昇腾创新实践课

毒蘑菇图像识别实验手册



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 2](#_Toc122541740)

[1.1 实验简介 2](#_Toc122541741)

[1.2 实验环境 2](#_Toc122541742)

[1.2.1 环境介绍 2](#_Toc122541743)

[1.2.2 环境搭建 2](#_Toc122541744)

[2 毒蘑菇图像识别实验 5](#_Toc122541745)

[2.1 实验介绍 5](#_Toc122541746)

[2.1.1 关于本实验 5](#_Toc122541747)

[2.1.2 实验目的 6](#_Toc122541748)

[2.1.3 背景知识 6](#_Toc122541749)

[2.1.4 实验设计 7](#_Toc122541750)

[2.2 实验过程 7](#_Toc122541751)

[2.2.1 数据文件准备 7](#_Toc122541752)

[2.2.2 环境准备 9](#_Toc122541753)

[2.2.3 参数设置 9](#_Toc122541754)

[2.2.4 数据展示 10](#_Toc122541755)

[2.2.5 数据处理 11](#_Toc122541756)

[2.2.6 网络定义 14](#_Toc122541757)

[2.2.7 模型训练 22](#_Toc122541758)

[2.2.8 模型评估 27](#_Toc122541759)

[2.2.9 效果展示 29](#_Toc122541760)

[2.2.10 模型保存 31](#_Toc122541761)

[2.3 实验总结 31](#_Toc122541762)

[2.4 课后作业：蘑菇分类模型Fine-Tune 31](#_Toc122541763)

[2.4.1 数据集 31](#_Toc122541764)

[2.4.2 实验设计要求 32](#_Toc122541765)

# 实验环境介绍

## 实验简介

本实验将使用kaggle的蘑菇数据集，采用MindSpore框架，在ModelArts平台上训练一个蘑菇分类器。

## 实验环境

### 环境介绍

实验、介绍、难度、软件环境、硬件环境：

实验环境介绍

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 实验介绍 | 难度 | 软件环境 | 开发环境 |
| 毒蘑菇图像识别识别实验 | 本实验将使用kaggle的蘑菇数据集，在ModelArts平台上训练一个蘑菇分类器。 | 困难 | Python3.7 MindSpore 1.7  Numpy 1.17.5  matplotlib 3.3.4  easydict 1.9 | 华为云ModelArts |

MindSpore是最佳匹配昇腾芯片的开源AI计算框架，支持Asend、GPU、CPU平台。MindSpore官网：https://www.mindspore.cn/

ModelArts是面向开发者的一站式AI开发平台，为机器学习与深度学习提供海量数据预处理及半自动化标注、大规模分布式Training、自动化模型生成，及端-边-云模型按需部署能力，帮助用户快速创建和部署模型，管理全周期AI工作流。

ModelArts成长地图：<https://support.huaweicloud.com/modelarts/index.html>

### 环境搭建

在华为云ModelArts创建Notebook工作环境：MindSpore1.7。

登录华为云网站

<https://www.huaweicloud.com/>

在搜索框输入ModelArts，进入ModelArts的控制台：

图形用户界面, 应用程序

描述已自动生成

创建开发环境

ModelArts区域选择华北-北京四，之后点击开发环境，进入到Notebook页面，点击创建，创建实验所需的工作环境：

电脑软件的截图

描述已自动生成

选择环境镜像、规格

项目自动停止时间可以设置为1小时；

**工作环境选择Ascend+ARM算法开发和训练基础镜像**，包含MindSpore1.7开发环境；

存储配置皆可，选择EVS更稳定。

规格：Ascend: 1\*Ascend910|CPU: 24核 96GB（单卡）

图形用户界面, 文本, 应用程序

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

配置完成之后“立即创建”，规格确认无误之后“提交”。

启动Notebook进入开发环境

当Notebook状态变为“运行中”时，点击右侧“打开”按钮打开Notebook。打开后选择右侧“MindSpore”按钮，进入Notebook环境：

图形用户界面, 应用程序, Word

描述已自动生成

# 毒蘑菇图像识别实验

## 实验介绍

### 关于本实验

全世界已知有2000多种野生食用菌的种类，云南占全国的80%、全世界的40%以上，全省境内有126个县城出产野生菌，每年吃菌的时间长达半年。

据说，每个云南人都有一个因吃菌中过毒的朋友。了解蘑菇种类，能帮助食客们分别毒蘑菇与可食用蘑菇。

本实验将使用蘑菇数据集，训练一个能分辨蘑菇种类的模型。

数据集介绍：

蘑菇数据集包含9个类，每个类有300~1600个长宽不等的彩色图像。

9个类完全相互排斥，且类之间没有重叠。

本实验使用ImageFolder格式管理数据集，每一类图片整理成单独的一个文件夹，数据集结构如下：

└─ImageFolder

├─train

│ class1Folder

│ class2Folder

│ ......

└─test

class1Folder

class2Folder

......

数据来源：<https://www.kaggle.com/maysee/mushrooms-classification-common-genuss-images> （后文实验步骤中可直接下载）



毒蘑菇数据集示意

数据集包含以下9种蘑菇：

Agaricus：双孢蘑菇，伞菌目，蘑菇科，蘑菇属，广泛分布于北半球温带，无毒。

Amanita：毒蝇伞，伞菌目，鹅膏菌科，鹅膏菌属，主要分布于我国黑龙江、吉林、四川、西藏、云南等地，有毒。

Boletus：丽柄牛肝菌，伞菌目，牛肝菌科，牛肝菌属，分布于云南、陕西、甘肃、西藏等地，有毒。

Cortinarius：掷丝膜菌，伞菌目，丝膜菌科，丝膜菌属，分布于湖南等地(夏秋季在山毛等阔叶林地上生长)，有毒。

Entoloma：霍氏粉褶菌，伞菌目，粉褶菌科，粉褶菌属，主要分布于新西兰北岛和南岛西部，有毒。

Hygrocybe：浅黄褐湿伞，伞菌目，蜡伞科，湿伞属，分布于香港(见于松仔园)，有毒。

Lactarius：松乳菇，红菇目，红菇科，乳菇属，广泛分布于亚热带松林地，无毒。

Russula：褪色红菇，伞菌目，红菇科，红菇属，分布于河北、吉林、四川、江苏、西藏等地，无毒。

Suillus：乳牛肝菌，牛肝菌目，乳牛肝菌科，乳牛肝菌属，分布于吉林、辽宁、山西、安徽、江西、浙江、湖南、四川、贵州等地，无毒。

### 实验目的

掌握在华为云ModelArts平台创建MindSpore开发环境。

掌握MindSpore实验流程。

熟悉ResNet50网络在图像分类中的应用。

### 背景知识

MindSpore基础知识，MindSpore进阶知识，图像数据预处理，卷积神经网络。

### 实验设计

1. 数据文件准备

2. 环境准备

3. 数据集下载和解压缩

4. 参数设置

5. 数据展示

6. 数据处理

7. 网络定义

8. 模型训练

9. 模型评估

10. 效果展示

11. 模型保存

## 实验过程

### 数据文件准备

本实验将使用kaggle的蘑菇数据集，在ModelArts平台上训练一个蘑菇分类器。

本实验需要以下第三方库：

split-folders 0.5.1

数据下载

在云开发平台notebook内下载处理后的[蘑菇数据集](https://www.kaggle.com/maysee/mushrooms-classification-common-genuss-images)。

!wget https://zhuanyejianshe.obs.cn-north-4.myhuaweicloud.com/chuangxinshijianke/MindSpore/archive.zip

下载后可刷新左侧窗口查看到archive.zip，并进行解压缩：

!unzip archive.zip

说明：原始数据集中目录如下，其中内层Mushrooms文件夹内的数据重复，现已手动删除。

Mushrooms

├── Agaricus

├── Amanita

├── Boletus

├── Cortinarius

├── Entoloma

├── Hygrocybe

├── Lactarius

├── Mushrooms # 重复数据，删除该文件夹

├── Russula

└── Suillus

数据清洗

Russula类别下的092\_43B354vYxm8.jpg 为损坏数据：



损坏数据

图片整改要求：

不能出现模糊区域；

可通过裁剪、删除操作进行修改。

代码：

# 删除损坏的数据

import os

bad\_data = os.path.join('Mushrooms','Russula','092\_43B354vYxm8.jpg')

if os.path.exists(bad\_data):

os.remove(bad\_data)

数据切分

使用[split-folders](https://pypi.org/project/split-folders/)切分文件夹形式数据。

【重要】切分前请确保数据文夹内仅包含以下9个文件夹：

Agaricus

Amanita

Boletus

Cortinarius

Entoloma

Hygrocybe

Lactarius

Russula

Suillus

代码：

!pip install split-folders

import splitfolders

splitfolders.ratio('Mushrooms', output="data", seed=1706, ratio=(.8, .0, .2)) # 这里Mushrooms是原始数据文件夹，data是切分后的数据文件夹

### 环境准备

Python环境导入

代码：

import mindspore.dataset as ds # 数据集载入

import mindspore.nn as nn # 各类网络层都在nn里面

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor # 回调函数

from mindspore.train import Model # 承载网络结构

from mindspore import load\_checkpoint # 读取最佳参数

from mindspore import context # 设置mindspore运行的环境

from easydict import EasyDict as ed # 超参数保存

import numpy as np # numpy

import matplotlib.pyplot as plt # 可视化

# 文件处理相关

import os

# 华为云文件传输相关

import moxing

MindSpore 环境设置

代码：

device\_target = context.get\_context('device\_target') # 获取运行装置（CPU，GPU，Ascend）

dataset\_sink\_mode = True if device\_target in ['Ascend','GPU'] else False # 是否将数据通过pipeline下发到装置上

context.set\_context(mode = context.GRAPH\_MODE, device\_target = device\_target) # 设置运行环境，静态图context.GRAPH\_MODE指向静态图模型，即在运行之前会把全部图建立编译完毕

print(f'device\_target: {device\_target}')

print(f'dataset\_sink\_mode: {dataset\_sink\_mode}')

输出：

device\_target: Ascend

dataset\_sink\_mode: True

### 参数设置

代码：

# 数据路径

train\_path = os.path.join('data', 'train')

test\_path = os.path.join('data', 'test')

# 超参数

config = ed({

# 训练参数

'batch\_size': 32,

'epochs': 150,

#网络参数

'class\_num': 9,

# 动态学习率调节

'warmup\_epochs': 5,

'lr\_init': 0.01,

'lr\_max': 0.1,

# 优化器参数

'momentum': 0.9,

'weight\_decay': 4e-5})

### 数据展示

代码：

# 创建图像标签列表

category\_dict = {0:'Agaricus',1:'Amanita',2:'Boletus',3:'Cortinarius',4:'Entoloma',

5:'Hygrocybe',6:'Lactarius',7:'Russula',8:'Suillus'}

# 载入展示用数据

demo\_ds = ds.ImageFolderDataset(test\_path, decode=True)

# 设置图像大小

plt.figure(figsize=(6, 6))

# 打印9张子图

i = 1

for dic in demo\_ds.create\_dict\_iterator():

plt.subplot(3,3,i)

plt.imshow(dic['image'].asnumpy()) # asnumpy：将 MindSpore tensor 转换成 numpy

plt.axis('off')

plt.title(category\_dict[dic['label'].asnumpy().item()])

i +=1

if i > 9 :

break

plt.show()

输出：



### 数据处理

计算数据集平均数和标准差

计算数据集平均数和标准差，数据标准化时使用

代码：

train\_ds = ds.ImageFolderDataset(train\_path, decode=True)

#计算数据集平均数和标准差，数据标准化时使用

tmp = np.asarray( [np.mean(x['image'], axis=(0, 1)) for x in train\_ds.create\_dict\_iterator(output\_numpy=True)] )

RGB\_mean = tuple(np.mean(tmp, axis=(0)))

RGB\_std = tuple(np.std(tmp, axis=(0)))

print(RGB\_mean)

print(RGB\_std)

输出：

(100.03388269705046, 94.57511259248079, 72.14921665851293)

(23.35913427414271, 20.336537235643164, 21.376613547858327)

自定义数据预处理preprocess.py 文件

定义数据预处理函数，函数功能包括：

1. 加载数据集

2. 打乱数据集

3. 图像特征处理（标准化、通道转换等）

4. 批量输出数据

5. 重复

因蘑菇数据集各个类的样本数差异较大，为确保评估公正性，这里对验证集使用PKSampler进行平均采样（欠采样）。

本实验附件配置文件已包含preprocess.py，在开发环境内从本地上传即可：

电脑软件截图

描述已自动生成

代码展示如下：

import mindspore.dataset.vision.c\_transforms as CV

import mindspore.dataset.transforms.c\_transforms as C

import mindspore.common.dtype as mstype

import mindspore.dataset as ds

def create\_dataset(data\_path, mean=None, std=None, repeat\_num=1, batch\_size=32, usage='train'):

"""

数据处理

Args:

dataset\_path (str): 数据路径

repeat\_num (int): 数据重复次数

batch\_size (int): 批量大小

usage (str): 训练或测试

Returns:

Dataset对象

"""

# 载入数据集

if usage=='train':

data = ds.ImageFolderDataset(data\_path)

else:

# 每类取63个样本（最小类样本总数）

sample\_num = 63

data = ds.ImageFolderDataset(data\_path, sampler=ds.PKSampler(sample\_num))

# 打乱数据集

data = data.shuffle(buffer\_size=10000)

# 设定resize和normalize参数

image\_size = 224

rgb\_mean = mean

rgb\_std = std

# 定义算子

if usage=='train':

trans = [

CV.RandomCropDecodeResize(image\_size, scale=(0.08, 1.0), ratio=(0.75, 1.333)),

CV.RandomHorizontalFlip(prob=0.5),

CV.Normalize(mean=mean, std=std),

CV.HWC2CHW()

]

else:

trans = [

CV.Decode(),

CV.Resize(256),

CV.CenterCrop(image\_size),

CV.Normalize(mean=mean, std=std),

CV.HWC2CHW()

]

type\_cast\_op = C.TypeCast(mstype.int32)

# 算子运算

data = data.map(operations=trans, input\_columns="image")

data = data.map(operations=type\_cast\_op, input\_columns="label")

# 批处理

if usage == 'train':

drop\_remainder = True

else:

drop\_remainder = False

data = data.batch(batch\_size, drop\_remainder=drop\_remainder)

# 重复

data = data.repeat(repeat\_num)

return data

调用preprocess.py 文件

代码：

from preprocess import create\_dataset

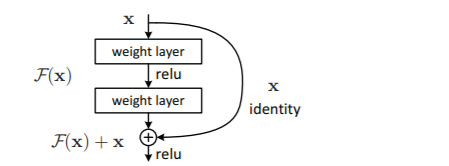
### 网络定义

本实验使用ResNet50网络。

ResNet出自论文《Deep Residual Learning for Image Recognition》。

ResNet又称残差网络，是由Microsoft Research提出的一种卷积神经网络，在2015年的ImageNet竞赛中获得图像分类和物体识别的优胜。ResNet提出了“残差块”的概念，通过跳跃连接的方式缓解了在深度神经网络中增加深度带来的梯度消失问题，这使得ResNet能够通过增加相当的深度来提高准确率。

残差块示意图：



残差结构示意

自定义残差网络network.py 文件

代码：

import numpy as np

import mindspore.nn as nn

from mindspore.ops import operations as P

from mindspore.common.tensor import Tensor

# 权重初始化函数

def \_weight\_variable(shape, factor=0.01):

init\_value = np.random.randn(\*shape).astype(np.float32) \* factor

return Tensor(init\_value)

# 卷积层

def \_conv3x3(in\_channel, out\_channel, stride=1):

weight\_shape = (out\_channel, in\_channel, 3, 3)

weight = \_weight\_variable(weight\_shape)

return nn.Conv2d(in\_channel, out\_channel,

kernel\_size=3, stride=stride, padding=0, pad\_mode='same', weight\_init=weight)

# 卷积层

def \_conv1x1(in\_channel, out\_channel, stride=1):

weight\_shape = (out\_channel, in\_channel, 1, 1)

weight = \_weight\_variable(weight\_shape)

return nn.Conv2d(in\_channel, out\_channel,

kernel\_size=1, stride=stride, padding=0, pad\_mode='same', weight\_init=weight)

# 卷积层

def \_conv7x7(in\_channel, out\_channel, stride=1):

weight\_shape = (out\_channel, in\_channel, 7, 7)

weight = \_weight\_variable(weight\_shape)

return nn.Conv2d(in\_channel, out\_channel,

kernel\_size=7, stride=stride, padding=0, pad\_mode='same', weight\_init=weight)

# 批量标准化层

def \_bn(channel):

return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,

gamma\_init=1, beta\_init=0, moving\_mean\_init=0, moving\_var\_init=1)

# 批量标准化层

def \_bn\_last(channel):

return nn.BatchNorm2d(channel, eps=1e-4, momentum=0.9,

gamma\_init=0, beta\_init=0, moving\_mean\_init=0, moving\_var\_init=1)

# 全连接层

def \_fc(in\_channel, out\_channel):

weight\_shape = (out\_channel, in\_channel)

weight = \_weight\_variable(weight\_shape)

return nn.Dense(in\_channel, out\_channel, has\_bias=True, weight\_init=weight, bias\_init=0)

# 残差块

class ResidualBlock(nn.Cell):

"""

ResNet V1 残差块定义

Args:

in\_channel (int): 输入通道数

out\_channel (int): 输出通道数

stride (int): 卷积层（3x3卷积核）的步长

Returns:

Tensor, 输出张量

Examples:

>>> ResidualBlock(3, 256, stride=2)

"""

expansion = 4

def \_\_init\_\_(self,

in\_channel,

out\_channel,

stride=1):

super(ResidualBlock, self).\_\_init\_\_()

channel = out\_channel // self.expansion

self.conv1 = \_conv1x1(in\_channel, channel, stride=1)

self.bn1 = \_bn(channel)

self.conv2 = \_conv3x3(channel, channel, stride=stride)

self.bn2 = \_bn(channel)

self.conv3 = \_conv1x1(channel, out\_channel, stride=1)

self.bn3 = \_bn\_last(out\_channel)

self.relu = nn.ReLU()

self.down\_sample = False

if stride != 1 or in\_channel != out\_channel:

self.down\_sample = True

self.down\_sample\_layer = None

if self.down\_sample:

self.down\_sample\_layer = nn.SequentialCell([\_conv1x1(in\_channel, out\_channel, stride),

\_bn(out\_channel)])

self.add = P.Add()

def construct(self, x):

identity = x

out = self.conv1(x)

out = self.bn1(out)

out = self.relu(out)

out = self.conv2(out)

out = self.bn2(out)

out = self.relu(out)

out = self.conv3(out)

out = self.bn3(out)

if self.down\_sample:

identity = self.down\_sample\_layer(identity)

out = self.add(out, identity)

out = self.relu(out)

return out

# ResNet backbone 结构配置

class ResNet\_backbone(nn.Cell):

"""

ResNet architecture.

Args:

block (Cell): 残差块

layer\_nums (list): 每个ResNet子模块的残差块数量

in\_channels (list): 每个ResNet子模块的输入通道数

out\_channels (list): 每个ResNet子模块的输出通道数

strides (list): 每个ResNet子模块的卷积层（3x3卷积核）步长

Returns:

Tensor, 输出张量

Examples:

>>> ResNet(ResidualBlock,

>>> [3, 4, 6, 3],

>>> [64, 256, 512, 1024],

>>> [256, 512, 1024, 2048],

>>> [1, 2, 2, 2])

"""

def \_\_init\_\_(self,

block,

layer\_nums,

in\_channels,

out\_channels,

strides):

super(ResNet\_backbone, self).\_\_init\_\_()

if not len(layer\_nums) == len(in\_channels) == len(out\_channels) == 4:

raise ValueError("the length of layer\_num, in\_channels, out\_channels list must be 4!")

self.conv1 = \_conv7x7(3, 64, stride=2)

self.bn1 = \_bn(64)

self.relu = nn.ReLU()

self.maxpool = nn.MaxPool2d(kernel\_size=3, stride=2, pad\_mode="same")

self.layer1 = self.\_make\_layer(block,

layer\_nums[0],

in\_channel=in\_channels[0],

out\_channel=out\_channels[0],

stride=strides[0])

self.layer2 = self.\_make\_layer(block,

layer\_nums[1],

in\_channel=in\_channels[1],

out\_channel=out\_channels[1],

stride=strides[1])

self.layer3 = self.\_make\_layer(block,

layer\_nums[2],

in\_channel=in\_channels[2],

out\_channel=out\_channels[2],

stride=strides[2])

self.layer4 = self.\_make\_layer(block,

layer\_nums[3],

in\_channel=in\_channels[3],

out\_channel=out\_channels[3],

stride=strides[3])

def \_make\_layer(self, block, layer\_num, in\_channel, out\_channel, stride):

"""

建立ResNet子模块

Args:

block (Cell): 残差块

layer\_num (int): 残差块数量

in\_channel (int): 输入通道数

out\_channel (int): 输出通道数

stride (int): 卷积层（3x3卷积核）的步长

Returns:

SequentialCell, ResNet子模块

Examples:

>>> \_make\_layer(ResidualBlock, 3, 128, 256, 2)

"""

layers = []

resnet\_block = block(in\_channel, out\_channel, stride=stride)

layers.append(resnet\_block)

for \_ in range(1, layer\_num):

resnet\_block = block(out\_channel, out\_channel, stride=1)

layers.append(resnet\_block)

return nn.SequentialCell(layers)

def construct(self, x):

x = self.conv1(x)

x = self.bn1(x)

x = self.relu(x)

c1 = self.maxpool(x)

c2 = self.layer1(c1)

c3 = self.layer2(c2)

c4 = self.layer3(c3)

c5 = self.layer4(c4)

#print(c5.shape)

#out = self.mean(c5, (2, 3))

return c5

# ResNet head 结构配置

class ResNet\_head(nn.Cell):

"""

ResNet architecture.

Args:

backbone\_out (int): backbone输出维度

num\_classes (int): 数据集分类数量

Returns:

Tensor, 输出张量

Examples:

>>> ResNet(ResidualBlock,

>>> [3, 4, 6, 3],

>>> [64, 256, 512, 1024],

>>> [256, 512, 1024, 2048],

>>> [1, 2, 2, 2],

>>> 10)

"""

def \_\_init\_\_(self,

backbone\_out,

num\_classes):

super(ResNet\_head, self).\_\_init\_\_()

self.avgpool = nn.AvgPool2d(kernel\_size=7)

self.flatten = nn.Flatten()

self.end\_point = \_fc(backbone\_out, num\_classes)

def construct(self, x):

x = self.avgpool(x)

x = self.flatten(x)

out = self.end\_point(x)

return out

# ResNet50\_backbone定义

def resnet50\_backbone():

"""

ResNet50

Args:

Returns:

Cell, ResNet50 backbone网络

Examples:

>>> net = resnet50(10)

"""

return ResNet\_backbone(ResidualBlock,

[3, 4, 6, 3],

[64, 256, 512, 1024],

[256, 512, 1024, 2048],

[1, 2, 2, 2])

# ResNet50\_head定义

def resnet50\_head(class\_num=10):

"""

ResNet50

Args:

class\_num (int): 数据集分类数

Returns:

Cell, ResNet50\_head网络

Examples:

>>> net = resnet50(10)

"""

return ResNet\_head(2048, class\_num)

参考上一步骤，配置文件已包含network.py，在开发环境内从本地上传即可；

图形用户界面, 文本, 应用程序

描述已自动生成

调用 network.py 文件

代码：

# 这里将网络分为backbone和head，backbone是ResNet包含残差块的部分，head是最后的全连接层。

from network import resnet50\_backbone, resnet50\_head

# 最终网络由backbone和head组成。

class ResNet50(nn.Cell):

"""

ResNet architecture.

Args:

backbone (Cell): ResNet50 backbone网络

head (Cell): ResNet50 head网络

Returns:

Tensor, 输出张量

Examples:

>>> ResNet50(resnet\_backbone,

>>> resnet\_head)

"""

def \_\_init\_\_(self, backbone, head):

super(ResNet50, self).\_\_init\_\_()

self.backbone = backbone

self.head = head

def construct(self, x):

x = self.backbone(x)

x = self.head(x)

return x

### 模型训练

载入数据集

代码：

# 训练集

train\_data = create\_dataset(data\_path=train\_path,

mean=RGB\_mean,

std=RGB\_std,

batch\_size=config.batch\_size,

usage='train',

repeat\_num=1)

# 测试集

test\_data = create\_dataset(data\_path=test\_path,

mean=RGB\_mean,

std=RGB\_std,

batch\_size=config.batch\_size,

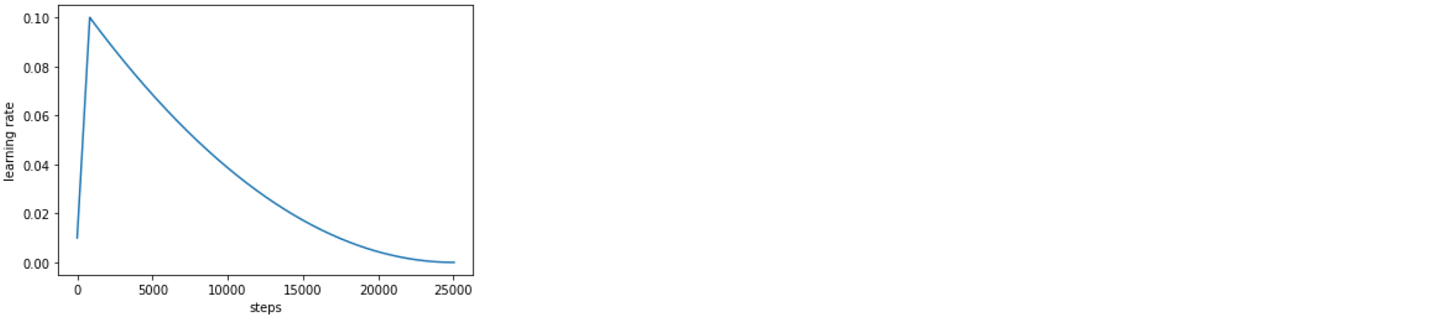
usage='test',

repeat\_num=1)

自定义动态学习率lr\_scheduler.py文件

设定动态学习率，加速模型收敛。

本实验学习率函数图如下：



学习率函数图

同上，上传lr\_scheduler.py至开发环境即可。

图形用户界面, 应用程序

描述已自动生成

代码展示如下：

import numpy as np

def get\_lr(total\_epochs,

steps\_per\_epoch,

lr\_init=0.01,

lr\_max=0.1,

warmup\_epochs=5):

"""

生成学习率数组

Args:

total\_epochs (int): 总epoch数

steps\_per\_epoch (float): 每个epoch多少step

lr\_init (float): 初始学习率

lr\_max (float): 最大学习率

warmup\_epochs (int): 预热epoch数

Returns:

numpy.ndarray，学习率数组

"""

lr\_each\_step = [] # 学习率数组（回传）

total\_steps = steps\_per\_epoch \* total\_epochs # 总step数

warmup\_steps = steps\_per\_epoch \* warmup\_epochs # 预热step数

# 计算预热阶段学习率递增值

if warmup\_steps != 0:

inc\_each\_step = (float(lr\_max) - float(lr\_init)) / float(warmup\_steps)

else:

inc\_each\_step = 0

# 学习率调整

for i in range(int(total\_steps)):

if i < warmup\_steps:

# 预热（学习率线性递增）

lr = float(lr\_init) + inc\_each\_step \* float(i)

else:

# 衰减（学习率指数递减）

base = ( 1.0 - (float(i) - float(warmup\_steps)) / (float(total\_steps) - float(warmup\_steps)) )

lr = float(lr\_max) \* base \* base

if lr < 0.0:

lr = 0.0

# 记录学习率

lr\_each\_step.append(lr)

lr\_each\_step = np.array(lr\_each\_step).astype(np.float32)

return lr\_each\_step

调用lr\_scheduler.py文件

代码：

from lr\_scheduler import get\_lr

# 训练step总数

train\_step\_size = train\_data.get\_dataset\_size()

# 学习率数组

lr = get\_lr(total\_epochs=config.epochs,

steps\_per\_epoch=train\_step\_size,

lr\_init=config.lr\_init,

lr\_max=config.lr\_max,

warmup\_epochs=config.warmup\_epochs)

构建网络

选择网络、选择损失函数、优化器、模型。

代码：

# 网络

backbone\_net = resnet50\_backbone() # backbone网络，保存后能提供后续迁移学习使用

head\_net = resnet50\_head(config.class\_num) # head网络，resnet50最后的全连接层

net = ResNet50(backbone\_net, head\_net)

# 损失函数

net\_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# 优化器

opt = nn.Momentum(net.trainable\_params(), lr, momentum=config.momentum, weight\_decay=config.weight\_decay)

# 模型

model = Model(net, loss\_fn = net\_loss,

optimizer = opt, metrics = {'accuracy','loss'})

自定义回调函数callbacks.py文件

自定义回调函数（EvalHistory中新增保存最佳模型的backbone网络参数的功能）

同上，上传callbacks.py至开发环境即可。

图形用户界面, 应用程序

描述已自动生成

代码展示：

from mindspore.train.callback import Callback

from mindspore import save\_checkpoint

import os, stat, copy

# 记录模型accuracy

class TrainHistroy(Callback):

def \_\_init\_\_(self, history):

super(TrainHistroy, self).\_\_init\_\_()

self.history = history

# 每个epoch结束时执行

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

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

loss = cb\_params.net\_outputs.asnumpy()

self.history.append(loss)

# 测试并记录模型在测试集的loss和accuracy，每个epoch结束时进行模型测试并记录结果，跟踪并保存准确率最高的模型的网络参数

class EvalHistory(Callback):

#保存accuracy最高的网络参数

best\_param = None

best\_param\_backbone = None

def \_\_init\_\_(self, model, backbone, loss\_history, acc\_history, eval\_data):

super(EvalHistory, self).\_\_init\_\_()

self.model = model

self.backbone = backbone

self.loss\_history = loss\_history

self.acc\_history = acc\_history

self.eval\_data = eval\_data

# 每个epoch结束时执行

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

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

res = self.model.eval(self.eval\_data, dataset\_sink\_mode=False)

if len(self.acc\_history)==0:

self.best\_param = copy.deepcopy(cb\_params.network)

self.best\_param\_backbone = copy.deepcopy(self.backbone)

elif res['accuracy']>=max(self.acc\_history):

self.best\_param = copy.deepcopy(cb\_params.network)

self.best\_param\_backbone = copy.deepcopy(self.backbone)

self.loss\_history.append(res['loss'])

self.acc\_history.append(res['accuracy'])

print('acc\_eval: ',res['accuracy'])

# 训练结束后执行

def end(self, run\_context):

# 保存最优网络参数

if os.path.exists('best\_param.ckpt'):

os.chmod('best\_param.ckpt', stat.S\_IWRITE)

if os.path.exists('best\_param\_backbone.ckpt'):

os.chmod('best\_param\_backbone.ckpt', stat.S\_IWRITE)

save\_checkpoint(self.best\_param, 'best\_param.ckpt')

save\_checkpoint(self.best\_param\_backbone, 'best\_param\_backbone.ckpt')

设置回调函数

from callbacks import TrainHistroy,EvalHistory

time\_cb = TimeMonitor(data\_size=train\_step\_size) # 监控每次迭代的时间

loss\_cb = LossMonitor() # 监控loss值

hist = {'loss':[], 'loss\_eval':[], 'acc\_eval':[]} # 训练过程记录

# 记录每次迭代的模型准确率

train\_hist\_cb = TrainHistroy(hist['loss'])

# 测试并记录模型在验证集的loss和accuracy，并保存最优网络参数

eval\_hist\_cb = EvalHistory(model = model,

backbone = backbone\_net,

loss\_history = hist['loss\_eval'],

acc\_history = hist['acc\_eval'],

eval\_data = test\_data)

cb = [time\_cb, loss\_cb, train\_hist\_cb, eval\_hist\_cb]

训练模型

代码：

model.train(config.epochs, train\_data, callbacks=cb)

输出：

epoch: 1 step: 167, loss is 1.9619308

epoch time: 116120.809 ms, per step time: 695.334 ms

acc\_eval: 0.23809523809523808

epoch: 2 step: 167, loss is 1.9744694

epoch time: 5230.776 ms, per step time: 31.322 ms

…

epoch: 149 step: 167, loss is 0.09291564

epoch time: 5655.406 ms, per step time: 33.865 ms

acc\_eval: 0.7601410934744268

epoch: 150 step: 167, loss is 0.14568897

epoch time: 5622.981 ms, per step time: 33.671 ms

acc\_eval: 0.7971781305114638

### 模型评估

可视化模型损失函数

代码：

# 定义loss记录绘制函数

def plot\_loss(hist):

plt.plot(hist['loss'], marker='.')

plt.plot(hist['loss\_eval'], marker='.')

plt.title('loss record')

plt.xlabel('epoch')

plt.ylabel('loss')

plt.grid()

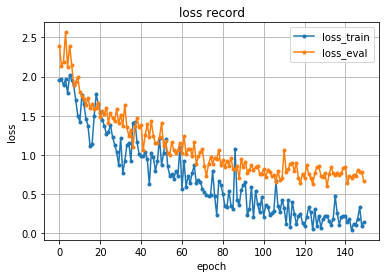
plt.legend(['loss\_train', 'loss\_eval'], loc='upper right')

plt.show()

plt.close()

plot\_loss(hist)

输出：



可视化模型准确率

代码：

def plot\_accuracy(hist):

plt.plot(hist['acc\_eval'], marker='.')

plt.title('accuracy history')

plt.xlabel('epoch')

plt.ylabel('acc\_eval')

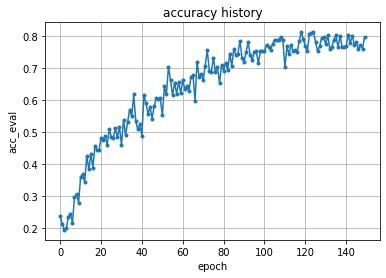
plt.grid()

plt.show()

plt.close()

plot\_accuracy(hist)

输出：



选择最优参数并验证

代码：

# 使用准确率最高的参数组合建立模型，并测试其在验证集上的效果

load\_checkpoint('best\_param.ckpt', net=net)

res = model.eval(test\_data, dataset\_sink\_mode=False)

print(res)

输出：

{'loss': 0.773123272591167, 'accuracy': 0.7918871252204586}

### 效果展示

代码：

# 创建图像标签列表

category\_dict = {0:'Agaricus',1:'Amanita',2:'Boletus',3:'Cortinarius',4:'Entoloma',

5:'Hygrocybe',6:'Lactarius',7:'Russula',8:'Suillus'}

ds\_test\_demo = create\_dataset(test\_path, mean=RGB\_mean, std=RGB\_std, batch\_size=1, usage='test')

# 将数据标准化至0~1区间

def normalize(data):

\_range = np.max(data) - np.min(data)

return (data - np.min(data)) / \_range

# 设置图像大小

plt.figure(figsize=(10,10))

i = 1

# 打印9张子图

for dic in ds\_test\_demo.create\_dict\_iterator():

# 预测单张图片

input\_img = dic['image']

output = model.predict(input\_img)

predict = np.argmax(output.asnumpy(),axis=1)[0] # 反馈可能性最大的类别

# 可视化

plt.subplot(3,3,i)

input\_image = np.squeeze(input\_img.asnumpy(),axis=0).transpose(1,2,0) # 删除batch维度

input\_image = normalize(input\_image) # 重新标准化，方便可视化

plt.imshow(input\_image)

plt.axis('off')

plt.title('True: %s\n Predict: %s'%(category\_dict[dic['label'].asnumpy().item()],category\_dict[predict]))

i +=1

if i > 9 :

break

plt.show()

输出：



### 模型保存

若要保存模型参数文件（ckpt），停止ModelArts环境前需先将保存在ModelArts暂存的模型参数文件best\_param\_backbon.ckpt下载到本地。

图形用户界面, 文本, 应用程序

描述已自动生成

## 实验总结

本实验介绍了MindSpore在华为云ModelArts平台使用Ascend 910芯片完成图像分类的任务，使用大规模的蘑菇图片数据集，搭建复杂的ResNet50的神经网络，调用华为云ModelArts平台的云端算力进行快速训练，最终实现毒蘑菇图像分类的任务。

## 课后作业：蘑菇分类模型Fine-Tune

上一个阶段使用ModelArts训练了一个蘑菇分类器，该分类器可以对9种蘑菇进行分类。

现在，请使用上一个阶段训练好的网络作为本作业的backbone网络，训练一个可以分类另外2种新蘑菇的分类器。

### 数据集

本作业使用kaggle的[另一个蘑菇数据集](https://www.kaggle.com/anilkrsah/deepmushroom)，只需对其中的两种新蘑菇进行分类：

Exidia：黑耳，又称为黑胶菌，分布于我国吉林、河北、山西、宁夏、青海、四川、广西、海南,西藏等地，无毒。

Inocybe：丝盖伞，又称毛丝盖菌、毛锈伞，分布于我国河北、吉林、江苏、山西、四川、云南、甘肃、新疆等地，有毒。

全量数据集可通过以下代码获取：

!wget https://zhuanyejianshe.obs.cn-north-4.myhuaweicloud.com/chuangxinshijianke/MindSpore/archive\_o.zip

### 实验设计要求

1. 环境准备

2. 参数设置

3. 数据处理

4. 网络定义

5. 特征提取

6. 模型训练（需要自定义动态学习率）

7. 模型评估（需要可视化loss和accuracy）

8. 效果展示

9. 模型保存

注：实验中需要的backbone.ckpt等文件点此下载：

!wget https://zhuanyejianshe.obs.cn-north-4.myhuaweicloud.com/chuangxinshijianke/MindSpore/peizhiwenjian-zuoye.zip