Fruit classification

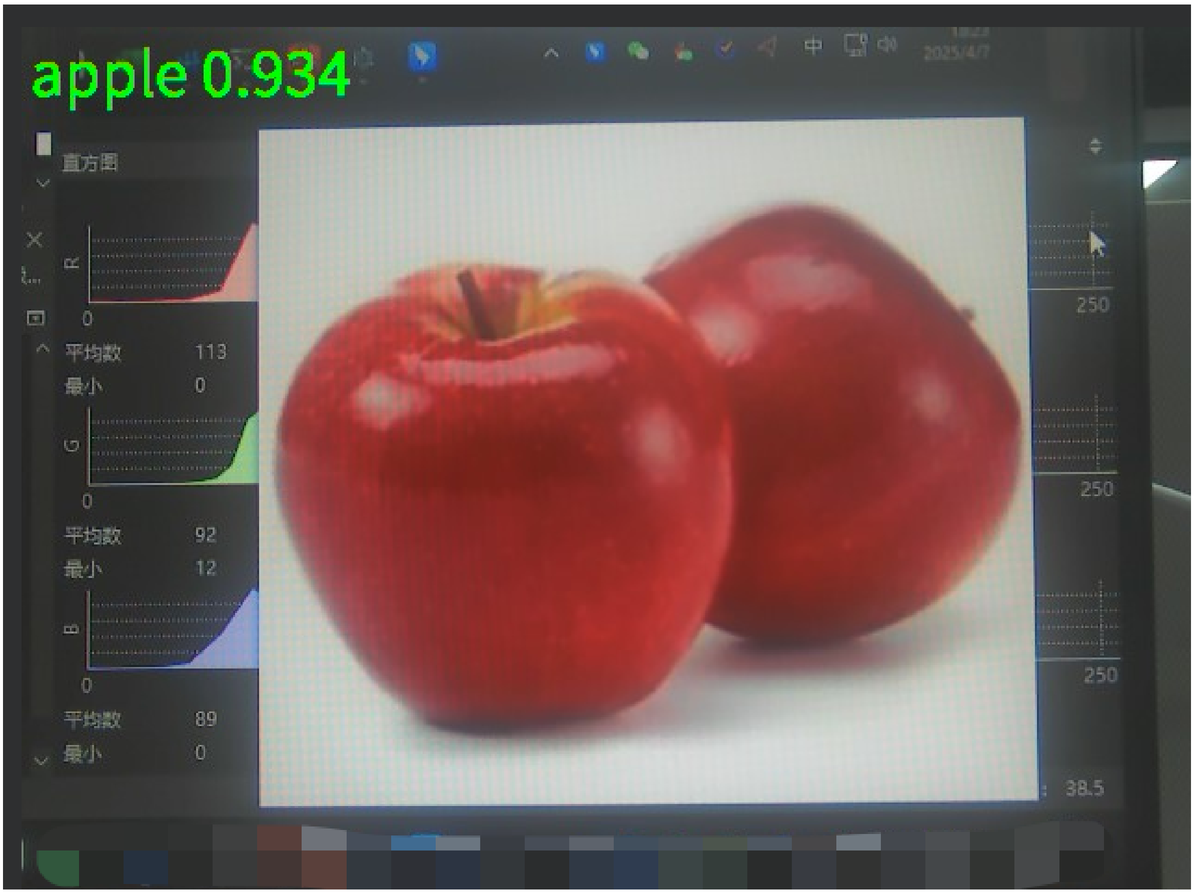
Effect Introduction

In this section, we will use a classification model trained based on yolov5 to perform simple fruit classification.

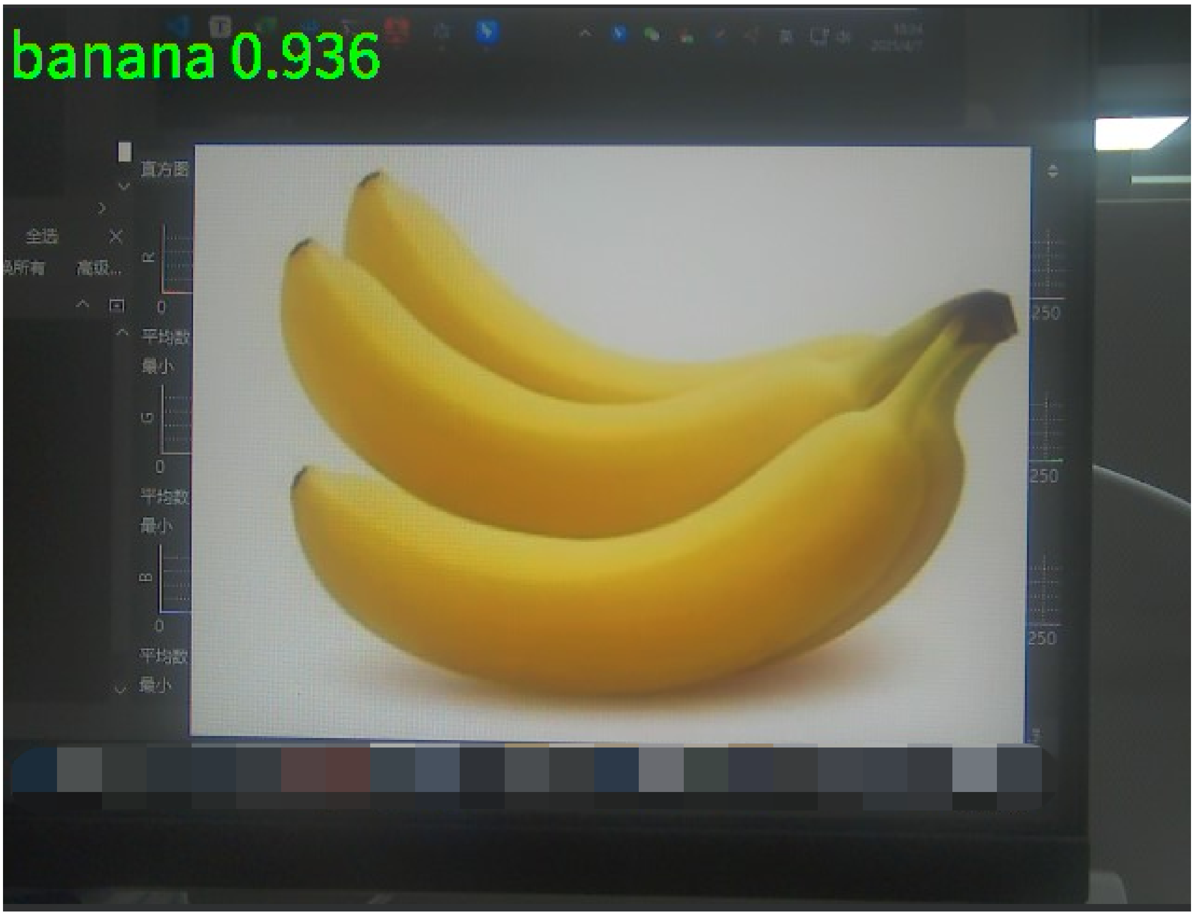
The training steps of this model can refer to the tutorial [13. Train your own model/Local deployment of yolo training environment]

> Note: Training requires a GPU environment and has a certain degree of operational difficulty

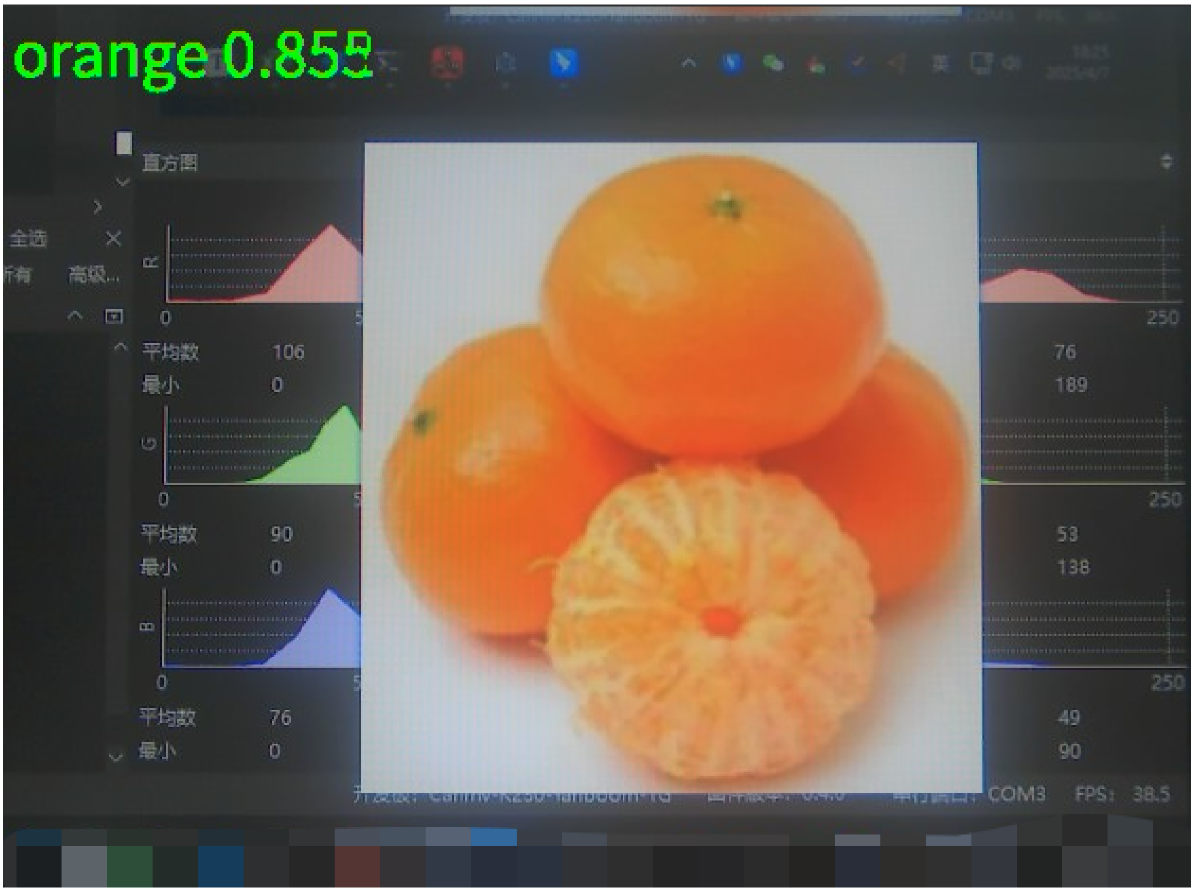
Identify Apples



Identify bananas



Identify oranges



Routine source code

```python
# 导入必要的库 / Import necessary libraries
from libs.PipeLine import PipeLine, ScopedTiming  # 导入PipeLine类和计时工具 / Import PipeLine class and timing tool
from libs.YOLO import YOLOv5  # 导入YOLOv5类 / Import YOLOv5 class
import os, sys, gc  # 导入操作系统、系统和垃圾回收相关库 / Import OS, system and garbage collection libraries
import ulab.numpy as np  # 导入numpy的兼容库ulab / Import numpy compatible library ulab
import image  # 导入图像处理库 / Import image processing library

if __name__ == "__main__":  # 主程序入口 / Main program entry

    # 设置图像和显示尺寸 / Set image and display sizes
    rgb888p_size = [1280, 720]  # 原始RGB图像尺寸 / Original RGB image size
    display_size = [640, 480]  # 显示尺寸 / Display size

    # 模型路径 / Model path
    kmodel_path = "/sdcard/kmodel/fruit_cls.kmodel"  # 水果分类模型路径 / Fruit classification model path

    # 标签列表 / Label list
    labels = ["apple", "banana", "orange"]  # 可识别的水果类别 / Recognizable fruit categories

    # 置信度阈值 / Confidence threshold
    confidence_threshold = 0.5  # 分类结果置信度阈值 / Classification confidence threshold

    # 模型输入尺寸 / Model input size
    model_input_size = [224, 224]  # 模型所需的输入图像尺寸 / Required input image size for the model

    # 初始化PipeLine / Initialize PipeLine
    pl = PipeLine(rgb888p_size=rgb888p_size,
                 display_size=display_size,
                 display_mode="lcd")  # 创建图像处理流水线 / Create image processing pipeline
    pl.create()  # 创建流水线资源 / Create pipeline resources

    # 初始化YOLOv5实例 / Initialize YOLOv5 instance
    yolo = YOLOv5(task_type="classify",  # 任务类型为分类 / Task type is classification
                 mode="video",  # 视频模式 / Video mode
                 kmodel_path=kmodel_path,  # 模型路径 / Model path
                 labels=labels,  # 标签列表 / Label list
                 rgb888p_size=rgb888p_size,  # 原始图像尺寸 / Original image size
                 model_input_size=model_input_size,  # 模型输入尺寸 / Model input size
                 display_size=display_size,  # 显示尺寸 / Display size
                 conf_thresh=confidence_threshold,  # 置信度阈值 / Confidence threshold
                 debug_mode=0)  # 调试模式关闭 / Debug mode off

    # 配置预处理过程 / Configure preprocessing
    yolo.config_preprocess()

    try:
        # 主循环，持续处理视频帧 / Main loop for continuous video frame processing
        while True:
            os.exitpoint()  # 检查退出点 / Check exit point

            # 计时整个处理周期 / Time the entire processing cycle
            with ScopedTiming("total", 1):
                # 获取一帧图像 / Get a frame
                img = pl.get_frame()

                # 执行推理 / Run inference
                res = yolo.run(img)

                # 在OSD层绘制结果 / Draw results on OSD layer
                yolo.draw_result(res, pl.osd_img)

                # 显示图像 / Display image
                pl.show_image()

                # 执行垃圾回收 / Perform garbage collection
                gc.collect()

    except Exception as e:
        # 异常处理 / Exception handling
        sys.print_exception(e)  # 打印异常信息 / Print exception information

    finally:
        # 清理资源 / Clean up resources
        yolo.deinit()  # 释放YOLO资源 / Release YOLO resources
        pl.destroy()  # 销毁PipeLine资源 / Destroy PipeLine resources
```

Code flow

Main working principle:

1. Get live video stream from camera
2. Preprocess each frame and resize it to fit the model input requirements
3. Feed the processed image into the neural network for inference
4. Filter classification results based on confidence threshold
5. Draw the recognition results on the display interface
6. Loop to process the next frame

Code structure

1. Import dependencies

```
from libs.PipeLine import PipeLine, ScopedTiming
from libs.YOLO import YOLOv5
import os,sys,gc
import ulab.numpy as np
import image
```

- `PipeLine`: Processing image pipeline
- `ScopedTiming`: Performance Timing Tool
- `YOLOv5`: Encapsulates the reasoning interface of the YOLO model

- `ulab.numpy`: NumPy compatible library under MicroPython, optimized for embedded devices

2. Configuration initialization

```
rgb888p_size=[1280,720]   # original image size
display_size=[640,480]   # Display size
kmodel_path="/sdcard/kmodel/fruit_cls.kmodel"  # Model path
labels = ["apple","banana","orange"]  # Category labels
confidence_threshold = 0.5   # confidence threshold
model_input_size=[224,224]  # Model input size
```

3. Pipeline and Model Instantiation

```
pl=PipeLine(rgb888p_size=rgb888p_size,display_size=display_size,display_mode="lcd")
pl.create()
yolo=YOLOv5(task_type="classify",mode="video",kmodel_path=kmodel_path,labels=labels,
           rgb888p_size=rgb888p_size,model_input_size=model_input_size,

display_size=display_size,conf_thresh=confidence_threshold,debug_mode=0)
yolo.config_preprocess()
```

4. Main Loop

```
try:
    while True:
        os.exitpoint()
        with ScopedTiming("total",1):
            img=pl.get_frame()      # Get a frame
            res=yolo.run(img)       # Run inference
            yolo.draw_result(res,pl.osd_img)  # draw result
            pl.show_image()          # Display image
            gc.collect()            # Garbage collection
```

5. Exception handling and resource release

```
except Exception as e:
    sys.print_exception(e)
finally:
    yolo.deinit()
    pl.destroy()
```