HCIA-华为昇腾-昇思MindSpore《AI计算系统》精品课程

HCIA-AI计算系统

精品课程

实验指导手册

版本:1.0



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

[4 基于MindSpore框架实现VGG17训练与验证 4](#_Toc124437865)

[4.1实验介绍 3](#_Toc490381853)

[4.1.2 数据集的介绍 4](#_Toc620982029)

[4.1.3 模型知识点的介绍 5](#_Toc74436983)

[4.2 实验环境要求 12](#_Toc1226890727)

[4.3 实验详细设计与实现 13](#_Toc214470195)

[4.3.1 数据集上传OBS 13](#_Toc1123007699)

[4.3.2 实验代码目录介绍 13](#_Toc156623610)

[4.3.3 实验步骤 14](#_Toc1705545695)

[4.4 实验总结 23](#_Toc494775709)

[4.5 实验任务与参考解答任务 23](#_Toc638659979)

[4.5.1 实验任务 23](#_Toc834999347)

[4.5.2 参考答案 2](#_Toc28391884)4

# 基于MindSpore框架实现VGG17训练与验证

## 实验介绍

本实验基于Modelarts平台，使用MindSpore深度学习框架，利用其成熟的算子库搭建VGG17神经网络模型，使用花卉数据集（雏菊、蒲公英、玫瑰、向日葵、郁金香）在Ascend910加速卡上进行训练和验证。

* 熟悉和使用MindSpore框架和ModelArts，熟悉MindSpore常见API的使用方法，熟悉ModelArts一站式模型训练和部署平台。
* 基于MindSpore框架构建VGG17网络。利用花卉数据集上完成模型训练（训练平台：ModelArts，可采用昇腾910芯片进行训练）。模型训练完成后，对模型进行保存。
* 基于昇腾310推理芯片作为计算平台，利用MindSpore框架导入训练好的模型，并在花卉测试数据集对构建的模型进行推理验证，输出推理性能以及测试集正确率。
* 本实验希望借助MindSpore帮助学生熟悉使用深度学习框架，感受框架封装基本操作的便捷。

### 数据集的介绍

我们示例中用到的图像花卉数据集，该数据集是开源数据集，总共包括5种花的类型：分别是daisy（雏菊，633张），dandelion（蒲公英，898张），roses（玫瑰，641张），sunflowers（向日葵，699张），tulips（郁金香，799张），保存在5个文件夹当中，总共3670张，大小大概在230M左右。为了在模型部署上线之后进行测试，数据集在这里分成了flower\_photos\_train和flower\_photos\_test两部分。

目录结构如下：

flower\_photos\_train

├── daisy

├── dandelion

├── roses

├── sunflowers

├── tulips

├── LICENSE.txt

flower\_photos\_test

├── daisy

├── dandelion

├── roses

├── sunflowers

├── tulips

├── LICENSE.txt

### 模型知识点的介绍

**4.1.2.1 MindSpore**

MindSpore是华为推出的深度学习框架，拥有自动微分、并行加持，一次训练，可多场景部署的特性。为全场景AI的模型开发、模型运行、模型部署提供端到端能力。 MindSpore支持深度学习算法在Ascend、GPU、CPU等硬件平台上开发和部署。



如需查看详情，请参见如下资源：

- [MindSpore教程](https://gitee.com/link?target=https%3A%2F%2Fwww.mindspore.cn%2Ftutorials%2Fzh-CN%2Fmaster%2Findex.html)

- [MindSpore Python API](https://gitee.com/link?target=https%3A%2F%2Fwww.mindspore.cn%2Fdocs%2Fzh-CN%2Fmaster%2Findex.html)

**4.1.2.2 VGG17网络**

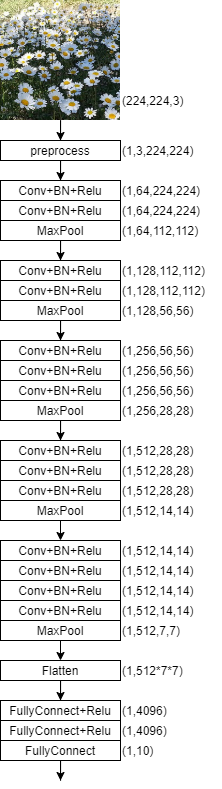
本次实验延续之前实验中使用的VGG16网络并进行一些变换提出VGG17网络，网络结构如下。VGG17网络的特点是：

* 在VGG中，使用了3个3x3卷积核来代替7x7卷积核，使用了2个3x3卷积核来代替5\*5卷积核，相比AlexNet中的较大卷积核（11x11，7x7，5x5），VGG网络层数更深，提升了网络性能。
* 池化层均采用相同的池化核参数，stride=2。
* 模型由若干卷积层和池化层堆叠的方式构成。

注：在构造网络时，还需要考虑BN(Batch Normalization)层和Relu层（BN层可以提高网络训练稳定性，Relu层是非线性激活层）。此外为了提高网络鲁棒性，加入了dropout层。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **算子** | **类型** | **输入通道数** | **输出通道数** | **窗口大小** | **边界扩充大小** | **步长** | **输出张量的高度和宽度** |
| layer1\_conv1 | 卷积 | 3 | 64 | 3 | 1 | 1 | 224×224 |
| layer1\_conv2 | 卷积 | 64 | 64 | 3 | 1 | 1 | 224×224 |
| layer1\_maxpool | 最大池化 | 64 | 64 | 2 | - | 2 | 112×112 |
| layer2\_conv1 | 卷积 | 64 | 128 | 3 | 1 | 1 | 112×112 |
| layer2\_conv2 | 卷积 | 128 | 128 | 3 | 1 | 1 | 112×112 |
| layer2\_maxpool | 最大池化 | 128 | 128 | 2 | - | 2 | 56×56 |
| layer3\_conv1 | 卷积 | 128 | 256 | 3 | 1 | 1 | 56×56 |
| layer3\_conv2 | 卷积 | 256 | 256 | 3 | 1 | 1 | 56×56 |
| layer3\_conv3 | 卷积 | 256 | 256 | 3 | 1 | 1 | 56×56 |
| layer3\_maxpool | 最大池化 | 256 | 256 | 2 | - | 2 | 28×28 |
| layer4\_conv1 | 卷积 | 256 | 512 | 3 | 1 | 1 | 28×28 |
| layer4\_conv2 | 卷积 | 512 | 512 | 3 | 1 | 1 | 28×28 |
| layer4\_conv3 | 卷积 | 512 | 512 | 3 | 1 | 1 | 28×28 |
| layer4\_maxpool | 最大池化 | 512 | 512 | 2 | - | 2 | 14×14 |
| layer5\_conv1 | 卷积 | 512 | 512 | 3 | 1 | 1 | 14×14 |
| layer\_conv2 | 卷积 | 512 | 512 | 3 | 1 | 1 | 14×14 |
| layer5\_conv3 | 卷积 | 512 | 512 | 3 | 1 | 1 | 14×14 |
| layer5\_conv4 | 卷积 | 512 | 512 | 3 | 1 | 1 | 14×14 |
| layer5\_maxpool | 最大池化 | 512 | 512 | 2 | - | 2 | 7×7 |
| flatten | 扁平化 | - | - | - | - | - | - |
| fullyconnect1 | 全连接 | 25088 | 4096 | - | - | - | - |
| fullyconnect2 | 全连接 | 4096 | 4096 | - | - | - | - |
| fullyconnect3 | 全连接 | 4096 | 4 | - | - | - | - |

VGG17的结构图如下：



**4.1.2.6 MindSpore常用函数介绍**

mindspore.nn模块中也提供了一系列的算子接口，主要是用于构建神经网络的预定义构建块或计算单元。在构建VGG网络模型时，我们主要用到了以下API，如下表所示。

|  |  |  |
| --- | --- | --- |
| **算子接口名称** | **功能描述** | **参数介绍** |
| mindspore.nn. Conv2d | 对输入Tensor计算二维卷积 | **in\_channels** (int) : Conv2d层输入Tensor的空间维度。  **out\_channels** (dict) : Conv2d层输出Tensor的空间维度。  **kernel\_size** (Union[int, tuple[int]]) ：卷积核的高和宽。  **padding** (Union[int, tuple[int]]) ：边界扩充。若padding类型为int，则四个方向上的填充都等于padding。  若padding为包含4个int的tuple，则上、下、左、右的填充分别等于padding[0]、padding[1]、padding[2]和padding[3]。值应该要大于等于0，默认值：0。  **stride** (Union[int, tuple[int]])：卷积步长，默认值：1。  **pad\_mode** (str)：填充模式，可选“same”，“valid”，“pad”，默认值：“same”。  “same”表示保留边界处的卷积结果，“valid”表示只进行有效卷积，对边界不处理。  **dilation** (Union[int, tuple[int]])：指二维卷积扩张的尺寸。  **has\_bias** (bool) ：Conv2d层是否添加偏置参数，若conv后接batchnorm层不添加偏置，反之添加，默认值：False。  **weight\_init** (Union[Tensor, str, Initializer, numbers.Number]) ：卷积核参数初始化的方法，可以是张量，字符串，初始化器或数字。  当使用str时，可选"TruncatedNormal"，"Uniform"，“Zero”等。默认值：“normal”，即用正态分布的随机数初始化卷积层的权重参数。  **bias\_init** (Union[Tensor, str, Initializer, numbers.Number]) ：偏置参数初始化的方法，默认值："zeros"。  **data\_format** (str) ：数据格式的可选值有“NHWC”，“NCHW”。默认值：“NCHW”。 |
| mindspore.nn.ReLU | 对输入Tensor做ReLU，返回的数据类型同输入Tensor | x (Tensor) ：用于计算ReLU的任意维度的Tensor。数据类型为number类 型，包括float32,float64,int32,uint8,int16,int64,bfloat16,uint16,half,uint32,uint64 和qint8。 |
| mindspore.nn.BatchNorm2d | 对输入Tensor进行批归一化 | num\_features (int) ：指定输入Tensor的通道数量，输入Tensor的size为 (N,C,H,W)。 |
| mindspore.nn.Dropout | Dropout是一种正则化手段，该算子根据丢弃概率 1−*keep\_prob1*，在训练过程中随机将一些神经元输出设置为0，通过阻止神经元节点间的相关性来减少过拟合 | **keep\_prob (float)** : 输入神经元保留率，数值范围在0到1之间。例如， rate=0.9，删除10%的神经元。默认值：0.5。  dtype (mindspore.dtype) : x 的数据类型。默认值：float32。 |
| mindspore.nn.SoftmaxCrossEntropyWithLogits | 计算预测值和真实值的交叉熵 | logits (Tensor) ：真实值Tensor，shape (N, C),数据类型为float16或 float32。  labels (Tensor) ：预测值Tensor，shape (N, )。如果 sparse 为True，则 labels 的类型为int32或int64。否则，labels 的类型与 logits 的类型相同。 |
| mindspore.nn.MaxPool2d | 对输入的多维数据进行二维的最大池化运算 | kernel\_size (Union[int, tuple[int]]) ：池化核的高和宽。默认值：1 。 stride (Union[int, tuple[int]]) ：池化操作的移动步长。默认值：1 。 pad\_mode (str) ：指定池化填充模式，取值为“same”或“valid”。默认值：“valid”。  data\_format (str) ：输入数据格式可为“NHWC” 或“NCHW”。默认值：“NCHW”。 |

关于更多API请参考MindSpore的API文档 <https://www.mindspore.cn/docs/zh-CN/r1.8/index.html>

## 实验环境要求

在动手进行实践之前，确保你已经正确安装了MindSpore。如果没有，可以通过MindSpore官网安装页面：<https://www.mindspore.cn/install/>，将MindSpore安装在你的电脑当中。

推荐环境：

版本：MindSpore 2.0

编程语言：Python 3.7

环境：ModelArts在线训练和评估平台。（可参考[ModelArts平台使用文档](https://support.huaweicloud.com/qs-modelarts/modelarts_06_0006.html)）

## 实验详细设计与实现

### 数据集上传OBS

将花卉数据集上传OBS桶，包括flower\_photos\_train和flower\_photos\_test两个数据集。上传步骤详见1.1.3.3。

### 实验代码目录介绍

本实验的代码目录介绍：

./ vgg17

├── data // 数据集文件夹

├── code // 代码文件夹

├── ascend310\_infer // Ascend310 推理目录

├── infer // MindX 推理目录

├── model\_utils

├── \_\_init\_\_.py // 初始化文件

├── config.py // 参数配置

├── device\_adapter.py // ModelArts的设备适配器

├── local\_adapter.py // 本地适配器

└── moxing\_adapter.py // ModelArts的模型适配器

├── scripts

├── run\_distribute\_train.sh // Ascend 分布式训练shell

脚本

├── run\_distribute\_train\_gpu.sh // GPU 分布式训练shell

脚本

├── run\_eval.sh // Ascend 验证shell脚本

├── run\_infer\_310.sh // Ascend310 推理shell脚本

├── src

├── utils

├── logging.py // 日志格式设置

├── sampler.py // 为数据集创建采样器

├── util.py // 工具函数

├── var\_init.py // 网络参数init方法

├── crossentropy.py // 损失计算

├── dataset.py // 创建数据集

├── linear\_warmup.py // 线性学习率

├── warmup\_cosine\_annealing\_lr.py // 余弦退火学习率

├── warmup\_step\_lr.py // 单次或多次迭代学习率

├──vgg.py // VGG架构

├── train.py // 训练脚本

├── eval.py // 评估脚本

├── postprocess.py // 后处理脚本

├── preprocess.py // 预处理脚本

├── mindspore\_hub\_conf.py // mindspore hub 脚本

├── flowerphotos\_config.yaml // flowerphotos 配置文件

├── export.py // 模型格式转换脚本

### 实验步骤

数据集加载

本实验顺承上次实验，采用[花卉数据集](https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz)，利用mindspore.dataset类下的ImageFolderDataset加载图片数据，同一个文件夹中的所有图片将被分配相同的label。

同时使用以下几种的数据增强操作，

* Random Crop： 对输入图像进行在随机位置的裁剪。
* RandomHorizontalFlip：对输入图像进行随机水平翻转。
* HWC2CHW：将输入图像的shape从 <H, W, C> 转换为 <C, H, W>。 如果输入图像的shape为 <H, W> ，图像将保持不变。
* Resize：对输入图像进行缩放。

代码：

#数据集加载

def vgg\_create\_dataset(data\_home, image\_size, batch\_size, rank\_id=0, rank\_size=1, training=True):

# 加载路径

data\_dir = os.path.join(data\_home, "train")

if not training:

data\_dir = os.path.join(data\_home, "test")

data\_set = de.ImageFolderDataset(data\_dir,

class\_indexing={'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4},

shuffle=False, num\_shards=rank\_size, shard\_id=rank\_id)

#数据增强的方法，上述提高的四种方法

transform\_img = vision.RandomCropDecodeResize([224,224], scale=(0.08, 1.0),

ratio=(0.75, 1.333)) # 改变尺寸

changeswap\_op = vision.HWC2CHW()

type\_cast\_op = C.TypeCast(mstype.float32)

random\_horizontal\_op = vision.RandomHorizontalFlip()

# map操作将指定函数操作于数据集的指定列数据

data\_set = data\_set.map(input\_columns="image", operations=transform\_img)

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

data\_set = data\_set.map(input\_columns="image", operations=random\_horizontal\_op)

data\_set = data\_set.map(input\_columns="image", operations=changeswap\_op)

# shuffle来进行数据集的混洗

data\_set = data\_set.shuffle(buffer\_size=data\_set.get\_dataset\_size())

# 连续 batch\_size 条数据合并为一个批处理数据

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

return data\_set

完善网络结构模块

本模块需要利用MindSpore.nn相关API完整搭建VGG17网络结构（网络结构对应文件相对路径为：(./code/src/vgg.py）。可参考gitee上的相关仓库([https://gitee.com/mindspore/models/tree/master/official/cv/vgg16](https://gitee.com/mindspore/models/tree/master/official/cv/vgg16 )) )。需熟悉使用MindSpore构建网络的相关API，可参考4.1.2.5 API接口介绍。

在利用MindSpore构建网络时，需要继承mindspore.nn.Cell类，并重写 \_\_init\_方法和construct方法。

* 在\_\_init\_\_函数中声明网络各个层和算子的定义。
* 在`construct`函数中完成网络的构造。

网络结构模块对应文件为（./code/src/vgg.py）。请同学按照VGG17的网络结构（如1.1.3.5小结所示的VGG17网络），以层级顺序搭建的方式搭建VGG17网络（如下图的参考示例中的example）。

代码：

class Vgg(nn.Cell):

"""

TO-DO.

参数:

num\_classes (int): Class numbers. Default: 5.

phase (int): 指定是训练/评估阶段

返回值:

Tensor, infer output tensor.

example：

"""

self.layer1\_conv1 = nn.Conv2d(in\_channels=3, out\_channels=64, kernel\_size=3,weight\_init='XavierUniform')

self.layer1\_bn1 = nn.BatchNorm2d(num\_features=64)

self.layer1\_relu1 = nn.LeakyReLU()

"""

def \_\_init\_\_(self, num\_classes=5, args=None, phase="train"):

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

dropout\_ratio = 0.5

if not args.has\_dropout or phase == "test":

dropout\_ratio = 1.0

def construct(self, x):

x = self.layer1\_conv1(x)

x = self.layer1\_bn1(x)

x = self.layer1\_relu1(x)

return x

def vgg17(num\_classes=1000, args=None, phase="train", \*\*kwargs):

"""

生成VGG17网络实例

参数:

num\_classes (int): 分类数

args (namespace): 参数

phase (str): 指定是训练/评估阶段

返回:

Cell, cell instance of Vgg17 neural network with Batch Normalization.

参考如下:

>>> vgg17(num\_classes=5, args=args, \*\*kwargs)

"""

net = Vgg(num\_classes=num\_classes, args=args, phase=phase, \*\*kwargs)

return net

参数配置

在 (./code/flowerphotos\_config.yaml) 中配置了VGG16的训练参数和评估参数。可自行尝试调整lr，batch\_size，max\_epoch等参数，提高模型精度。

代码：

num\_classes: 5 # 数据集类数

lr: 0.01 # 学习率

lr\_init: 0.01 # 初始学习率

lr\_max: 0.1 # 最大学习率

lr\_epochs: '30,60,90,120' # 基于变化lr的轮次

lr\_scheduler: "step" # 学习率模式

warmup\_epochs: 5 # 热身轮次数

batch\_size: 64 # 输入张量批次大小

max\_epoch: 100 # 训练轮次，只对训练有效

momentum: 0.9 # 动量

weight\_decay: 0.0005 # 权重衰减

loss\_scale: 1.0 # 损失放大

label\_smooth: 0 # 标签平滑

label\_smooth\_factor: 0 # 标签平滑因子

buffer\_size: 10 # 混洗缓冲区大小

image\_size: '224,224' # 图像大小

pad\_mode: 'same' # conv2d的填充方式

padding: 0 # conv2d的填充值

has\_bias: False # conv2d是否有偏差

batch\_norm: True # 在conv2d中是否有batch\_norm

keep\_checkpoint\_max: 10 # 只保留最后一个keep\_checkpoint\_max检查点

initialize\_mode: "XavierUniform" # conv2d init模式

has\_dropout: True # 是否使用Dropout层

模型训练与评估

MindSpore提供mindspore.Model 用于模型训练和推理的API，Model会根据用户传入的参数封装可训练或推理的实例。

* 模型训练

使用mindspore.Model.train接口执行模型训练，train接口的常用参数如下：

* epoch：训练执行轮次，通常每个epoch都会使用全量数据集进行训练。
* train\_dataset：一个训练数据集迭代器。
* callbacks：训练过程中需要执行的回调对象或者回调对象列表。

训练文件对应为（./code/train.py），相关代码如下：

代码：

# 定义神经网络

network = vgg17(config.num\_classes, config)

# 若存在权重文件，将参数加载到网络中

if config.pre\_trained:

load\_param\_into\_net(network, load\_checkpoint(config.pre\_trained))

# 定义学习率调整策略

if config.lr\_scheduler == 'exponential':

lr = warmup\_step\_lr(config.lr,

config.lr\_epochs,

config.steps\_per\_epoch,

config.warmup\_epochs,

config.max\_epoch,

gamma=config.lr\_gamma)

elif config.lr\_scheduler == 'cosine\_annealing':

lr = warmup\_cosine\_annealing\_lr(config.lr,

config.steps\_per\_epoch,

config.warmup\_epochs,

config.max\_epoch,

config.T\_max,

config.eta\_min)

elif config.lr\_scheduler == 'step':

lr = lr\_steps(0, lr\_init=config.lr\_init, lr\_max=config.lr\_max, warmup\_epochs=config.warmup\_epochs,

total\_epochs=config.max\_epoch, steps\_per\_epoch=batch\_num)

else:

raise NotImplementedError(config.lr\_scheduler)

# 定义优化器函数

opt = Momentum(params=get\_param\_groups(network),

learning\_rate=Tensor(lr),

momentum=config.momentum,

weight\_decay=config.weight\_decay,

loss\_scale=config.loss\_scale)

#定义损失函数，初始化模型参数

if config.dataset == "flower\_photos":

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

model = Model(network, loss\_fn=loss, optimizer=opt, metrics={'acc'},

amp\_level="O2", keep\_batchnorm\_fp32=False, loss\_scale\_manager=None)

# 定义回调函数

time\_cb = TimeMonitor(data\_size=batch\_num)

loss\_cb = LossMonitor()

callbacks = [time\_cb, loss\_cb]

if config.rank\_save\_ckpt\_flag:

ckpt\_config = CheckpointConfig(save\_checkpoint\_steps=config.ckpt\_interval \* config.steps\_per\_epoch,

keep\_checkpoint\_max=config.keep\_checkpoint\_max)

save\_ckpt\_path = os.path.join(config.outputs\_dir, 'ckpt\_' + str(config.rank) + '/')

print(save\_ckpt\_path)

ckpt\_cb = ModelCheckpoint(config=ckpt\_config,

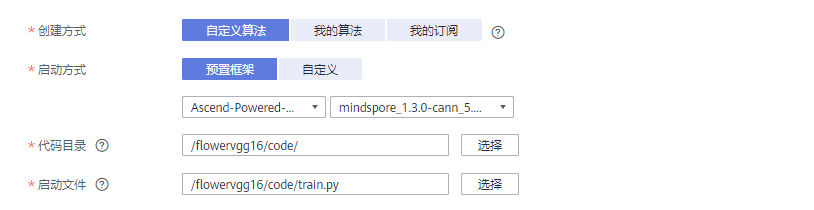
directory=save\_ckpt\_path,

prefix='{}'.format(config.rank))

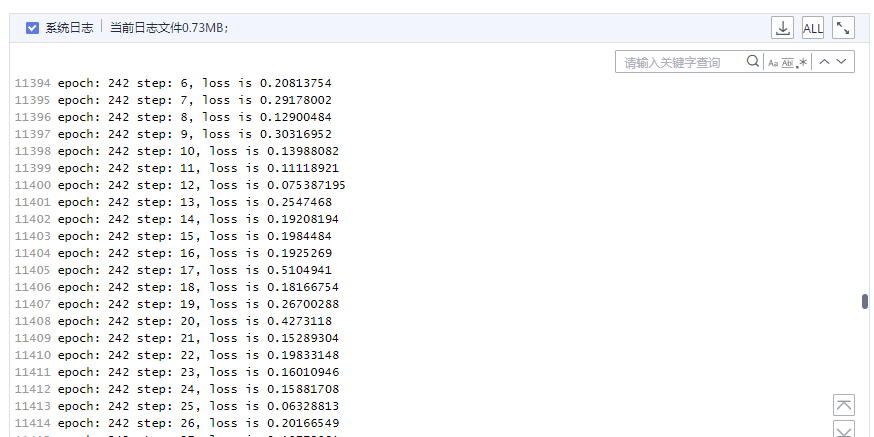
callbacks.append(ckpt\_cb)

#进行模型训练

model.train(config.max\_epoch, dataset, callbacks=callbacks)

在Modelarts上训练，训练页面设置如下：

训练得到如下文件：



如果在命令行运行，指令如下：

python train.py

* 模型评估

使用测试数据集，使用mindspore.Model.eval接口进行评估，eval接口参数如下：

* valid\_dataset：评估模型的数据集。
* callbacks：评估过程中需要执行的回调对象或回调对象列表。
* dataset\_sink\_mode：数据是否直接下沉至处理器进行处理。

model.eval 会返回一个字典，里面是传入metrics的指标和结果。

评估文件对应为（./code/eval.py），相关代码如下：

代码：

if config.dataset == "flower\_photos":

# 构建网络实例

net = vgg17(num\_classes=config.num\_classes, args=config,phase="test")

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

model = Model(net, loss\_fn=loss, metrics={'acc'})

param\_dict = load\_checkpoint(config.pre\_trained)

load\_param\_into\_net(net, param\_dict)

# 定义为评估阶段

net.set\_train(False)

dataset = vgg\_create\_dataset(config.data\_dir, config.image\_size, config.per\_batch\_size, training=False)

# 开始评估

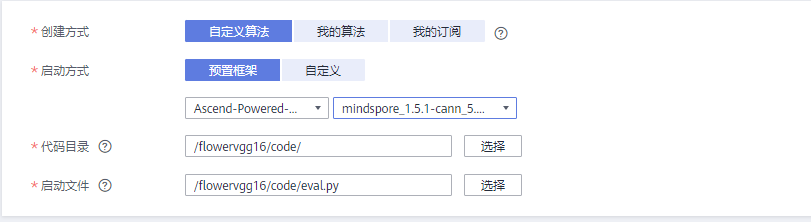
res = model.eval(dataset)

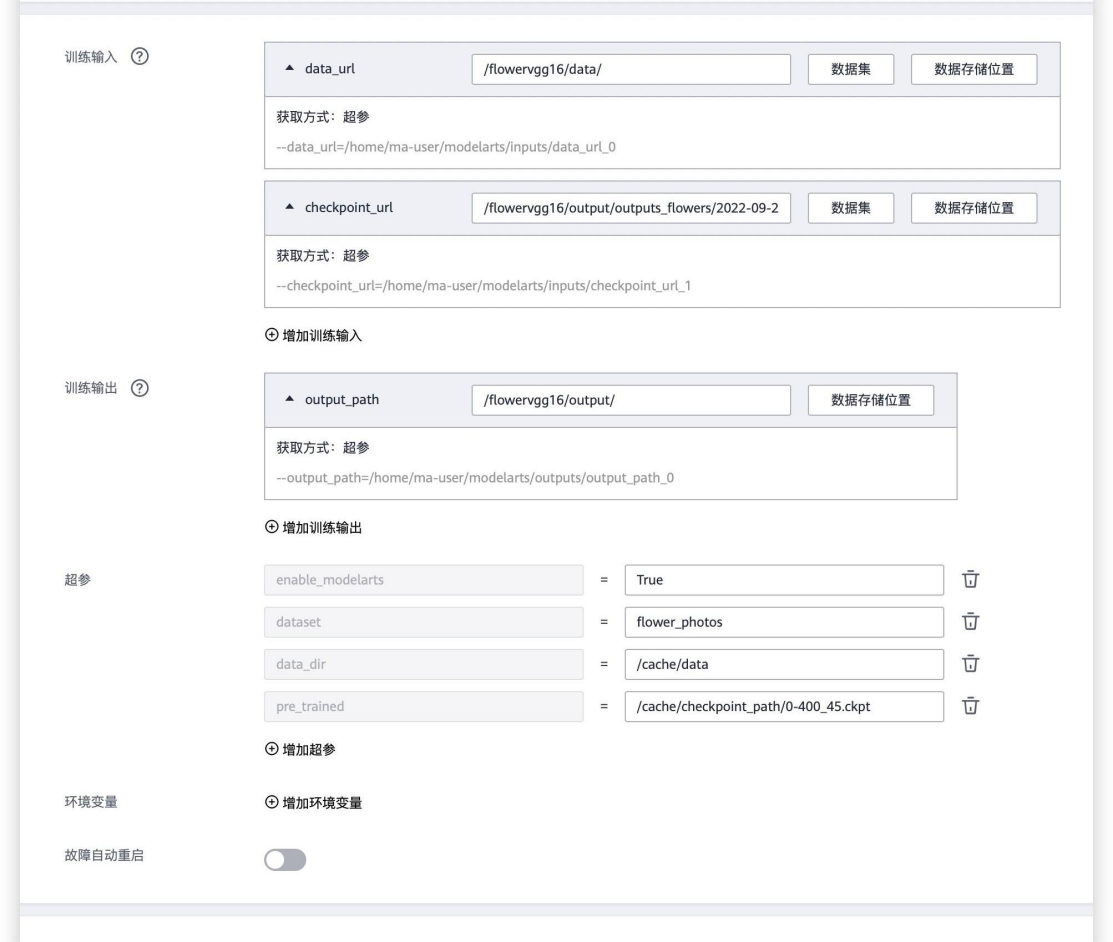
print("result: ", res)

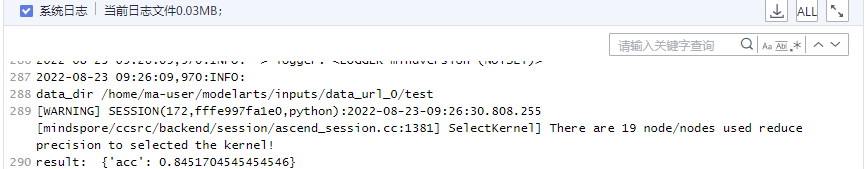
在Modelarts上训练，评估页面设置如下：

需要在训练输入中指定权重文件和数据集文件的目录，其中权重文件需要指定到ckpt存放的文件夹，

然后pre\_trained超参 = /cache/checkpoint\_path/权重文件名.ckpt







如果在命令行运行，指令如下：

python eval.py

模型推理与部署

训练好的模型需要在更多的数据集进行推理和更多平台上进行部署。我们在训练中可以添加检查点（CheckPoint）用于保存模型的参数，以便进行推理及再训练使用。如果想继续在不同的硬件平台上做推理，可通过网络和Checkpoint格式文件生成对应的MindIR文件。

我们将花卉测试集送入训练好的模型执行推理。MindIR是MindSpore框架实现跨平台部署的文件格式，可以将Ascend AI处理器、GPU、CPU等不同平台上训练得到的Checkpoint导出为同一的MindIR格式。MindIR通过统一的表达定义了网络的逻辑结构和算子属性，将MindIR和硬件平台解耦，实现一次训练在不同的平台上部署。

* 导出MindIR文件

导出模型的对应文件为（./code/export.py），运行export.py文件，基于训练完成后的网络模型（即CKPT文件）转换为MindIR格式。

python export.py --config\_path [YMAL\_CONFIG\_PATH] --ckpt\_file [CKPT\_PATH] --file\_name [FILE\_NAME] --file\_format [FILE\_FORMAT]

注：batch\_Size需设置为1，目前花卉数据集仅支持batch\_Size为1的推理，即每次只进行一张图片的推理。

export.py的核心代码如下:

代码：

# 定义网络

if config.dataset == "flower\_photos":

net = vgg17(num\_classes=config.num\_classes, args=config)

# 加载训练好的权重文件

load\_checkpoint(config.ckpt\_file, net=net)

net.set\_train(False)

# 定义网络输入

input\_data = Tensor(np.zeros([config.batch\_size, 3, config.image\_size[0], config.image\_size[1]]), mstype.float32)

# 导出MindIR文件

export(net, input\_data, file\_name=config.file\_name, file\_format=config.file\_format)

## 实验总结

本实验基于Modelarts平台，使用MindSpore深度学习框架，利用其成熟的算子库搭建VGG17神经网络模型，使用花卉数据集（雏菊、蒲公英、玫瑰、向日葵、郁金香）在Ascend910加速卡上进行训练和验证。另外，本实验提出一个思考题，同学们可以对比开启混合精度前后，模型运行速度的快慢，并调研在给定数据集和不改变网络结构的前提下，提升模型训练速度的方法。本实验希望借助MindSpore帮助学生熟悉使用深度学习框架，感受框架封装基本操作的便捷。

## 实验任务与参考解答任务

### 实验任务

基于MindSpore框架构建VGG17网络。利用花卉数据集上完成模型训练（训练平台：ModelArts，可采用昇腾910芯片进行训练）。模型训练完成后，对模型进行保存。

基于昇腾310推理芯片作为计算平台，利用MindSpore框架导入训练好的模型，并在花卉测试数据集对构建的模型进行推理验证，输出推理性能以及测试集正确率。

### 参考答案

完善网络模块的代码如下：

class Vgg(nn.Cell):

"""

TO-DO.

参数:

num\_classes (int): Class numbers. Default: 5.

phase (int): 指定是训练/评估阶段

返回值:

Tensor, infer output tensor.

example：

self.layer1\_conv1 = nn.Conv2d(in\_channels=3, out\_channels=64, kernel\_size=3,weight\_init='XavierUniform')

self.layer1\_bn1 = nn.BatchNorm2d(num\_features=64)

self.layer1\_relu1 = nn.LeakyReLU()

"""

def \_\_init\_\_(self, num\_classes=5, args=None, phase="train"):

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

dropout\_ratio = 0.5

if not args.has\_dropout or phase == "test":

dropout\_ratio = 1.0

self.layer1\_conv1 = nn.Conv2d(in\_channels=3, out\_channels=64, kernel\_size=3,weight\_init='XavierUniform')

self.layer1\_bn1 = nn.BatchNorm2d(num\_features=64)

self.layer1\_relu1 = nn.ReLU()

self.layer1\_conv2 = nn.Conv2d(in\_channels=64, out\_channels=64, kernel\_size=3,weight\_init='XavierUniform')

self.layer1\_bn2 = nn.BatchNorm2d(num\_features=64)

self.layer1\_relu2 = nn.ReLU()

self.layer1\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer2\_conv1 = nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3,weight\_init='XavierUniform')

self.layer2\_bn1 = nn.BatchNorm2d(num\_features=128)

self.layer2\_relu1 = nn.ReLU()

self.layer2\_conv2 = nn.Conv2d(in\_channels=128, out\_channels=128, kernel\_size=3,weight\_init='XavierUniform')

self.layer2\_bn2 = nn.BatchNorm2d(num\_features=128)

self.layer2\_relu2 = nn.ReLU()

self.layer2\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer3\_conv1 = nn.Conv2d(in\_channels=128, out\_channels=256, kernel\_size=3,weight\_init='XavierUniform')

self.layer3\_bn1 = nn.BatchNorm2d(num\_features=256)

self.layer3\_relu1 = nn.ReLU()

self.layer3\_conv2 = nn.Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3,weight\_init='XavierUniform')

self.layer3\_bn2 = nn.BatchNorm2d(num\_features=256)

self.layer3\_relu2 = nn.ReLU()

self.layer3\_conv3 = nn.Conv2d(in\_channels=256, out\_channels=256, kernel\_size=3,weight\_init='XavierUniform')

self.layer3\_bn3 = nn.BatchNorm2d(num\_features=256)

self.layer3\_relu3 = nn.ReLU()

self.layer3\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer4\_conv1 = nn.Conv2d(in\_channels=256, out\_channels=512, kernel\_size=3,weight\_init='XavierUniform')

self.layer4\_bn1 = nn.BatchNorm2d(num\_features=512)

self.layer4\_relu1 = nn.ReLU()

self.layer4\_conv2 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3,weight\_init='XavierUniform')

self.layer4\_bn2 = nn.BatchNorm2d(num\_features=512)

self.layer4\_relu2 = nn.ReLU()

self.layer4\_conv3 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3,weight\_init='XavierUniform')

self.layer4\_bn3 = nn.BatchNorm2d(num\_features=512)

self.layer4\_relu3 = nn.ReLU()

self.layer4\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.layer5\_conv1 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3,weight\_init='XavierUniform')

self.layer5\_bn1 = nn.BatchNorm2d(num\_features=512)

self.layer5\_relu1 = nn.ReLU()

self.layer5\_conv2 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3,weight\_init='XavierUniform')

self.layer5\_bn2 = nn.BatchNorm2d(num\_features=512)

self.layer5\_relu2 = nn.ReLU()

self.layer5\_conv3 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3,weight\_init='XavierUniform')

self.layer5\_bn3 = nn.BatchNorm2d(num\_features=512)

self.layer5\_relu3 = nn.ReLU()

self.layer5\_conv4 = nn.Conv2d(in\_channels=512, out\_channels=512, kernel\_size=3,weight\_init='XavierUniform')

self.layer5\_bn4 = nn.BatchNorm2d(num\_features=512)

self.layer5\_relu4 = nn.ReLU()

self.layer5\_maxpool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.flatten = nn.Flatten()

self.fullyconnect1 = nn.Dense(512 \* 7 \* 7, 4096)

self.relu\_1 = nn.ReLU()

self.dropout\_1 = nn.Dropout(dropout\_ratio)

self.fullyconnect2 = nn.Dense(4096, 4096)

self.relu\_2 = nn.ReLU()

self.dropout\_1 = nn.Dropout(dropout\_ratio)

self.fullyconnect3 = nn.Dense(4096, num\_classes)

def construct(self, x):

x = self.layer1\_conv1(x)

x = self.layer1\_bn1(x)

x = self.layer1\_relu1(x)

x = self.layer1\_conv2(x)

x = self.layer1\_bn2(x)

x = self.layer1\_relu2(x)

x = self.layer1\_maxpool(x)

x = self.layer2\_conv1(x)

x = self.layer2\_bn1(x)

x = self.layer2\_relu1(x)

x = self.layer2\_conv2(x)

x = self.layer2\_bn2(x)

x = self.layer2\_relu2(x)

x = self.layer2\_maxpool(x)

x = self.layer3\_conv1(x)

x = self.layer3\_bn1(x)

x = self.layer3\_relu1(x)

x = self.layer3\_conv2(x)

x = self.layer3\_bn2(x)

x = self.layer3\_relu2(x)

x = self.layer3\_conv3(x)

x = self.layer3\_bn3(x)

x = self.layer3\_relu3(x)

x = self.layer3\_maxpool(x)

x = self.layer4\_conv1(x)

x = self.layer4\_bn1(x)

x = self.layer4\_relu1(x)

x = self.layer4\_conv2(x)

x = self.layer4\_bn2(x)

x = self.layer4\_relu2(x)

x = self.layer4\_conv3(x)

x = self.layer4\_bn3(x)

x = self.layer4\_relu3(x)

x = self.layer4\_maxpool(x)

x = self.layer5\_conv1(x)

x = self.layer5\_bn1(x)

x = self.layer5\_relu1(x)

x = self.layer5\_conv2(x)

x = self.layer5\_bn2(x)

x = self.layer5\_relu2(x)

x = self.layer5\_conv3(x)

x = self.layer5\_bn3(x)

x = self.layer5\_relu3(x)

x = self.layer5\_conv4(x)

x = self.layer5\_bn4(x)

x = self.layer5\_relu4(x)

x = self.layer5\_maxpool(x)

x = self.flatten(x)

x = self.fullyconnect1(x)

x = self.relu\_1(x)

x = self.dropout\_1(x)

x = self.fullyconnect2(x)

x = self.relu\_2(x)

x = self.dropout\_1(x)

x = self.fullyconnect3(x)

return x