华为“智能基座”系列课程

基于昇腾的

深度学习实战

版本：2.2



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验环境介绍 4](#_Toc93413793)

[1.1 实验介绍 4](#_Toc93413794)

[1.1.1 关于本实验 4](#_Toc93413795)

[1.1.2 实验环境 4](#_Toc93413796)

[1.1.3 实验目的 4](#_Toc93413797)

[1.1.4 实验清单 4](#_Toc93413798)

[2 昇腾 AscendCL应用开发快速入门 6](#_Toc93413799)

[2.1 实验介绍 6](#_Toc93413800)

[2.2 实验目的 6](#_Toc93413801)

[2.3 实验环境要求 6](#_Toc93413802)

[2.4 实验原理 6](#_Toc93413803)

[2.5 实验步骤 7](#_Toc93413804)

[2.5.1 项目文件下载至Ubuntu服务器（在Ubuntu服务器内执行） 7](#_Toc93413805)

[2.5.2 拷贝项目文件classification\_python至开发板内（在Ubuntu服务器内执行） 8](#_Toc93413806)

[2.5.3 访问开发板（在Ubuntu服务器内执行） 9](#_Toc93413807)

[2.5.4 ATC工具简介 9](#_Toc93413808)

[2.5.5 使用ATC工具在开发板内进行模型转换（在开发板内执行） 13](#_Toc93413809)

[2.5.6 模型推理（在开发板内执行） 13](#_Toc93413810)

[2.5.7 拷出推理结果（在Ubuntu服务器内执行） 14](#_Toc93413811)

[2.6 实验代码流程 15](#_Toc93413812)

[2.7 开发步骤以及关键代码解析 16](#_Toc93413813)

[2.7.1 运行管理资源申请 17](#_Toc93413814)

[2.7.2 加载模型文件，构建模型输出内存 17](#_Toc93413815)

[2.7.3 读取本地图像数据并进行预处理 18](#_Toc93413816)

[2.7.4 构建模型输入数据，进行模型推理 19](#_Toc93413817)

[2.7.5 模型推理 20](#_Toc93413818)

[2.7.6 解析模型推理结果 20](#_Toc93413819)

[2.7.7 资源释放 21](#_Toc93413820)

[2.8 实验总结 22](#_Toc93413821)

[3 基于MobileNetv2的垃圾分类 23](#_Toc93413822)

[3.1 实验介绍 23](#_Toc93413823)

[3.1.1 实验目的 23](#_Toc93413824)

[3.1.2 实验原理 23](#_Toc93413825)

[3.1.3 实验环境 23](#_Toc93413826)

[3.2 环境准备 24](#_Toc93413827)

[3.2.1 登录华为云ModelArts 24](#_Toc93413828)

[3.2.2 数据集下载 24](#_Toc93413829)

[3.2.3 ModelArts环境 24](#_Toc93413830)

[3.3 MobileNetV2训练 26](#_Toc93413831)

[3.3.1 数据集介绍 26](#_Toc93413832)

[3.3.2 导入实验所需模块 27](#_Toc93413833)

[3.3.3 配置参数 27](#_Toc93413834)

[3.3.4 训练策略 29](#_Toc93413835)

[3.3.5 整网训练 30](#_Toc93413836)

[3.3.6 模型推理 33](#_Toc93413837)

[3.4 导出AIR模型文件 34](#_Toc93413838)

[3.5 关闭项目 35](#_Toc93413839)

[3.6 垃圾分类端侧推理 35](#_Toc93413840)

[3.6.1 垃圾分类项目文件下载（在Ubuntu服务器内执行） 35](#_Toc93413841)

[3.6.2 拷贝项目文件garbage\_classification至开发板内（在Ubuntu服务器内执行） 36](#_Toc93413842)

[3.6.3 访问开发板（在Ubuntu服务器内执行） 37](#_Toc93413843)

[3.6.4 模型转换（在开发板内执行） 38](#_Toc93413844)

[3.6.5 模型推理（在开发板内执行） 38](#_Toc93413845)

[3.6.6 拷出推理结果（在Ubuntu服务器内执行） 39](#_Toc93413846)

[3.7 实验小结 40](#_Toc93413847)

[4 昇腾实践之目标检测 41](#_Toc93413848)

[4.1 实验介绍 41](#_Toc93413849)

[4.1.1 实验目的 41](#_Toc93413850)

[4.1.2 实验原理 41](#_Toc93413851)

[4.1.3 实验环境 42](#_Toc93413852)

[4.2 实验步骤 42](#_Toc93413853)

[4.2.1 目标检测项目文件下载（在Ubuntu服务器内执行） 42](#_Toc93413854)

[4.2.2 拷贝项目文件objectdetection\_python至开发板内（在Ubuntu服务器内执行） 43](#_Toc93413855)

[4.2.3 访问开发板（在Ubuntu服务器内执行） 43](#_Toc93413856)

[4.2.4 模型转换（在开发板内执行） 44](#_Toc93413857)

[4.2.5 模型推理（在开发板内执行） 44](#_Toc93413858)

[4.2.6 拷出推理结果（在Ubuntu服务器内执行） 46](#_Toc93413859)

[4.3 实验小结 47](#_Toc93413860)

[5 昇腾实践之图像分割 48](#_Toc93413861)

[5.1 实验介绍 48](#_Toc93413862)

[5.1.1 实验背景 48](#_Toc93413863)

[5.1.2 实验目的 48](#_Toc93413864)

[5.1.3 实验原理 48](#_Toc93413865)

[5.1.4 实验环境 49](#_Toc93413866)

[5.2 实验步骤 49](#_Toc93413867)

[5.2.1 图像分割项目文件下载（在Ubuntu服务器内执行） 49](#_Toc93413868)

[5.2.2 拷贝项目文件deeplabv3\_pascal\_pic至开发板内（在Ubuntu服务器内执行） 50](#_Toc93413869)

[5.2.3 访问开发板（在Ubuntu服务器内执行） 51](#_Toc93413870)

[5.2.4 模型推理（在开发板内执行） 51](#_Toc93413871)

[5.2.5 拷出推理结果（在Ubuntu服务器内执行） 52](#_Toc93413872)

[5.3 实验小结 53](#_Toc93413873)

# 实验环境介绍

## 实验介绍

### 关于本实验

本章实验会使用已经训练好的模型，在Atlas200DK上进行部署推理。

### 实验环境

在进行该实验前需要提前搭建环境，环境搭建详情请参考《Atlas 200DK合设环境搭建指南》。

注意：本章实验只是将PC端的Ubuntu服务器作为连接开发板的工具，只执行代码拷贝命令，不作为开发环境；Atlas 200DK开发板上为合设环境，既是发环境，又是运行环境，所以模型转换和推理代码皆在开发板上运行。

### 实验目的

通过本章实验，可以实现多个昇腾案例，步骤包含模型转换和在Atlas200DK上进行推理部署，使学员熟悉昇腾应用开发流程，加深对昇腾相关理论和技术的理解。

### 实验清单

|  |  |  |  |
| --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 合设环境 |
| 昇腾 AscendCL应用开发快速入门 | 基于GoogLeNet分类网络实现图像分类 | 初级 | Ubuntu服务器，Atlas200DK |
| 基于MobileNetv2的垃圾分类 | 基于MobilenetV2模型实现垃圾分类 | 初级 | Ubuntu服务器，Atlas200DK |
| 昇腾实践之目标检测 | 基于YOLOv3检测网络模型实现图像目标检测 | 初级 | Ubuntu服务器，Atlas200DK |
| 昇腾实践之图像分割 | 使用DeepLabv3+模型实现图像的语义分割 | 初级 | Ubuntu服务器，Atlas200DK |

# 昇腾 AscendCL应用开发快速入门

* 1. 实验介绍

本实验是基于Atlas 200DK的实验项目，基于GoogLeNet分类网络编写示例代码，将该示例代码部署到Atlas 200DK上，通过读取本地图像数据作为输入，对图像中的物体进行识别，并将分类结果展示出来。

## 实验目的

本实验是昇腾AscendCL应用开发快速入门实验，目的在于让学员：

了解如何在Atlas 200DK上部署应用；

了解如何在Atlas 200DK上运行管理资源申请；

了解如何加载模型文件并构建输出内存；

了解如何获取本地图像进行预处理；

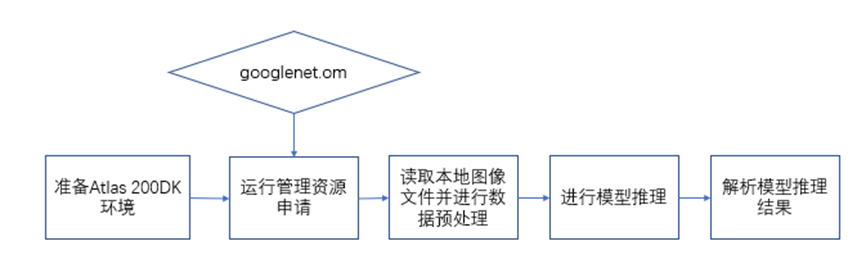
了解如何在Atlas 200DK上进行模型推理并解析推理结果。

* 1. 实验环境要求

Ubuntu18.04及以上版本；

Atlas 200DK 合设环境。

* 1. 实验原理



实验流程

**运行管理资源申请**：用于初始化系统内部资源，固定的调用流程。

**加载模型文件并构建输出内存**：从文件加载离线模型数据，需要由用户自行管理模型运行的内存，根据内存中加载的模型获取模型的基本信息包含模型输入、输出数据的数据buffer大小；由模型的基本信息构建模型输出内存，为接下来的模型推理做好准备。

**获取本地图像并进行预处理**：从本地存放图像数据的目录中循环读取图像数据并使用DVPP的JPEGD功能将图片数据解码为YUV420SP，并使用DVPP的resize功能对图像数据进行缩放至模型要求的输入图像分辨率。

**模型推理和解析模型推理结果**：根据构建好的模型输入数据进行模型推理，根据模型的输出，解析图片分类的结果，获取当前图像中识别出的物体类别以及对应的置信度，将分类结果标注在图片上进行展示。

* 1. 实验步骤

打开Ubuntu服务器，连接开发板。

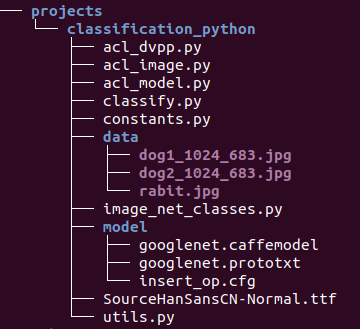
### 项目文件下载至Ubuntu服务器（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端，创建projects文件夹，下载项目文件classification\_python.zip，解压文件，得到classification\_python项目文件夹。

mkdir projects

wget https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/CANN/classification\_python.zip

unzip classification\_python.zip



项目文件目录

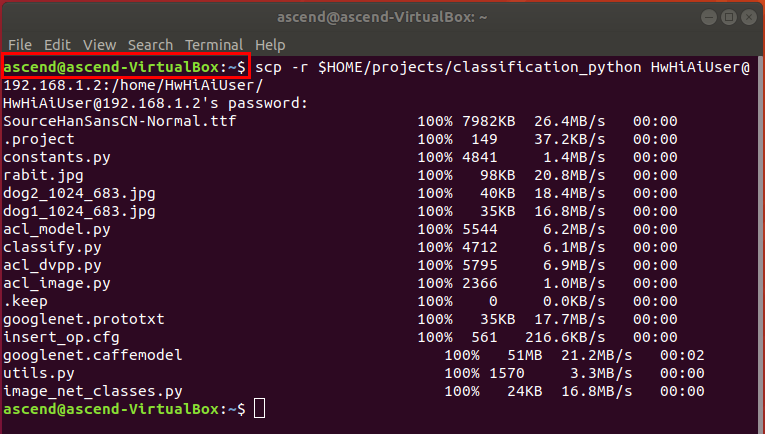


目录文件说明

### 拷贝项目文件classification\_python至开发板内（在Ubuntu服务器内执行）

在Ubuntu服务器内新建终端，输入以下命令拷贝项目文件classification\_python至开发板内，密码为Mind@123，其中$HOME/projects/classification\_python 为Ubuntu内的项目文件目录，HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects 为开发板内的项目文件目录。

scp -r $HOME/projects/classification\_python HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects

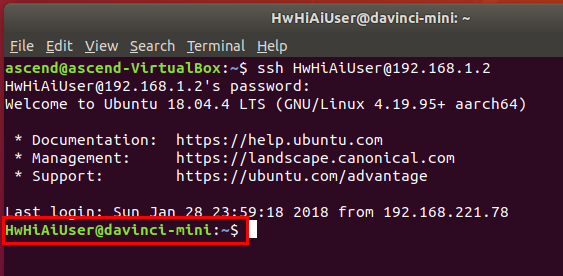


拷贝文件

### 访问开发板（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端Terminal，输入如下命令访问开发板，密码为Mind@123

ssh HwHiAiUser@192.168.1.2



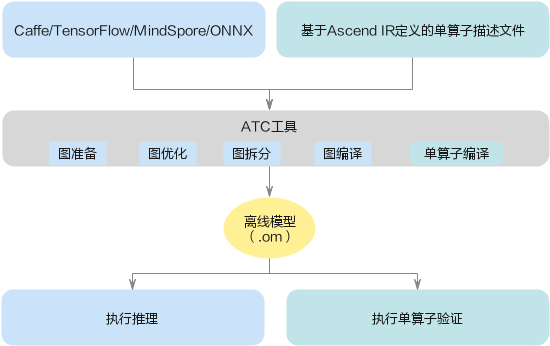
访问开发板

### ATC工具简介

ATC即模型和算子编译器。用于离线模型转换、自定义算子开发、IR构图开发等场景。

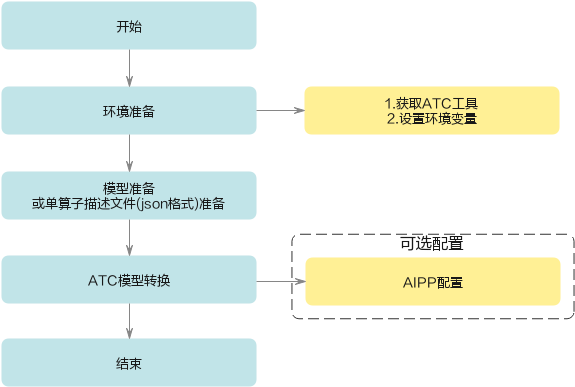
用户可以将开源框架网络模型或Ascend IR定义的单算子描述文件（json格式）通过ATC工具转换成适配昇腾AI处理器的离线模型。

ATC工具功能架构：



ATC工具功能架构

ATC工具运行流程：



ATC工具运行流程

ATC工具总体约束：

在进行模型转换前，请务必查看如下约束要求：

如果要将FasterRCNN等网络模型转成适配昇腾AI处理器的离线模型，则务必参见[定制网络修改（Caffe）](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0100.html)先修改prototxt模型文件。

支持原始框架类型为Caffe、TensorFlow、MindSpore、ONNX的模型转换，当原始框架类型为Caffe、MindSpore、ONNX时，输入数据类型为FP32、FP16（通过设置入参--input\_fp16\_nodes实现，MindSpore框架不支持该参数）、UINT8（通过配置数据预处理实现）；当原始框架类型为TensorFlow时，输入数据类型为FP16、FP32、UINT8、INT32、INT64、BOOL（原始框架类型为TensorFlow时，不支持输入输出数据类型为INT64，需要用户自行将INT64的数据类型修改为INT32类型）。

当原始框架类型为Caffe时，模型文件（.prototxt）和权重文件（.caffemodel）的op name、op type必须保持名称一致（包括大小写）。

当原始框架类型为TensorFlow时，只支持FrozenGraphDef格式。

不支持动态shape的输入，例如：NHWC输入为[?,?,?,3]多个维度可任意指定数值。模型转换时需指定固定数值。

对于Caffe框架网络模型：输入数据最大支持四维，转维算子（reshape、expanddim等）不能输出五维。

模型中的所有层算子除const算子外，输入和输出需要满足dim!=0。

只支持算子规格参考中的算子，并需满足算子限制条件。

ATC参数概览：

须知：

如果通过atc --help命令查询出的参数未解释在表1，则说明该参数预留或适用于其他芯片版本，用户无需关注。

使用atc命令进行模型转换转换时，命令有两种方式，用户根据实际情况进行选择：

* + atc param1=value1 param2=value2 ...（value值前面不能有空格，否则会导致截断，param取的value值为空）
  + atc param1 value1 param2 value2 ...

参数是否必选以--mode为0和3为准。

ATC参数

|  |  |  |  |
| --- | --- | --- | --- |
| ATC参数名称 | 参数简述（具体说明见参数描述章节） | 是否必选 | 默认值 |
| [--help或--h](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0040.html) | 显示帮助信息。 | 否 | 不涉及 |
| [--mode](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0041.html) | 运行模式。 | 否 | 0 |
| [--model](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0043.html) | 原始模型文件路径与文件名。 | 是 | 不涉及 |
| [--weight](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0044.html) | 权重文件路径与文件名。 | 否 | 不涉及 |
| [--om](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0045.html) | 需要转换为json格式的离线模型或原始模型文件的路径和文件名。 | 否 | 不涉及 |
| [--framework](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0046.html) | 原始框架类型。 | 是 | 不涉及 |
| [--input\_format](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0047.html) | 输入数据格式。 | 否 | Caffe默认为NCHW； |
| TensorFlow默认为NHWC |
| [--input\_shape](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0048.html) | 模型输入数据的shape。 | 否 | 不涉及 |
| [--input\_shape\_range](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0049.html) | 指定模型输入数据的shape范围。 | 否 | 不涉及 |
| [--dynamic\_batch\_size](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0050.html) | 设置动态batch档位参数，适用于执行推理时，每次处理图片数量不固定的场景。 | 否 | 不涉及 |
| [--dynamic\_image\_size](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0051.html) | 设置输入图片的动态分辨率参数。适用于执行推理时，每次处理图片宽和高不固定的场景。 | 否 | 不涉及 |
| [--dynamic\_dims](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0052.html) | 设置ND格式下动态维度的档位。适用于执行推理时，每次处理任意维度的场景。 | 否 | 不涉及 |
| [--singleop](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0053.html) | 单算子定义文件，将单个算子Json文件转换成适配昇腾AI处理器的离线模型。 | 否 | 不涉及 |
| [--output](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0055.html) | 如果是开源框架的网络模型，存放转换后的离线模型的路径以及文件名。 | 是 | 不涉及 |
| 如果是单算子描述文件，存放转换后的单算子模型的路径。 |
| [--output\_type](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0056.html) | 指定网络输出数据类型或指定某个输出节点的输出类型。 | 否 | 不涉及 |
| [--check\_report](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0057.html) | 预检结果保存文件路径和文件名。 | 否 | check\_result.json |
| [--json](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0058.html) | 离线模型或原始模型文件转换为json格式文件的路径和文件名。 | 否 | 不涉及 |
| [--soc\_version](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0060.html) | 模型转换时指定芯片版本。 | 是 | 不涉及 |
| [--aicore\_num](https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0062.html) | 设置模型编译时使用的aicore数目。 | 否 | 默认值为最大值 |

有关更多ATC工具的初级功能和高级功能请参考：

初级功能：<https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0007.html>

高级功能：<https://support.huaweicloud.com/atctool-cann502alpha3infer/atlasatc_16_0013.html>

### 使用ATC工具在开发板内进行模型转换（在开发板内执行）

model文件夹内有原始网络模型googlenet.prototxt，对应的权重文件googlenet.caffemodel，以及aipp配置文件insert\_op.cfg，我们可以调用ATC模型转换工具将其转换成om模型。

在开发板内进入项目文件garbage\_classification内

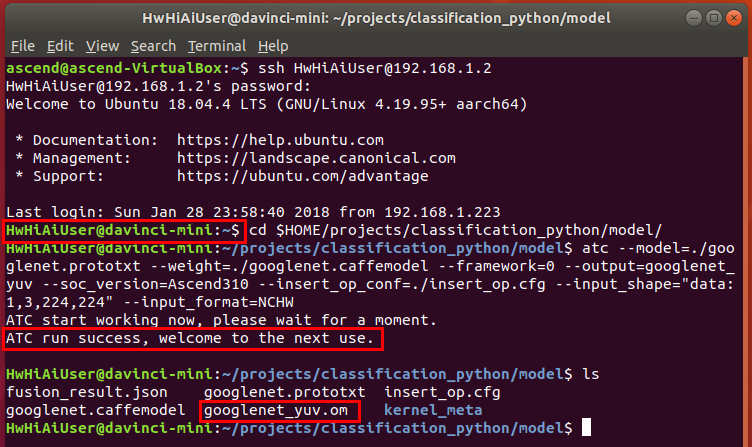
cd $HOME/projects/classification\_python/model/

执行模型转换命令

atc --model=./googlenet.prototxt --weight=./googlenet.caffemodel --framework=0 --output=googlenet\_yuv --soc\_version=Ascend310 --insert\_op\_conf=./insert\_op.cfg --input\_shape="data:1,3,224,224" --input\_format=NCHW

输出“ATC run success”即代表ATC模型转换成功。

输入“ls”查看转换好的om模型



ATC模型转换

### 模型推理（在开发板内执行）

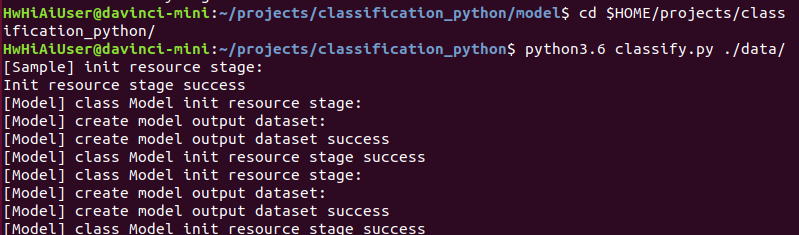
切换到classification\_python/文件夹内

cd $HOME/projects/classification\_python/

执行模型推理代码

classify.py为模型推理代码，对data文件夹内的图片进行推理。

python3.6 classify.py ./data/



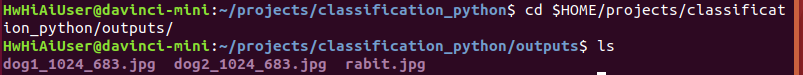
模型推理

查看推理结果

推理完成后在outputs文件夹内得到推理结果。

cd $HOME/projects/classification\_python/outputs/

ls

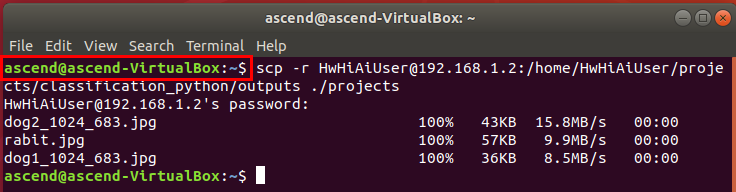


查看推理结果

### 拷出推理结果（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端，输入以下命令将推理结果图片拷贝至Ubuntu服务器的projects文件夹内，HwHiAiUser@192.168.1.2密码为Mind@123。

scp -r HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects/classification\_python/outputs ./projects



拷出推理结果

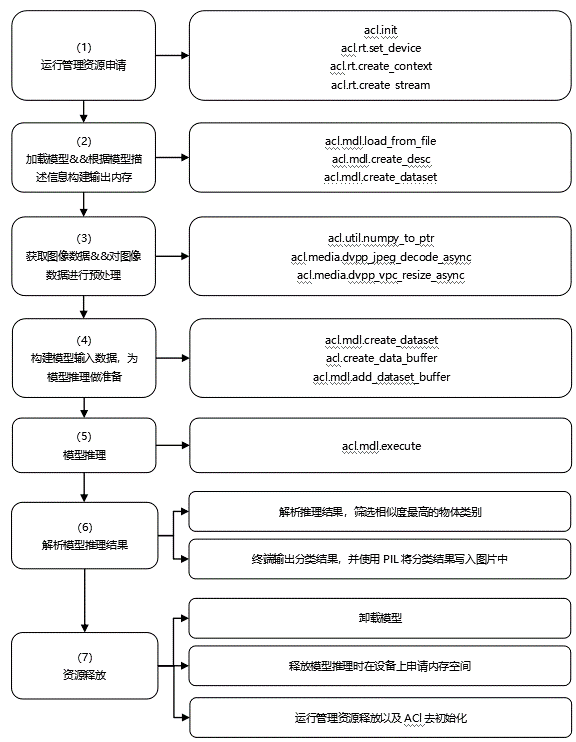
即可在Ubuntu的projects/outputs 文件夹内查看推理结果图片。



推理结果图片

* 1. 实验代码流程

图片分类代码流程图如下所示：



代码流程图

## 开发步骤以及关键代码解析

将模型部署到Atlas 200DK的步骤一般是这样的：

首先，模型转换，得到在昇腾AI处理器上能够跑起来的离线模型，经过上面的实验准备环节，已经得到了om模型。

然后，基于ACL接口进行应用开发。

### 运行管理资源申请

运行管理资源申请的功能封装在了函数Classify.\_init\_resource 中。

函数源码如下所示，ACL相关功能接口的调用已在函数中进行了说明：

def \_init\_resource(self):

print("[Sample] init resource stage:")

ret = acl.init()

check\_ret("acl.rt.set\_device", ret)

ret = acl.rt.set\_device(self.device\_id)

check\_ret("acl.rt.set\_device", ret)

self.context, ret = acl.rt.create\_context(self.device\_id)

check\_ret("acl.rt.create\_context", ret)

self.stream, ret = acl.rt.create\_stream()

check\_ret("acl.rt.create\_stream", ret)

self.run\_mode, ret = acl.rt.get\_run\_mode()

check\_ret("acl.rt.get\_run\_mode", ret)

print("Init resource stage success")

### 加载模型文件，构建模型输出内存

加载本地om模型文件到内存中，并创建并获取模型的描述信息，此函数功能封装在Model.init\_resource。

函数定义及相关源码注释如下所示：

def init\_resource(self):

print("[Model] class Model init resource stage:")

self.model\_id, ret = acl.mdl.load\_from\_file(self.model\_path)

check\_ret("acl.mdl.load\_from\_file", ret)

self.model\_desc = acl.mdl.create\_desc()

ret = acl.mdl.get\_desc(self.model\_desc, self.model\_id)

check\_ret("acl.mdl.get\_desc", ret)

output\_size = acl.mdl.get\_num\_outputs(self.model\_desc)

self.\_gen\_output\_dataset(output\_size)

print("[Model] class Model init resource stage success")

self.\_get\_output\_info(output\_size)

return SUCCESS

根据模型的描述信息，获取模型的每路输出数据在设备上所需的空间大小。

此函数功能封装在Model.\_gen\_output\_dataset中。

ACL库内置数据类型说明：aclmdlDataset主要用于描述模型推理时的输入数据或输出数据，模型可能存在多个输入、多个输出，每个输入或输出的内存地址、内存大小用aclDataBuffer类型的数据来描述。

函数定义及相关源码注释如下所示：

def \_gen\_output\_dataset(self, size):

print("[Model] create model output dataset:")

dataset = acl.mdl.create\_dataset()

for i in range(size):

temp\_buffer\_size = acl.mdl.\

get\_output\_size\_by\_index(self.model\_desc, i)

temp\_buffer, ret = acl.rt.malloc(temp\_buffer\_size,

ACL\_MEM\_MALLOC\_NORMAL\_ONLY)

check\_ret("acl.rt.malloc", ret)

dataset\_buffer = acl.create\_data\_buffer(temp\_buffer,

temp\_buffer\_size)

\_, ret = acl.mdl.add\_dataset\_buffer(dataset, dataset\_buffer)

if ret:

acl.destroy\_data\_buffer(dataset)

check\_ret("acl.destroy\_data\_buffer", ret)

self.output\_dataset = dataset

print("[Model] create model output dataset success")

### 读取本地图像数据并进行预处理

读取本地图像数据后，使用DVPP的JPEGD功能将图片数据解码为YUV420SP并使用DVPP的resize功能对图像数据进行缩放至模型要求的输入图像分辨率；将读取到的图像数据拷贝至设备侧申请的内存空间中，为接下来构建模型输入数据做好准备。

DVPP的解码功能函数封装在Dvpp.jpegd中。

函数定义及相关源码注释如下所示：

def jpegd(self, image):

print('[Dvpp] vpc decode stage:')

device\_image = image.copy\_to\_device(self.\_run\_mode)

output\_desc, out\_buffer = self.\_gen\_jpegd\_out\_pic\_desc(image)

ret = acl.media.dvpp\_jpeg\_decode\_async(self.\_dvpp\_channel\_desc,

device\_image.data(),

image.size,

output\_desc,

self.\_stream)

if ret != ACL\_ERROR\_NONE:

print("dvpp\_jpeg\_decode\_async failed ret={}".format(ret))

return None

ret = acl.rt.synchronize\_stream(self.\_stream)

if ret != ACL\_ERROR\_NONE:

print("dvpp\_jpeg\_decode\_async failed ret={}".format(ret))

return None

width = align\_up16(image.width)

height = align\_up2(image.height)

return AclImage(out\_buffer, width, height, yuv420sp\_size(width, height))

DVPP的大小缩放功能函数封装在Dvpp.resize中。

函数定义及相关源码注释如下所示：

def resize(self, image, resize\_width, resize\_height):

print('[Dvpp] vpc resize stage:')

input\_desc = self.\_gen\_input\_pic\_desc(image)

output\_desc, out\_buffer, out\_buffer\_size = \

self.\_gen\_resize\_out\_pic\_desc(resize\_width, resize\_height)

ret = acl.media.dvpp\_vpc\_resize\_async(self.\_dvpp\_channel\_desc,

input\_desc,

output\_desc,

self.\_resize\_config,

self.\_stream)

check\_ret("acl.media.dvpp\_vpc\_resize\_async", ret)

ret = acl.rt.synchronize\_stream(self.\_stream)

check\_ret("acl.rt.synchronize\_stream", ret)

print('[Dvpp] vpc resize stage success')

stride\_width = align\_up16(image.width)

stride\_height = align\_up2(image.height)

return AclImage(out\_buffer, stride\_width,

stride\_height, out\_buffer\_size)

### 构建模型输入数据，进行模型推理

构建模型的输入数据，本样例中的googlenet模型有一路输入，此输入是描述设备侧包含有图像数据的内存空间。

构建模型输入数据的功能函数封装在Model.\_gen\_input\_dataset中。

函数定义及相关源码注释如下所示：

def \_gen\_input\_dataset(self, data, data\_size):

self.input\_dataset = acl.mdl.create\_dataset()

input\_dataset\_buffer = acl.create\_data\_buffer(data, data\_size)

\_, ret = acl.mdl.add\_dataset\_buffer(

self.input\_dataset,

input\_dataset\_buffer)

if ret:

ret = acl.destroy\_data\_buffer(self.input\_dataset)

check\_ret("acl.destroy\_data\_buffer", ret)

### 模型推理

根据已经加载到内存中，要进行推理的模型ID、构建好的模型推理输入数据，调用ACL库中模型推理接口进行模型推理。

模型推理功能函数封装在了Model.execute中。

函数定义及相关源码注释如下所示：

def execute(self, data, data\_size):

self.\_gen\_input\_dataset(data, data\_size)

ret = acl.mdl.execute(self.model\_id,

self.input\_dataset,

self.output\_dataset)

check\_ret("acl.mdl.execute", ret)

return self.\_output\_dataset\_to\_numpy()

### 解析模型推理结果

模型的输出为1000个float型数据，1000个float型数据代表1000中物体类别，每个float型数据为归一化数据范围0 ~ 1。代表该类物体识别的置信度，选取其中置信度最高的类别，作为图像的分类结果。

模型推理结果解析的功能函数封装在Classify.post\_process中。

函数定义及相关源码注释如下所示：

def post\_process(self, infer\_output, image\_file):

print("post process")

data = infer\_output[0]

vals = data.flatten()

top\_k = vals.argsort()[-1:-6:-1]

print("images:{}".format(image\_file))

print("======== top5 inference results: =============")

for n in top\_k:

object\_class = get\_image\_net\_class(n)

print("label:%d confidence: %f, class: %s" % (n, vals[n], object\_class))

#使用pillow，将置信度最高的类别写在图片上，并保存到本地

if len(top\_k):

object\_class = get\_image\_net\_class(top\_k[0])

output\_path = os.path.join("./outputs", os.path.basename(image\_file))

origin\_img = Image.open(image\_file)

draw = ImageDraw.Draw(origin\_img)

font = ImageFont.truetype("SourceHanSansCN-Normal.ttf", size=30)

draw.text((10, 50), object\_class, font=font, fill=255)

origin\_img.save(output\_path)

### 资源释放

推理接收后要卸载模型，并释放与模型推理相关的资源，此功能函数封装在Model.\_\_del\_\_中。

函数定义及相关源码注释如下所示：

def \_\_del\_\_(self):

self.\_release\_dataset()

if self.model\_id:

ret = acl.mdl.unload(self.model\_id)

check\_ret("acl.mdl.unload", ret)

if self.model\_desc:

ret = acl.mdl.destroy\_desc(self.model\_desc)

check\_ret("acl.mdl.destroy\_desc", ret)

print("Model release source success")

释放存放图像数据的内存空间的功能函数封装在AclImage.destroy中。

函数定义及相关源码注释如下所示：

# 释放存放图像数据的内存空间

def destroy(self):

if self.\_memory\_type == MEMORY\_DEVICE:

acl.rt.free(self.\_data)

elif self.\_memory\_type == MEMORY\_HOST:

acl.rt.free\_host(self.\_data)

elif self.\_memory\_type == MEMORY\_DVPP:

acl.rt.free\_dvpp(self.\_data)

运行管理资源释放以及ACL去初始化的功能函数封装在Classify.\_\_del\_\_中。

函数定义及相关源码注释如下所示：

def \_\_del\_\_(self):

if self.\_model:

del self.\_model

if self.\_dvpp:

del self.\_dvpp

if self.stream:

acl.rt.destroy\_stream(self.stream)

if self.context:

acl.rt.destroy\_context(self.context)

acl.rt.reset\_device(self.device\_id)

acl.finalize()

print("[Sample] class Samle release source success")

* 1. 实验总结

本实验主要介绍如何在Atlas 200DK上进行简单地应用开发，通过读取本地图像数据作为输入，在Atlas 200DK上对图像中的物体进行识别分类，并将分类结果展示出来。

# 基于MobileNetv2的垃圾分类

## 实验介绍

本文档主要介绍垃圾分类代码开发并部署在Atlas 200 DK开发板上执行的方法。通过Atlas 200 DK开发板来实现垃圾分类推理实验，通过读取本地图像数据作为输入，对图像中的垃圾物体进行检测，并且将检测结果图片保存到文件中。用户可以通过垃圾分类项目对Atlas 200 DK开发板在AI方面的应用有全面的认识。

### 实验目的

了解熟悉垃圾分类应用代码的编写（Python语言）；

掌握将应用部署在Atlas 200 DK开发板上的操作；

了解Linux操作系统的基本使用；

掌握atc命令进行模型转换的基本操作。

### 实验原理

Garbage\_yuv.om

读取本地图像文件并进行数据预处理

加载模型文件构建输出内存

运行管理资源申请

进行模型推理

解析模型推理结果

本实验是基于Atlas 200DK的图像分类项目，基于garbage\_yuv分类网络编写的示例代码，该示例代码部署在Atlas 200DK上 ，通过读取本地图像数据作为输入，对图像中的物体进行识别分类，并将分类的结果展示出来。

### 实验环境

华为云ModelArts；

Ubuntu18.04及以上版本；

Atlas 200DK 合设环境。

## 环境准备

### 登录华为云ModelArts

进入华为云网址：<https://www.huaweicloud.com/>

通过华为账号登入；

在搜索框搜索“ModelArts”，进入ModelArts控制台。

电脑屏幕截图

描述已自动生成

ModelArts控制台

### 数据集下载

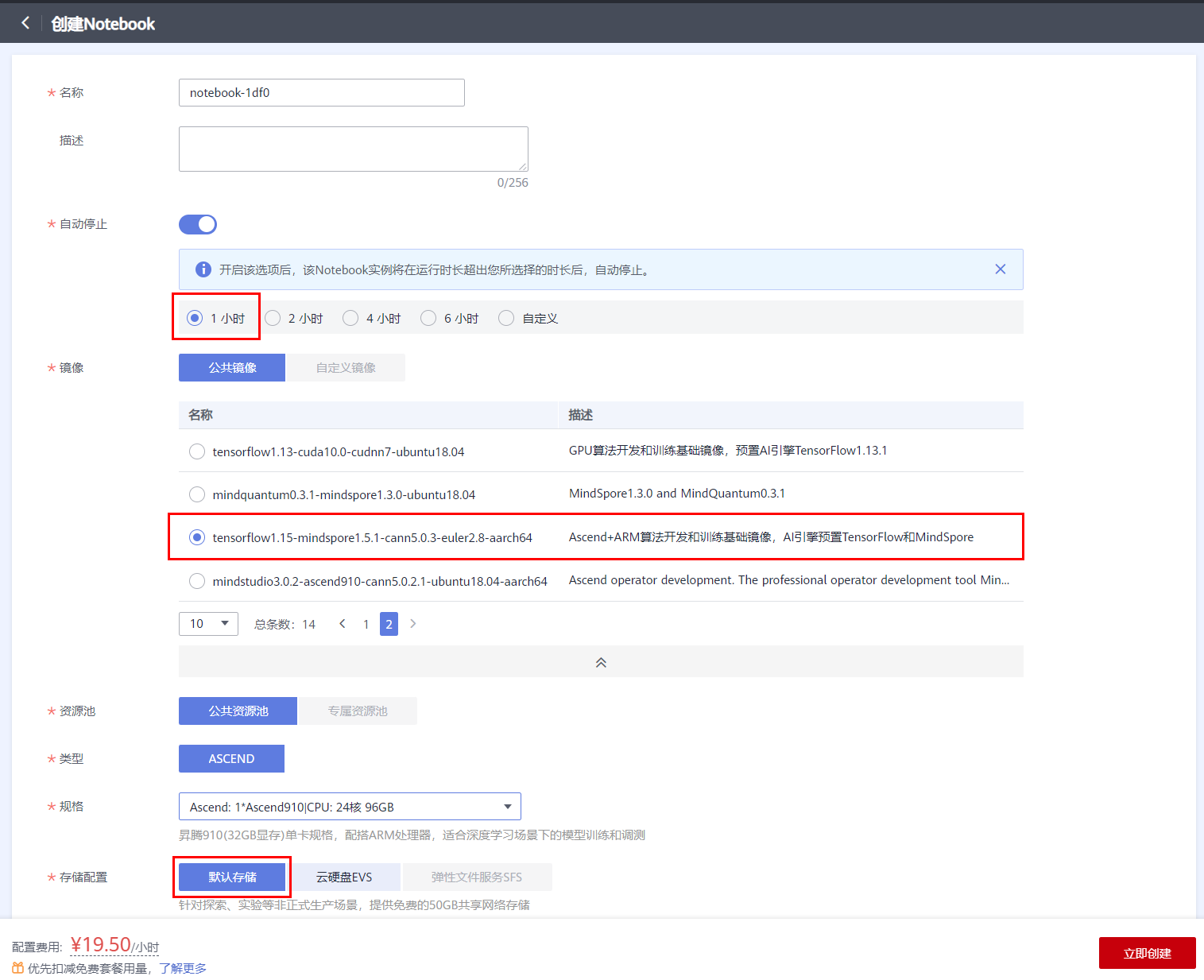
下载实验代码及数据集的压缩包：

<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/CANN/mobilenetv2_garbage.zip>

### ModelArts环境

创建Notebook

进入ModelArts控制台，点击开发环境中的Notebook，进入Notebook界面点击创建。



创建Notebook

上传数据集和代码的压缩包

将mobilenetv2\_garbage.zip压缩包上传至项目环境。

图形用户界面, 应用程序

描述已自动生成

上传数据集和代码

解压缩文件

新建Terminal，输入如下命令解压缩文件。

cd work

unzip mobilenetv2\_garbage.zip

解压完成后结果：

图形用户界面, 文本, 应用程序

描述已自动生成

解压完成后结果

新建或打开garbage.ipynb文件，开始编辑代码。

新建或打开Notebook，Kernel为MindSpore-python3.7-aarch64，开始编译。

图形用户界面, 文本, 应用程序

描述已自动生成

编译代码

## MobileNetV2训练

### 数据集介绍

MobileNetV2的代码默认使用ImageFolder格式管理数据集，数据集结构如下：

图示

中度可信度描述已自动生成

数据集结构

### 导入实验所需模块

将模块导入，具体如下：

import math

import numpy as np

import os

import random

import shutil

import time

from matplotlib import pyplot as plt

from easydict import EasyDict

from PIL import Image

import mindspore as ms

from mindspore import context

from mindspore import nn

from mindspore import Tensor

from mindspore.train.model import Model

from mindspore.train.serialization import load\_checkpoint, save\_checkpoint, export

from mindspore.train.callback import Callback, LossMonitor, ModelCheckpoint, CheckpointConfig

from dataset import create\_dataset # 数据处理脚本

from mobilenetV2 import MobileNetV2Backbone, MobileNetV2Head, mobilenet\_v2 # 模型定义脚本

os.environ['GLOG\_v'] = '2' # Log Level = Error

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend", device\_id=0) # 设置采用图模式执行，设备为Ascend#

### 配置参数

配置后续训练、验证、推理用到的参数：

# 垃圾分类数据集标签，以及用于标签映射的字典。

garbage\_classes = {

'干垃圾': ['贝壳', '打火机', '旧镜子', '扫把', '陶瓷碗', '牙刷', '一次性筷子', '脏污衣服'],

'可回收物': ['报纸', '玻璃制品', '篮球', '塑料瓶', '硬纸板', '玻璃瓶', '金属制品', '帽子', '易拉罐', '纸张'],

'湿垃圾': ['菜叶', '橙皮', '蛋壳', '香蕉皮'],

'有害垃圾': ['电池', '药片胶囊', '荧光灯', '油漆桶']

}

class\_cn = ['贝壳', '打火机', '旧镜子', '扫把', '陶瓷碗', '牙刷', '一次性筷子', '脏污衣服',

'报纸', '玻璃制品', '篮球', '塑料瓶', '硬纸板', '玻璃瓶', '金属制品', '帽子', '易拉罐', '纸张',

'菜叶', '橙皮', '蛋壳', '香蕉皮',

'电池', '药片胶囊', '荧光灯', '油漆桶']

class\_en = ['Seashell', 'Lighter','Old Mirror', 'Broom','Ceramic Bowl', 'Toothbrush','Disposable Chopsticks','Dirty Cloth',

'Newspaper', 'Glassware', 'Basketball', 'Plastic Bottle', 'Cardboard','Glass Bottle', 'Metalware', 'Hats', 'Cans', 'Paper',

'Vegetable Leaf','Orange Peel', 'Eggshell','Banana Peel',

'Battery', 'Tablet capsules','Fluorescent lamp', 'Paint bucket']

index\_en = {'Seashell': 0, 'Lighter': 1, 'Old Mirror': 2, 'Broom': 3, 'Ceramic Bowl': 4, 'Toothbrush': 5, 'Disposable Chopsticks': 6, 'Dirty Cloth': 7,

'Newspaper': 8, 'Glassware': 9, 'Basketball': 10, 'Plastic Bottle': 11, 'Cardboard': 12, 'Glass Bottle': 13, 'Metalware': 14, 'Hats': 15, 'Cans': 16, 'Paper': 17,

'Vegetable Leaf': 18, 'Orange Peel': 19, 'Eggshell': 20, 'Banana Peel': 21,

'Battery': 22, 'Tablet capsules': 23, 'Fluorescent lamp': 24, 'Paint bucket': 25}

# 训练超参

config = EasyDict({

"num\_classes": 26,

"image\_height": 224,

"image\_width": 224,

#"data\_split": [0.9, 0.1],

"backbone\_out\_channels":1280,

"batch\_size": 64,

"eval\_batch\_size": 8,

"epochs": 10,

"lr\_max": 0.05,

"momentum": 0.9,

"weight\_decay": 1e-4,

"save\_ckpt\_epochs": 1,

"save\_ckpt\_path": "./ckpt",

"dataset\_path": "./data\_en",

"class\_index": index\_en,

"pretrained\_ckpt": "./mobilenetV2-200\_1067.ckpt" # mobilenetV2-200\_1067.ckpt mobilenetv2\_ascend.ckpt

})

展示部分处理后的数据：

ds = create\_dataset(dataset\_path=config.dataset\_path, config=config, training=False)

print(ds.get\_dataset\_size())

data = ds.create\_dict\_iterator(output\_numpy=True).\_get\_next()

images = data['image']

labels = data['label']

for i in range(1, 5):

plt.subplot(2, 2, i)

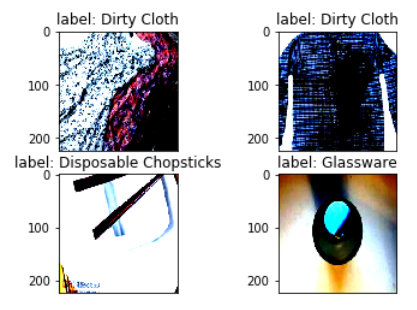
plt.imshow(np.transpose(images[i], (1,2,0)))

plt.title('label: %s' % class\_en[labels[i]])

plt.xticks([])

plt.show()

展示结果如下图所示：



测试结果

### 训练策略

一般情况下，模型训练时采用静态学习率，如0.01。随着训练步数的增加，模型逐渐趋于收敛，对权重参数的更新幅度应该逐渐降低，以减小模型训练后期的抖动。所以，模型训练时可以采用动态下降的学习率，常见的学习率下降策略有：

polynomial decay/square decay;

cosine decay;

exponential decay;

stage decay.

这里使用cosine decay下降策略：

def cosine\_decay(total\_steps, lr\_init=0.0, lr\_end=0.0, lr\_max=0.1, warmup\_steps=0):

"""

Applies cosine decay to generate learning rate array.

Args:

total\_steps(int): all steps in training.

lr\_init(float): init learning rate.

lr\_end(float): end learning rate

lr\_max(float): max learning rate.

warmup\_steps(int): all steps in warmup epochs.

Returns:

list, learning rate array.

"""

lr\_init, lr\_end, lr\_max = float(lr\_init), float(lr\_end), float(lr\_max)

decay\_steps = total\_steps - warmup\_steps

lr\_all\_steps = []

inc\_per\_step = (lr\_max - lr\_init) / warmup\_steps if warmup\_steps else 0

for i in range(total\_steps):

if i < warmup\_steps:

lr = lr\_init + inc\_per\_step \* (i + 1)

else:

cosine\_decay = 0.5 \* (1 + math.cos(math.pi \* (i - warmup\_steps) / decay\_steps))

lr = (lr\_max - lr\_end) \* cosine\_decay + lr\_end

lr\_all\_steps.append(lr)

return lr\_all\_steps

### 整网训练

在模型训练过程中，可以添加检查点（Checkpoint）用于保存模型的参数，以便进行推理及中断后再训练使用。使用场景如下：

训练后推理场景

1. 模型训练完毕后保存模型的参数，用于推理或预测操作。
2. 训练过程中，通过实时验证精度，把精度最高的模型参数保存下来，用于预测操作。

再训练场景

1. 进行长时间训练任务时，保存训练过程中的Checkpoint文件，防止任务异常退出后从初始状态开始训练。
2. Fine-tuning（微调）场景，即训练一个模型并保存参数，基于该模型，面向第二个类似任务进行模型训练。

这里加载ImageNet数据上预训练的MobileNetv2进行Fine-tuning，**只训练最后修改的FC层**，并在训练过程中保存Checkpoint。

def switch\_precision(net, data\_type):

if context.get\_context('device\_target') == "Ascend":

net.to\_float(data\_type)

for \_, cell in net.cells\_and\_names():

if isinstance(cell, nn.Dense):

cell.to\_float(ms.float32)

在面对复杂网络时，往往需要进行几十甚至几百次的epoch训练。在训练之前，很难掌握在训练到第几个epoch时，模型的精度能达到满足要求的程度，所以经常会采用一边训练的同时，在相隔固定epoch的位置对模型进行精度验证，并保存相应的模型，等训练完毕后，通过查看对应模型精度的变化就能迅速地挑选出相对最优的模型。流程如下：

定义回调函数EvalCallback，实现同步进行训练和验证；

定义训练网络并执行；

将不同epoch下的模型精度绘制出折线图并挑选最优模型Checkpoint。

当我们训练深度学习神经网络的时候通常希望能获得最好的泛化性能。但是深度学习神经网络很容易过拟合。当网络在训练集上表现越来越好，错误率越来越低的时候，就极有可能出现了过拟合。我们可以设计一种早停法，比如验证精度连续5次不在上升就停止训练，这样能避免继续训练导致过拟合的问题。

class EvalCallback(Callback):

def \_\_init\_\_(self, model, eval\_dataset, history, eval\_epochs=1):

self.model = model

self.eval\_dataset = eval\_dataset

self.eval\_epochs = eval\_epochs

self.history = history

self.acc\_max = 0

# acc连续5次<=过程中的最大值，则停止训练

self.count\_max = 5

self.count = 0

def epoch\_begin(self, run\_context):

self.losses = []

self.startime = time.time()

def step\_end(self, run\_context):

cb\_param = run\_context.original\_args()

loss = cb\_param.net\_outputs

self.losses.append(loss.asnumpy())

def epoch\_end(self, run\_context):

cb\_param = run\_context.original\_args()

cur\_epoch = cb\_param.cur\_epoch\_num

train\_loss = np.mean(self.losses)

time\_cost = time.time() - self.startime

if cur\_epoch % self.eval\_epochs == 0:

metric = self.model.eval(self.eval\_dataset, dataset\_sink\_mode=False)

self.history["epoch"].append(cur\_epoch)

self.history["eval\_acc"].append(metric["acc"])

self.history["eval\_loss"].append(metric["loss"])

self.history["train\_loss"].append(train\_loss)

self.history["time\_cost"].append(time\_cost)

if self.acc\_max < metric["acc"]:

self.count = 0

self.acc\_max = metric["acc"]

else:

self.count += 1

if self.count == self.count\_max:

run\_context.request\_stop()

print("epoch: %d, train\_loss: %f, eval\_loss: %f, eval\_acc: %f, time\_cost: %f" %(cur\_epoch, train\_loss, metric["loss"], metric["acc"], time\_cost))

在原始数据集上训练：

from mindspore.train.loss\_scale\_manager import FixedLossScaleManager

LOSS\_SCALE = 1024

def train():

train\_dataset = create\_dataset(dataset\_path=config.dataset\_path, config=config)

eval\_dataset = create\_dataset(dataset\_path=config.dataset\_path, config=config)

step\_size = train\_dataset.get\_dataset\_size()

backbone = MobileNetV2Backbone() #last\_channel=config.backbone\_out\_channels

# Freeze parameters of backbone. You can comment these two lines.

for param in backbone.get\_parameters():

param.requires\_grad = False

# load parameters from pretrained model

load\_checkpoint(config.pretrained\_ckpt, backbone)

# head = MobileNetV2Head(num\_classes=config.num\_classes, last\_channel=config.backbone\_out\_channels)

head = MobileNetV2Head(input\_channel=backbone.out\_channels, num\_classes=config.num\_classes)

network = mobilenet\_v2(backbone, head)

# define loss, optimizer, and model

loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

loss\_scale = FixedLossScaleManager(LOSS\_SCALE, drop\_overflow\_update=False)

lrs = cosine\_decay(config.epochs \* step\_size, lr\_max=config.lr\_max)

opt = nn.Momentum(network.trainable\_params(), lrs, config.momentum, config.weight\_decay, loss\_scale=LOSS\_SCALE)

model = Model(network, loss, opt, loss\_scale\_manager=loss\_scale, metrics={'acc', 'loss'})

history = {'epoch': [], 'train\_loss': [], 'eval\_loss': [], 'eval\_acc': [], 'time\_cost':[]}

eval\_cb = EvalCallback(model, eval\_dataset, history)

cb = [eval\_cb]

ckpt\_cfg = CheckpointConfig(save\_checkpoint\_steps=config.save\_ckpt\_epochs \* step\_size, keep\_checkpoint\_max=config.epochs)

ckpt\_cb = ModelCheckpoint(prefix="mobilenetv2", directory=config.save\_ckpt\_path, config=ckpt\_cfg)

cb.append(ckpt\_cb)

model.train(50, train\_dataset, callbacks=cb, dataset\_sink\_mode=False)

return history

在不同epoch下的模型进度绘制出折现图并挑选最优模型checkpoint：

if os.path.exists(config.save\_ckpt\_path):

shutil.rmtree(config.save\_ckpt\_path)

history = train()

plt.plot(history['epoch'], history['train\_loss'], label='train\_loss')

plt.plot(history['epoch'], history['eval\_loss'], 'r', label='val\_loss')

plt.legend()

plt.show()

plt.plot(history['epoch'], history['eval\_acc'], 'r', label = 'val\_acc')

plt.legend()

plt.show()

CKPT = 'mobilenetv2-%d\_40.ckpt' % (np.argmax(history['eval\_acc']) + 1) # 挑选最优模型Checkpoint，根据“eval\_acc”最大进行赋值

print("Chosen checkpoint is", CKPT)

经过观察发现epoch=11时最优，如下图所示：

图表, 折线图

描述已自动生成

Checkpoint折线图

因此这里的结果是将CKPT取值为11：

CKPT=”mobilenetv2-11\_40.ckpt”

### 模型推理

加载模型Checkpoint进行推理，使用load\_checkpoint接口加载数据时，需要把数据传入给原始网络，而不能传递给带有优化器和损失函数的训练网络。

def image\_process(image):

"""Precess one image per time.

Args:

image: shape (H, W, C)

"""

mean=[0.485\*255, 0.456\*255, 0.406\*255]

std=[0.229\*255, 0.224\*255, 0.225\*255]

image = (np.array(image) - mean) / std

image = image.transpose((2,0,1))

img\_tensor = Tensor(np.array([image], np.float32))

return img\_tensor

def infer\_one(network, image\_path):

image = Image.open(image\_path).resize((config.image\_height, config.image\_width))

logits = network(image\_process(image))

pred = np.argmax(logits.asnumpy(), axis=1)[0]

print(image\_path, class\_en[pred])

def infer():

backbone = MobileNetV2Backbone(last\_channel=config.backbone\_out\_channels)

head = MobileNetV2Head(input\_channel=backbone.out\_channels, num\_classes=config.num\_classes)

network = mobilenet\_v2(backbone, head)

load\_checkpoint(os.path.join(config.save\_ckpt\_path, CKPT), network)

for i in range(91, 100):

infer\_one(network, f'data\_en/test/Cardboard/000{i}.jpg')

infer()

## 导出AIR模型文件

导出AIR模型文件，用于后续Atlas 200 DK上的模型转换与推理。当前仅支持MindSpore+Ascend环境。

backbone = MobileNetV2Backbone(last\_channel=config.backbone\_out\_channels)

head = MobileNetV2Head(input\_channel=backbone.out\_channels, num\_classes=config.num\_classes)

network = mobilenet\_v2(backbone, head)

load\_checkpoint(os.path.join(config.save\_ckpt\_path, CKPT), network)

input = np.random.uniform(0.0, 1.0, size=[1, 3, 224, 224]).astype(np.float32)

export(network, Tensor(input), file\_name='mobilenetv2.air', file\_format='AIR') # MindSpore 1.0

# export(network, Tensor(input), file\_name='mobilenetv2.pb', file\_format='GEIR') # MindSpore 0.5

# export(network, Tensor(input), file\_name='mobilenetv2.onnx', file\_format='ONNX')

将生成的air文件下载到本机：

图形用户界面, 文本, 应用程序

描述已自动生成

下载air模型

## 关闭项目

云端项目训练完毕之后记得停止项目运行，释放资源，避免欠费。

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

停止项目

## 垃圾分类端侧推理

打开Ubuntu服务器，连接开发板。

### 垃圾分类项目文件下载（在Ubuntu服务器内执行）

垃圾分类项目文件下载

<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/CANN/garbage_classification.zip>

解压garbage\_classification.zip文件

解压文件，得到garbage\_classification文件夹。

将mobilenetv2.air模型文件放入garbage\_classification/model 文件夹内

文本

描述已自动生成

项目目录结构

### 拷贝项目文件garbage\_classification至开发板内（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端Terminal，拷贝实验文件至开发板环境内，密码为Mind@123，

其中$HOME/projects/garbage\_classification 为Ubuntu内的项目文件目录，HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects 为开发板内的项目文件目录。

scp -r $HOME/projects/garbage\_classification HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects

文本

描述已自动生成

拷贝项目文件至开发板内

### 访问开发板（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端Terminal，输入如下命令访问开发板，密码为Mind@123

ssh HwHiAiUser@192.168.1.2

文本

描述已自动生成

访问开发板

### 模型转换（在开发板内执行）

在开发板内进入项目文件garbage\_classification内

cd $HOME/projects/garbage\_classification/

进入model文件夹内

cd $HOME/projects/garbage\_classification/model/

执行模型转换命令

atc --model=./mobilenetv2.air --framework=1 --output=garbage\_yuv --soc\_version=Ascend310 --insert\_op\_conf=./insert\_op\_yuv.cfg --input\_shape="data:1,3,224,224" --input\_format=NCHW

得到转换后的om模型

文本

描述已自动生成

模型转换成功

### 模型推理（在开发板内执行）

切换到garbage\_classification/文件夹内

cd $HOME/projects/garbage\_classification/

执行模型推理代码

classify\_test.py为模型推理代码，

python3.6 src/classify\_test.py ./data/

文本

描述已自动生成

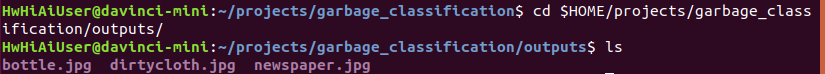
推理成功

查看推理结果

推理完成后在outputs文件夹内得到推理结果。

cd $HOME/projects/garbage\_classification/outputs/

ls



查看推理结果

### 拷出推理结果（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端，输入以下命令将推理结果图片拷贝至Ubuntu服务器的projects文件夹内，HwHiAiUser@192.168.1.2密码为Mind@123。

其中HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects/garbage\_classification/outputs是开发板推理结果目录，./projects是Ubuntu系统目录。

scp -r HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects/garbage\_classification/outputs ./projects

文本

描述已自动生成

拷出推理结果图片

即可查看推理结果图片，图片上有垃圾分类的标签：

冰箱里有许多饮料

描述已自动生成

推理结果图片

## 实验小结

本章实验介绍了昇腾垃圾分类实现过程，包括数据训练，模型转换，在Atlas200DK上进行部署推理，通过实验使学员熟悉昇腾应用开发流程，加深对昇腾相关理论的理解。

# 昇腾实践之目标检测

## 实验介绍

本实验是基于Atlas 200DK的图像目标检测项目，基于yolov3算法模型编写的示例代码，该示例代码部署在Atlas 200DK上 ，通过读取本地图像数据作为输入，对图像中的物体进行目标检测，得到检测出的物体置信度以及检测框的坐标值，最后使用Pillow模块将检测结果标注在原始图像上并写到本地文件中。

### 实验目的

本章实验的主要目的是掌握ATC命令进行模型转换的基本操作，了解模型推理相关基础知识和主要步骤，熟悉Linux编程。

### 实验原理

Yolov3\_yuv.om

解析模型推理结果

进行模型推理

读取本地图像文件

并进行数据预处理

加载模型文件构建输出内存

运行管理资源申请

**运行管理资源申请：** 用于初始化系统内部资源，固定的调用流程。

**加载模型文件并构建输出内存：** 从文件加载离线模型数据，需要由用户自行管理模型运行的内存，根据内存中加载的模型获取模型的基本信息包含模型输入、输出数据的数据buffer大小；由模型的基本信息构建模型输出内存，为接下来的模型推理做好准备。

**获取本地图像并进行预处理：** 从本地存放有图像数据的目录中循环读取图像数据并使用DVPP的JPEGD功能将图片数据解码为YUV420SP，并使用DVPP的resize功能对图像数据进行缩放至模型要求的输入图像分辨率。

**模型推理：** 根据构建好的模型输入数据进行模型推理。

**解析推理结果：**根据模型输出，解析目标检测的结果，得到图像数据中检测到的目标框的坐标值，相应的物体类别以及置信度，并使用Pillow模块将检测结果标注在图片上进行展示。

### 实验环境

Ubuntu18.04及以上版本；

Atlas 200DK 合设环境。

## 实验步骤

打开Ubuntu服务器，连接开发板。

### 目标检测项目文件下载（在Ubuntu服务器内执行）

目标检测项目文件下载

<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/CANN/objectdetection_python.zip>

解压objectdetection\_python.zip文件

解压文件，得到objectdetection\_python文件夹。

文本

描述已自动生成

项目目录结构

工程目录说明如下表所示：

|  |  |  |
| --- | --- | --- |
| **一级目录** | **二级目录/文件** | **说明** |
| Model |  | 存放转换后的om模型文件 |
| data | \*.jpg | 待处理的本地图片数据 |
| Objectdetection\_python | acl\_dvpp.py | DVPP 图片预处理功能源文件 |
| acl\_image.py | 读取本地图片数据源文件 |
| acl\_model.py | 模型加载及模型推理源文件 |
| constants.py | 存放公共变量的源文件 |
| object\_detect.py | 主函数调用源文件 |
| utils.cpp | 存放公共函数的源文件 |
| SourceHanSansCN-Normal.ttf | 开源的字体库，用于PIL画框时使用 |

工程目录

### 拷贝项目文件objectdetection\_python至开发板内（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端Terminal，拷贝实验文件至开发板环境内，密码为Mind@123，

其中$HOME/projects/objectdetection\_python 为Ubuntu内的项目文件目录，HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects 为开发板内的项目文件目录。

scp -r $HOME/projects/objectdetection\_python HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects

文本

描述已自动生成

拷贝项目文件至开发板内

### 访问开发板（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端Terminal，输入如下命令访问开发板，密码为Mind@123

ssh HwHiAiUser@192.168.1.2

文本

描述已自动生成

访问开发板

### 模型转换（在开发板内执行）

在开发板内进入项目文件objectdetection\_python内

cd $HOME/projects/objectdetection\_python/

进入model文件夹内

cd $HOME/projects/objectdetection\_python/model/

执行模型转换命令

atc --model=yolov3.prototxt --weight=yolov3.caffemodel --framework=0 --output=yolov3\_yuv --soc\_version=Ascend310 --insert\_op\_conf=aipp\_nv12.cfg

得到转换后的om模型

文本

描述已自动生成

模型转换成功

### 模型推理（在开发板内执行）

切换到objectdetection\_python文件夹内

cd $HOME/projects/objectdetection\_python/

执行模型推理代码

object\_detect.py为模型推理代码。

python3.6 object\_detect.py ./data/

文本

描述已自动生成

推理成功

推理结果解析：

文本

描述已自动生成

6

5

4

3

2

1

推理结果解析

如上图可以得到如下检测结果：

1：当前检测图片中目标检测框的个数；

2/3/4：3个检测框详细信息（检测框坐标、置信度和类别编号）；

5：当前检测图片文件路径和文件名；

6：当前检测图片的检测信息（类别编号、检测框坐标和置信度）。

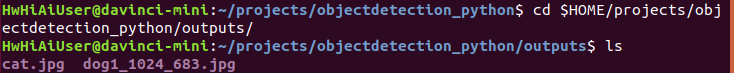
推理成功

查看推理结果

推理完成后在outputs文件夹内得到推理结果。

cd $HOME/projects/objectdetection\_python/outputs/

ls



查看推理结果

### 拷出推理结果（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端，输入以下命令将推理结果图片拷贝至Ubuntu服务器的projects文件夹内，HwHiAiUser@192.168.1.2密码为Mind@123。

其中HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects/objectdetection\_python/outputs是开发板推理结果目录，./projects是Ubuntu系统下的文件夹，需要在执行该命令前在当前路径下创建。

scp -r HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects/objectdetection\_python/outputs ./projects

文本

描述已自动生成

拷出推理结果图片

即可查看推理结果图片：

狗的照片上写着字

描述已自动生成

推理结果图片

## 实验小结

本章实验介绍了基于YOLOv3模型的目标检测，包括模型转换、在Atlas200DK上进行部署推理，通过实验使学员熟悉昇腾应用开发流程，加深对昇腾相关理论的理解。

# 昇腾实践之图像分割

* 1. 实验介绍
     1. 实验背景

本实验是基于Atlas 200DK的语义分割项目，基于DeepLabv3\_plus语义分割网络，该网络需要部署在Atlas 200DK上，通过读取本地图像数据作为输入，对图像中的物体进行目语义分割，得到分割后的结果，最后使用OpenCV写到本地文件中。

* + 1. 实验目的

掌握模型转换工具基本操作；

掌握ACL接口的基本使用方法；

掌握在Atlas200DK上运行python推理代码的方法。

* + 1. 实验原理

deeplabv3\_plus.om

后处理

运行资源管理申请

读取本地图像文件进行数据预处理

加载模型文件构建输出内存

进行模型推理

**运行管理资源申请：** 用于初始化系统内部资源，固定的调用流程。

**加载模型文件并构建输出内存：** 从文件加载离线模型数据，需要由用户自行管理模型运行的内存，根据内存中加载的模型获取模型的基本信息包含模型输入、输出数据的数据buffer大小；由模型的基本信息构建模型输出内存，为接下来的模型推理做好准备。

**获取本地图像并进行预处理：** 从本地存放有图像数据的目录中循环读取图像数据并OpenCV的resize功能对图像数据进行缩放至模型要求的输入图像分辨率。

**模型推理：** 根据构建好的模型输入数据进行模型推理。

**解析推理结果：** 根据模型输出，解析模型的推理结果。使用OpenCV处理推理后的图像数据。

* + 1. 实验环境

Ubuntu18.04及以上版本；

Atlas 200DK 合设环境（固件与驱动版本为1.0.10， CANN版本为5.0.2alpha003），合设环境内已预装多个Python版本，本实验使用Python3.6版本。

## 实验步骤

打开Ubuntu服务器，连接开发板。

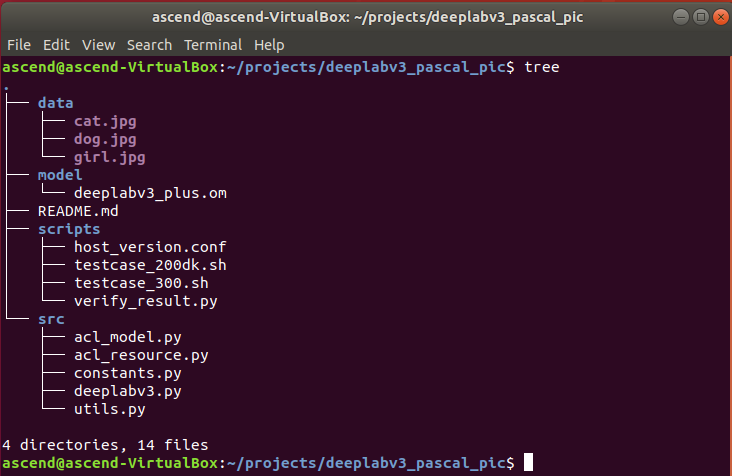
### 图像分割项目文件下载（在Ubuntu服务器内执行）

图像分割项目文件下载

<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/CANN/deeplabv3_pascal_pic.zip>

解压deeplabv3\_pascal\_pic.zip文件

解压文件，得到deeplabv3\_pascal\_pic文件夹。



项目目录结构

工程目录说明如下表所示：

工程目录

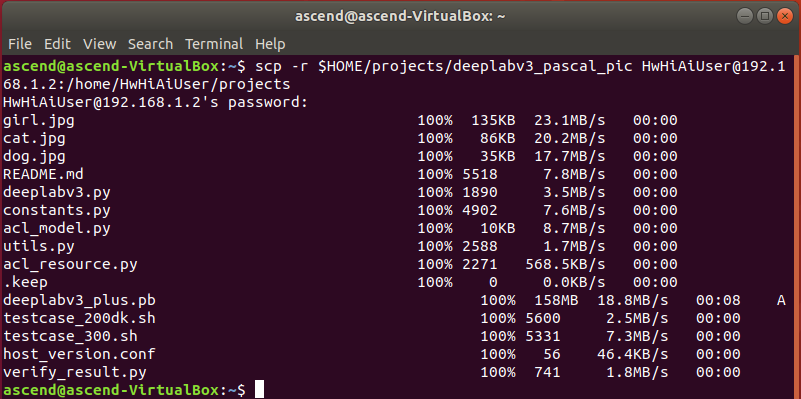
|  |  |  |
| --- | --- | --- |
| **一级目录** | **二级目录/文件** | **说明** |
| model |  | 存放转换好的om模型文件 |
| scripts | host\_version.conf | 设备型号 |
| testcase\_200dk.sh | 200DK测试脚本 |
| testcase\_300.sh | 300测试脚本 |
| verify\_result.py | 验证图片是否相同 |
| data | \*.jpg | 待处理的本地图片数据 |
| src | acl\_model.py | 模型加载及模型推理源文件 |
| constants.py | 存放公共变量的源文件 |
| deeplabv3.py | 主函数调用源文件 |
| utils.py | 存放公共函数的源文件 |

### 拷贝项目文件deeplabv3\_pascal\_pic至开发板内（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端Terminal，拷贝实验文件至开发板环境内，密码为Mind@123，

其中$HOME/projects/deeplabv3\_pascal\_pic 为Ubuntu内的项目文件目录，HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects 为开发板内的项目文件目录。

scp -r $HOME/projects/deeplabv3\_pascal\_pic HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects

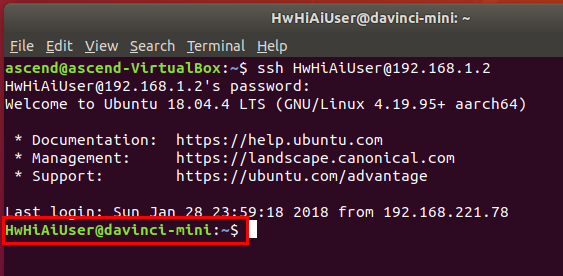


拷贝项目文件至开发板内

### 访问开发板（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端Terminal，输入如下命令访问开发板，密码为Mind@123

ssh HwHiAiUser@192.168.1.2



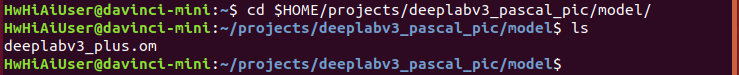
访问开发板

### 模型推理（在开发板内执行）

在开发板内进入项目文件deeplabv3\_pascal\_pic内

cd $HOME/projects/deeplabv3\_pascal\_pic/model/

ls



查看om模型

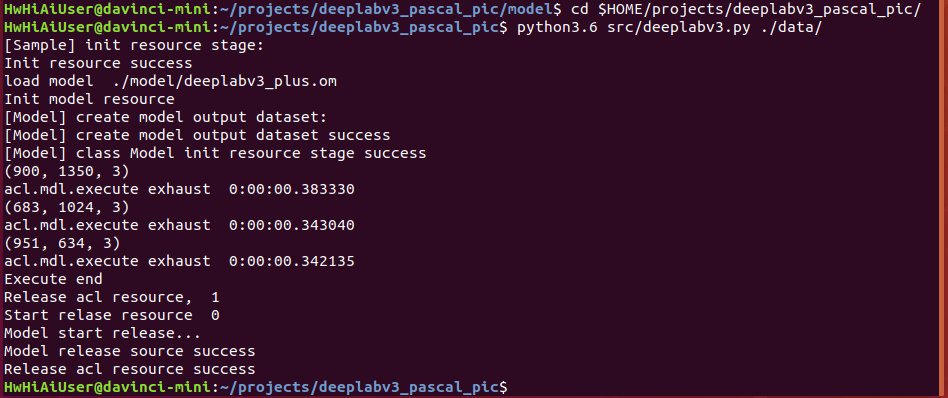
切换到deeplabv3\_pascal\_pic文件夹内

cd $HOME/projects/deeplabv3\_pascal\_pic/

执行模型推理代码

deeplabv3.py为模型推理代码，./data/为数据集路径。

python3.6 src/deeplabv3.py ./data/



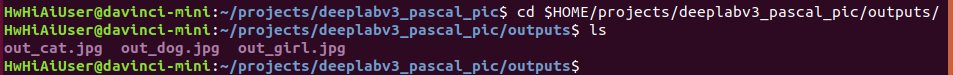
推理成功

查看推理结果

推理完成后在outputs文件夹内得到推理结果。

cd $HOME/projects/deeplabv3\_pascal\_pic/outputs/

ls



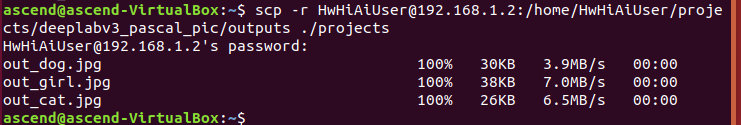
查看推理结果

### 拷出推理结果（在Ubuntu服务器内执行）

在Ubuntu服务器新建终端，输入以下命令将推理结果图片拷贝至Ubuntu服务器的projects文件夹内，HwHiAiUser@192.168.1.2密码为Mind@123。

其中HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects/deeplabv3\_pascal\_pic/outputs是开发板推理结果目录，./projects是Ubuntu系统目录。

scp -r HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects/deeplabv3\_pascal\_pic/outputs ./projects



拷出推理结果图片

即可查看推理结果图片：



推理结果图片

* 1. 实验小结

本章实验介绍了昇腾图像分割案例的完整实现过程，在Atlas200DK上进行推理部署，通过实验使学员熟悉昇腾应用开发流程，加深对昇腾相关理论和技术的理解。