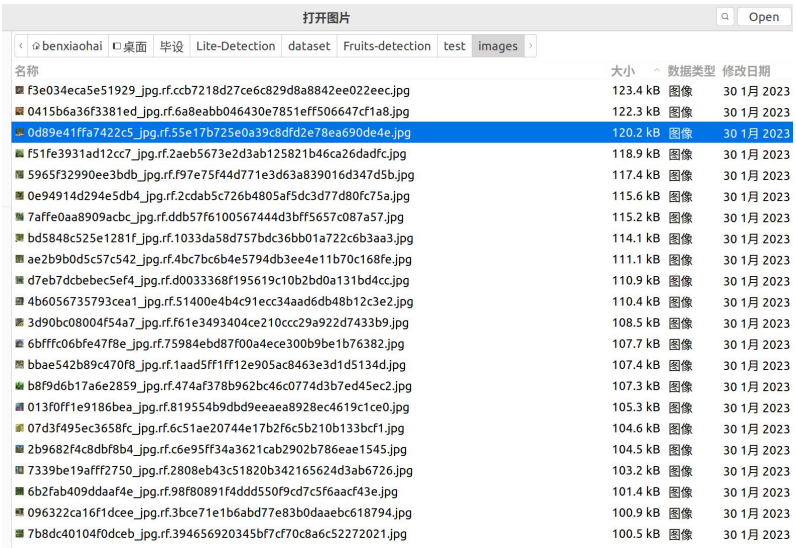


图 3-6 图片输入界面

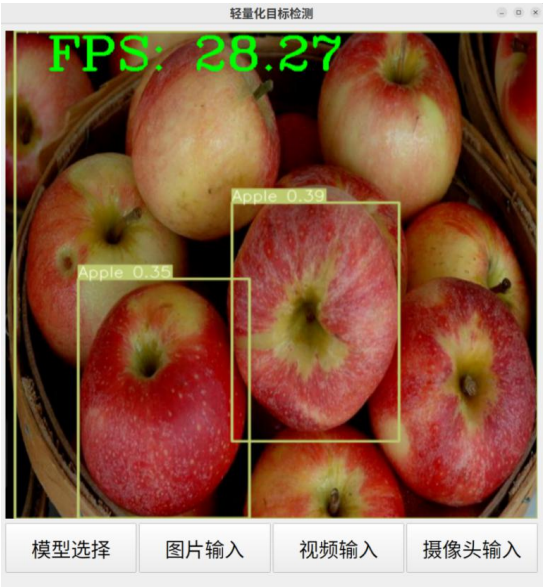


图 3-7 图片检测结果展示界面

点击“视频输入”按钮，系统会指定用户选择指定的视频文件导入，如图 3-8 所示。用户选择视频后，系统会进行模型推理，完成对视频的目标检测任务。并将目标检测的结果展示在可视化窗口区域，并展示检测物体的置信度以及对应的检测速度 FPS，如图 3-9 所示。

图 3-8 视频输入界面

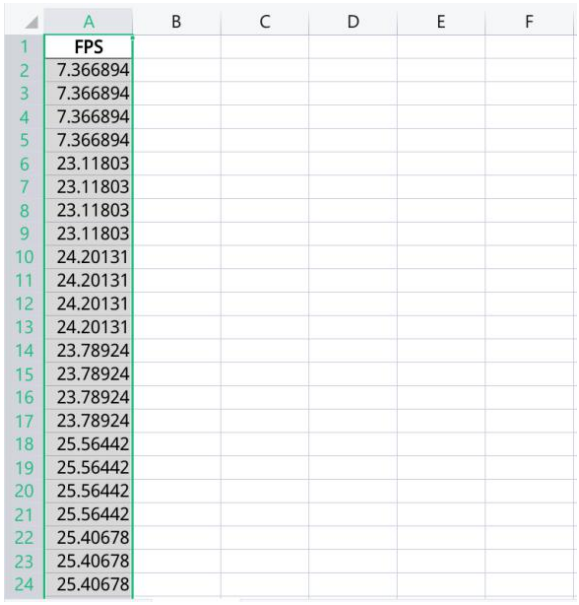
图 3-9 视频目标检测结果展示界面

点击“摄像头输入”按钮，系统会使用摄像头作为信息输入源。系统根据摄像头内容，实时进行模型推理，完成对输入内容的目标检测任务。并将目标检测的结果展示在可视化窗口区域，并展示检测物体的置信度以及对应的检测速度 FPS。如图 3-10 所示。

图 3-10 摄像头目标检测结果展示界面

每次运行完后，系统会自动导出当前模型对应的目标检测的 FPS 统计表，以便后续比较。

如图 3-11 所示。



	A	B	C	D	E	F
1	FPS					
2	7.366894					
3	7.366894					
4	7.366894					
5	7.366894					
6	23.11803					
7	23.11803					
8	23.11803					
9	23.11803					
10	24.20131					
11	24.20131					
12	24.20131					
13	24.20131					
14	23.78924					
15	23.78924					
16	23.78924					
17	23.78924					
18	25.56442					
19	25.56442					
20	25.56442					
21	25.56442					
22	25.40678					
23	25.40678					
24	25.40678					

图 3-11 当前目标检测模型的 FPS 统计结果

3.3 系统测试

在轻量化目标检测系统的功能测试中，如表 3.1 所示，设计了 7 个测试用例，这些测试用例旨在对系统的各个功能模块进行全面测试。

表 3.1 系统功能测试

用例编号	测试用例描述	预期结果	测试结果	所属功能模块
1	点击“模型选择”	进入本地文件夹	与预期相符	模型选择
2	选择本地 onnx 模型文件	正常导入平台	与预期相符	模型选择
3	点击”图片输入”	进入本地文件夹	与预期相符	数据加载
4	选择本地图片文件	显示图像目标检测结果及 FPS	与预期相符	显示交互
5	点击”视频输入”	进入本地文件夹	与预期相符	数据加载
6	选择本地视频文件	显示图像目标检测结果及 FPS，并保存检测视频	与预期相符	显示交互
7	点击”视频输入”	摄像头开启，显示图像目标检	与预期相符	数据加载

测结果及 FPS，并保存检测视
频

3.4 本章小结

本章节利用 Python 语言和 PyQt 工具，结合第二章介绍的模型，设计并构建了一个轻量级的目标检测软件，该软件适用于多种不同的目标检测模型，最后使用测试用例测试了系统的主要功能，使用测试数据测试了系统的准确性和有效性。

4 总结与展望

4.1 总结

本文针对 YOLO 系列算法，构建了一种轻量化目标检测软件。可供用户选择不同的检测模型。平台运行在树莓派上，本文主要工作如下：

（1）在 Fruits detection 数据集中，本研究采用了 YOLOv5、YOLOv5-Lite、YOLOv7 这三种网络架构进行实验，并展示了相应的成效，最终导出模型的检测速度 FPS。通过对比不同网络的性能，最终实验结果表明，YOLOv5-lite 网络在该任务上具有最优的性能。

（2）分析网络架构，阐述了 YOLOv5、YOLOv5-lite、YOLOv7 各自的特点，分析了 YOLOv5-lite 为什么检测速度优于 YOLOv5、YOLOv7。

（3）开发轻量化目标检测软件，本文基于 PyQt，使用三种目标检测模型，实现了图片、视频、摄像头三种输入源的目标检测，并将结果的可视化展示。该系统运行在树莓派上，可以方便地导入模型和实现目标检测，并提供友好的用户界面，方便用户选择不同的模型以及查看检测结果。

4.2 工作展望

本文所采用的目标检测模型能够实现基本的检测，但仍存在以下问题：

（1）本系统在树莓派上运行速度较慢。由于 PyQt 界面的使用，在树莓派这种资源有限的设备上运行，界面运行速度很慢。因此，如何构建轻量化的界面，或者远程连接树莓派运行目标检测模型，将检测结果返回给本地终端，需要后续的改进。

（2）本系统支持的模型较少，只适用 YOLOv5、YOLOv5-lite、YOLOv7。对于其他目标检测模型，效果支持并不好。如何解决多种模型的选择，以及不同模型的推理算法统一问题，有待后续的研究。

（3）本系统并没有做出算法创新，只是对现有模型的复现和应用。如何在现有模型的基

础上，对模型进行改进，后续需要继续深入研究。

参考文献

[1] 方路平,何杭江,周国民.目标检测算法研究综述[J].计算机工程与应用, 2018, 54(13):9.DOI:CNKI:SUN:JSGG.0.2018-13-002.

[2] ZOU Zhengxia, SHI Zhenwei, GUO Yuhong, et al. Object detection in 20 years: A survey[J]. arXiv preprint arXiv:1905.05055, 2019.

[3] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]//2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05). Ieee, 2005, 1: 886-893.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, pp. 1137–1149,June 2017.

[5] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

[6] 杨玉敏,廖育荣,林存宝,等.轻量化卷积神经网络目标检测算法综述[J].舰船电子工程, 2021, 041(004):31-36..

[7] 刘卫东,姜青山,李勇,等.基于先期毁伤的舰空导弹网络化协同反导火力分配[J].舰船科学技术, 2011, 33(2):5.DOI:10.3404/j.issn.1672-7649.2011.02.023.

[8] 刘东涛.有限雷达资源条件下的舰空导弹发射时序模型研究磁[J].舰船电子工程, 2015, 000(008):23-26..

[9] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2014: 580-587.

[10] Girshick R. Fast r-cnn[C]//Proceedings of the IEEE international conference on computer vision. 2015: 1440-1448.

[11] Ren S, He K, Girshick R, et al. Faster r-cnn: Towards real-time object detection with region proposal networks[J]. Advances in neural information processing systems, 2015, 28.

[12] He K, Gkioxari G, Dollár P, et al. Mask r-cnn[C]//Proceedings of the IEEE international conference on computer vision. 2017: 2961-2969.

[13] Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C]//Computer Vision–ECCV

2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, P art I 14. Springer International Publishing, 2016: 21-37.

[14] Wang C Y, Yeh I H, Liao H Y M. YOLOv9: Learning What You Want to Learn Using Programma ble Gradient Information[J]. arxiv preprint arxiv:2402.13616, 2024.

[15] Kim K H, Hong S, Roh B, et al. Pvanet: Deep but lightweight neural networks for real-time object detection[J]. arxiv preprint arxiv:1608.08021, 2016.

[16] Huang R, Pedoeem J, Chen C. YOLO-LITE: a real-time object detection algorithm optimized for no n-GPU computers[C]//2018 IEEE international conference on big data (big data). IEEE, 2018: 2503-2510.

[17] Zhang Y, Bi S, Dong M, et al. The implementation of CNN-based object detector on ARM embedd ed platforms[C]//2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Com puting and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech). IEEE, 2 018: 379-382.

[18] Zheng Z, Wang P, Liu W, et al. Distance-IoU loss: Faster and better learning for bounding box regr ession[C]//Proceedings of the AAAI conference on artificial intelligence. 2020, 34(07): 12993-13000.

[19] 贾世娜. 基于改进 YOLOv5 的小目标检测算法研究[D]. 硕士学位论文]. 南昌: 南昌大学, 2022.

[20] Zhang X , Zhou X , Lin M ,et al.ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices[J]. 2017.DOI:10.48550/arXiv.1707.01083.

[21] Wang C Y , Bochkovskiy A , Liao H Y M .YOLOv7: Trainable bag-of-freebies sets new state-of-th e-art for real-time object detectors[J].arXiv e-prints, 2022.DOI:10.48550/arXiv.2207.02696.

[22] 郑尚坡,陈德富,邱宝象,等.基于树莓派与 YOLOv5-Lite 模型的行人检测系统设计[J].计算机时代, 2023 (9):116-119.

[23] Ma N, Zhang X, Zheng H T, et al. Shufflenet v2: Practical guidelines for efficient cnn architecture design[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 116-131.

[24] 杨攀.基于深度学习的车型识别方法研究与应用[D].西南石油大学,2018.

[25] 宋志明.基于改进区域建议网络的目标检测算法[D].西安电子科技大学,2019.DOI:10.27389/d.cnki.gxadu. 2019.001518.

[26] 李易.多类型车道线的检测与识别方法研究[D].西安石油大学,2021.DOI:10.27400/d.cnki.gxasc.2021.000660.