

# 深度学习

## 问题1

我现在打算用yolo模型来做巡线到十字路口根据数字选择分支，但我不大会，该怎么部署这个模型

## 问题2

我们怎么从yolov5s.pt 基础模型开始进行微调，可以从我给你提供的这些下载下来训练吗，用markdown输出

## 问题1

好的，当然。这是一个使用 Markdown 格式化后的版本，结构更清晰，便于阅读。

---

这是一个非常棒且非常典型的机器人综合项目，它结合了传统的计算机视觉（巡线）和深度学习（目标检测），是学习和实践的绝佳案例。

我们不建议自始至终都只用一个复杂的YOLO模型来完成所有事情（包括巡线和数字识别），因为这样效率很低，而且调试困难。

我们推荐一种更稳定、更高效的**混合策略**：

- **平时**：使用简单、快速的OpenCV颜色处理方法进行巡线。
  - **遇到十字路口时**：激活YOLO模型，专门用来识别数字，并根据结果做出决策。
- 

## 模型部署与开发分步指南

整个过程分为三大步：**1. 模型训练（在PC上） -> 2. 模型转换（在PC上） -> 3. 板端部署（在机器人上）。**

### 第一步：数据准备与模型训练（在PC上完成）

您的首要任务是训练一个能认识数字的YOLO模型。

1. **收集数据**：

- 在您要比赛或运行的场地上，用机器人的摄像头**拍摄大量包含数字（例如 "1", "2", "3"）的图片**。
- **关键：**图片要包含各种角度、光照条件、距离远近、甚至轻微模糊的情况，数据越多，模型越鲁棒。至少为每个数字类别准备100-200张图片。

## 2. 标注数据：

- 使用标注工具（如 `labelimg` 或在线的 `CVAT`、`Roboflow`）为您的图片打上标签。
- 操作很简单：在每张图片的数字周围画一个框（`bounding box`），并给它指定一个类别（例如，框住数字"1"，就给它打上"1"的标签）。

## 3. 选择并训练模型：

- **选择模型：**为了在边缘设备上流畅运行，请选择轻量级的YOLO模型，例如 **YOLOv5s** 或 **YOLOv8n**。
- **开始训练：**使用您标注好的数据集，在带有GPU的电脑上开始训练。YOLOv5和YOLOv8都有非常成熟的开源代码库，您只需要按照它们的官方文档指引，配置好数据集路径，就可以一键开始训练。
- **训练结果：**训练完成后，您会得到一个模型权重文件，通常是 `.pt` 格式。这个文件就是您定制的、能认识数字的AI模型。

# 第二步：模型转换（在PC上完成）

这一步是将您在PC上训练好的 `.pt` 模型，转换成能在地平线BPU上高效运行的格式。

1. **准备工具：**下载并安装地平线官方的“**天工开物**”(**Togetheo**) **AIoT DDK** 开发工具包。
2. **执行转换：**使用DDK中提供的模型转换工具，对您的 `.pt` 文件进行转换和**量化**。这个过程会生成一个后缀为 `.bin` 或 `.hbm` 的文件。
  - 这个过程是整个部署流程中最具技术挑战性的一环，您需要**严格遵循地平线官方的模型转换文档**来操作。

# 第三步：板端部署（机器人上的代码）

这是项目的核心，我们需要编写一个**状态机（State Machine）**来控制机器人的行为。

## 1. 定义机器人的三种状态：

- `FOLLOWING_LINE` (正在巡线)
- `AT_CROSSROADS` (到达路口，准备识别)
- `TURNING` (正在转弯)

2. **编写代码逻辑** 在您的ROS2节点中（可以基于之前的 `follower` 节点修改），`image_callback` 函数需要根据当前状态执行不同操作。

Python

# 这是一个简化的伪代码结构

```
class AutonomousNavigator(Node):
```

```
    def __init__(self):
```

```
        # ... 此处进行各种初始化 ...
```

```
        super().__init__('autonomous_navigator')
```

```
        self.state = "FOLLOWING_LINE"
```

```
        # 在这里加载转换好的 .hbm 模型
```

```
        self.yolo_model = bpu_infer_lib.Infer(...)
```

```
        self.yolo_model.load_model("您的数字识别模型.hbm")
```

```
        # ... 其他初始化 ...
```

```
    def image_callback(self, msg):
```

```
        image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
```

```
        if self.state == "FOLLOWING_LINE":
```

```
            # --- 执行OpenCV颜色巡线 ---
```

```
            # 使用您之前写的基于颜色阈值的巡线代码。
```

```
            # 它简单、快速、可靠。
```

```
            # --- 判断是否到达十字路口 ---
```

```
            # 判断条件可以是：
```

```
            # 1. 检测到的黄线突然消失。
```

```
            # 2. 检测到一条横向的黑线（路口停止线）。
```

```
            if self.is_at_crossroads(image):
```

```
                self.get_logger().info("到达十字路口，切换到识别状态！")
```

```
                self.stop_robot() # 让机器人停下或减速
```

```
                self.state = "AT_CROSSROADS"
```

```
        elif self.state == "AT_CROSSROADS":
```

```
            # --- 在这个状态下，激活YOLO模型 ---
```

```
            # 1. 对图像进行预处理（缩放、填充等），使其符合模型输入要求。
```

```
            # 2. 调用 self.yolo_model.forward() 进行推理。
```

```
            # 3. 解析模型的输出，找到识别到的数字和它的置信度。
```

```
            recognized_number = self.run_yolo_inference(image)
```

```
            if recognized_number is not None:
```

```
                self.get_logger().info(f"成功识别数字：{recognized_number},
```

```
                准备转弯！")
```

```
                self.target_direction =
```

```

self.decide_direction(recognized_number)
    self.state = "TURNING"

elif self.state == "TURNING":
    # --- 根据识别结果执行转弯动作 ---
    # 例如:
    # if self.target_direction == "left":
    #     self.turn_left()
    # elif self.target_direction == "right":
    #     self.turn_right()

    # 在转弯动作完成后（可以通过延时或里程计判断）
    self.get_logger().info("转弯完成，恢复巡线状态！")
    self.state = "FOLLOWING_LINE"

```

### 3. 调试建议

- **分步测试：**先确保您的纯OpenCV巡线模块能稳定工作。再单独测试YOLO识别模块，看看它能否在您把数字卡片放到摄像头前时正确识别。
- **打印日志：**在代码的关键位置（如状态切换、识别出数字时）使用 `self.get_logger().info()` 打印日志，方便您实时了解机器人当前的状态。
- **可视化：**使用 `rqt_image_view` 查看机器人摄像头传回的图像，以及您处理后（画上检测框）的图像，这对于调试至关重要。

这个方案将一个复杂问题分解成了多个简单、可控的子任务，是实现您目标的稳健路径。祝您项目顺利！

## 问题2

好的，我们来详细梳理一下如何从零开始，使用 `yolov5s.pt` 基础模型进行微调，来训练一个能识别您自定义数字的模型。

首先，回答您最关键的问题：

### 一个至关重要的前提：`.bin` 文件无法用于训练！

您从地平线模型库下载的那些 `.bin` 文件，是**不能**用来进行训练或微调的。

- **.pt 文件（您需要自己准备的）**
  - 这是PyTorch框架保存的模型**权重文件**。您可以把它想象成模型的“源代码”或“配方”。它包含了模型的所有参数，这些参数可以在训练过程中被不断学习和优化。
  - 这是我们进行“微调训练”的起点。

- **.bin / .hbm 文件（您下载的/最终生成的）**
  - 这是经过地平线“天工开物”工具链**编译、量化和优化后的可执行二进制文件**。您可以把它想象成一个编译好的 `.exe` 程序。
  - 它的目的是在地平线BPU上高效运行，其内部结构已经固化，**无法再进行任何训练或修改**。

所以，我们的目标是：**获取官方的 `yolov5s.pt` -> 用您的数据训练它 -> 得到您自己的 `digits.pt` -> 再把它转换为 `.bin` 文件。**

---

## YOLOv5s 微调详细步骤 (在PC上完成)

整个微调过程都在带有NVIDIA GPU的电脑上进行，而不是在RDK X5开发板上。

### 第一步：搭建训练环境 (在PC上)

1. **克隆YOLOv5官方代码库** 打开终端，运行git命令，将YOLOv5的完整项目代码下载到您的电脑。

Bash

```
git clone https://github.com/ultralytics/yolov5.git
```

2. **进入项目目录并安装依赖** 这个项目依赖PyTorch, OpenCV等库。 `requirements.txt` 文件已经列出了所有需要的库。

Bash

```
cd yolov5
pip install -r requirements.txt
```

3. **下载 `yolov5s.pt` 预训练权重** 这是我们微调的“基础”。它已经在大型数据集（COCO）上进行了预训练，具备了很好的特征提取能力，我们只需要在此基础上教它认识数字即可。您可以直接从YOLOv5的GitHub Releases页面下载，或者使用命令：

Bash

```
wget
https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt
```

## 第二步：准备自定义数据集

这是最耗时但也是最重要的一步。

1. **创建文件夹结构** 在 `yolov5` 目录下，创建一个 `dataset` 文件夹，并组织成如下结构：

```
yolov5/
├── dataset/
│   ├── images/
│   │   ├── train/ <-- 存放训练图片
│   │   └── val/   <-- 存放验证图片
│   └── labels/
│       ├── train/ <-- 存放训练标签
│       └── val/   <-- 存放验证标签
└── ... (其他yolov5文件)
```

2. **收集并划分图片** 将您拍摄的所有数字图片，按大约8:2的比例，分别放入 `images/train` 和 `images/val` 文件夹。

3. **标注图片**

- 使用 `labelimg` 等标注工具，为 `images` 文件夹下的每一张图片进行标注。
- 对于每张图片，在数字周围画框，并指定类别。例如，数字“1”是第0类，数字“2”是第1类，数字“3”是第2类。
- 标注完成后，`labelimg` 会在 `labels` 文件夹下生成与图片同名的 `.txt` 文件。例如 `pic1.jpg` 会对应生成 `pic1.txt`。

4. **创建数据集配置文件 (.yaml)** 在 `yolov5/data` 目录下，创建一个新的文件，例如 `digits.yaml`，内容如下：

YAML

```
# 训练集和验证集的图片路径
train: ../dataset/images/train
val: ../dataset/images/val

# 类别数量
nc: 3

# 类别名称（顺序必须和标注时一致！）
names: ['1', '2', '3']
```

## 第三步：开始微调训练

一切准备就绪后，在 `yolov5` 的根目录下，打开终端，运行以下命令开始训练：

Bash

```
python train.py --img 640 --batch 16 --epochs 100 --data data/digits.yaml --weights yolov5s.pt --name digits_run
```

### 命令参数解释：

- `--img 640`：训练时输入的图片大小，保持640以匹配地平线模型的要求。
- `--batch 16`：批处理大小，根据您的GPU显存大小调整，如果显存小，就调小这个值（如8或4）。
- `--epochs 100`：训练轮次，表示要将整个数据集完整训练100遍。您可以先从较小的轮次开始尝试。
- `--data data/digits.yaml`：指向我们刚刚创建的数据集配置文件。
- `--weights yolov5s.pt`：**这是微调的关键！** 它告诉程序从 `yolov5s.pt` 的权重开始训练，而不是从零开始。
- `--name digits_run`：为本次训练任务命名，训练结果会保存在 `runs/train/digits_run` 文件夹下。

## 第四步：获取训练结果

训练过程会在终端显示进度条和各项指标。训练完成后：

- 找到 `yolov5/runs/train/digits_run/weights/` 目录。
- 里面会有一个 `best.pt` 和 `last.pt` 文件。`best.pt` 是训练过程中验证集上表现最好的模型。

**这个 `best.pt` 文件，就是您最终得到的、认识您自定义数字的专属模型。**

接下来，您就可以拿着这个 `best.t` 文件，去执行我们之前讨论的“**第二步：模型转换**”，用“天工开物”DDK将其转换为可以在RDK X5上运行的 `.bin` 文件了。