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

1.1 实验介绍

1.1.1 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。

1.1.2 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。

1.1.3 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。

1.1.4 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

2 基于 MobileNetv2 的垃圾分类

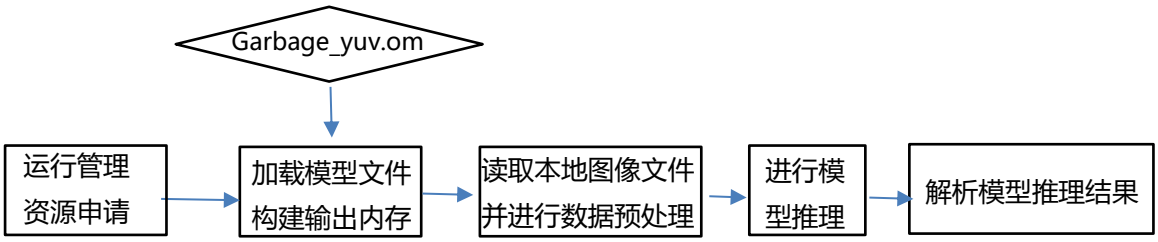
2.1 实验介绍

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。

2.1.1 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

2.1.2 实验原理



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。

2.1.3 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

2）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。

2.2 环境准备

进入华为云网址：https://www.huaweicloud.com/

通过华为账号登入

2.2.1 创建 OBS

进入 obs 控制台界面，点击创建桶

图2-1 obs 控制界面

配置如下，点击“立即创建”

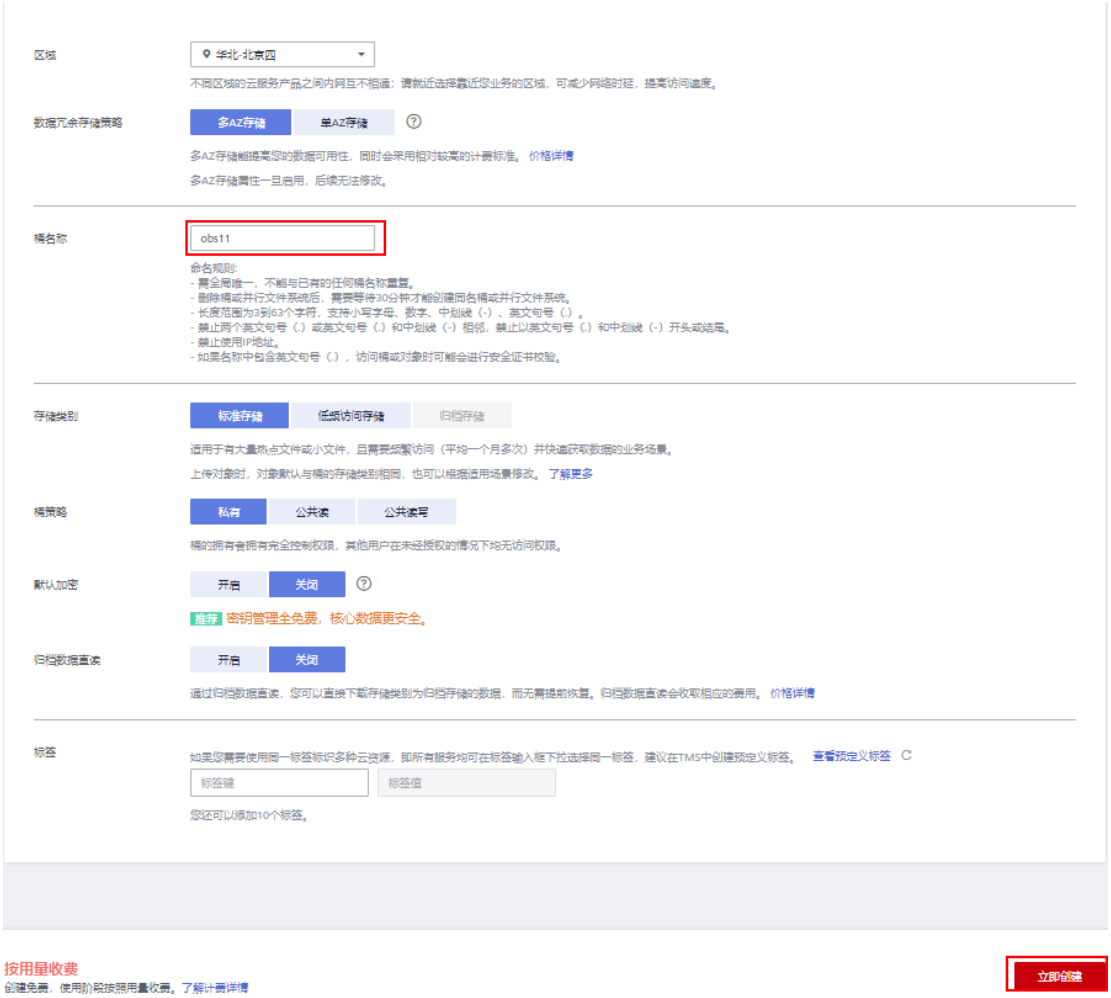


图2-2 obs 配置

在弹出的窗口中点击“确认”

2.2.2 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

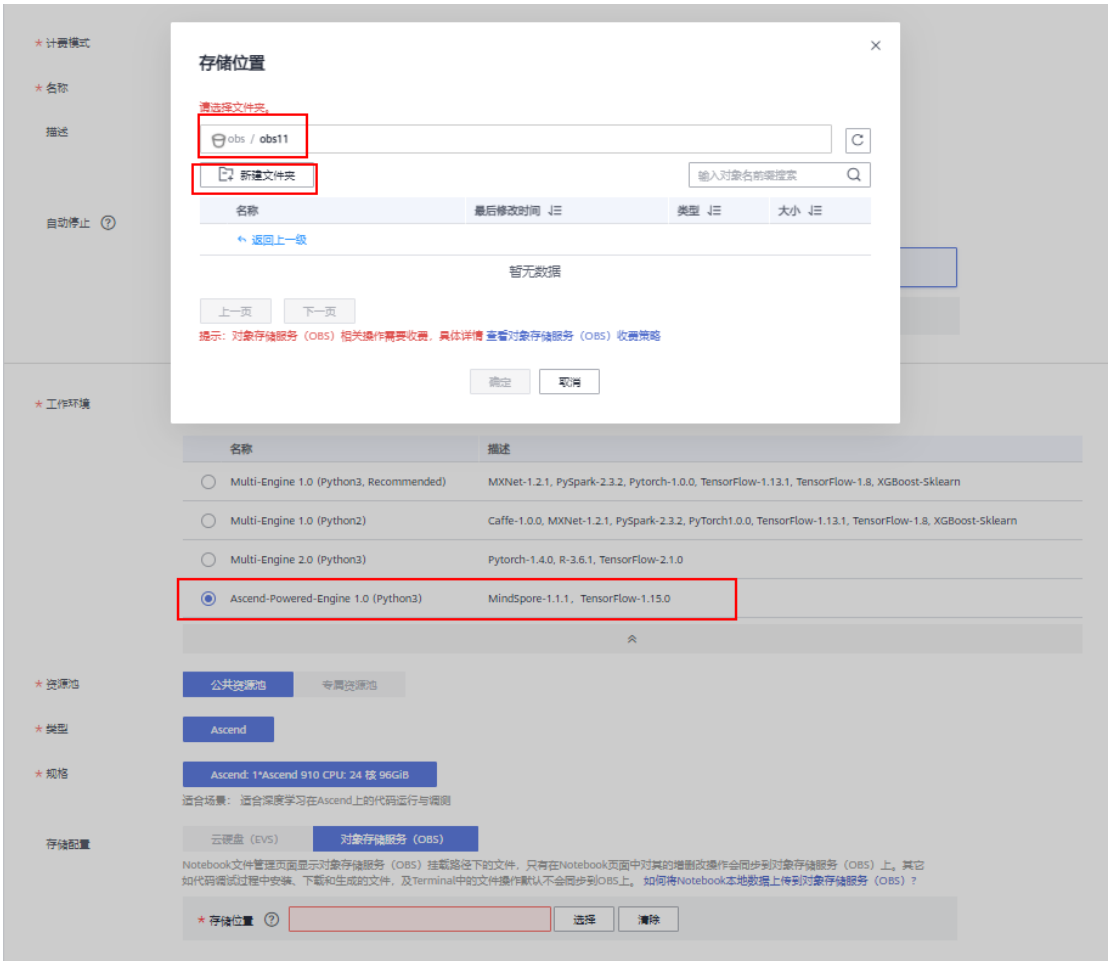


图2-3 Notebook 创建

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

图2-4 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

图2-5 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：

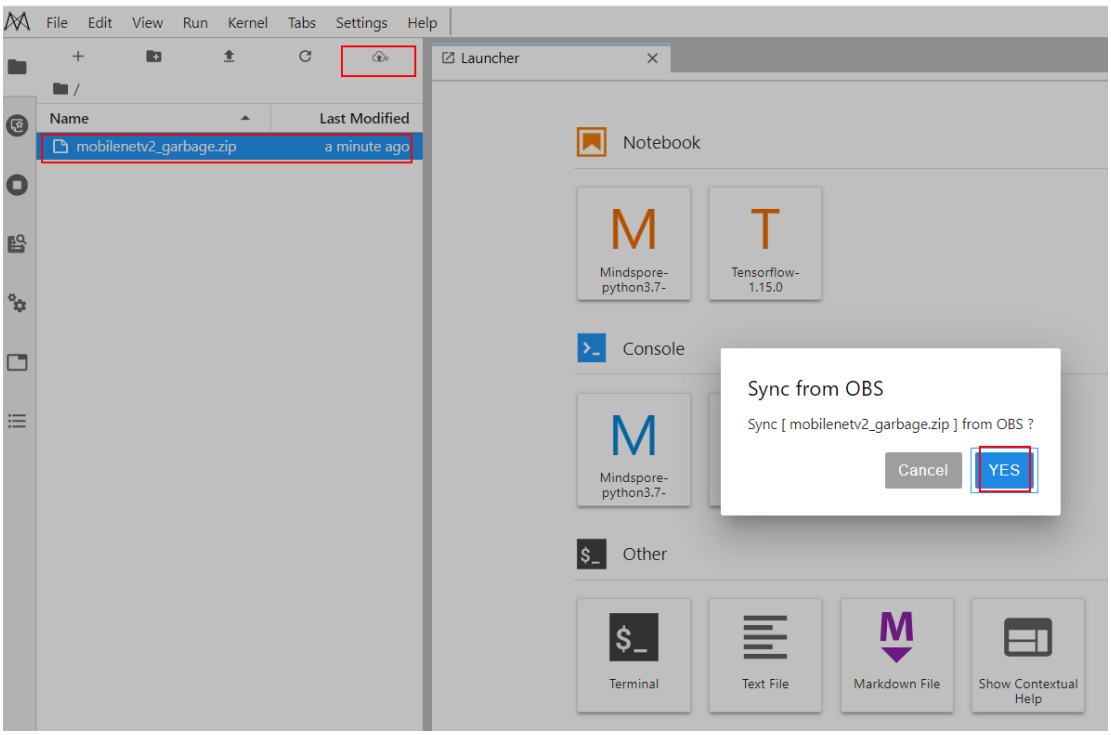


图2-6 同步文件

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。

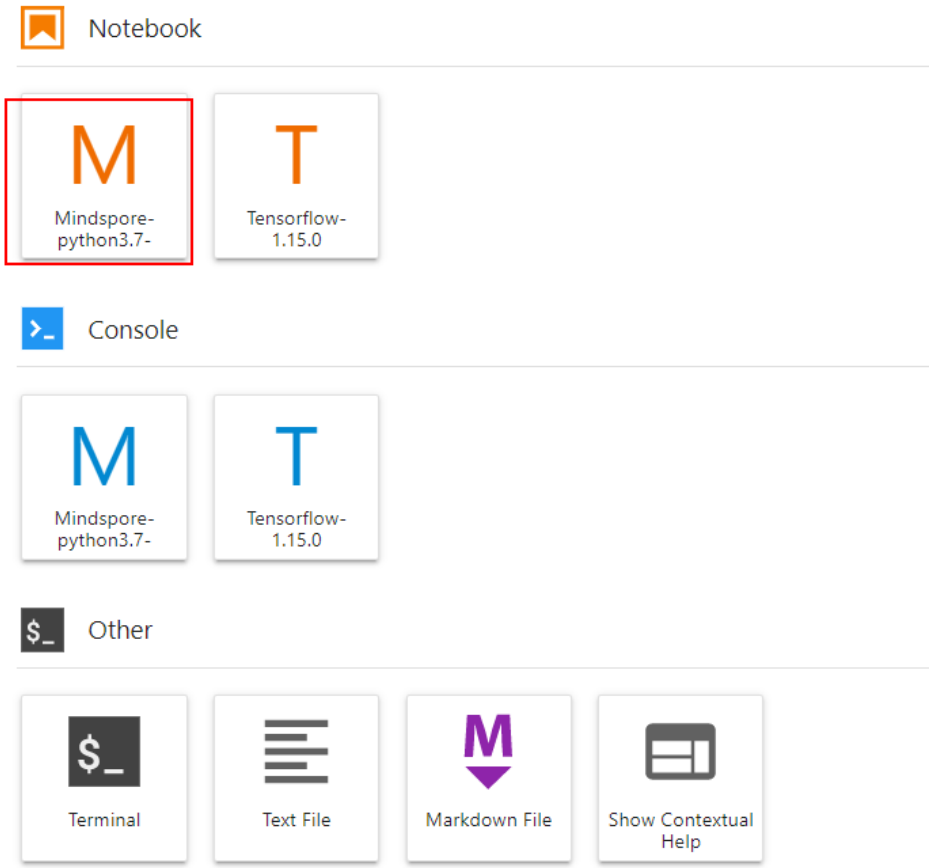


图2-7 Python 编辑

2.3 MobileNetV2 训练

2.3.1 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：

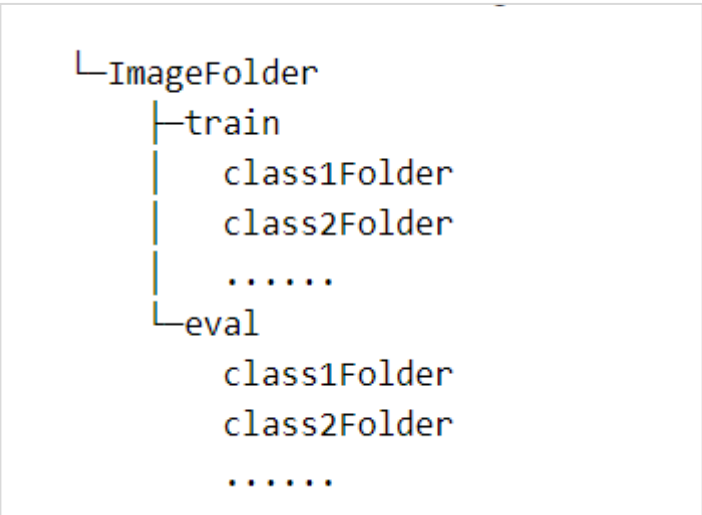


图2-8 数据集结构

2.3.2 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#
```

2.3.3 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：

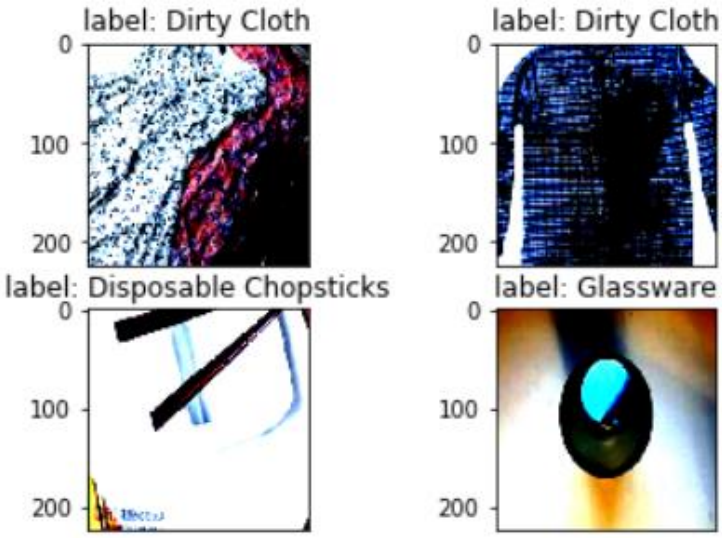


图2-9 测试结果

2.3.4 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
```

2.3.5 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：

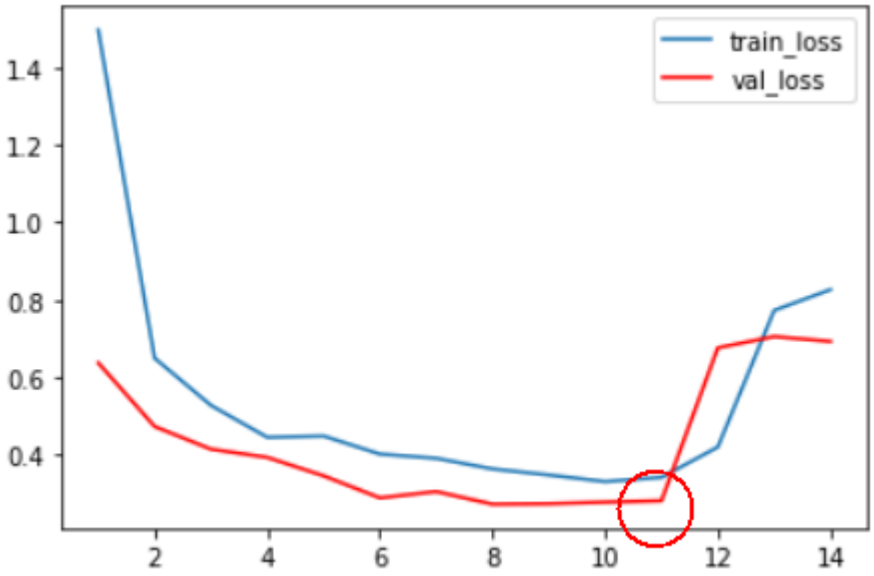


图2-10 Checkpoint 折线图

将 CKPT 取值为 11

```
CKPT="mobilenetv2-11_40.ckpt"
```

2.3.6 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()
```

2.4 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

步骤 1　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')
```

步骤 2　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')
```

步骤 3　进入 obs 桶内，将 air 文件下载保存至本地。

2.5 模型转换

2.5.1 创建垃圾分类项目

步骤 1　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：

```
ascend@ascend-VirtualBox:~$ mkdir ~/projects
ascend@ascend-VirtualBox:~$ ls
 AscendProjects      Documents       models      projects    Videos
'~AscendProjects'    Downloads       Music       Public
 Desktop             get-docker.sh   Pictures    Templates
```

图2-11 创建文件

步骤 2　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
```

步骤 3　将下载好的 insert_op_yuv.cfg 和 mobilenetv2.air 文件存放至 model 目录内。

2.5.2 启动 Docker 开发环境

步骤 1　查看 docker 镜像 images

```
docker images
```

步骤 2　查看 docker 容器 container

```
docker ps -a
```

步骤 3　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。

```
ascend@ascend-VirtualBox:~$ docker run -t -i --privileged -v /dev:/dev -v /tmp:/
tmp -v ~/projects:/home/Ascend/projects --net=host -e DISPLAY=$DISPLAY taotaoba/
develop-env:20.2
```

图2-12 Docker 环境

步骤 4　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
```

步骤 5　切换到 Ascend 用户

```
su Ascend
bash
```

执行成功，如下所示：

图2-13 切换至 Ascend 用户

查看当前目录

```
pwd
```

步骤 6　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:/
```

步骤 7　进入 docker 目录，查看是否有 cfg 与 air 文件

```
cd projects/garbage_classification/model/
ls
```

步骤 8　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

表2-1 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

步骤 9　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
```

2.6 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。

2.6.1 编译推理代码

步骤 1　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/
```

步骤 2　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
```

步骤 3　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
```

步骤 4　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()
```

步骤 5　将扩展包 atlas_utils 直接拖拽到 ubantu 文件目录内

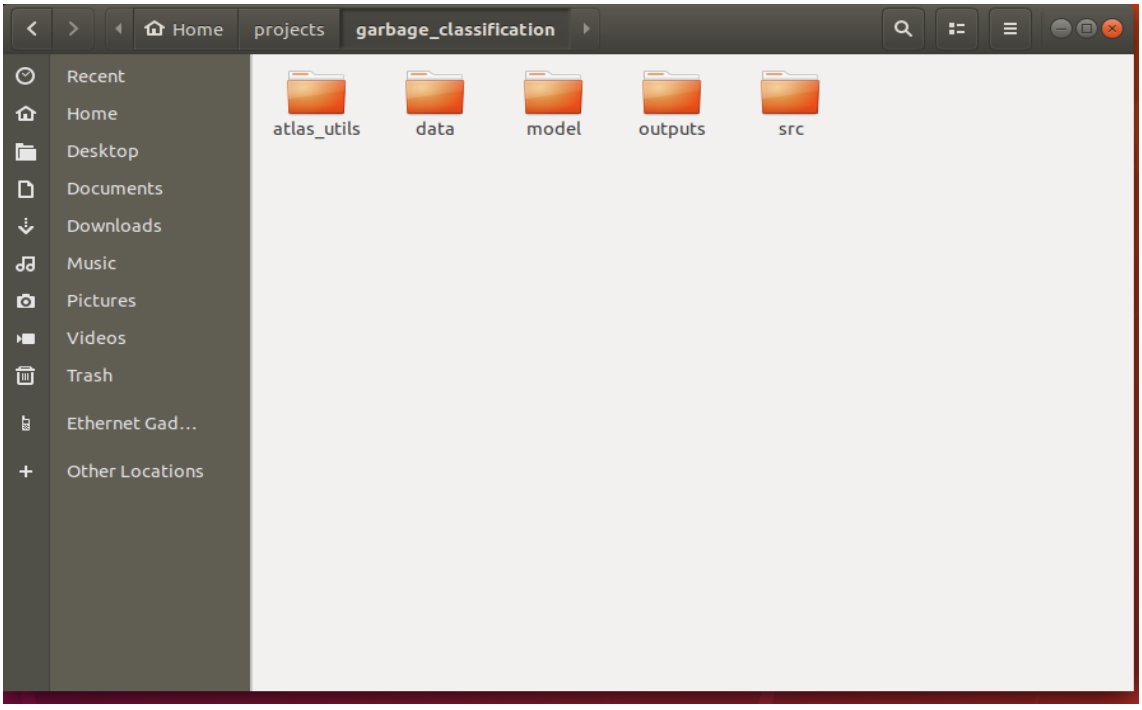


图2-14 Garbage_classification

2.6.2 连接开发板到虚拟机

步骤 1　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

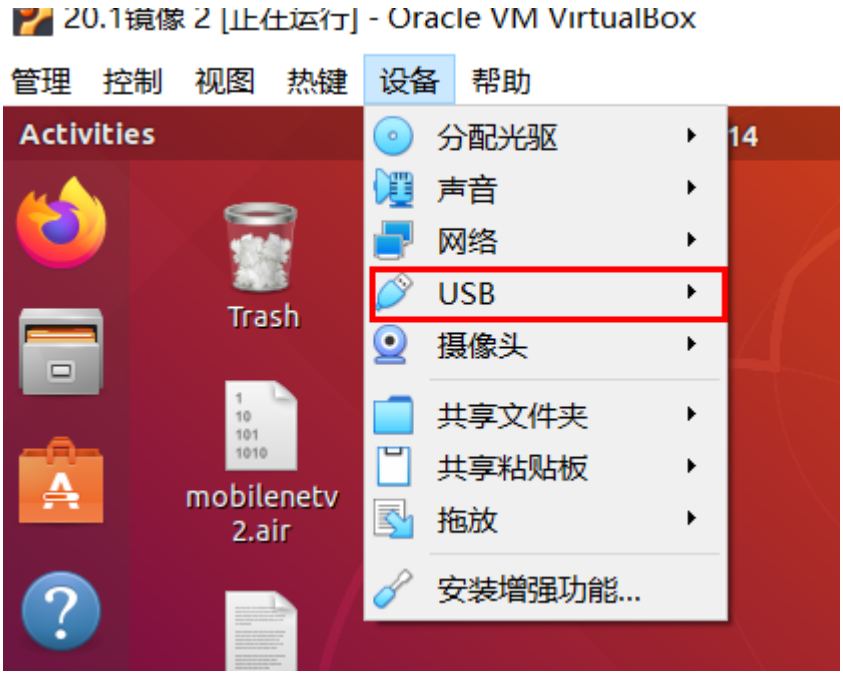


图2-15

步骤 2　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
```

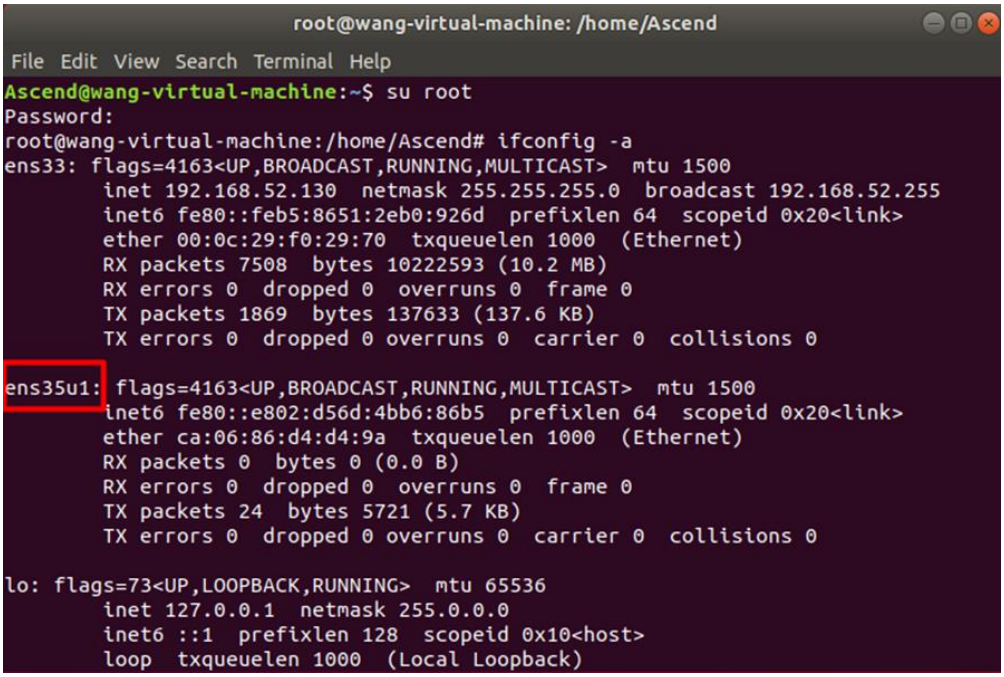


图2-16 查看开发板虚拟网卡名称

步骤 3　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
```

步骤 4　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]
```

步骤 5　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
```

2.6.3 开发板端侧推理

步骤 1　切换到 Atlas 200 DK

```
ssh HwHiAiUser@192.168.1.2
```

步骤 2　新建 projects 目录

```
mkdir -p ~/projects
```

步骤 3　将文件拷贝到开发板内

```
scp -r $HOME/projects/garbage_classification HwHiAiUser@192.168.1.2:/home/HwHiAiUser/projects
```

步骤 4　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）

步骤 5　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：

```
ascend@ascend-VirtualBox:~/projects/garbage_classification$ scp -r HwHiAiUser@19
2.168.1.2:/home/HwHiAiUser/garbage_classification/outputs ./projects/
HwHiAiUser@192.168.1.2's password:
dirtycloth.jpg                                100%   11KB   1.9MB/s   00:00
newspaper.jpg                                 100%   15KB   2.5MB/s   00:00
bottle.jpg                                    100%   48KB   4.1MB/s   00:00
```

图2-17 推理结果

步骤 6　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：

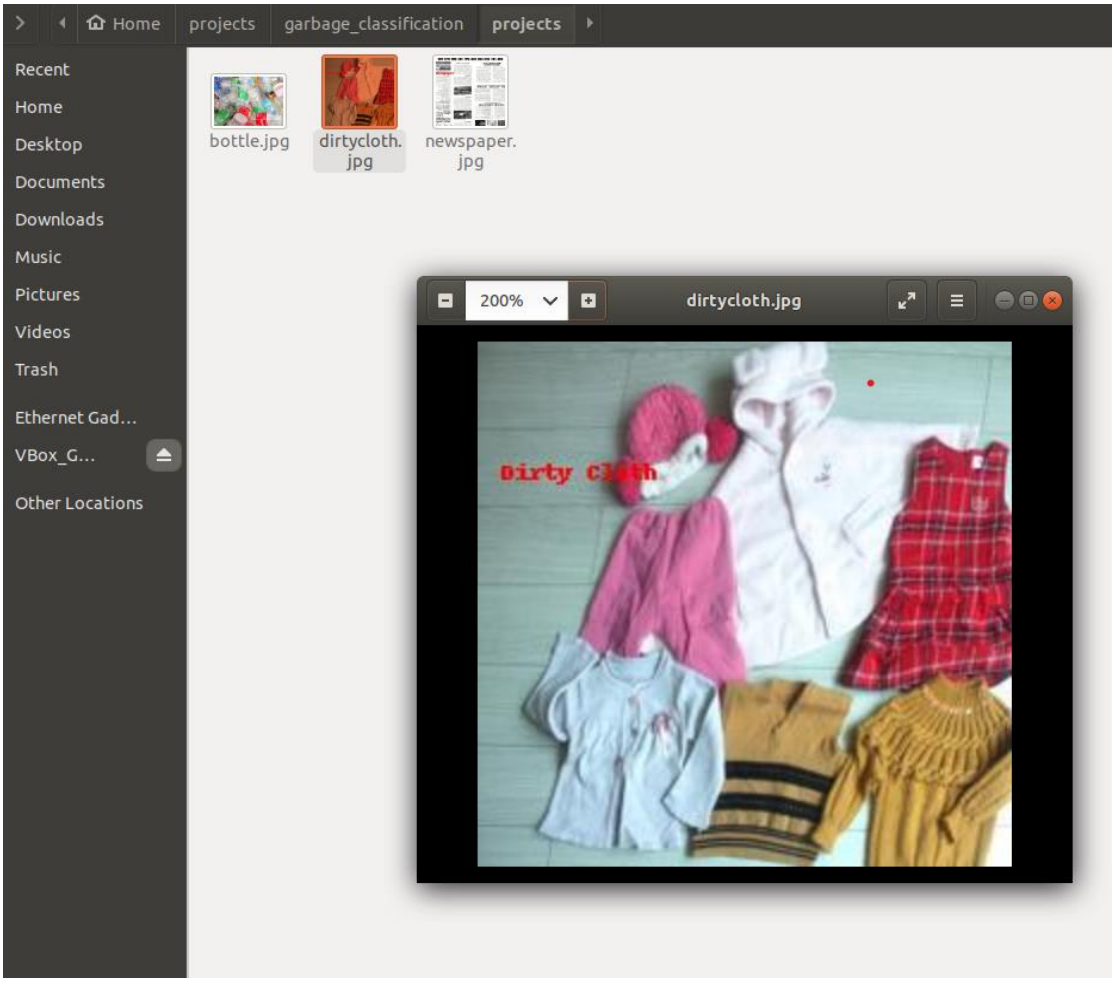


图2-18 垃圾分类结果

2.7 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。