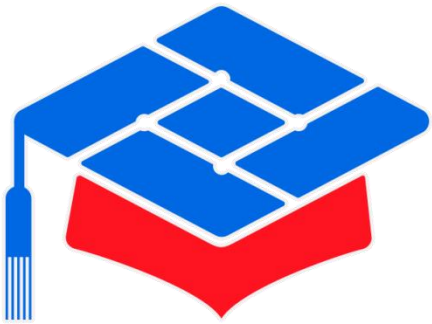


第七届

全国大学生集成电路创新创业大赛

报告类型：设计报告

参赛杯赛：第七届全国大学生集成电路创新创业大赛-算能杯

作品名称：“Overwatch”基于 TPU 实现的无人驾驶机器人

队伍编号：CICC5557

团队名称：只要名字长就不会有人忘记对不队

目录

绪论：

无人驾驶技术一直是人工智能领域的研究热点，其在改善交通安全、提高出行效率和降低能源消耗方面具有巨大潜力。随着计算能力和算法的不断进步，智能无人驾驶技术取得了显著的突破。本文将介绍我们团队所开发的基于 TPU 的智 能无人驾驶机器人系统，以及其中关键的 VGG 模型移植方案。

Overwatch 是一款基于 TPU（Tensor Processing Unit）的智能无人驾驶机器人，采用 ROS（Robot Operating System）作为机器人操作系统，并搭载多种市面上常见的 AI 模型。这些 AI 模型通过集成与优化，使机器人实现了智能化的 无人驾驶功能。在项目研究的后期，我们团队专注于 VGG（Visual Geometry Group）的模型移植方案，该方案具备高度灵活性，允许使用任意数据集来快速训练 VGG 模型文件。

通过结合团队研发的工具链和算能提供的 SDK 工具，我们成功为 VGG 模型的 快速部署提供了可行且高效的方法。这种 VGG 模型的部署迁移方法在网络公开资 料中是首次被提及，为 TPU 芯片的开发提供了更多宝贵的思路与实践经验。

在无人驾驶机器人的设计中，我们充分利用了 TPU 芯片的强大计算能力，使 机器人能够在实时场景下高效地处理大量数据，实现智能决策和精准控制。ROS 的支持使我们能够轻松管理机器人的各种传感器数据，执行复杂的任务规划和路 径规划。在实际应用中，我们的系统展现了卓越的性能和稳定性。通过我们的研究成果，成功将 VGG 模型应用于无人驾驶场景，解决了计算资源有限和部署繁琐的问题。同时，我们的工作也对 TPU 芯片的应用探索提供了新 的思路，为智能无人驾驶技术的发展迈出了重要的一步。我们对未来进一步优化 和扩展我们的系统充满信心，不断努力实现更加智能、安全的无人驾驶技术。

1. 系统应用场景说明

机器人通过无人驾驶技术，智能识别技术，可以用于各类人工驾驶的场景，本项目旨在实现机器人各项功能，具有数字化地图测绘，包括医疗援助，自动充电，违章检测在内的校园巡逻功能。

① 数字化地图测绘：

高精度地图，即高分辨率地图，一方面指地图的绝对精度更高；另一方面指地图能够格式化存储交通场景中的各种交通要素[1]。机器人通过雷达收集环境数据，将路线及障碍物转化为数字信息，通过高精度地图，机器人可以迅速规划全局路线。高精地图也可作为校园建筑测绘，排除消防隐患的依据。

② 无人驾驶：

无人驾驶分为，违章检测，医疗援助，自动充电三大子功能，违章检测可以检测环境中的违停车辆，并对其车牌进行识别和上传，医疗援助功能接收援助请求，自主完成医疗工具的配送，自动充电技术通过电量检测，当电量不足时，可以使机器人自主前往就近充电站进行充电。

③ AI 模型快速部署：

由于机器人在不同场景下，对不同的目标和环境有不同的检测需求，需要一套高效且可移植的模型生成及部署方案，本项目应用了如 volov5，lpnet，resnet 等市面上常见的模型应用，同时使用 vgg16 模型，首创出一套可以快速生成，训练，转换，基于 TPU 的推演的应用框架，完成了自定义模型的快速部署以适应需工作环境的功能。

2. 系统硬件和软件架构说明

2.1 系统硬件架构

系统硬件由主控板、电机驱动板、电机、电池、 轮胎，摄像头，雷达等组成。

从上到下依次为传感器层，控制层、电机驱动层，传感器层使用 Astra Pro Plus 体感深度相机 RPLIDAR A1 雷达，控制层分为上位机系统和下位机系统，在上位机的选择上，选择了搭载了算能自主研发的第三代 TPUBM168 芯片的少林派开发板，采用少林派作为上位机的方案对于视觉处理和大量的数据处理具有巨大的优势。下位机则选择了系统资源丰富的 ESP32C3_UNO 峨眉派开发板。峨眉派对于多路 PWM 控制模块拥有集成库，开发较为容易，且能耗低，在使用时通过 PlatformIO 进行编译烧录，调用 Arduino 和 Hal 库实现对电机的控制和与上位机之间的通信。直流减速电机以及电机控制模块，采用 L289N 电机驱动方案。通过 PWM 波控制电机转速。项目软件工作架构如下图所示。

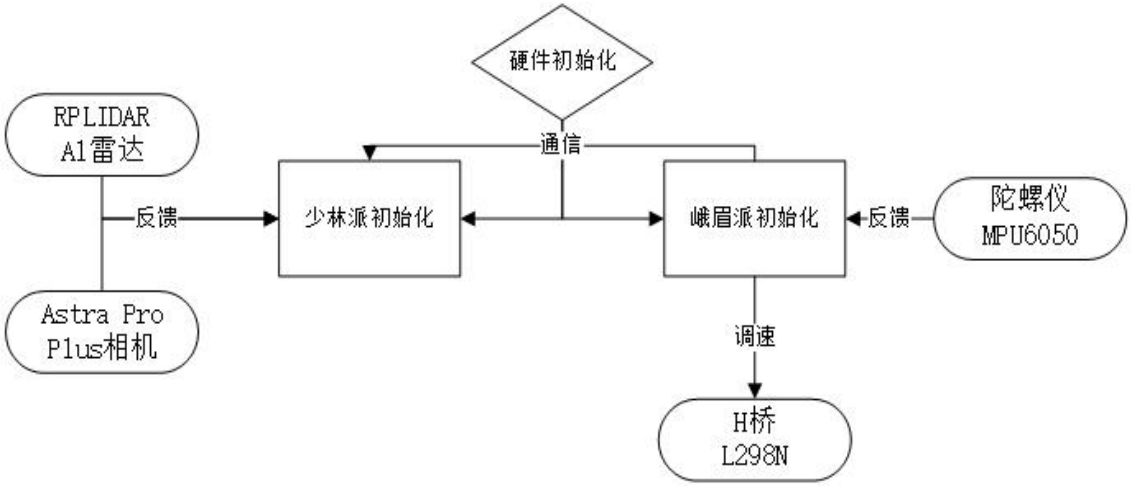


图 1 系统硬件框图

2.2 系统软件架构

软件框架围绕数字化地图测绘和巡逻驶两大功能主要使用了无人驾驶和雷达测绘两大核心技术，巡逻功能采用了无人驾驶，目标检测，路线规划等技术，项目软件工作架构如图所示。

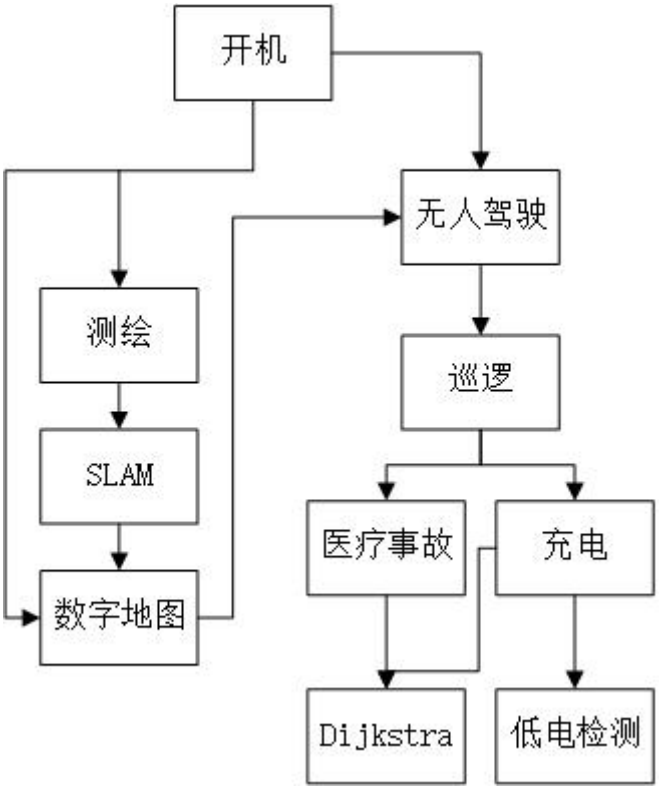


图 2 系统软件件框图

3. 关键硬件和软件技术实现方案

3.1 硬件技术实现方案:

本项目的硬件技术实现方案分两个部分来介绍。

3.1.1 下位机：

本项目底盘以峨眉派为核心，通过引脚向 L298N 输入 PWM 信号，串口读取陀螺仪数据，实现机器人的 PID 调速。

3.1.2 上位机：

地图测绘主要通过遥控来控制机器人移动，通过雷达检测周围环境构建数字化地图。在巡逻时将相机识别到的图像传回少林派，少林派对数据分析识别后控制机器人完成各个功能。

3.2 软件技术实现方案:

3.2.1 数字化地图测绘实现：

SLAM（Simultaneous Localization and Mapping，同时定位与地图构建）是一种用于机器人导航和环境感知的技术，它可以实现机器人在未知环境中同时确定自身位置和构建环境地图。SLAM 的可视化界面提供了对 SLAM 系统实时结果的可视化展示，以便用户能够直观地观察机器人的定位和地图构建过程。

数据获取：传感器采集环境的视觉或几何数据，并将其传输给 SLAM 系统进行处理。

数据处理：系统根据传感器数据的处理结果，估计机器人在环境中的位置和姿态，并同时构建环境的地图。定位是通过将当前的传感器数据与之前的数据进行比较和匹配来实现的，而地图构建则是通过整合多个时刻的传感器数据来逐步构建环境的结构和特征。

可视化展示：

通过可视化界面，用户可以直观地观察机器人在环境中的运动轨迹、定位精度以及环境地图的构建情况。可视化工具能够接收 SLAM 系统发布的图像话题，并将其显示在图像窗口中，以便用户观察和分析。

最终利用 ROS 中的地图构建模块，依据消防机器人[2]通过激光雷达构建地图的方案，按照实际环境特征选择 Gmapping 算法，Hector 算法或 Karto 算法将机器人运动轨迹的位置点及传 感器测量数据集转化为地图模型。

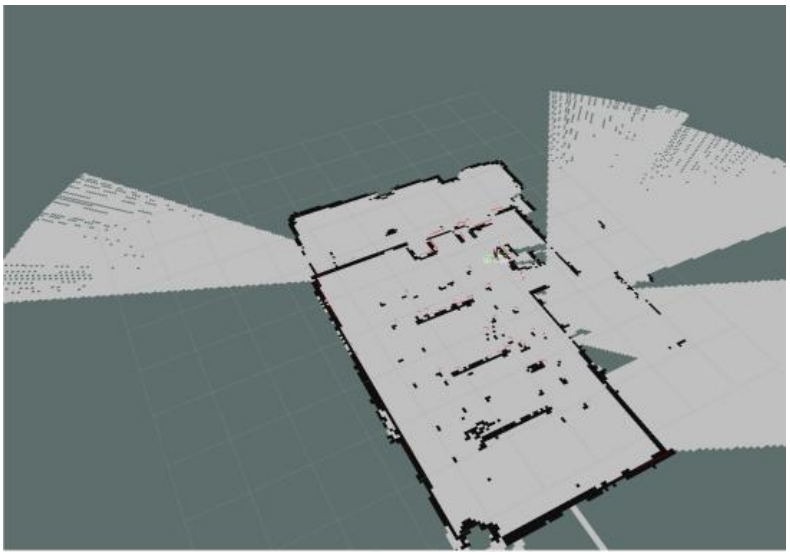


图 3 真实环境下使用激光雷达建模

3.2.2 无人驾驶：

无人驾驶功能由自主巡逻，目标检测，医疗援助，低电返航三个子功能组成，

①巡逻：

ROS（Robot Operating System）是一个灵活、模块化的机器人操作系统，它提供了一系列的工具、库和软件框架，用于帮助开发者构建和控制机器人系统。

ROS 的设计理念是模块化和分布式的，它将机器人系统拆分成多个独立的模块，每个模块负责不同的功能。这些模块可以是传感器驱动、运动控制、感知处理、路径规划等等。开发者可以根据自己的需求选择和组合这些模块，构建出符合自己要求的机器人系统。

ROS 提供了一套强大的通信机制，使得不同模块之间可以进行数据交换和消息传递。这种通信机制基于发布-订阅模式，即一个模块可以发布消息，其他模块可以订阅这些消息并进行相应的处理。这种松耦合的通信方式使得开发者可以更加灵活地组织和扩展机器人系统。

关于导航部分对高精度雷达的使用，需要使用到实 时动态（载波相位差分技术[3]修正误差，机器人能够利用数字地图得到机器人坐标后，障碍物检测以神经网络学习为基础,对障碍物进行预警。方案利用神经网络进行检测。

②识别：

由于在无人驾驶中的雷达，避障等功能占用了处理器的大量资源，。因此我们采用了轻量化的 YOLOv5 算法对本功能的目标进行检测。我们通过深度可分离卷积代替普通卷积层[4]的方法减少内存，在一定程度上实现轻量级 YOLOv5 算法。

除此之外我们还利用了 Retinaface 和 YOLO3D 对人脸和 3d 车辆进行检测。

Retinaface 会先对要识别的图片通过 Resnet50 或者 mobilenet 进行特征提取。然后再使用 SSH 和 FPN 进行特征强化，其次使用 ClassHead、BoxHead、LandmarkHead 网络从特征获取预测结果。最后，对预测结果进行 decode 解码并通过 NMS 非极大抑制去除重复检测值得出最终结果。以下是对网络框架结构源码的详细解析。

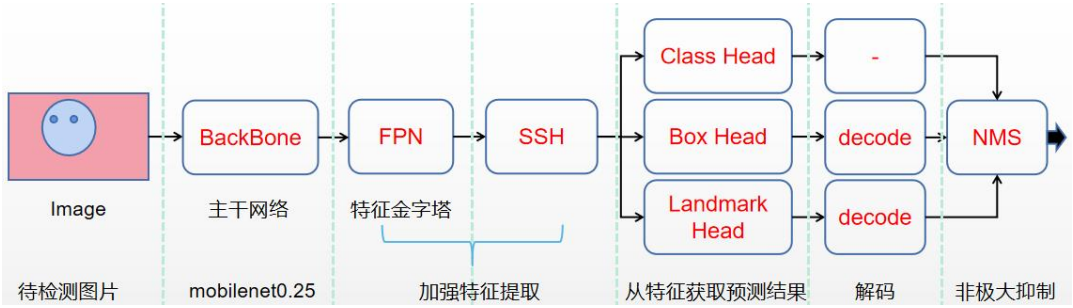


图 4Retinaface 模型框架

我们首先安装完成开源模型的下载和配置。之后，我们还需要识别的数据集和训练完成的模型。由于我们用的是 BM164，用 pytorch 训练完后的模型还需要转换成 bmodel 模型。下面是我们通过小车拍摄到的画面经过 Retinaface 处理后的结果。

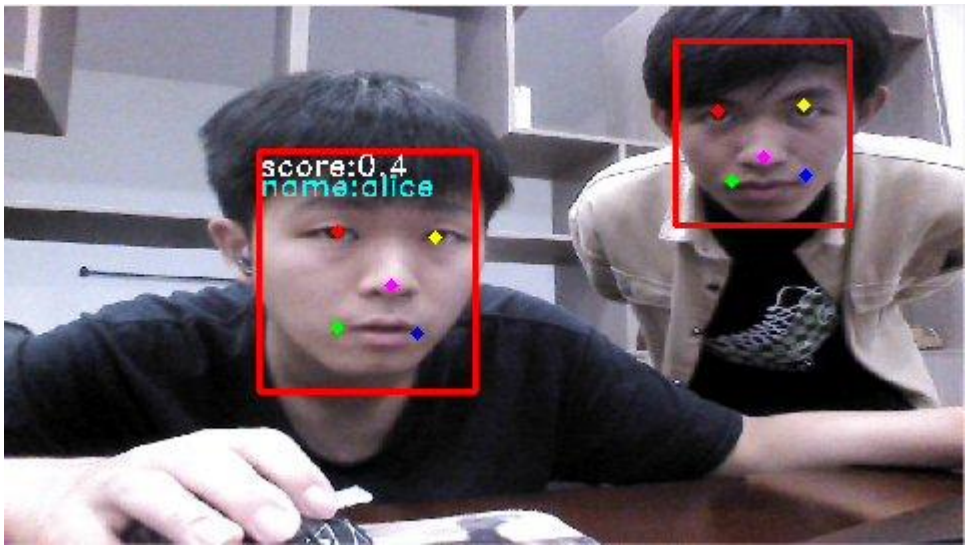


图 4Retinaface 输出图片

YOLO3D 是一种基于 YOLO 算法的三维物体检测方法，它结合了 2D 物体检测和 3D 物体定位的能力。下面将详细介绍 YOLO3D 的工作原理和特点。

YOLO3D 的网络结构与 YOLO 类似，由卷积层、池化层和全连接层组成。它采用了多尺度特征融合的方法，通过不同层次的特征图来检测不同尺寸的物体。输入和输出： YOLO3D 的输入是一张 RGB 图像，输出是一系列边界框和相应的类别概率、三维位置和尺寸信息。每个边界框由 5 个坐标值表示，分别是边界框的中心坐标、宽度、高度和深度。物体检测和定位： YOLO3D 通过在输入图像上的网格中预测边界框来进行物体检测。每个网格预测出多个边界框，每个边界框都有一个类别概率，表示该边界框中物体属于不同类别的概率。

与 YOLO 不同的是，YOLO3D 还会为每个边界框预测出物体的三维位置和尺寸信息。通过使用 3D 卷积和 3D 池化操作，YOLO3D 可以从输入图像中提取出物体的三维特征，然后通过全连接层进行预测。以下是我们使用 yolov3d 对道路上的小车进行检测。

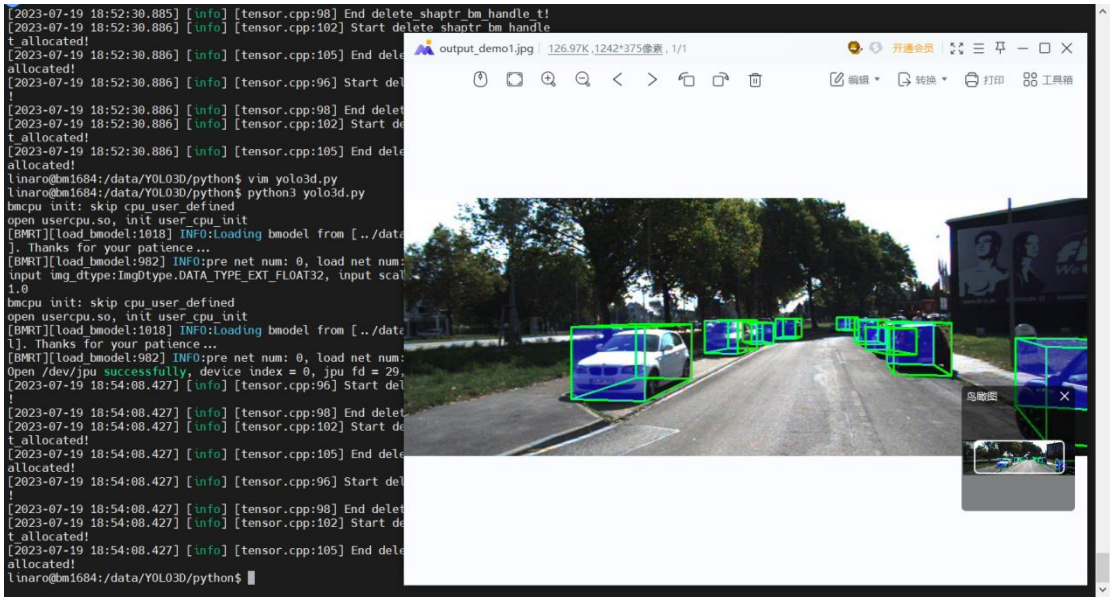


图 5YOLO3D 输出

③：低电量保护功能实现：

电源检测模块检测电池电量，结合机器人当前位置，计算出是否触发机器人低电量保护，机器人根据数字地图选择最近充电点返回充电。如何精准的返回充电站作为一大难题亟需解决，我们采用了雷达导航方案，使用之前雷达建图的地图信息，通过 2D NAV GOAL 功能实现较为精准的导航。

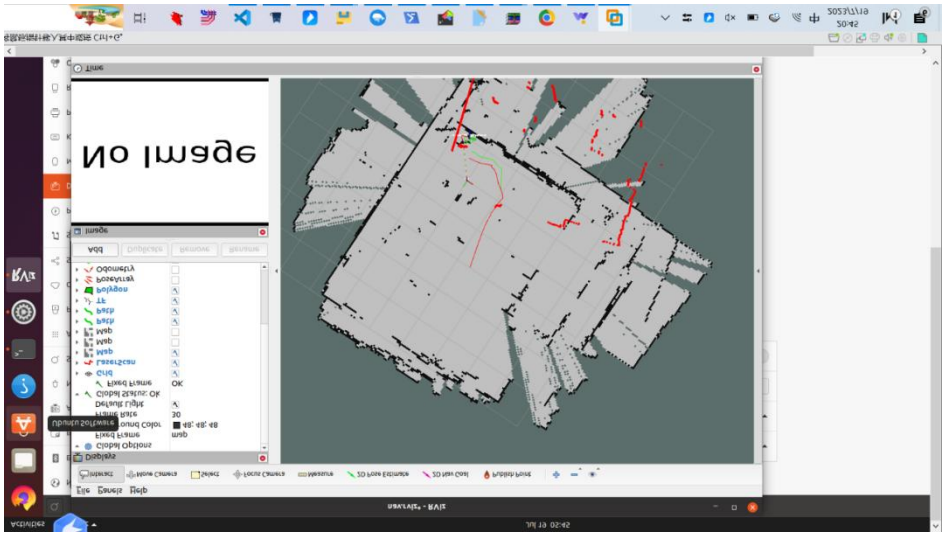


图 6 真实环境下使用激光雷达定点导航

④：医疗援助

机器人随身携带医疗箱，在校师生可以通小程序预约医疗帮助，在获取求助信息后，机器人会优先处理援助任务，利用数字地图快速确定求助者位置，若求助信

号不唯一，则机器人收集信息后利用 Dijkstra 算法[5]计算最优路径，到达**目的地**附近后机器人会进行广播，方便求助者及时取得医疗工具。

3.2.3AI 模型快速部署：

此功能的是继初赛后的主要研究方向，各类成熟 ai 模型的移植与应用，为上一项无人驾驶功能提供了更多的可行性，我们使用了 lpnet 对车牌进行检测，使用了 yolov3d 对机器人周围的 3d 物体进行识别，ai 模型的应用为机器人的大脑注入源源不断的动力以保证无人驾驶在各种场景下的稳定性，对于不同场景，如学校，工地，商场，处理传统的目标检测，巡线，导航功能，机器人应当对不同场景中的特殊标记也具有一定的识别功能，项目中使用了 Vgg16 神经网络，以 pytorch 为框架，开发了一套适用任意数据集的模型生成，转换，部署，推理应用。用户只需要提供符合规定结构的数据集，就可以快速生成，并部署特定的 AI 模型，并使用项目中提供的推理工具，进行基于 TUP 的模型推理。我们按照官方文档的要求，自主编写了 vgg16 的移植文档，附在附录中，以下是对该功能的重点介绍。

① 简介：

VGG16 是一种深度卷积神经网络模型，由牛津大学的研究团队开发。它在 2014 年的 ImageNet 图像分类竞赛中取得了显著的成绩，被广泛应用于计算机视觉任务中，如图像分类、目标检测和图像分割等。VGG16 的网络结构相对简单，由重复堆叠的卷积层和池化层构成。这种简单的结构使得网络易于理解和实现，同时也降低了出错的可能性。结构清晰的 VGG16 模型也为后续的改进和扩展提供了便利。

② 数据集

我们首先是拍摄了校园中的路标路牌作为数据集，但这样的数量是远远不够的，因此我们团队编写了一套爬虫程序可以在网络上爬取我们所需求的数据集，但对于本次所识别的路牌，网络上的数据太过于杂乱，我们需要不断筛除爬取数据中差异过大的图片。此外为了提高准确率，我们也找到了鲸盒社区中的一份数据集以供使用。

我们设计该模型具有很好的移植性，只要数据集放置如下结构即可训练：

下载数据

|　├──类 1

|　├──类 2

|　.......

|　└──类 n

③模型训练环境：

编程语言：Python 3.6.13
深度学习主要依赖库：
Torch 1.8.0+cu113
torchvision 0.11.3 ⑧
NumPy ③④⑤⑥⑦
1.19.5
pandas
1.1.5
数据集爬虫主要依赖库：
josn
Request
操作系统：Windows 10
开发工具：vscode,jupyter

④模型的训练：

我们基于 pytorch 框架，由于我们在后续的模型转换中发现了 droupout 算子由于版本兼容问题,出现了不支持转化问题,我们团队优化了 pytorch 自带的 vgg16 模型，通过 GPU 对该模型进行训练。模型训练 50 批次。生成 pth 文件。在模型训练中我们使用了数据增强技术和迁移学习技术：

```python
train_transforms = transforms.Compose([
    transforms.Resize([224, 224]),  # 将输入图片resize成统一尺寸
    transforms.RandomHorizontalFlip(), # 随机水平翻转
    transforms.ToTensor(),          # 将PIL Image或numpy.ndarray转换为tensor，并归一化到[0,1]之间
    transforms.Normalize(           # 标准化处理-->转换为标准正太分布（高斯分布），使模型更容易收敛
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])  # 其中 mean=[0.485,0.456,0.406]与std=[0.229,0.224,0.225] 从数据集
])
```

图 7 数据增强技术

迁移学习：使用大型标准数据集对模型进行训练，之后将模型学习到的通用特征的知识迁移到目标任务中，此时再利用目标任务的样本对迁移的知识进行微调并解决具体的问题。

```python
model1 = models.vgg16(pretrained=True).to(device)
vgg=model1.features#获取vgg16的特征提取层
for param in vgg.parameters():
    param.requires_grad = False
#print(vgg16.classifier)
class MyVggNet(nn.Module):
    def __init__(self):
        super(MyVggNet, self).__init__()
        # 预训练的vgg16特征提取层
        self.vgg = vgg
        # 添加新的全连接层
        self.classify = nn.Sequential(
            nn.Linear(25088, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 5),
            nn.Softmax(dim=1)
        )
```

图 8Vgg 中的迁移学习技术

⑤pth 文件的 jit 化

在 PyTorch 中，torch.jit.trace 函数用于将模型转换为 Torch 脚本（Torch Script）。Torch 脚本是一种中间表示形式，可以提供模型的静态图形表示，从而实现更高效的模型部署和推断。

我们团队开发的 pt JIT 化工具可以将 pt 成功的转为 torch 脚本，考虑到大部分模型使用 GPU 模型训练，而算能提供的 SDK 仅支持，我们的转换工具可以将 GPU 上的数据移动到 CPU 上。

原始模型	Ygg16.pth

原始模型	Ygg16.pth
概述	基于 VGG 的路标识别模型。
骨干网络	VGG16
训练集	自制数据集
运算量	未说明
输入数据	[batch_size, 3, 224, 224],
输出数据	[batch_size, 224,224], FP32
前处理	resize,减均值,除方差,HWC->CHW
后处理	ctc_decode

⑥ 模型转换

模型转换验证和程序编译必须在开发环境中完成，我们需要一台 x86 主机作为开发环境，并且在我们提供的基于 Ubuntu18.04 的 docker 镜像中，使用我们的 SophonSDK 进行模型转换和量化。如果我们的 x86 主机插有 PCIe 加速卡可使用 PCIe 模式，如果没有可使用 CModel 模式。从宿主机 SDK 根目录下执行脚本进入 docker 环境进入 docker 容器时，我们应将 jit 化后的模型例程拷贝或映射至容器目录下。为了方便操作，我们使用了 xmoba 工具进行文件的查看与调试。

在 docker 容器内安装依赖库及和设置环境变量

```
cd $REL_TOP/scripts
./install_lib.sh nntc
# source envsetup_cmodel.sh  # for CMODEL MODE
```

生成 FP32 BModel

在本工程目录下执行以下命令，使用 bmnetp 编译生成 FP32 BModel

```
python3 -m bmnetp --model=traced_model4.pt --shapes="[1, 3, 224, 224]"
--net_name=VGG16 --opt=1 --dyn=false --outdir=compiled_model3
```

```
--target=BM1684 --cmp=true
```

上述脚本会在 data/models/fp32bmodel/下生成 lprnet_fp32_1b4b.bmodel、lprnet_fp32_1b.bmodel、lprnet_fp32_4b.bmodel、文件，即转换好的 FP32 BModel。

使用 bm_model.bin --info {path_of_bmodel}查看 lprnet_fp32_1b4b.bmodel 具体信息如下：

```
chip: BM1684

create time: Tue Jul 18 00:00:17 2023



========================================

net 0: [VGG16]  static

------------

stage 0:

input: x.1, [1, 3, 224, 224], float32, scale: 1

output: 107, [49, 10], float32, scale: 1
```

⑦Python 例程推理

由于在相关文档和公开资料中暂未发现 sdk 对 vgg 的推理方式，我们团队通过研究 ResNet18 的推理代码，了解相关库函数的功能，仿写了适用 vgg.bmodel 的推理文件。我们开发的 vggtest.py 推理代码适用使用 OpenCV 解码、OpenCV 前处理、SAIL 推理。

Python 代码无需编译，无论是 x86 PCIe 平台还是 arm SoC 平台配置好环境之后就可直接运行。

工程目录下的 python 目录提供了一系列 python 例程以供参考使用，具体情况如下：

#	例程主文件	说明

#	例程主文件	说明
1	Vggtest.py	使用 OpenCV 解码、OpenCV 前处理、SAIL 推理

模型推理

以 Vggtest.py.py 的推理为例,我们希望所开发的 vgg 模型推演工具具有强大的移植性，我们为其设置了多个接口，具体参数说明如下：

python3 test.py --img_path ../data/vgg/1.jpg
--bmodel ../data/vgg/compilation.bmodel --tpu_id 0

--img_path:推理图片路径，可输入单张图片的路径，也可输入整个图片文件夹的路径；

--bmodel:用于推理的 bmodel 路径；

--tpu_id:用于推理的 tpu 设备 id。

测试数据集并保存预测结果的实例如下：

测试单张图片

python3 test.py --img_path ../data/vgg/1.jpg
--bmodel ../data/vgg/compilation.bmodel --tpu_id 0

执行完成后，会打印预测结果、推理时间、准确率等信息。

```
  -1.93212390e-01 -1.70112640e-01]
 [-1.60855129e-02 -2.03645632e-01  4.16578650e-02  4.22625616e-02
   3.01629305e-03 -1.39177904e-01  2.47835740e-02 -4.64780703e-02
  -1.98652163e-01  4.78569046e-03]]
Time consuming: 0.08834 sec
--------------------------------------------------------------------
right
```

图 8 模型推演信息

为了彰显我们项目中的部署的迅捷和全面性，在项目的后期阶段，我们不仅成功集成了我们自身的工具链，还为该工具链设计了一个用户友好的网页界面，以实现一键转换和快速部署的功能。

我们的工具链的网页界面拥有多个重要功能，其中之一是数据集上传功能。用户可以轻松地自行上传其所需的数据集，并设置数据的不同类别和属性。这为个性化的训练提供了便捷的手段。

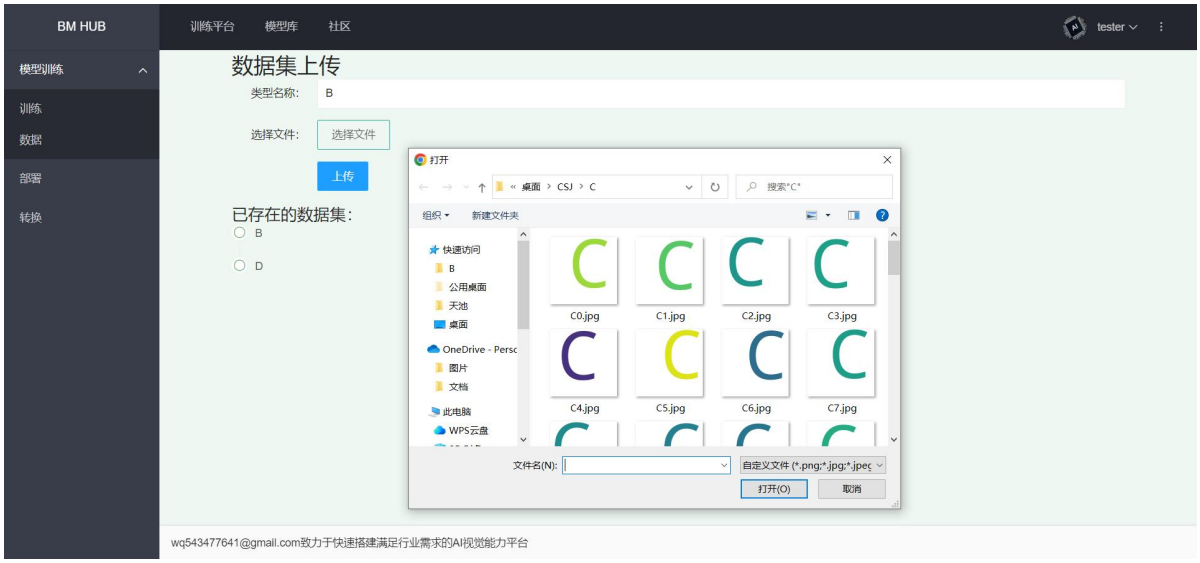


图 9 BM · HUB 数据集上传功能

此外，我们的在线训练功能为用户提供了强大的在线训练和转换服务。通过我们的平台，用户可以即时启动训练过程，使用各种预设的模型和参数。更为令人振奋的是，在训练完成后，我们的工具链会自动为用户生成一份详尽的推演代码。这些代码可以直接供用户使用，或者根据需要封装为接口函数，以支持用户进一步开发新的 AI 应用。

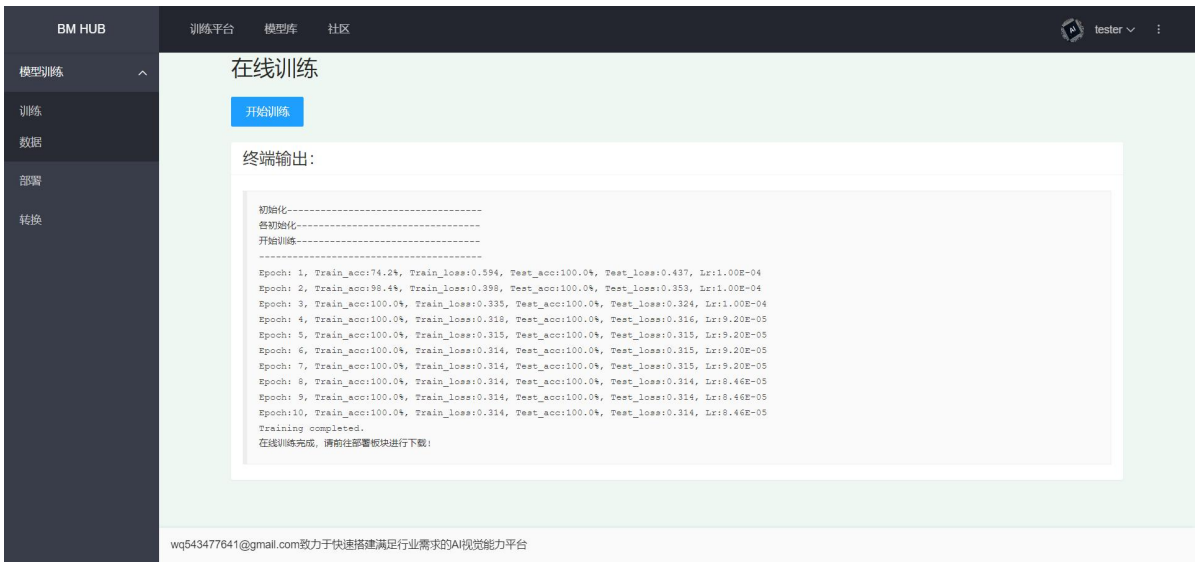


图 10BM · HUB 在线训练功能

关于在线部署功能，我们提供了两种便捷选择。首先，在模型训练完成后，用户可以选择将训练结果以压缩包形式下载。这个压缩包中包含了经过转换的bmodel 文件以及相应的推演代码，让用户可以在自己的环境中方便地进行部署。其次，对于连接到我们平台的设备，我们还提供了直接将模型与代码部署到设备的选项。需要注意的是，在实际操作中，考虑到模型文件通常较大，而且设备通常位于用户的身边，我们经过实测得出手动部署的效果往往优于自动部署。

图 11BM · HUB 在线部署功能

通过这些功能，我们为用户提供了高度定制化和高效率的模型训练、转换和部署过程，使得人工智能的应用变得更加简便、迅速和灵活。

4. 关键性能指标

本项目的关键性能指标可以包括以下几个方面：

模型推演速度测试

TUP	0.08834
CPU	0.1398s
GPU	0.1387

图 12 模型推演速度数据

感知性能：机器人的感知性能是指其感知环境和识别物体的准确性和鲁棒性。在此，我们通过车牌识别实验来测量机器人的感知性能。

```
INFO:root:img:皖SZ788K.jpg, res:皖SZ788K
INFO:root:img:皖SZH382.jpg, res:皖SZH382
INFO:root:img:川X90621.jpg, res:川X90621
INFO:root:ACC = 0.8910
INFO:root:------------------ Inference Time Info ----------------
INFO:root:inference_time(ms): 2.04
INFO:root:total_time(ms): 2778.61, img_num: 1000
INFO:root:average latency time(ms): 2.78, QPS: 359.891961
```

图 13 测量感知性能数据

通过实验，并通过公式 ACC=TP/CN 我们得出车牌识别实验中推理的准确率在 0.89 左右，说明机器人具有较好的感知性能。

控制性能：机器人的控制性能是指其实施车辆控制命令的准确性和稳定性。在调节 PID 的过程中，我们使用了 VOFA 软件来对机器人的 PID 曲线进行拟合，得到了较为稳定的控制性能。

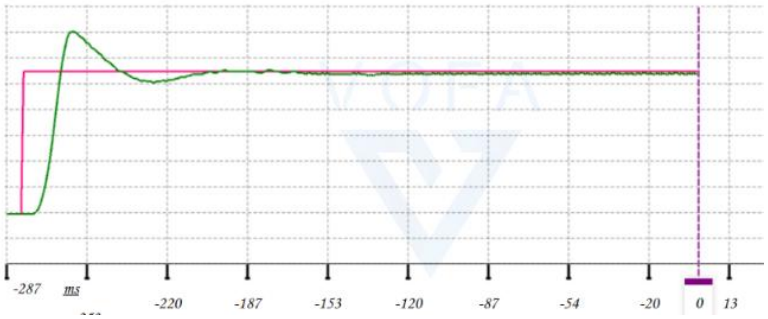


图 14 测量控制性能

经实验，机器人采用的电池及其管理系统，在实验环境中，具有良好的续航功能。

Vgg16 模型的训练曲线：

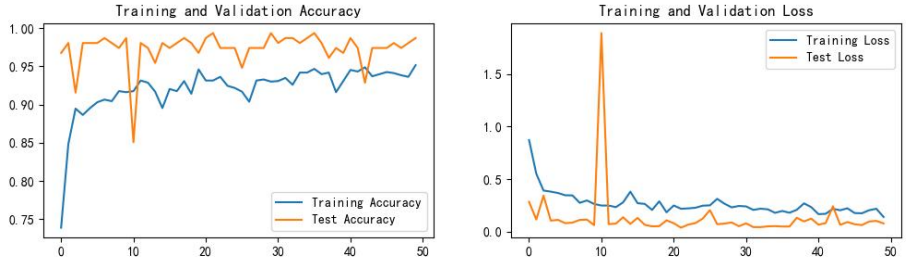


图 1VGG 损失参数

除感知性能，控制性能，能效性等性能已通过实验验证的关键性指标外，项目的定位精度，决策性能，安全性能，故障容忍等性能均在实验中展现出良好的状态。

5. 关键技术创新点

1. 采用基于深度学习的无人驾驶场景识别。
2. 使用 ROS 系统基于激光雷达进行场景的建模并使用。
3. 探索了 Dijkstra 算法优化路径。
4. 复现深度可分离卷积实现的轻量级 YOLOv3D 技术。
5. 使用雷达导航功能实现低电量保护方法。

“Overwatch”是一款基于 TPU（Tensor Processing Unit）的智能无人驾驶机器人，该系统采用 ROS（Robot Operating System）作为机器人操作系统，并搭载多种市面上常见的 AI 模型。通过这些 AI 模型的集成与优化，实现了智能化的无人驾驶功能。在项目研究的后期，我们团队开发了一套基于 VGG（Visual Geometry Group）的模型移植方案，该方案具备高度灵活性，允许使用任意数据集来快速训练 VGG 模型文件。

我们的移植方案结合了团队研发的工具链和算能提供的 SDK 工具，为 VGG 模型的快速部署提供了可行且高效的方法。这个 VGG 模型的部署迁移方法在网络公开资料中是首次被提及，为 TPU 芯片的开发提供了更多宝贵的思路与实践经验。

我们的无人驾驶机器人通过 TPU 芯片的强大计算能力，能够在实时场景下高效地处理大量数据，实现智能决策和精准控制。通过 ROS 的支持，我们能够轻松管理机器人的各种传感器数据、执行复杂的任务规划和路径规划。在无人驾驶的实际应用中，我们的系统表现出卓越的性能和稳定性。

通过我们的研究成果，我们成功将 VGG 模型应用于无人驾驶场景，有效地解决了计算资源有限和部署繁琐的问题。同时，我们的工作也对 TPU 芯片的应用探索提供了新的思路，为智能无人驾驶技术的发展迈出了重要的一步。我们对于未来进一步优化和扩展我们的系统充满信心，为实现更加智能、安全的无人驾驶技术不断努力

6.总结与展望

“Overwatch”是一款基于 TPU（Tensor Processing Unit）的智能无人驾驶机 器人，该系统采用 ROS（Robot Operating System）作为机器人操作系统，并搭 载多种市面上常见的 AI 模型。通过这些 AI 模型的集成与优化，实现了智能化的 无人驾驶功能。在项目研究的后期，我们团队开发了一套基于 VGG（Visual Geometry Group）的模型移植方案，该方案具备高度灵活性，允许使用任意数据 集来快速训练 VGG 模型文件。

我们的移植方案结合了团队研发的工具链和算能提供的 SDK 工具，为 VGG 模型的 快速部署提供了可行且高效的方法。这个 VGG 模型的部署迁移方法在网络公开资 料中是首次被提及，为 TPU 芯片的开发提供了更多宝贵的思路与实践经验。

我们的无人驾驶机器人通过 TPU 芯片的强大计算能力，能够在实时场景下高效地 处理大量数据，实现智能决策和精准控制。通过 ROS 的支持，我们能够轻松管理 机器人的各种传感器数据、执行复杂的任务规划和路径规划。在无人驾驶的实际

应用中，我们的系统表现出卓越的性能和稳定性。通过我们的研究成果，我们成功将 VGG 模型应用于无人驾驶场景，有效地解决了 计算资源有限和部署繁琐的问题。同时，我们的工作也对 TPU 芯片的应用探索提

供了新的思路，为智能无人驾驶技术的发展迈出了重要的一步。我们对于未来进 一步优化和扩展我们的系统充满信心，为实现更加智能、安全的无人驾驶技术不 断努力

参考文献

[1] 蔡忠亮,雷飞仪,蒋子捷等.无人驾驶小车的地图驱动机制研究[J].测绘地理信息,2021,46(05):1-7.DOI:10.14188/j.2095-6045.2020603.

[2] 范一赢,张慧贤,布占伟等.基于 ROS 的消防机器人激光雷达地图构建方法研究[J].工业控制计算机,2022,35(11):56-58.

[3] 蔡忠亮,雷飞仪,蒋子捷等.无人驾驶小车的地图驱动机制研究[J].测绘地理信息,2021,46(05):1-7.DOI:10.14188/j.2095-6045.2020603.

[4] 甄然,刘颖,孟凡华等.基于 YOLOv5 的轻量化目标检测算法[J/OL].无线电工程 :1-9[2023-05-28].http://kns.cnki.net/kcms/detail/13.1097.TN.20230328.1521.002.html.

[5] 先梦瑜.一种基于 Dijkstra 的物流配送路径优化算法设计[J].电子设计工程,2023,31(02):20-24.DOI:10.14022/j.issn1674-6236.2023.02.005.

附录

附录 1，vgg 模型移植技术文档：

1. 简介

VGG16 是一种深度卷积神经网络模型，由牛津大学的研究团队开发。它在 2014 年的 ImageNet 图像分类竞赛中取得了显著的成绩，被广泛应用于计算机视觉任务中，如图像分类、目标检测和图像分割等。

VGG16 的网络结构相对简单，由重复堆叠的卷积层和池化层构成。这种简单的结构使得网络易于理解和实现，同时也降低了出错的可能性。结构清晰的 VGG16 模型也为后续的改进和扩展提供了便利。

学习复杂特征能力强：VGG16 模型采用了多个卷积层和全连接层，这使得网络能够学习到更加复杂和抽象的特征表示。通过多个卷积层的叠加，VGG16 可以提取出不同尺度和层次的图像特征，从而提高了分类性能。这使得 VGG16 在许多计算机视觉任务中表现出色。

良好的性能和泛化能力：VGG16 在 2014 年的 ImageNet 图像分类竞赛中取得了显著的成绩，证明了其优秀的性能和泛化能力。VGG16 的网络结构和参数经过大规模的训练和调优，使得它具备很强的图像分类能力，并且对于不同的数据集和任务都能够取得不错的效果。这种强大的泛化能力使得 VGG16 成为了许多计算机视觉研究和应用的重要工具。

2. 数据集

我们首先是拍摄了校园中的路标路牌作为数据集，但这样的数量是远远不够的，因此我们团队编写了一套爬虫程序可以在网络上爬取我们所需求的数据集，但对于本次所识别的路牌，网络上的数据太过于杂乱，我们需要不断筛除爬取数据中差异过大的图片。此外为了提高准确率，我们也找到了鲸盒社区中的一份数据集以供使用。

我们设计该模型具有很好的移植性，只要数据集放置如下结构即可训练：

```
下载数据
|    ├──类 1
|    ├──类 2
|    .......
|    └──类 n
```

3. 模型来源与训练

3.1 模型训练环境:

编程语言：Python 3.6.13
深度学习主要依赖库：
Torch 1.8.0+cu113
torchvision 0.11.3
NumPy
1.19.5
pandas
1.1.5
数据集爬虫主要依赖库：
josn
Request
操作系统：Windows 10
开发工具：vscode,jupyter

3.2 模型的训练

我们基于 pytorch 框架，由于我们在后续的模型转换中发现了 gropout 算子由于版本兼容问题，出现了不支持转化问题，我们团队优化了 pytorch 自带的 vgg16 模型，通过 GPU 对该模型进行训练。模型训练 100 批次。生成 pth 文件。

数据增强技术：随机水平翻转

```
train_transforms = transforms.Compose([
    transforms.Resize([224, 224]),  # 将输入图片resize成统一尺寸
    transforms.RandomHorizontalFlip(), # 随机水平翻转
    transforms.ToTensor(),          # 将PIL Image或numpy.ndarray转换为tensor，并归一化到[0,1]之间
    transforms.Normalize(           # 标准化处理-->转换为标准正太分布（高斯分布），使模型更容易收敛
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])  # 其中 mean=[0.485,0.456,0.406]与std=[0.229,0.224,0.225] 从数据集
])
```

数据增强技术技术

迁移学习：使用大型标准数据集对模型进行训练，之后将模型学习到的通用特征的知识迁移到目标任务中，此时再利用目标任务的样本对迁移的知识进行微调并解决具体的问题。

```python
model1 = models.vgg16(pretrained=True).to(device)
vgg=model1.features#获取vgg16的特征提取层
for param in vgg.parameters():
    param.requires_grad = False
#print(vgg16.classifier)
class MyVggNet(nn.Module):
    def __init__(self):
        super(MyVggNet, self).__init__()
        # 预训练的vgg16特征提取层
        self.vgg = vgg
        # 添加新的全连接层
        self.classify = nn.Sequential(
            nn.Linear(25088, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 5),
            nn.Softmax(dim=1)
        )
```

Vgg 中的迁移学习技术

4. 准备工作

4.1 pth 文件的 jit 化

在 PyTorch 中，torch.jit.trace 函数用于将模型转换为 Torch 脚本（Torch Script）。Torch 脚本是一种中间表示形式，可以提供模型的静态图形表示，从而实现更高效的模型部署和推断。

我们团队开发的 pt JIT 化工具可以将 pt 成功的转为 torch 脚本，考虑到大部分模型使用 GPU 模型训练，而算能提供的 SDK 仅支持

4.2 准备开发环境

模型转换验证和程序编译必须在开发环境中完成，我们需要一台 x86 主机作为开发环境，并且在我们提供的基于 Ubuntu18.04 的 docker 镜像中，使用我们的 SophonSDK 进行模型转换和量化。如果我们的 x86 主机插有 PCIe 加速卡可使用 PCIe 模式，如果没有可使用 CModel 模式。

从宿主机 SDK 根目录下执行脚本进入 docker 环境

进入 docker 容器时，我们应将 jit 化后的模型例程拷贝或映射至容器目录下。为了方便操作，我们使用了 xmoaba 工具进行文件的查看与调试。

在 docker 容器内安装依赖库及和设置环境变量

```
cd $REL_TOP/scripts
./install_lib.sh nntc
# source envsetup_cmodel.sh  # for CMODEL MODE
```

4.3 准备模型与数据

模型与数据集统一从宿主机上迁移至 docker 中。

原始模型	Ygg16.pth
概述	基于 VGG 的路标识别模型。
骨干网络	VGG16
训练集	自制数据集
运算量	未说明
输入数据	[batch_size, 3, 224, 224],
输出数据	[batch_size, 224,224], FP32
前处理	resize,减均值,除方差,HWC->CHW
后处理	ctc_decode

5. 模型转换

模型转换的过程需要在 x86 下的 docker 开发环境中完成。以下操作均在 x86 下的 docker 开发环境中完成。

5.1 生成 FP32 BModel

在本工程目录下执行以下命令，使用 bmnetp 编译生成 FP32 BModel，

python3 -m bmnetp --model=traced_model4.pt --shapes="[1, 3, 224, 224]" --net_name=VGG16 --opt=1 --dyn=false --outdir=compiled_model3 --target=BM1684 --cmp=true

上述脚本会在 data/models/fp32bmodel/下生成 lprnet_fp32_1b4b.bmodel、lprnet_fp32_1b.bmodel、lprnet_fp32_4b.bmodel、文件，即转换好的 FP32 BModel，使用 bm_model.bin --info {path_of_bmodel} 查看 lprnet_fp32_1b4b.bmodel 具体信息如下：

```
chip: BM1684
create time: Tue Jul 18 00:00:17 2023

=========================================
net 0: [VGG16]  static
------------
stage 0:
```

```
    input: x.1, [1, 3, 224, 224], float32, scale: 1
    output: 107, [49, 10], float32, scale: 1
```

6. 推理测试

6.1 环境配置

对于推理环境，可以选用 x86 环境或 armSoc 环境，在次我们仅用 X86 环境进行推理测试

对于 x86 with PCIe 平台，程序执行所需的环境变量执行 source envsetup_pcie.sh 时已经配置完成。

由于 Python 例程用到 sail 库，需安装 Sophon Inference：

6.2 Python 例程推理

由于在相关文档和公开资料中暂未发现 sdk 对 vgg 的推理方式，我们团队通过研究 ResNet18 的推理代码，了解相关库函数的功能，仿写了适用 vgg.bmodel 的推理文件。我们开发的 wggtest.py 推理代码适用使用 OpenCV 解码、OpenCV 前处理、SAIL 推理。Python 代码无需编译，无论是 x86 PCIe 平台还是 arm SoC 平台配置好环境之后就可直接运行。工程目录下的 python 目录提供了一系列 python 例程以供参考使用，具体情况如下：

#	例程主文件	说明
1	Vggtest.py	使用 OpenCV 解码、OpenCV 前处理、SAIL 推理

以 Vggtest.py.py 的推理为例,我们希望所开发的 vgg 模型推演工具具有强大的移植性，我们为其设置了多个接口，具体参数说明如下：

```
python3 test.py --img_path ../data/vgg/1.jpg
--bmodel ../data/vgg/compilation.bmodel --tpu_id 0
```

--img_path:推理图片路径，可输入单张图片的路径，也可输入整个图片文件夹的路径；

--bmodel:用于推理的 bmodel 路径；

--tpu_id:用于推理的 tpu 设备 id。

测试数据集并保存预测结果的实例如下：

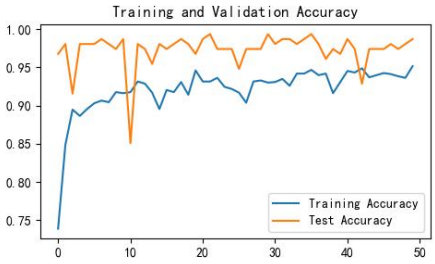
```
# 测试单张图片
```

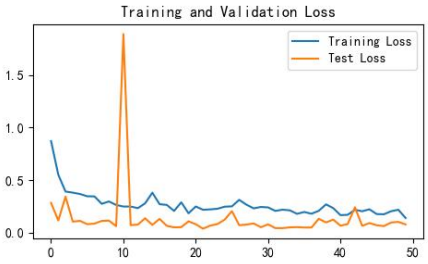
```
python3 test.py --img_path ../data/vgg/1.jpg
--bmodel ../data/vgg/compilation.bmodel --tpu_id 0

# 测试整个文件夹

python3 test.py --img_path ../data/vgg/test
--bmodel ../data/vgg/compilation.bmodel --tpu_id 0
```

执行完成后，会打印预测结果、推理时间、准确率等信息。

```
  -1.93212390e-01 -1.70112640e-01]
 [-1.60855129e-02 -2.03645632e-01  4.16578650e-02  4.22625616e-02
   3.01629305e-03 -1.39177904e-01  2.47835740e-02 -4.64780703e-02
  -1.98652163e-01  4.78569046e-03]]
Time consuming: 0.08834 sec
--------------------------------------------------------------------
right
```

6.3 测试结果





附录 2 模型移植代码

Jit.py 将 vgg16 经行 jit 处理并将数据迁移至 cpu

```
    import torch
    import torch.nn as nn
    import torchvision.transforms as transforms
    from torchvision import models
    import torch
    from torch import nn
    import torch.nn.functional as F
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    vgg= models.vgg16(pretrained=True).to(device)
    print(device)

    del vgg.classifier[2]
    del vgg.classifier[4]
    vgg.classifier[4] = nn.Linear(4096, 17)
    model =vgg
    print(vgg)

    model.load_state_dict(torch.load('Vgg_16.pth',
map_location=device))
    model = model.to(device)

    example_input = torch.randn(1, 3, 32, 32).to(device)
    traced_model = torch.jit.trace(model, example_input)
    traced_model.save('vgg_16.pt')

    print("Model traced and saved successfully.")
    model = torch.jit.load('vgg_16.pt', map_location="cpu")
    model.eval()
    input_data = torch.randn(1, 3, 32, 32)
    traced_model = torch.jit.trace(model, [input_data])
    traced_model_name = "vgg_16.pt"
    traced_model.save('vgg_16.pt')
```

Vgg16_test.py 在 BM1684 上使用 TPU 对模型进行推演

```
    import cv2
    import time
```

```
    import argparse
    import numpy as np
    from PIL import Image
    import sophon.sail as sail


    def preprocess(img):
        image = Image.fromarray(img)
        resized_img = np.array(image.resize((224,224), resample=2))
        out = np.array(resized_img / 255., dtype=np.float32)
        return out.transpose((2, 0, 1))


    def postprocess(output):
        class_names =  ['bridge', 'left', 'right', 'walk', 'right',
'NO_right', 'crossroads', 'door', 'narrow', 'construction','waring']
        predicted_class_idx = np.argmax(output)
        l=[]


        predicted_class = class_names[predicted_class_idx]
        return predicted_class

        predicted_class = np.argmax(output)
        print( class_names[max_index])
        return class_names[max_index]


    def infer(bmodel_path, img_path, tpu_id=0):
        start_time = time.time()

        img = cv2.imread(img_path)
        img_array = preprocess(img)

        net = sail.Engine(bmodel_path, tpu_id, sail.IOMode.SYSIO)
        graph_name = net.get_graph_names()[0]
        print(net.get_graph_names())

        input_names = net.get_input_names(graph_name)

        output_names = net.get_output_names(graph_name)
        input_data   =   {input_names[0]:   np.expand_dims(img_array,
axis=0)}
```

```
        prediction = net.process(graph_name, input_data)

        end_time = time.time()
        timer = end_time - start_time
        print("-" * 66)
        print("Predicted                    class                    index:
{}".format(prediction[output_names[0]]))
        print("Time consuming: %.5f sec" % timer)
        print("-" * 66)
        res = postprocess(prediction[output_names[0]])
        print("The predicted class is {}".format(res))
        return res


    if __name__ == '__main__':
        PARSER = argparse.ArgumentParser(description='VGG16 inference
with Sail')
        PARSER.add_argument('--bmodel', default='vgg16.bmodel')
        PARSER.add_argument('--img_path', default='2.png')
        PARSER.add_argument('--tpu_id',      default=0,      type=int,
required=False)
        ARGS = PARSER.parse_args()
        infer(ARGS.bmodel, ARGS.img_path, ARGS.tpu_id)
```

Trainvgg.py

```
    import torch
    import torch.nn as nn
    from torchvision import transforms, datasets, utils
    import matplotlib.pyplot as plt
    import numpy as np
    import torch.optim as optim

    import model

    import os
    import json
    import time
    import os
    from torchvision import models
    import torch
    from torch import nn
    os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

```
    os.environ['CUDA_LAUNCH_BLOCKING'] = '1' # 下面老是报错 shape 不一致
    train_loss = []
    train_acc  = []
    test_loss  = []
    test_acc   = []



    #device : GPU or CPU
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print(device)



    #数据转换
    data_transform = {
        "train":
transforms.Compose([transforms.RandomResizedCrop(224),

transforms.RandomHorizontalFlip(),
                                        transforms.ToTensor(),
                                        transforms.Normalize((0.5,  0.5, 0.5), (0.5, 0.5, 0.5))]),
        "val": transforms.Compose([transforms.Resize((224, 224)),  # cannot 224, must (224, 224)
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.5,  0.5, 0.5), (0.5, 0.5, 0.5))])}
    #data_root = os.path.abspath(train_dataset.path.join(os.getcwd(), "../.."))  # get data root path
    data_root = os.getcwd()
    image_path = data_root + "/Hollywood_data/"  # flower data set path
    train_dataset = datasets.ImageFolder(root=image_path + "/train",

transform=data_transform["train"])
    train_num = len(train_dataset)

    #数据集部分展示
    # examples = iter(train_dataset)
    # example_data,example_label = next(examples)
    # train_image,label = train_dataset[0]
    # plt.imshow(train_image[2])
    # plt.title("labels:%d"%label)
    # plt.show()
```

```
    # {'daisy':0, 'dandelion':1, 'roses':2, 'sunflower':3, 'tulips':4}
    flower_list = train_dataset.class_to_idx
    cla_dict = dict((val, key) for key, val in flower_list.items())
    # write dict into json file
    json_str = json.dumps(cla_dict, indent=4)
    with open('class_indices.json', 'w') as json_file:
        json_file.write(json_str)

    batch_size = 8
    train_loader = torch.utils.data.DataLoader(train_dataset,
                                               batch_size=batch_size,
shuffle=True,
                                               num_workers=0)

    validate_dataset = datasets.ImageFolder(root=image_path + "/val",

transform=data_transform["val"])
    val_num = len(validate_dataset)
    validate_loader = torch.utils.data.DataLoader(validate_dataset,

batch_size=batch_size, shuffle=True,
                                                  num_workers=0)

    test_data_iter = iter(validate_loader)
    test_image, test_label = test_data_iter.next()
    #print(test_image[0].size(),type(test_image[0]))
    #print(test_label[0],test_label[0].item(),type(test_label[0]))


    #显示图像，之前需把 validate_loader 中 batch_size 改为 4
    def imshow(img):
        img = img / 2 + 0.5  # unnormalize
        npimg = img.numpy()
        plt.imshow(np.transpose(npimg, (1, 2, 0)))
        plt.show()

    print(' '.join('%5s' % cla_dict[test_label[j].item()] for j in
range(8)))
    imshow(utils.make_grid(test_image))


    # net = AlexNet(num_classes=17, init_weights=True)
    net = models.vgg16(pretrained=True)
```

```
    del net.classifier[2]
    del net.classifier[4]
    # print(net)
    # num_features = net.fc.in_features
    net.classifier[4] = nn.Linear(4096, 17)
    print(net)
    net.to(device)
    #损失函数:这里用交叉熵
    loss_function = nn.CrossEntropyLoss()
    #优化器 这里用 Adam
    optimizer = optim.Adam(net.parameters(), lr=1e-4)
    #训练参数保存路径
    # save_path = './AlexNet.pth'
    # save_path = './resnet34.pth'
    # save_path = './Vgg16.pth'
    # save_path = './GoogLeNet.pth'
    # save_path = './resnet18.pth'
    # save_path = './Mobile.pth'
    save_path = './Vgg_16.pth'
    #训练过程中最高准确率
    best_acc = 0.0
    epochs = 50
    #开始进行训练和测试，训练一轮，测试一轮
    for epoch in range(epochs):
        size = len(train_loader.dataset)
        # train
        net.train()    #训练过程中，使用之前定义网络中的 dropout
        running_loss = 0.0
        train_ac = 0
        t1 = time.perf_counter()
        for step, data in enumerate(train_loader, start=0):
            images, labels = data
            optimizer.zero_grad()
            outputs = net(images.to(device))
            loss = loss_function(outputs, labels.to(device))
            loss.backward()
            optimizer.step()

            # print statistics
            running_loss += loss.item()
            # print train process
            rate = (step + 1) / len(train_loader)
            a = "*" * int(rate * 50)
            b = "." * int((1 - rate) * 50)
```

```
            print("\rtrain                                                    loss:
{:^3.0f}%[{}->{}]{:.3f}".format(int(rate * 100), a, b, loss), end="")
            train_ac          +=          (outputs.argmax(1)          ==
labels.to(device)).type(torch.float).sum().item()
        train_ac /= size
        print()
        print(time.perf_counter()-t1)

        # validate
        net.eval()    #测试过程中不需要 dropout，使用所有的神经元
        acc = 0.0  # accumulate accurate number / epoch
        test_lo = 0
        num_batches = len(validate_loader)
        with torch.no_grad():
            for val_data in validate_loader:
                val_images, val_labels = val_data
                outputs = net(val_images.to(device))
                test_lo                                                  +=
loss_function(outputs,val_labels.to(device))
                predict_y = torch.max(outputs, dim=1)[1]
                acc             +=             (predict_y             ==
val_labels.to(device)).sum().item()
            val_accurate = acc / val_num
            if val_accurate > best_acc:
                best_acc = val_accurate
                torch.save(net.state_dict(), save_path)
            print('[epoch %d] train_loss: %.3f  test_accuracy: %.3f' %
                  (epoch + 1, running_loss / step, val_accurate))
            test_lo /= num_batches
            train_loss.append(running_loss / step)
            test_loss.append(test_lo)
            train_acc.append(train_ac)
            test_acc.append(val_accurate)
    import matplotlib.pyplot as plt
    #隐藏警告
    import warnings
    warnings.filterwarnings("ignore")                #忽略警告信息
    plt.rcParams['font.sans-serif']    = ['SimHei'] # 用来正常显示中文
标签
    plt.rcParams['axes.unicode_minus'] = False      # 用来正常显示负号
    plt.rcParams['figure.dpi']         = 100        #分辨率

    epochs_range = range(epochs)
```

```
    plt.figure(figsize=(12, 3))
    plt.subplot(1, 2, 1)

    plt.plot(epochs_range, train_acc, label='Training Accuracy')
    plt.plot(epochs_range, test_acc, label='Test Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, train_loss, label='Training Loss')
    plt.plot(epochs_range, test_loss, label='Test Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()

    print('Finished Training')
```

附录 3 BMHUB 使用说明书

BM HUB 在线训练平台使用说明

欢迎使用我们的在线训练平台，为您提供便捷的 AI 模型训练和部署体验。以下是详细的使用说明，以帮助您顺利完成训练和部署过程。

步骤 1：访问

打开浏览器，访问我们的在线训练平台网址。

步骤 2：上传数据

您将进入平台的主界面。点击“上传数据”按钮。

您可以选择将训练所需的数据文件上传至平台。确保数据文件格式正确且数据完整。

步骤 3：在线训练

在数据上传完成后，点击“开始训练”按钮。

在训练页面，您可以选择预定义的模型架构或上传自定义模型架构。

选择训练参数，如批量大小、学习率等。这些参数将影响训练过程和模型性能。

确认设置后，点击“开始训练”以启动训练过程。

步骤 4：一键部署

训练完成后，您将回到平台主界面。现在您可以选择一键部署或手动部署。

点击“一键部署”按钮。系统将自动将推演代码与经过训练的模型导入嵌入式设备。

确认您的设备已连接并准备就绪，然后点击“开始部署”。

步骤 5：手动部署

如果您选择手动部署，点击“手动部署”按钮。

您将获得经过转化的模型文件和推演代码的下载链接。

将这些文件下载到您的开发环境中，按照我们提供的指南进行配置和部署。

注意事项：

1. 在训练和部署过程中，确保您的设备连接稳定的互联网。

2. 确保上传的数据符合隐私和法律要求，不包含敏感信息。

3. 对于手动部署，请遵循所提供的文档进行设备配置和代码部署。

感谢您选择我们的在线训练平台。如您在使用过程中遇到问题，请随时联系我们的客户支持团队获取帮助。祝您训练和部署愉快！

附录·4 BHMUH 代码（主要）

App.py BM·HUB 的后端代码

```
    # app.py
    from flask import Flask, render_template, request,
jsonify,redirect,url_for,flash,send_file
    import os
    import threading
    import sys
    import time
    from train2 import train_model

    from werkzeug.utils import secure_filename
    UPLOAD_FOLDER = 'uploads'  # 指定文件上传的文件夹
    training_thread = None
    terminal_output = []
    # 如果文件夹不存在，则创建
    if not os.path.exists(UPLOAD_FOLDER):
        os.makedirs(UPLOAD_FOLDER)



    app = Flask(__name__)
    app.config['UPLOAD_FOLDER'] = 'uploads'
    app.secret_key = 'your_secret_key'

    if not os.path.exists(app.config['UPLOAD_FOLDER']):
        os.makedirs(app.config['UPLOAD_FOLDER'])

    @app.route('/')
    def index():
        return render_template('index.html')
    @app.route('/up', methods=['GET', 'POST'])
    def up():
        existing_folders = [folder for folder in
os.listdir(app.config['UPLOAD_FOLDER']) if
os.path.isdir(os.path.join(app.config['UPLOAD_FOLDER'], folder))]

        if request.method == 'POST':
            type_name = request.form['type_name']
            if type_name:
                type_folder =
os.path.join(app.config['UPLOAD_FOLDER'], type_name)
                if not os.path.exists(type_folder):
```

```
                    os.makedirs(type_folder)

                files = request.files.getlist('files[]')
                for file in files:
                    if file and allowed_file(file.filename):
                        filename = secure_filename(file.filename)
                        file_path = os.path.join(type_folder, filename)
                        file.save(file_path)

                flash('文件上传成功', 'success')
                return redirect(url_for('up'))

        return                              render_template('upload.html',
existing_folders=existing_folders)

    def allowed_file(filename):
        ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
        return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS
    @app.route('/train')
    def train():
        return render_template('train.html')
    @app.route('/start_training', methods=['POST'])
    def start_training():
        global training_thread
        global terminal_output

        if training_thread is None or not training_thread.is_alive():
            terminal_output = []
            training_thread   =   threading.Thread(target=train_model,
args=(terminal_output,))
            training_thread.start()
            return jsonify({'status': 'Training started.'})
        else:
            return   jsonify({'status':   'Training   is   already   in
progress.'})

    @app.route('/get_terminal_output', methods=['GET'])
    def get_terminal_output():
        global terminal_output
        return jsonify({'output': terminal_output})
    @app.route('/download_model', methods=['GET'])
    def download_model():
        model_path = 'bmodel\\ALL.zip'  # 替换为实际模型路径
```

```
        return send_file(model_path, as_attachment=True)
    @app.route('/bushu', methods=['GET'])
    def bushu():
        model_path = 'bmodel\\ALL.zip'  # 替换为实际模型路径
        return render_template('bushu.html')
    @app.route('/help')
    def help():
        return render_template('help.html')
    if __name__ == '__main__':
        app.run(host='0.0.0.0',debug=True)
```

Trainonlian.Py 在线训推一体代码

```
    import torch
    import torch.nn as nn
    import torchvision.transforms as transforms
    import torchvision
    from torchvision import transforms, datasets
    import os,PIL,pathlib,warnings
    from torchvision import models
    import torch.optim
    import copy
    import subprocess
    import time
    from ldocker import codepor
    from zip import zipmodel
    def train_model(terminal_output):
        # 模型训练代码
        terminal_output.append("          初          始          化
----------------------------------")
        time.sleep(1)
        terminal_output.append("       各       初       始       化
--------------------------------")
        time.sleep(1)
        terminal_output.append("       开       始       训       练
--------------------------------")
        time.sleep(1)

terminal_output.append("----------------------------------------")
        time.sleep(1)
        try:
            warnings.filterwarnings("ignore")              #忽略警告信
```

```
息
            os.environ['CUDA_LAUNCH_BLOCKING'] = '1'

            device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
            # device = torch.device("cpu")
            print(device)
            data_dir = 'uploads'
            data_dir = pathlib.Path(data_dir)

            data_paths  = list(data_dir.glob('*'))
            classeNames  =  [str(path).split("\\")[1]  for  path  in
data_paths]
            print(classeNames)

            train_transforms = transforms.Compose([
                transforms.Resize([224, 224]),  # 将输入图片 resize 成
统一尺寸
                transforms.RandomHorizontalFlip(), # 随机水平翻转
                transforms.ColorJitter()# 随机颜色改变翻转
                transforms.ToTensor(),              # 将 PIL Image 或
numpy.ndarray 转换为 tensor，并归一化到[0,1]之间
                transforms.Normalize(           # 标准化处理-->转换为
标准正太分布（高斯分布），使模型更容易收敛
                    mean=[0.485, 0.456, 0.406],
                    std=[0.229,    0.224,    0.225])       #    其    中
mean=[0.485,0.456,0.406]与 std=[0.229,0.224,0.225] 从数据集中随机抽样
计算得到的。
            ])

            test_transform = transforms.Compose([
                transforms.Resize([224, 224]),  # 将输入图片 resize 成
统一尺寸
                transforms.ToTensor(),              # 将 PIL Image 或
numpy.ndarray 转换为 tensor，并归一化到[0,1]之间
                transforms.Normalize(           # 标准化处理-->转换为
标准正太分布（高斯分布），使模型更容易收敛
                    mean=[0.485, 0.456, 0.406],
                    std=[0.229,    0.224,    0.225])       #    其    中
mean=[0.485,0.456,0.406]与 std=[0.229,0.224,0.225] 从数据集中随机抽样
计算得到的。
            ])
            total_data                                              =
datasets.ImageFolder("uploads",transform=train_transforms)
```

```
            train_size = int(0.8 * len(total_data))
            test_size  = len(total_data) - train_size
            train_dataset,                test_dataset                =
torch.utils.data.random_split(total_data, [train_size, test_size])
            batch_size = 8

            train_dl = torch.utils.data.DataLoader(train_dataset,

batch_size=batch_size,
                                                   shuffle=True,
                                                   num_workers=0)
            test_dl = torch.utils.data.DataLoader(test_dataset,

batch_size=batch_size,
                                                   shuffle=True,
                                                   num_workers=0)
            # model = models.googlenet(num_classes=4, aux_logits=True,
init_weights=True).to(device) # 加载预训练的 vgg16 模型
            model1 = models.vgg16(pretrained=False).to(device)
            vgg=model1.features#获取 vgg16 的特征提取层
            for param in vgg.parameters():
                param.requires_grad = False
            #print(vgg16.classifier)
            class MyVggNet(nn.Module):
                def __init__(self):
                    super(MyVggNet, self).__init__()
                    # 预训练的 vgg16 特征提取层
                    self.vgg = vgg
                    # 添加新的全连接层
                    self.classify = nn.Sequential(
                        nn.Linear(25088, 512),
                        nn.ReLU(),

                        nn.Linear(512, 256),
                        nn.ReLU(),

                        nn.Linear(256, len(classeNames)),
                        nn.Softmax(dim=1)
                    )

                # 定义网络的前向传播
                def forward(self, x):
                    x = self.vgg(x)
                    x = x.view(x.size(0), -1)  # 多维度的 tensor 展平成
```

一维

```
                output = self.classify(x)
                return output

        model=MyVggNet()

        print(model)
        # for param in model.parameters():
        #     param.requires_grad = False # 冻结模型的参数，这样子在训练的时候只训练最后一层的参数

        # 修改 classifier 模块的第 6 层（即：(6): Linear(in_features=4096, out_features=2, bias=True)）
        # 注意查看我们下方打印出来的模型
        #             model.classifier._modules['6']             = nn.Linear(4096,len(classeNames)) # 修改 vgg16 模型中最后一层全连接层，输出目标类别个数
        model.to(device)
        def train(dataloader, model, loss_fn, optimizer):
            size = len(dataloader.dataset)  # 训练集的大小
            num_batches  =  len(dataloader)      # 批次数目，(size/batch_size，向上取整)

            train_loss, train_acc = 0, 0  # 初始化训练损失和正确率

            for X, y in dataloader:  # 获取图片及其标签
                X, y = X.to(device), y.to(device)

                # 计算预测误差
                # with torch.no_grad():
                pred = model(X)           # 网络输出
                loss = loss_fn(pred, y)  # 计算网络输出和真实值之间的差距，targets 为真实值，计算二者差值即为损失

                # 反向传播
                optimizer.zero_grad()  # grad 属性归零
                loss.backward()        # 反向传播
                optimizer.step()       # 每一步自动更新

                # 记录 acc 与 loss
                train_acc           +=      (pred.argmax(1)      == y).type(torch.float).sum().item()
                train_loss += loss.item()
```

```
                train_acc  /= size
                train_loss /= num_batches

                return train_acc, train_loss
            lambda1 = lambda epoch: 0.92 ** (epoch // 4)
            optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
            scheduler  =  torch.optim.lr_scheduler.LambdaLR(optimizer,
lr_lambda=lambda1) #选定调整方法
            def test (dataloader, model, loss_fn):
                size        = len(dataloader.dataset)  # 测试集的大小
                num_batches = len(dataloader)            # 批次数目,
(size/batch_size，向上取整)
                test_loss, test_acc = 0, 0

                # 当不进行训练时，停止梯度更新，节省计算内存消耗
                with torch.no_grad():
                    for imgs, target      =      imgs.to(device),
target.to(device)

                        # 计算 loss
                        target_pred = model(imgs)
                        loss        = loss_fn(target_pred, target)

                        test_loss += loss.item()
                        test_acc      +=   (target_pred.argmax(1)   ==
target).type(torch.float).sum().item()

                test_acc  /= size
                test_loss /= num_batches

                return test_acc, test_loss
            import copy

            loss_fn    = nn.CrossEntropyLoss() # 创建损失函数
            epochs     =10

            train_loss = []
            train_acc  = []
            test_loss  = []
            test_acc   = []

            best_acc = 0    # 设置一个最佳准确率，作为最佳模型的判别指
标
```

```
            for epoch in range(epochs):
                # 更新学习率（使用自定义学习率时使用）
                # adjust_learning_rate(optimizer, epoch, learn_rate)

                model.train()
                epoch_train_acc,  epoch_train_loss  =  train(train_dl,
model, loss_fn, optimizer)
                scheduler.step() # 更新学习率（调用官方动态学习率接口
时使用）

                model.eval()
                epoch_test_acc, epoch_test_loss = test(test_dl, model,
loss_fn)

                # 保存最佳模型到 best_model
                if epoch_test_acc > best_acc:
                    best_acc   = epoch_test_acc
                    best_model = copy.deepcopy(model)

                train_acc.append(epoch_train_acc)
                train_loss.append(epoch_train_loss)
                test_acc.append(epoch_test_acc)
                test_loss.append(epoch_test_loss)

                # 获取当前的学习率
                lr = optimizer.state_dict()['param_groups'][0]['lr']

                template    =    ('Epoch:{:2d},    Train_acc:{:.1f}%,
Train_loss:{:.3f}, Test_acc:{:.1f}%, Test_loss:{:.3f}, Lr:{:.2E}')
                print(template.format(epoch+1,    epoch_train_acc*100,
epoch_train_loss,
                                      epoch_test_acc*100,
epoch_test_loss, lr))
                terminal_output.append(template.format(epoch+1,
epoch_train_acc*100, epoch_train_loss,
                                      epoch_test_acc*100,
epoch_test_loss, lr))
                print(terminal_output)
            # 保存最佳模型到文件中
            PATH = './best_model5.pt'  # 保存的参数文件名
            torch.save(model.state_dict(), PATH)

            print('Done')
```

```
            from PIL import Image
            import matplotlib.pyplot as plt
            classes = list(total_data.class_to_idx)

            def   predict_one_image(image_path,   model,   transform,
classes):

                test_img = Image.open(image_path).convert('RGB')
                plt.imshow(test_img)  # 展示预测的图片

                test_img = transform(test_img)
                img = test_img.to(device).unsqueeze(0)

                model.eval()
                output = model(img)

                _,pred = torch.max(output,1)
                pred_class = classes[pred]
                print(f'预测结果是：{pred_class}')
                plt.rcParams['font.sans-serif']    = ['SimHei'] # 用来
正常显示中文标签
            plt.rcParams['axes.unicode_minus'] = False      # 用来正常
显示负号
            plt.rcParams['figure.dpi']          = 100        #分辨率

            model=MyVggNet()
            model.load_state_dict(torch.load('best_model5.pt',
map_location=device))
            model = model.to(device)
            example_input = torch.randn(1, 3, 224, 224).to(device)
            traced_model = torch.jit.trace(model, example_input)
            traced_model.save('vgg_16.pt')
            print("Model traced and saved successfully.")
            model = torch.jit.load('vgg_16.pt', map_location="cpu")
            model.eval()
            input_data = torch.randn(1, 3, 224, 224)
            traced_model = torch.jit.trace(model, [input_data])
            traced_model_name = "vgg_16.pt"
            traced_model.save('vgg_16.pt')
            terminal_output.append("Training completed.")
            combined_command ="copy C:\\Users\\admin\\Desktop\\集创赛
\\vgg_16.pt C:\\shareFolder\\1"
            subprocess.run(combined_command, shell=True, check=True)
            combined_command = 'docker exec -it yingshi /bin/bash -c "ls
```

```
&& cd / && ls && cd shareData/1 && ls && cd /workspace/sophonsdk_v3.0.0/scripts/ && chmod +777 *.sh && source envsetup_cmodel.sh && cd /shareData/1 &&chmod +x 1.sh && ./1.sh"'
            subprocess.run(combined_command, shell=True, check=True)
            combined_command                                    ="copy C:\\shareFolder\\1\\compiled_model3\\compilation.bmodel C:\\Users\\admin\\Desktop\\集创赛\\bmodel\\"
            subprocess.run(combined_command, shell=True, check=True)
            codepor(classeNames)
            zipmodel()

            terminal_output.append("在线转换完成！请前往部署板块进行下载!")
        except:
            terminal_output.append("--出错--")

        # plt.figure(figsize=(12, 3))
        # plt.subplot(1, 2, 1)

        # plt.plot(epochs_range, train_acc, label='Training Accuracy')
        # plt.plot(epochs_range, test_acc, label='Test Accuracy')
        # plt.legend(loc='lower right')
        # plt.title('Training and Validation Accuracy')

        # plt.subplot(1, 2, 2)
        # plt.plot(epochs_range, train_loss, label='Training Loss')
        # plt.plot(epochs_range, test_loss, label='Test Loss')
        # plt.legend(loc='upper right')
        # plt.title('Training and Validation Loss')
        # plt.show()
        # 预测训练集中的某张照片
        # for i in range(10,100):
        #
predict_one_image(image_path='./aoteman/saiwen/0'+str(i)+'.jpg',
        #                       model=model,
        #                       transform=train_transforms,
        #                       classes=classes)
```