1、软件简介

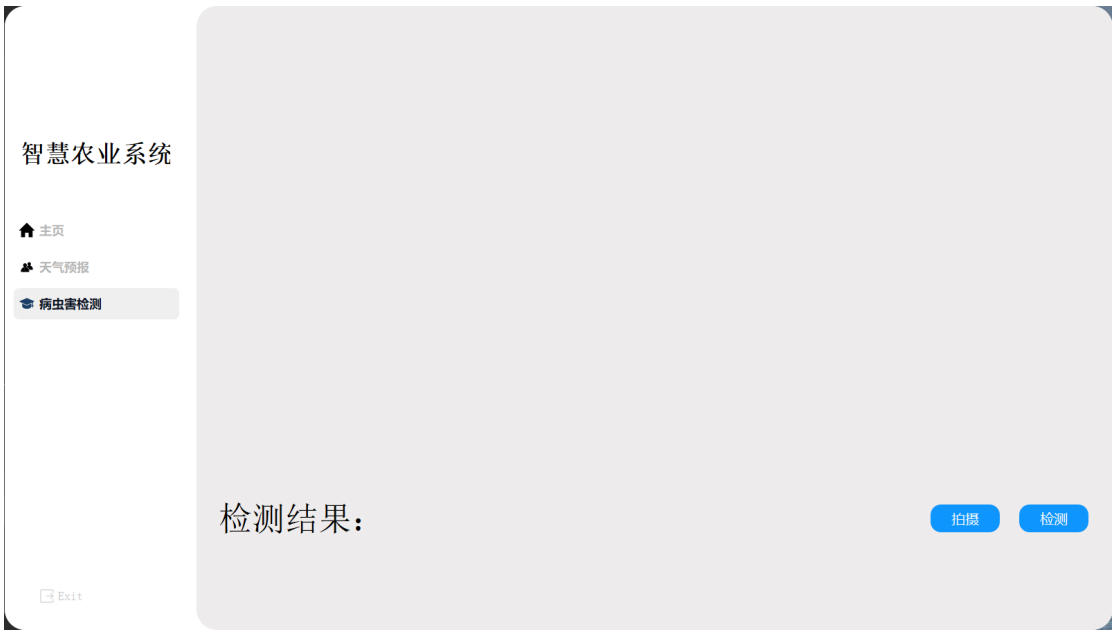
本项目主要通过传感器、摄像头采集农田信息。通过对图像信息数据的数字化处理和分析，得到模型的识别结果并反馈给用户，我们设计的智慧农业管理系统系统通过传感器、监测设备等手段，可以实时、准确地获取土壤的温度、湿度、光照等信息，为农民提供精确的决策依据。农业管理系统可以对农田进行全面规划和管理，提高土地的利用率和农作物的产量，农业管理系统能够充分利用信息技术，实现对土地、作物、气象等方面的数据采集和分析，帮助农民科学合理地进行农业生产。我们的软件实现了病虫害、温湿度、CO2 、TVOC、光照等全程监测，帮助管理决策，可检测病虫害 14 种，识别率达 95%，实时检测，单次仅需 0.15s，通过多病虫害精准检测、多参数环境建模，提升全周期数字化管理决策能力

2.使用说明

打开软件，显示当前状态，土壤情况，天气情况，以及种子情况

下方显示工作模式，右边显示各项传感器的参数，包括电池电量，空气温湿度，光照强度，TVOC 和二氧化碳浓度

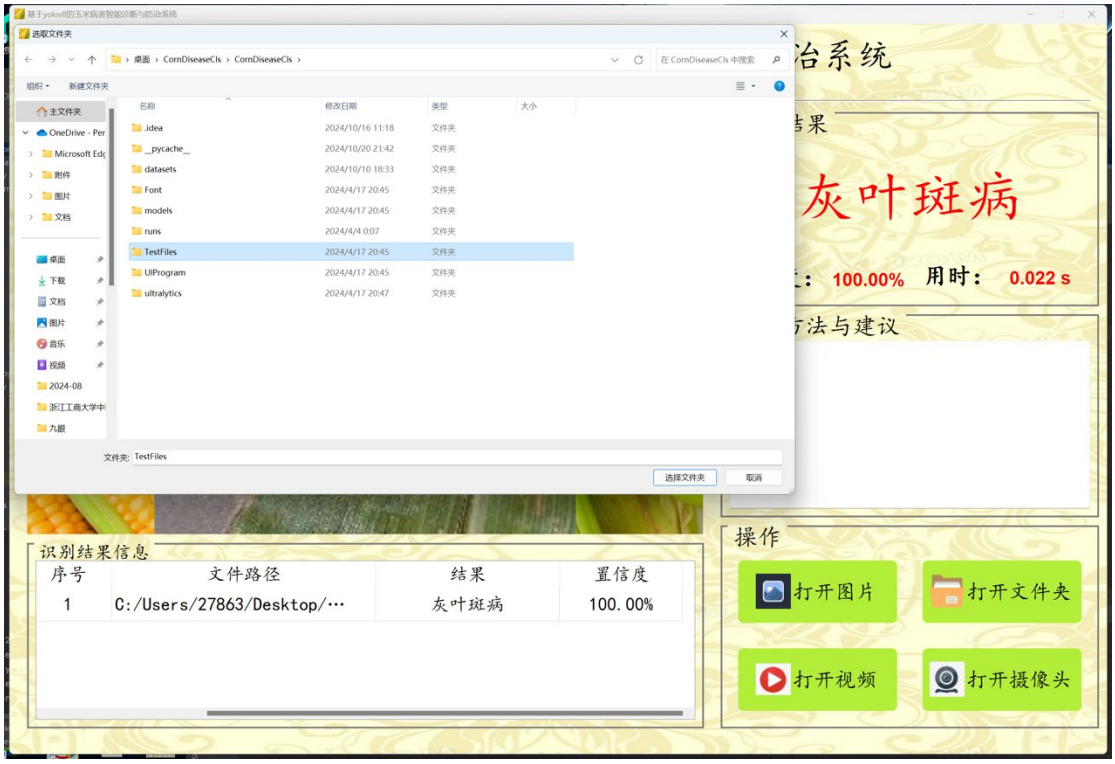
主页下面是三栏，有天气预报功能和病虫害检测功能



病虫害检测主页，主要有四项功能，单张图片检测，批量图片检测，视频检测和摄像头检测，检测结果通过表格的形式展示，右边显示结果，置信度和用时

选中一张图片

叶片的识别结果为灰叶病，下面给出防治方法



选中一个文件夹

给出所有的识别结果，在表格内，可以选中任意一个数据

给出防治建议

打开摄像头通过手机摄像头进行实时监测

3.需求分析

粮食安全对于中国这一发展中的大国至关重要，它不仅是经济发展和社会稳定的基石，也是国家安全的关键要素。鉴于中国复杂的地理环境，尤其是丘陵地带占据了相当大的一部分，这些区域的粮食生产能力对维护全国粮食安全发挥重

要作用。丘陵地形的起伏不平、土壤肥力不均等自然因素限制了农业生产的效率和产量。因此，增强丘陵地区的粮食生产能力，对于实现全国粮食安全具有关键作用。

从世界范围看，联合国大会在2015年通过的《2030年可持续发展议程》将消除饥饿、实现粮食安全、改善营养和促进可持续农业作为17个可持续目标之一。到2050年，持续的人口和人均粮食消费量的增长意味着全球的粮食需求仍将不断增长，全球90亿人口所需要的粮食预计比现在多59%-98%。如何提高粮食产量和维护粮食安全已成为全球的关注焦点。随着全球人口的增长和粮食需求的攀升，丘陵地区必须寻找更加高效、可持续的农业生产方式，以满足日益增长的粮食需求。

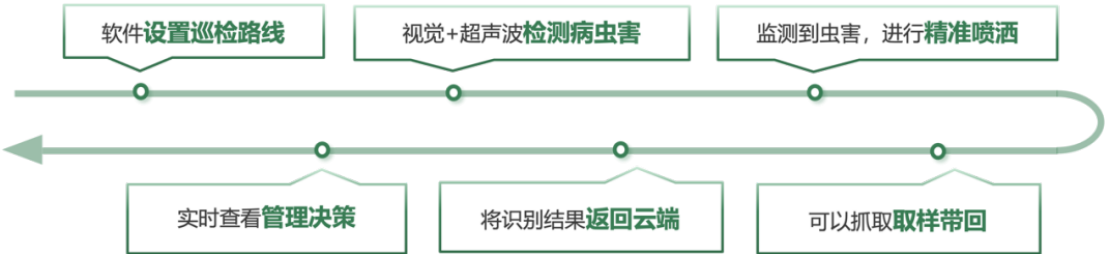
在中国，随着工业化、城镇化、信息化和农业现代化的同步发展，耕地资源的供需矛盾愈发突出。目前，全国已有8个省份、74个地级市人均耕地面积低于联合国粮农组织标准测算的0.8亩警戒线，已到了不能实现本区粮食自给的水平。鉴于丘陵地区的耕地资源相对有限，且受地形限制，土地碎片化现象严重，这进一步加剧了耕地资源的紧张状况。

《“十四五”全国种植业发展规划》进一步规划保障了粮食等重要农产品供给安全的决策部署，所示，这一规划强调了提高农产品产量和质量的目标，以满足国内需求的增长。此外，《规划》还强调了农业自动化、数字化建设的重要性，这表明政府致力农业高新技术的发展，以推动农业现代化进程。

2023年2月，中共中央、国务院印发的《数字中国建设整体布局规划》提出“要深入实施数字乡村发展行动，以数字化赋能乡村产业发展、乡村建设和乡村治理”.这为我国乡村数字化建设提供了方向指引和政策指导。

日本通过运用物联网、大数据等现代信息技术，实现了精准管理、精准作业，极大地提高了农业生产的智能化水平和资源利用效率。这种精细化农业的发展模式，不仅提高了农产品的产量和质量，还减少了对环境的负面影响，为我国农业现代化提供了有益的借鉴契合了国内农业数字化、现代化建设的大方向。通过结合先进的农业技术和政策支持，我们有信心助力农业实现更高水平的可持续发展。

智能化：产品具备智能化功能，可以根据作物生长情况和病虫害情况进行智能化的识别治理，提高喷洒效率、减少农药的使用量。



4.基于 MQTT 协议进行环境信息传输

随着物联网的快速发展，嵌入式设备和传感器在各种应用中越来越普遍。这些设备需要通过网络进行数据传输和通信。MQTT 是一种轻量级的发布/订阅消息传输协议，适用于低功耗设备和网络带宽有限的环境，如图所示。相比之下，HTTP 协议较为复杂，且需要更多的网络带宽和电力资源。

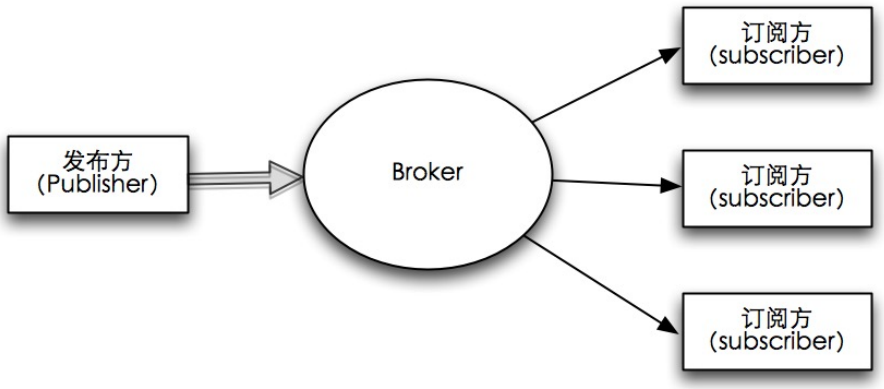
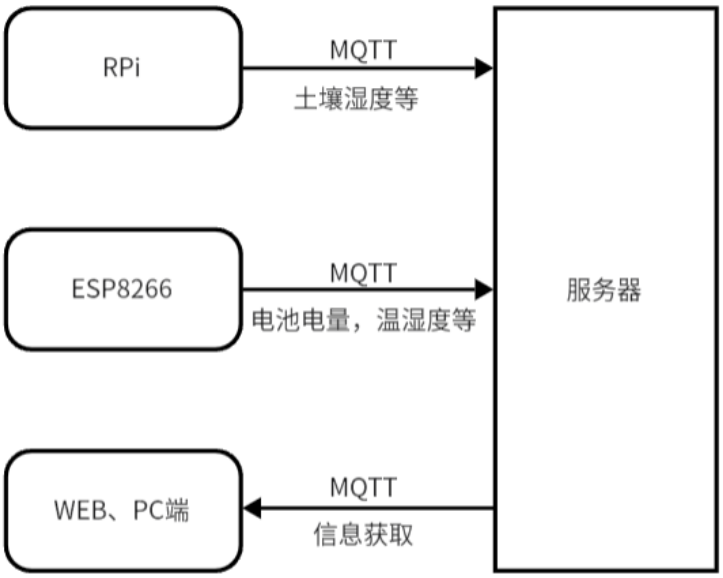


图 MQTT 原理图

在嵌入式设备中，低功耗和轻量级是非常重要的考虑因素。而 MQTT 协议在轻量级、低功耗和发布/订阅模式等方面具有优势，更适合在嵌入式系统中使用，如所示。



图信息传输架构图

本项目的 MQTT Broker 使用的是 EMQ-X，具有高并发、高吞吐的能力，并且可以将各种数据存储在 Redis 数据库中，后续在 WEB 端可以通过这些数据进行数据分析，并反馈给用户，提出相应的策略。

5.基于 MJPG 实现实时图像抓取

在本项目中，我们使用 MJPG（开源软件）来抓取视频，并将抓取的视频传送至 WEB 端进行实时显示

与其他流传输软件相比，mjpg_streamer 具有几个显著的优势。首先，它采用 MJPEG 格式编码视频流，这种格式在保留图像质量的同时能够有效地减少文件大小，从而降低网络传输的带宽要求。其次，mjpg_streamer 使用 V4L2 驱动程序来访问摄像头设备，这意味着它能够充分利用 Linux 系统的性能和稳定性。此外，由于 mjpg_streamer 是通过 C 语言实现的，因此具有出色的运行速度和高效的系统资源利用率，使得它能够在不占用过多系统资源的情况下实现高质量的视频流传输。

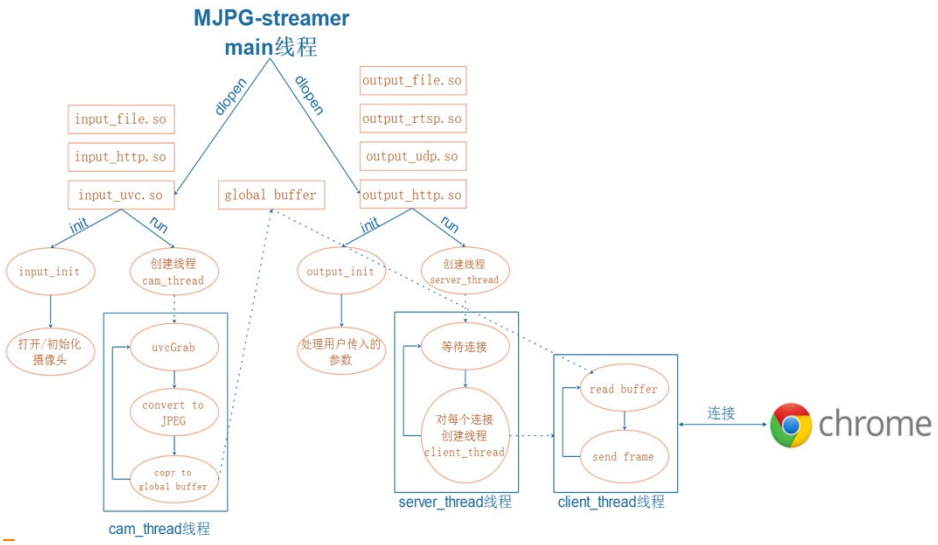


图 MJPG 原理图

经过实验验证，使用 MJPG 抓取视频相比于其他软件会有更快的速度，因为它可以提供更好的图像质量和更小的文件大小。在实验中，使用 MJPG 抓取视频延迟小于 80ms，而使用 OpenCV 抓取视频延迟大于 100ms，使用 Linux 内置摄像机拍摄延迟大于 90ms。

基于我们的模型转换算法，将训练好的模型软件转化为 Tensorflow Lite 可部署在安卓设备的轻量化模型，基于 Android Studio 的交互编程实现各个页面的跳转。我们通过编写 mMoudle 方法成功实现在安卓端加载轻量模型并进行计算：

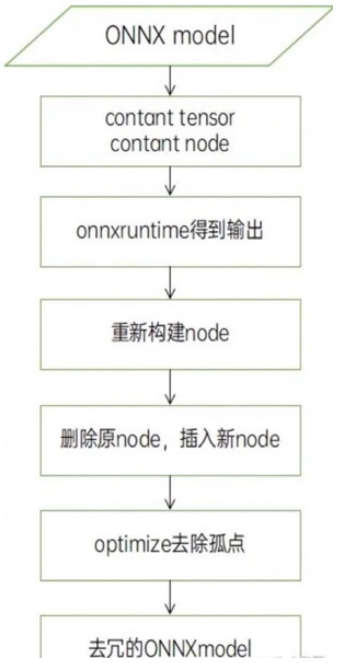
```
// 读取文件，加载模型
try {
    mModule = LiteModuleLoader.load(assetFilePath(getApplicationContext(),  assetName: "yolov5s.torchscript.ptl"));
    BufferedReader br = new BufferedReader(new InputStreamReader(getAssets().open( fileName: "classes.txt")));
    String line;
    List<String> classes = new ArrayList<>();//classes 放置classes
    while ((line = br.readLine()) != null){
        classes.add(line);
    }
    //
    PrePostProcessor.mClasses = new String[classes.size()];
    classes.toArray(PrePostProcessor.mClasses);
} catch (IOException e) {
    Log.e( tag: "Object Detection", msg: "Error reading assets",e);
    finish();
}
```

6. 轻量化模型转换

Yolo 模型较重，适合于计算机运行，因此部署在边缘设备成为难题。针对两套方

案，我们设计了更轻和更重的模型版本，以分别解决缺乏高效 GPU 支持的移动设备上的 CPU 推断系统和在桌面上运行的精度要求更高的问题。

我们的第一种方案以实时移动 GPU 推断系统为目标，在云端服务器运行，设计了较大的模型版本，此版本精度更高，识别更准确；而部署在移动设备本地运行便需要更轻的版本。因此，我们设计了新的轻量化模型算法，通过万能模型 onnx 为媒介，将训练的大模型转换为 pytorchlite、tflite、NCNN 轻量化模型，边缘设备通过 onnx 部署，分别可以部署在不同的设备上。我们的算法在力求减少所需算力的同时尽量保证了精度不下降太多。轻量化模型使得大部分设备能够轻松运行，下图为转换代码和核心思路如下：



7 网络环境设计及部署

使用移动网络模块连接主控以及 ESP8266。这两块开发板之间的网络通信属于局域网通信，连接速度快，延迟低。

MQTT 服务器是一台 2 核 4G 的服务器，具有高并发的特性，并且具有公网 IP，方便处理两块开发板上的 MQTT 请求。

WEB 端及 PC 端都是运行在本地计算机上的。每位客户买了此产品后，都可以在自己本地计算机运行一个 WEB 页面，此页面可以显示这位客户的机器在田间传来的各种数据。

远程终端是利用内网穿透主控的 22 端口，实现远程 SSH 控制。用户可在本地计算机远程控制田间的主控板，输入给定的指令，可令其进行移动、返回等操作。

8 模型训练

团队利用大量数据集训练，模型覆盖多种场景。数据集制作采用拍摄与图像增广算法，已经实现了对 14 种常见套种虫害的识别，训练精度高达 90.5%，识别平均置信度高达 0.86，识别准确率极高。

我们使用了病虫害数据集 IP-CitrusPests 13，用于病虫害小目标检测训练。

在病虫害监测功能中，我们采用了 YOLOv8 作为基础的目标检测算法。相比其他目标检测算法，YOLOv8 具有几个显著的优势。首先，它具有更快的检测速度和更高的准确性，能够实现实时的目标检测和定位。其次，YOLOv8 采用了最新的模型优化技术，能够在保持高性能的同时减小模型的体积，降低了模型部署和运行的成本。此外，YOLOv8 在训练过程中采用了一系列的数据增强和模型优化技术，能够更好地适应不同环境下的病虫害监测需求。

从技术对比的角度来看，相比传统的目标检测算法，YOLOv8 在病虫害监测中具有更高的灵敏度和准确性。它能够快速而准确地识别病虫害区域，为后续的病害分析和防治提供了可靠的数据支持

同时，YOLOv8 在实时性和稳定性上也具有明显的优势，能够满足病虫害监测对实时性和稳定性的要求，如所示。

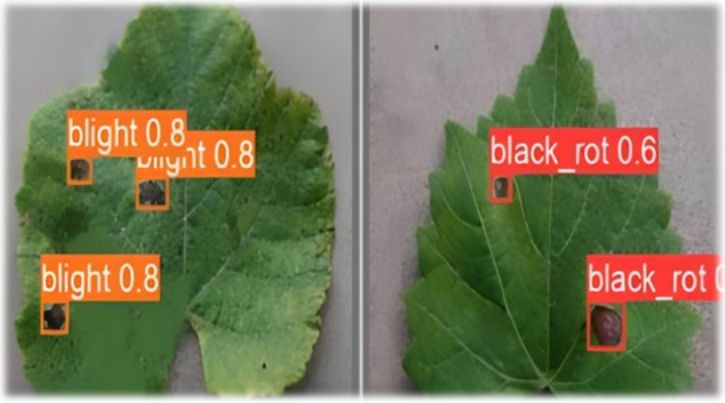
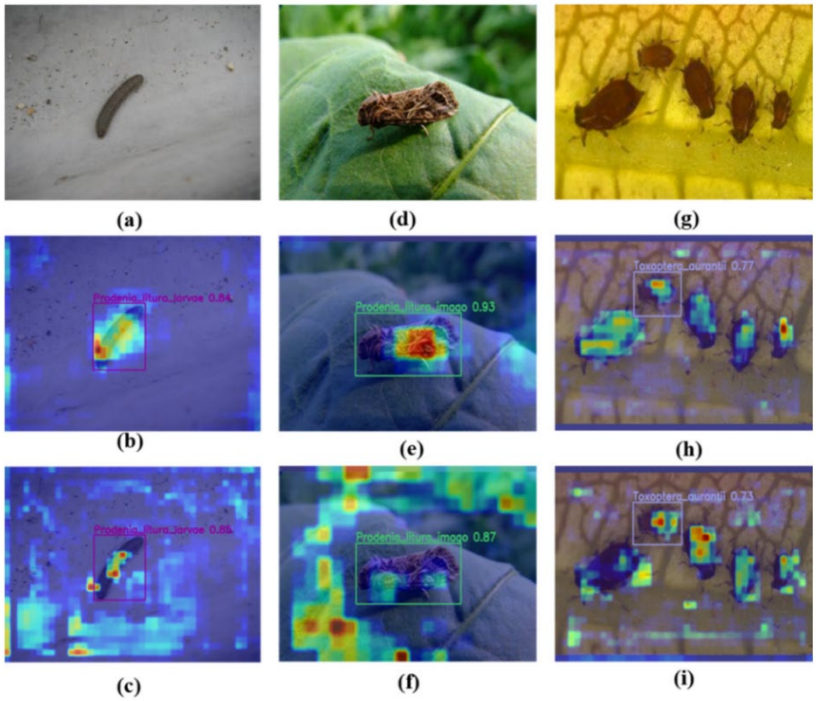
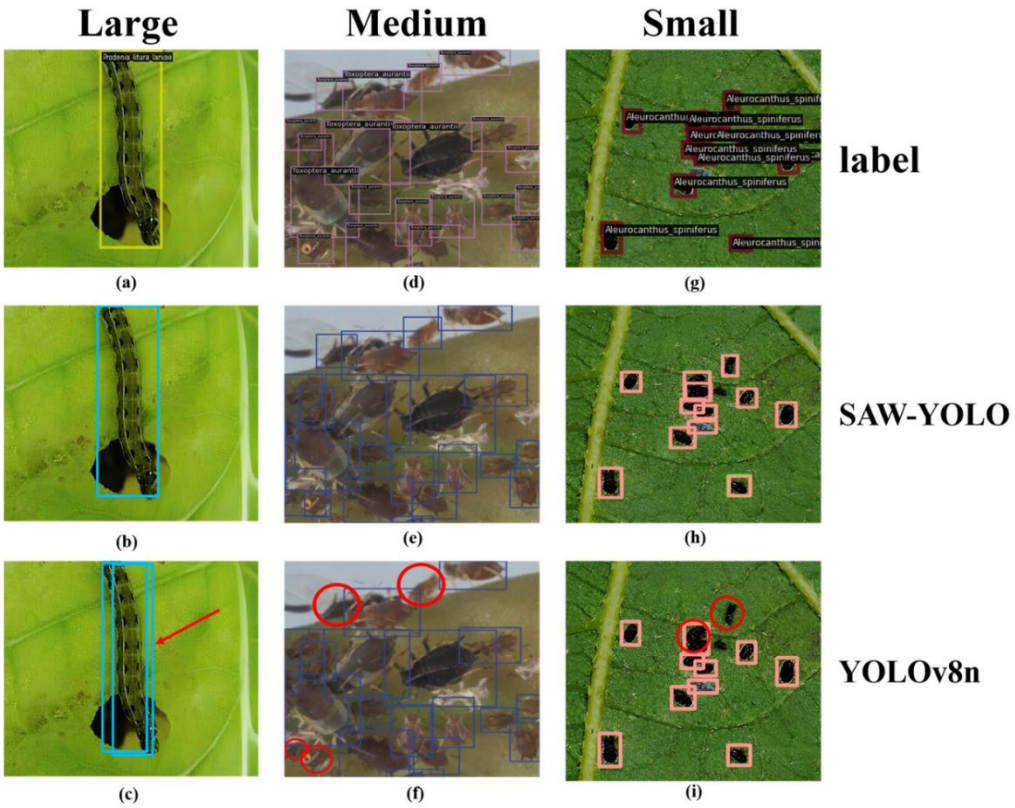


图 YOLOv8 病虫害监测效果

我们使用 OpenCV 技术、Pytorch 技术建立了一套专属的目标检测系统

我们使用原生 YOLO 格式根据 IP 102 提供的类别信息注释数据集。此外，我们将数据集以 7：2：1 的比例分为训练集、验证集和测试集，我们使用了 COCO 评估指标，并根据每个病虫害样本的像素面积足迹将其分配到基于大小的层（小、中或大）





9 软件设计

```
def __init__(self, parent=None):
    super().__init__(parent)
    self.setupUi(self)
    self.start_x = None
    self.start_y = None
    self.setAttribute(QtCore.Qt.WA_TranslucentBackground)
    self.setWindowFlags(Qt.FramelessWindowHint)  # 设置窗口标志：隐藏窗口边框
    # 显示视频
    frame_layout = QVBoxLayout(self.frame_5)
    self.frame_web_view = QFrame(self)
    frame_layout.addWidget(self.frame_web_view)
    self.web_view = QWebEngineView(self.frame_web_view)
    frame_layout2 = QVBoxLayout(self.frame_web_view)
    frame_layout2.addWidget(self.web_view)
    # 加载视频页面
    self.web_view.load(QUrl('http://192.168.43.52:8080/?action=stream'))
    self.web_view.loadFinished.connect(self.on_load_finished)
    # 订阅主题，实时更新
```

使用 QtWebEngineWidgets 中的 QWebEngineView 来显示摄像头云传输画面

```
def effect_shadow_style(self, widget):
    effect_shadow = QtWidgets.QGraphicsDropShadowEffect(self)
    effect_shadow.setOffset(0, 8)  # 偏移
    effect_shadow.setBlurRadius(48)  # 阴影半径
    effect_shadow.setColor(QColor(162, 129, 247))  # 阴影颜色
    widget.setGraphicsEffect(effect_shadow)

def effect_shadow_style2(self, widget):
    effect_shadow = QtWidgets.QGraphicsDropShadowEffect(self)
    effect_shadow.setOffset(0, 8)  # 偏移
    effect_shadow.setBlurRadius(48)  # 阴影半径
    effect_shadow.setColor(QColor(253, 139, 133))  # 阴影颜色
    widget.setGraphicsEffect(effect_shadow)

def effect_shadow_style3(self, widget):
    effect_shadow = QtWidgets.QGraphicsDropShadowEffect(self)
    effect_shadow.setOffset(0, 8)  # 偏移
```

界面设计

```
def takePhoto():
    img_url = "http://192.168.43.52:8080/?action=snapshot"

    response = requests.get(img_url)

    if response.status_code == 200:
        filename = "snapshot.jpg"
        with open(filename, "wb") as f:
            f.write(response.content)
        myWin.label_12.setPixmap(QPixmap("snapshot.jpg"))
        myWin.label_12.setAlignment(Qt.AlignCenter)
        print(f"Successfully downloaded {filename}!")
    else:
        print(f"Request failed with code {response.status_code}!")
```

截取某一张图片

```
    # 加载css渲染效果
    style_file = 'UIProgram/style.css'
    qssStyleSheet = QSSLoader.read_qss_file(style_file)
    self.setStyleSheet(qssStyleSheet)

    self.conf = 0.5
    self.unkonwn_text = '无法识别'

    # 视频与摄像头检测频率，每5帧检测一次
    self.detection_frequency = 5
```

设置视频流的检测频率

```
def signalconnect(self):
    self.ui.PicBtn.clicked.connect(self.open_img)
    self.ui.VideoBtn.clicked.connect(self.vedio_show)
    self.ui.CapBtn.clicked.connect(self.camera_show)
    self.ui.FilesBtn.clicked.connect(self.detact_batch_imgs)
    self.ui.tableWidget.cellClicked.connect(self.on_cell_clicked)
```

每个按钮的功能设计

```
# 加载检测模型
self.model = YOLO(Config.model_path, task='classify')
self.model(np.zeros((48, 48, 3)).astype(np.uint8), device=self.device)  #预先加载推理模型
```

加载模型

```
# 表格
self.ui.tableWidget.verticalHeader().setSectionResizeMode(QHeaderView.Fixed)
self.ui.tableWidget.verticalHeader().setDefaultSectionSize(40)
self.ui.tableWidget.setColumnWidth(0, 100)  # 设置列宽
self.ui.tableWidget.setColumnWidth(1, 300)
self.ui.tableWidget.setColumnWidth(2, 210)
self.ui.tableWidget.setColumnWidth(3, 140)
# self.ui.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)  # 表格铺满
# self.ui.tableWidget.horizontalHeader().setSectionResizeMode(0, QHeaderView.Interactive)
# self.ui.tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)  # 设置表格不可编辑
self.ui.tableWidget.setSelectionBehavior(QAbstractItemView.SelectRows)  # 设置表格整行选中
self.ui.tableWidget.verticalHeader().setVisible(False)  # 隐藏列标题
self.ui.tableWidget.setAlternatingRowColors(True)  # 表格背景交替
```

编写表格代码

```
file_path, _ = QFileDialog.getOpenFileName(None, '打开图片', './', "Image files (*.jpg *.jpeg *.png *.bmp)")
if not file_path:
    return

self.org_path = file_path
self.org_img = tools.img_cvread(self.org_path)

# 目标检测
t1 = time.time()
self.results = self.model(self.org_path, conf=self.conf)[0]
print(self.results)
```

调用 Qt 的 Qfilediolog 完成打开文件

进行目标检测

```
for file_name in os.listdir(directory):
    full_path = os.path.join(directory, file_name)
    if os.path.isfile(full_path) and file_name.split('.')[-1].lower() in img_suffix:
        img_path = full_path
        self.org_img = tools.img_cvread(img_path)
        # 目标检测
        t1 = time.time()
        self.results = self.model(self.org_img, conf=self.conf)[0]
        self.conf_list = self.results.probs.data.tolist()

        cls_index = self.results.probs.top1
        conf = self.conf_list[cls_index]
        if conf > self.conf:
            show_conf = '{:.2f}%'.format(conf * 100)
            res_name = Config.CH_names[cls_index]
```

通过循环读取一整个文件夹的图片，进行

检测，并给出检测速度，置信度和检测结果

以及病虫害防治等建议

未来规划

作物视觉识别技术升级：引入更先进的作物视觉识别技术，结合更多维度传感器提高对作物病害的识别准确度和速度，以便及时采取相应的防治措施。

数据采集和分析功能：随着产品的实际使用，后台病虫害识别专家库将不断丰富，增加数据采集和分析功能，帮助农民更好地了解作物生长情况，提供决策支持。

用户界面和操作体验改进：不断优化产品的用户界面和操作体验，使其更易于操作和维护，提高用户满意度，升级可以通过 OTA（Over-The-Air）实现。