基于CANN的MobileNetv2垃圾分类实验-MS1.1



华为技术有限公司

目录

[1 实验环境介绍 2](#_Toc70408125)

[1.1 实验介绍 2](#_Toc70408126)

[1.1.1 关于本实验 2](#_Toc70408127)

[1.1.2 实验环境 2](#_Toc70408128)

[1.1.3 实验目的 2](#_Toc70408129)

[1.1.4 实验清单 2](#_Toc70408130)

[2 基于MobileNetv2的垃圾分类 4](#_Toc70408131)

[2.1 实验介绍 4](#_Toc70408132)

[2.1.1 实验目的 4](#_Toc70408133)

[2.1.2 实验原理 4](#_Toc70408134)

[2.1.3 实验环境 4](#_Toc70408135)

[2.2 环境准备 5](#_Toc70408136)

[2.2.1 创建OBS 5](#_Toc70408137)

[2.2.2 ModelArts环境 6](#_Toc70408138)

[2.3 MobileNetV2训练 9](#_Toc70408139)

[2.3.1 数据集介绍 9](#_Toc70408140)

[2.3.2 导入实验所需模块 10](#_Toc70408141)

[2.3.3 配置参数 10](#_Toc70408142)

[2.3.4 训练策略 12](#_Toc70408143)

[2.3.5 整网训练 13](#_Toc70408144)

[2.3.6 模型推理 16](#_Toc70408145)

[2.4 导出AIR模型文件 17](#_Toc70408146)

[2.5 模型转换 18](#_Toc70408147)

[2.5.1 创建项目文件夹 18](#_Toc70408148)

[2.5.2 启动Docker开发环境 18](#_Toc70408149)

[2.5.3 创建垃圾分类项目文件夹 18](#_Toc70408150)

[2.5.4 上传原始air格式模型文件 19](#_Toc70408151)

[2.5.5 执行模型转换命令 20](#_Toc70408152)

[2.6 应用代码修改 21](#_Toc70408153)

[2.6.1 复制文件 21](#_Toc70408154)

[2.6.2 获取数据 21](#_Toc70408155)

[2.6.3 推理过程 21](#_Toc70408156)

[2.7 执行推理 23](#_Toc70408157)

[2.7.2 上传文件 24](#_Toc70408158)

[2.7.3 运行Python代码 24](#_Toc70408159)

[2.7.4 查看结果 25](#_Toc70408160)

[2.8 实验小结 25](#_Toc70408161)

# 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，环境同时需要运行环境和开发环境，开发环境默认为Ubuntu18+Docker镜像，运行环境默认为Atlas200DK环境。环境搭建详情请参考环境搭建实验手册。

### 实验目的

通过本章实验，可以实现多个CANN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CANN相关理论的理解。

### 实验清单

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 开发环境 | 运行环境 |
| 基于MobileNetv2的垃圾分类 | 基于MobilenetV2模型实现垃圾分类 | 初级 | Ubuntu18+Docker镜像 | Atlas200DK |

# 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

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

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

实验环境需从硬件和软件两个方面进行准备：

1. 硬件配件准备环境：

使用Atlas 200 DK前，需自行购买相关配件，包含制作Atlas 200 DK启动系统的micro SD卡、读卡器，与Ubuntu虚拟机相连接的Type-C数据线

1. 软件部署环境：

Linux + Docker镜像，镜像环境配置请参考相应指导手册。

## 环境准备

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

通过华为账号登入

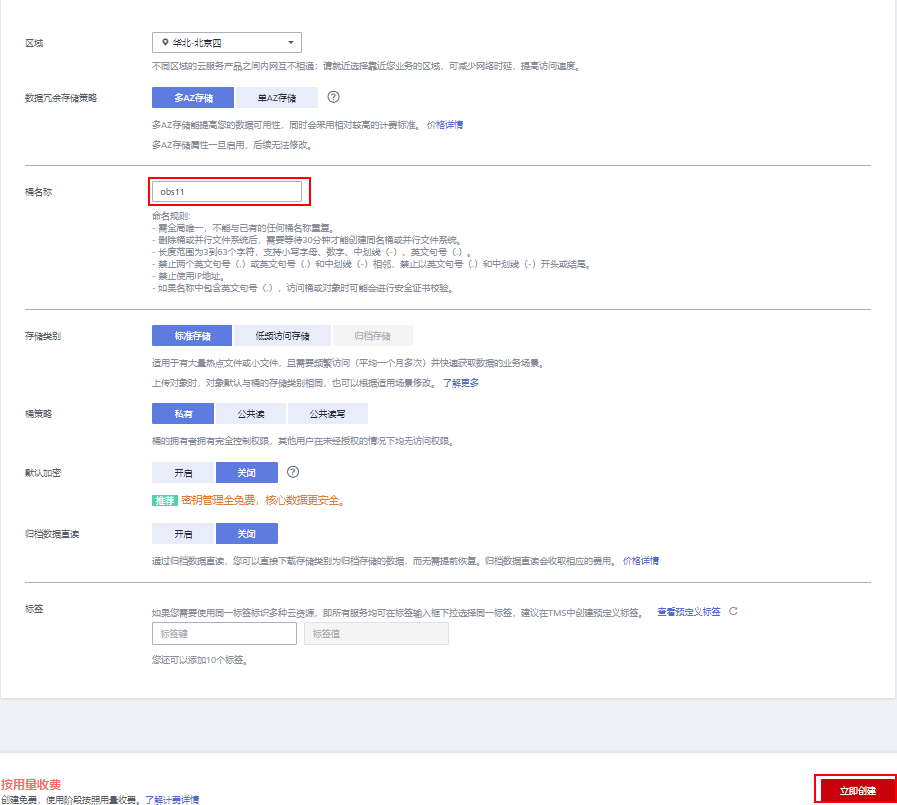
### 创建OBS

进入obs控制台界面，点击创建桶



obs控制界面

配置如下，点击“立即创建”

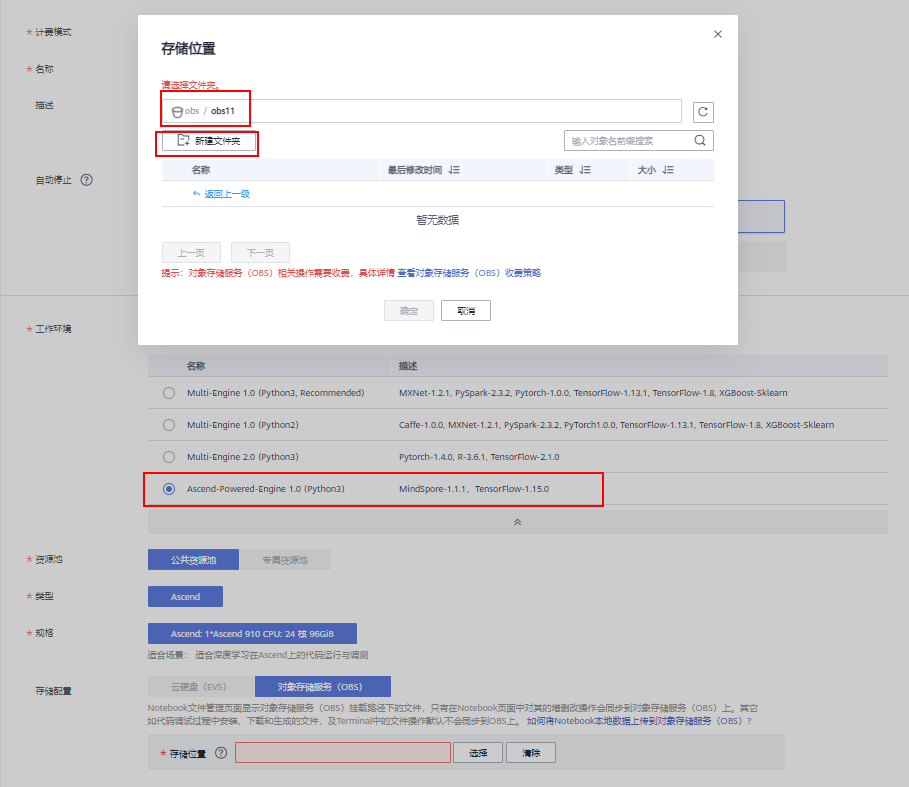


obs配置

在弹出的窗口中点击“确认”

### ModelArts环境

进入ModelArts控制台，点击开发环境中的Notebook，进入Notebook界面点击创建。工作环境选择“Ascend-Power-Engine 1.0 (Python3)”,存储位置选择之前创建的obs文件夹



Notebook创建

成功后点击“返回notebook列表”

返回obs界面，进入之前创建的桶，点击上传对象



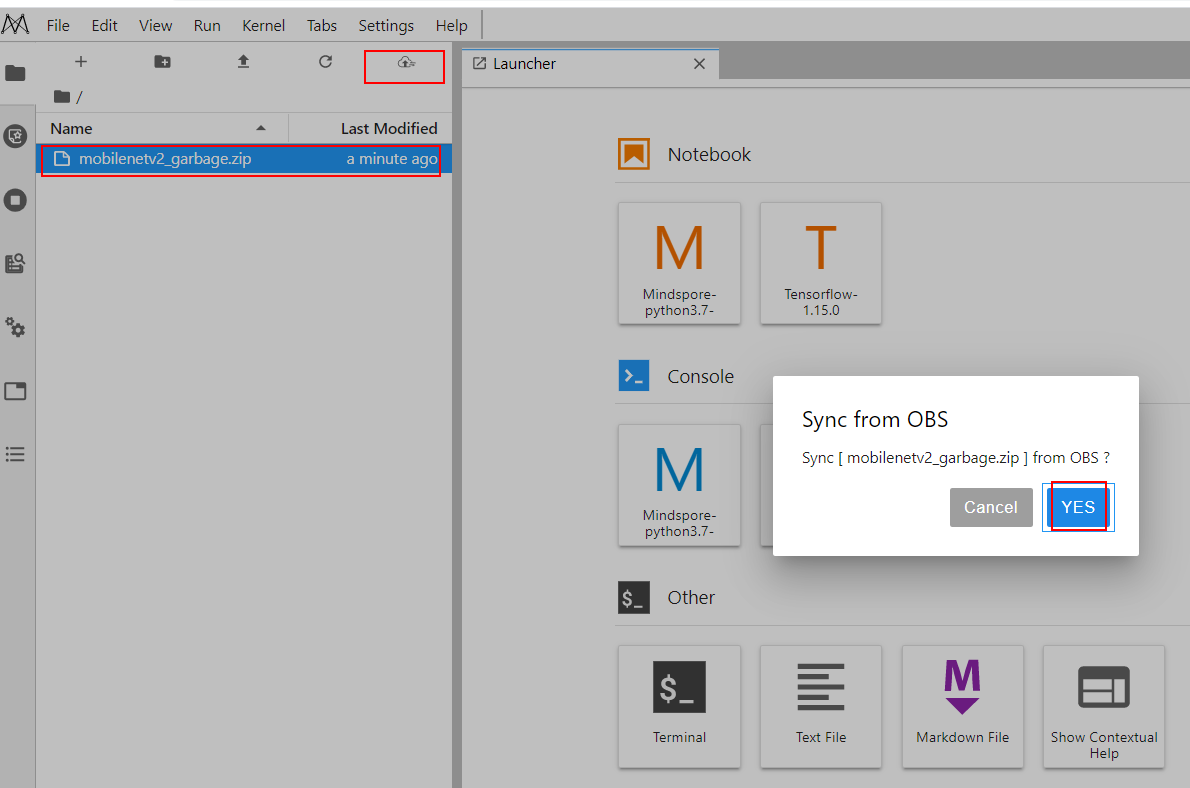
进入obs文件夹

将mobilenetv2\_garbage.zip文件通过拖拽或者添加的方式上传到obs中



obs上传文件

在notebook中打开JupyterLab界面，将所上传文件“mobilenetv2\_garbage.zip”同步到训练主机上，所下图所示：



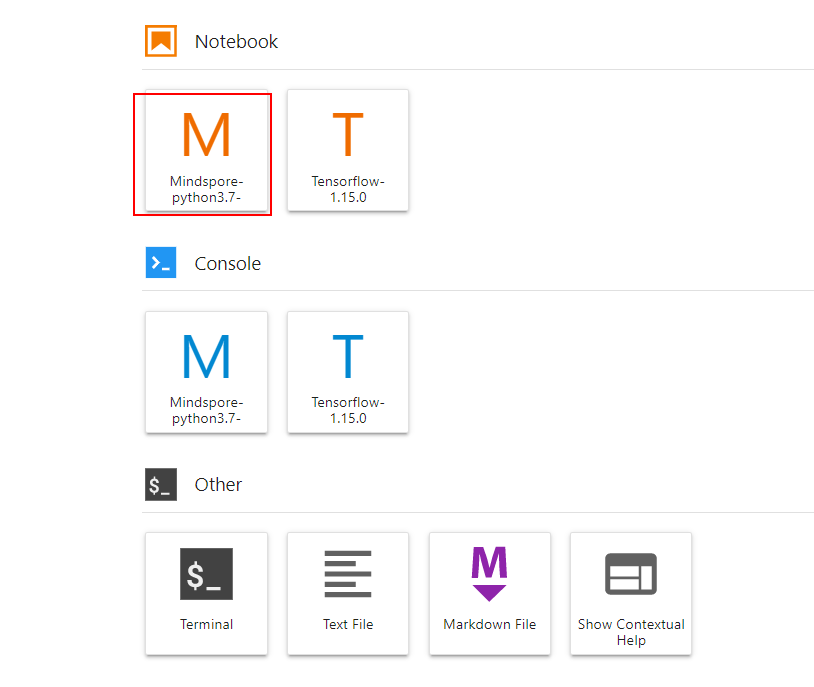
同步文件

进入Terminal环境中，解压当前文件夹

cd work

unzip mobilenetv2\_garbage.zip

解压完后点击“Mindspore-python3.7-”进入Python编辑界面，进行编译。

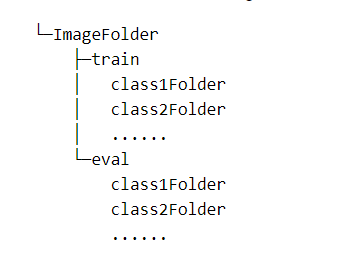


Python编辑

## 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：



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'] = '3' # Log level includes 3(ERROR), 2(WARNING), 1(INFO), 0(DEBUG).

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)

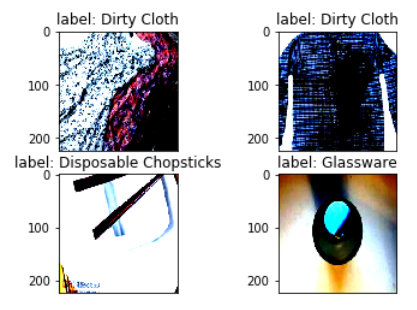
plt.imshow(np.transpose(images[i], (1,2,0)))

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')

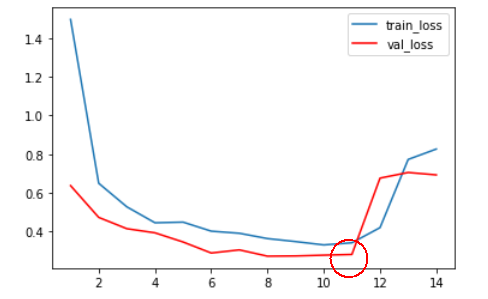
plt.legend()

plt.show()

CKPT = 'mobilenetv2-%d\_40.ckpt' % (np.argmax(history['eval\_acc']) + 1) # 挑选最优模型Checkpoint，根据数据量和batch\_size修改81值

print("Chosen checkpoint is", CKPT)

经过观察发现epoch=11时最优，如下图所示：



Checkpoint折线图

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

模型保存

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文件存放到所创的obs文件中：

import moxing as mox

mox.file.copy('/home/ma-user/work/mobilenetv2.air', 'obs://obs11/123/mobilenetv2.air')

进入obs桶内，将air文件下载保存至本地。

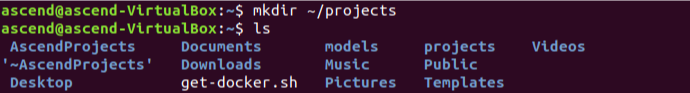
## 模型转换

### 创建垃圾分类项目

进入Ubuntu环境后，创建项目文件夹，用于存放代码以及模型

mkdir -p /home/wang/projects/garbage\_classification/model

ls命令检查文件是否创建成功：



创建文件

下载insert\_op\_yuv.cfg文件

insert\_op\_yuv.cfg下载链接： <https://c7xcode.obs.cn-north-4.myhuaweicloud.com/models/garbage_picture/insert_op_yuv.cfg>

或使用命令行下载：

wget https://c7xcode.obs.cn-north-4.myhuaweicloud.com/models/garbage\_picture/insert\_op\_yuv.cfg

将下载好的insert\_op\_yuv.cfg 和mobilenetv2.air文件存放至model目录内。

### 启动Docker开发环境

查看docker 镜像 images

docker images

查看docker容器container

docker ps -a

创建docker容器，会直接进入该容器内

docker run -it --privileged -v /dev:/dev -v /tmp:/tmp --net=host -e DISPLAY=$DISPLAY taotaoba/develop-env:cann3.3.a1

命令执行成功后进入容器，如下所示。



Docker环境

再次查看docker容器，记下container ID，用于后续拷贝文件。

可选：docker容器常见命令

\*启动容器

docker start CONTAINER ID

\*进入容器

docker exec -it CONTAINER ID /bin/bash

\*关闭容器

docker stop CONTAINER ID

\*删除docker容器

docker container rm CONTAINER ID

切换到Ascend用户

su Ascend

bash

执行成功，如下所示：



切换至Ascend用户

查看当前目录

pwd

新建terminal，通过ID在容器和宿主机之间拷贝文件，拷贝宿主机文件到Docker

docker cp path1 CONTAINER ID:path2

# docker cp ~/projects/ fbb62ac8056f:/

进入docker目录，查看是否有cfg与air文件

cd projects/garbage\_classification/model/

ls

使用ATC工具进行模型转换

ATC工具简介：<https://support.huaweicloud.com/atctool-cann330alphaXinfer/atlasatc_16_0003.html>

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 转换模型：

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

ATC参数说明：

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| --model | 原始模型文件路径与文件名。 | 是 | 不涉及 |
| --framework | 原始框架类型。 | 是 | 不涉及 |
| --input\_format | 输入数据格式。 | 否 | Caffe默认为NCHW； |
| TensorFlow默认为NHWC |
| --input\_shape | 模型输入数据的shape。 | 否 | 不涉及 |
| --output | 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 |
| --soc\_version | 模型转换时指定芯片版本。 | 是 | 不涉及 |
| --insert\_op\_conf | 插入算子的配置文件路径与文件名，例如aipp预处理算子。 | 否 | 不涉及 |

ATC参数说明

执行成功，输出如下提示

ATC run success, welcome to the next use.

执行ls命令，确认是否生成对应om文件

拷贝Docker文件到宿主机，将om文件拷贝到model目录下

docker cp CONTAINER ID:path1 path2

# docker cp fbb62ac8056f:/projects/garbage\_classification/model/garbage\_yuv.om ~/projects/garbage\_classification/model

## 端侧推理

完成以上步骤后，我们得到了所需要的网络模型。对Python模板工程进行修改和补充，构建垃圾分类算法应用。基于转换后的om模型 - garbage\_yuv.om完成推理任务。

### 编译推理代码

创建测试集文件夹

在Ubuntu系统内创建测试集文件夹data

mkdir -p ~/projects/garbage\_classification/data

cd ~/projects/garbage\_classification/data/

获取测试图片

获取测试图片：

wget https://c7xcode.obs.cn-north-4.myhuaweicloud.com/models/garbage\_picture/newspaper.jpg

wget https://c7xcode.obs.cn-north-4.myhuaweicloud.com/models/garbage\_picture/bottle.jpg

wget https://c7xcode.obs.cn-north-4.myhuaweicloud.com/models/garbage\_picture/dirtycloth.jpg

新建src文件夹存放推理代码 classify\_test.py

新建文件夹src存放推理代码文件：

mkdir -p ~/projects/garbage\_classification/src

将推理代码放到src文件夹内

编写Python脚本文件classify\_test.py：

#!/usr/bin/env python

# encoding: utf-8

import sys

import os

path = os.path.dirname(os.path.abspath(\_\_file\_\_))

import numpy as np

import acl

import base64

from PIL import Image, ImageDraw, ImageFont

from atlas\_utils.acl\_dvpp import Dvpp

import atlas\_utils.constants as const

from atlas\_utils.acl\_model import Model

from atlas\_utils.acl\_image import AclImage

from atlas\_utils.acl\_resource import AclResource

SRC\_PATH = os.path.realpath(\_\_file\_\_).rsplit("/", 1)[0]

MODEL\_PATH = os.path.join(SRC\_PATH, "../model/garbage\_yuv.om")

MODEL\_WIDTH = 224

MODEL\_HEIGHT = 224

image\_net\_classes = [

"Seashel", "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"]

def get\_image\_net\_class(class\_id):

if class\_id >= len(image\_net\_classes):

return "unknown"

else:

return image\_net\_classes[class\_id]

def pre\_process(image, dvpp):

"""preprocess"""

image\_input = image.copy\_to\_dvpp()

yuv\_image = dvpp.jpegd(image\_input)

print("decode jpeg end")

resized\_image = dvpp.resize(yuv\_image,

MODEL\_WIDTH, MODEL\_HEIGHT)

print("resize yuv end")

return resized\_image

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

print("post process")

data = infer\_output[0]

vals = data.flatten()

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

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

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

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

draw = ImageDraw.Draw(origin\_image)

font = ImageFont.load\_default()

font.size =50

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

origin\_image.save(output\_path)

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

return

def construct\_image\_info():

"""construct image info"""

image\_info = np.array([MODEL\_WIDTH, MODEL\_HEIGHT,

MODEL\_WIDTH, MODEL\_HEIGHT],

dtype = np.float32)

return image\_info

def main():

if (len(sys.argv) != 2):

print("The App arg is invalid")

exit(1)

acl\_resource = AclResource()

acl\_resource.init()

model = Model(MODEL\_PATH)

dvpp = Dvpp(acl\_resource)

image\_dir = sys.argv[1]

images\_list = [os.path.join(image\_dir, img)

for img in os.listdir(image\_dir)

if os.path.splitext(img)[1] in const.IMG\_EXT]

#Create a directory to store the inference results

if not os.path.isdir(os.path.join(SRC\_PATH, "../outputs")):

os.mkdir(os.path.join(SRC\_PATH, "../outputs"))

image\_info = construct\_image\_info()

for image\_file in images\_list:

image = AclImage(image\_file)

resized\_image = pre\_process(image, dvpp)

print("pre process end")

result = model.execute([resized\_image,])

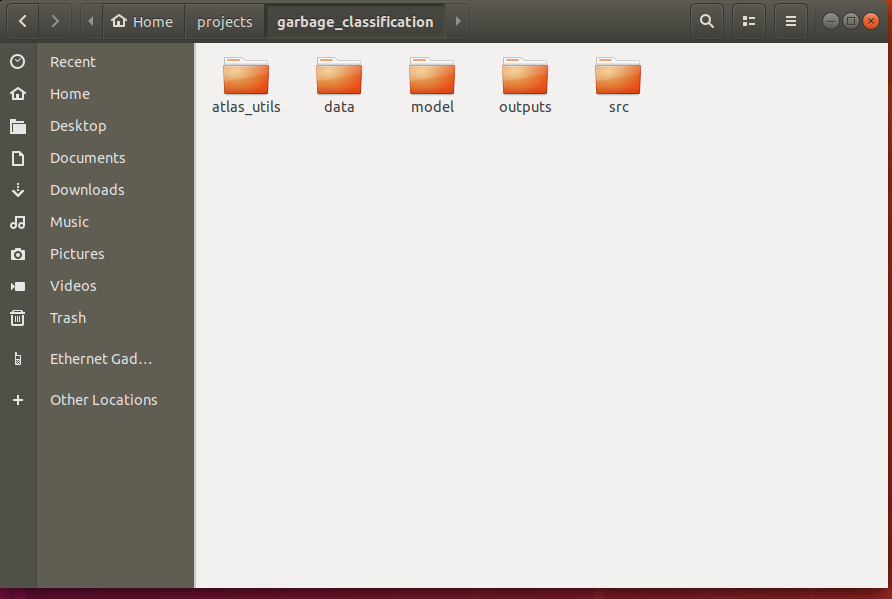
post\_process(result, image\_file)

print("process "+image\_file+" end")

if \_\_name\_\_ == '\_\_main\_\_':

main()

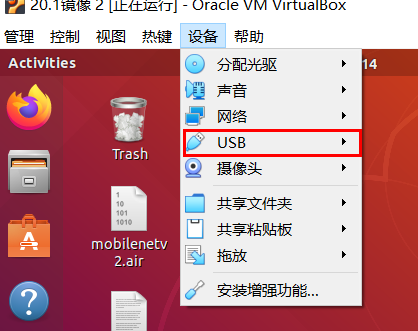
将扩展包atlas\_utils直接拖拽到ubantu文件目录内



Garbage\_classification

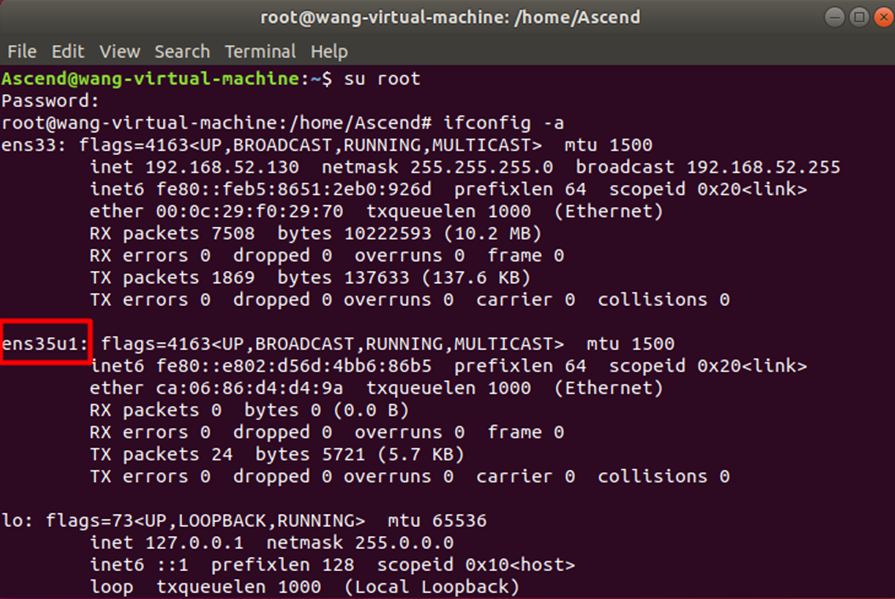
### 连接开发板到虚拟机

Atlas插入sd卡， 插入电源，四个指示灯全亮之后，通过usb-typeC 连接线连接电脑，在ubuntu系统的设备->USB处，勾选HISILICON 设备，如图所示



开发板连接Ubuntu系统成功后，打开一个终端，输入以下命令查看虚拟网卡名称。

ifconfig



查看开发板虚拟网卡名称

输入以下命令配置netplan，使得Ubuntu系统可以访问开发板。

vi /etc/netplan/01-network-manager-all.yaml

按i键，进入插入模式，随后把以下内容复制到文件当中，红色部分根据实际情况进行替换，然后按Esc，输入：后输入wq！退出并保存文件。

network:

version: 2

renderer: NetworkManager

ethernets:

ens35u1: #配置的网卡名称,使用ifconfig -a查看得到

dhcp4: no #dhcp4关闭

addresses: [192.168.1.223/8] #设置本机IP及掩码

gateway4: 255.255.255.0 #设置网关

nameservers:

addresses: [114.114.114.114]

配置完成后输入以下命令，使配置生效，至此，开发板已经可以正常和Ubuntu系统连接。

netplan apply

### 开发板端侧推理

切换到Atlas 200 DK

ssh [HwHiAiUser@192.168.1](mailto:HwHiAiUser@192.168.1).2

新建projects目录

mkdir -p ~/projects

将文件拷贝到开发板内

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

执行python推理代码

cd $HOME/ projects/garbage\_classification/

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

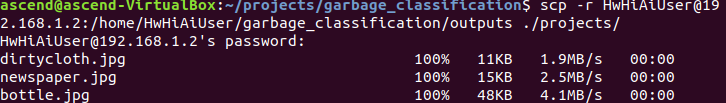
运行完成后，会在outputs目录下生成带推理结果的jpg图片。

（由于在Atlas200DK上无法直接查看结果，因此需要将output目录下的文件拷贝回Ubuntu系统进行查看，输入以下命令拷贝）

将推理结果从开发板复制到ubuntu系统

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

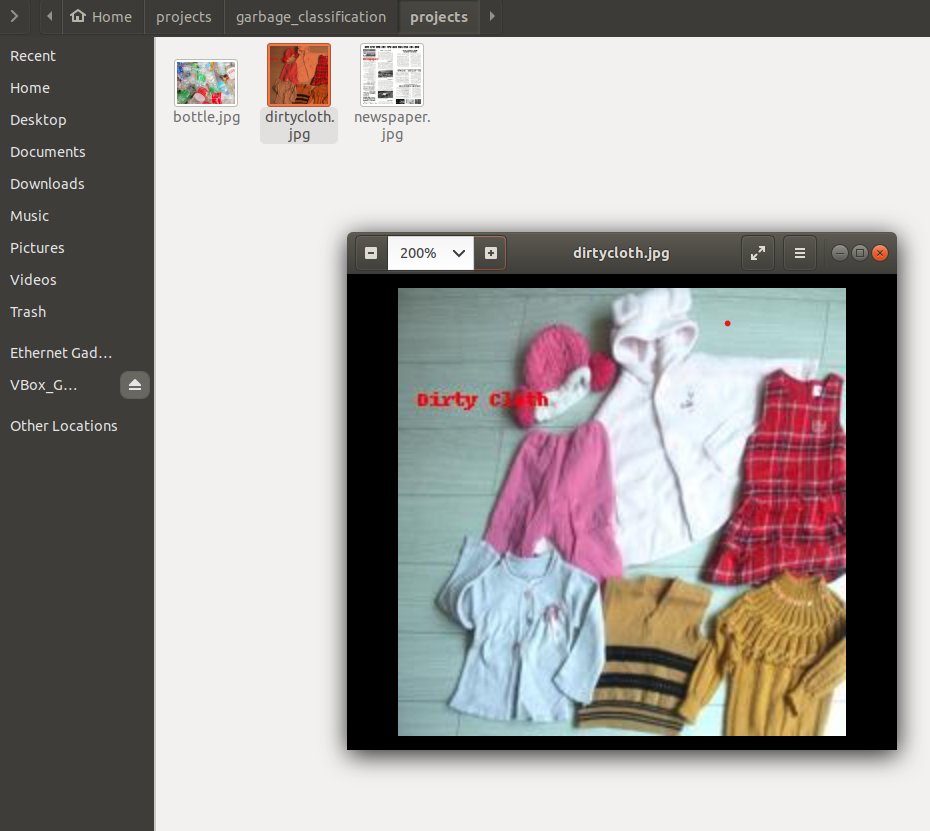
运行环境结果如下：



推理结果

查看结果

在本地Ubuntu上找到对应文件夹查看分类结果，如图所示：



垃圾分类结果

## 实验小结

本章实验介绍了CANN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CANN相关理论的理解。