1 花卉识别实验

1.1 花卉识别实验介绍

随着电子技术的迅速发展,人们使用便携数码设备(如手机、相机等)获取花卉图像越来越方便,如何自动识别花卉种类受到了广泛的关注。由于花卉所处背景的复杂性,以及花卉自身的类间相似性和类内多样性,利用传统的手工提取特征进行图像分类的方法,并不能很好地解决花卉图像分类这一问题。

图1-1 玫瑰图像

本实验基于华为自研的 MindSpore 深度学习开源框架，搭建卷积神经网络模型，实现花卉的识别。与传统图像分类方法不同,卷积神经网络无需人工提取特征,可以根据输入图像,自动学习包含丰富语义信息的特征,得到更为全面的花卉图像特征描述,可以很好地表达图像不同类别的信息。通过本实验的学习，相信大家对卷积神经网络会有深入的理解，对 MindSpore 深度学习框架的使用会有更深的掌握。

【实验环境要求】：

1、python3.7

2、MindSpore1.7

3、ModelArts 平台

1.2 实验总体设计

该实验的设计可以从两个角度分析，一是功能结构，二是体系结构。

1.2.1 功能结构

花卉识别实验总体设计如图 2-2 所示，该实验可以划分为数据处理、模型构建、图像识别三个主要的子实验。其中数据处理子实验包括数据读取、数据集划分、图像预处理三部分；模型构建子实验主要包括模型定义、模型训练以及模型部署三个部分；图像识别子实验内容主要包括模型准备、模型应用两部分。

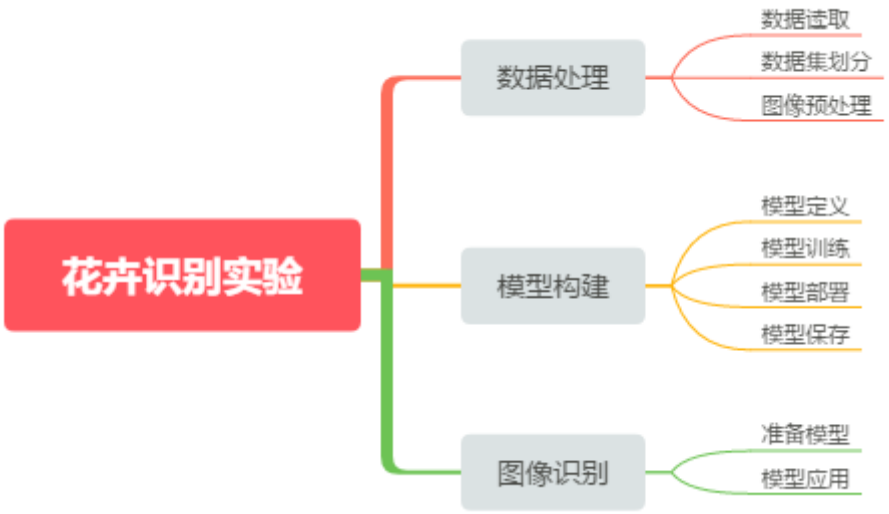


图1-2 实验整体功能结构图

1.2.2 体系结构

按照体系结构划分，整个实验的体系结构可以划分为三部分，分别为模型训练、模型保存和模型推理，如图 2-3 所示。各层侧重点各不相同。训练层运行在安装有 mindspore 框架的服务器，最好配置计算加速卡。推断层运行于开发环境，能够支持卷积神经网络的加速。展示层运行于客户端应用程序，能够完成图像选择并实时显示推断层的计算结果。各层之间存在单向依赖关系。推断层需要的网络模型由训练层提供，并根据需要进行必要的格式转换或加速重构。相应的，展示层要显示的元数据需要由推断层计算得到。

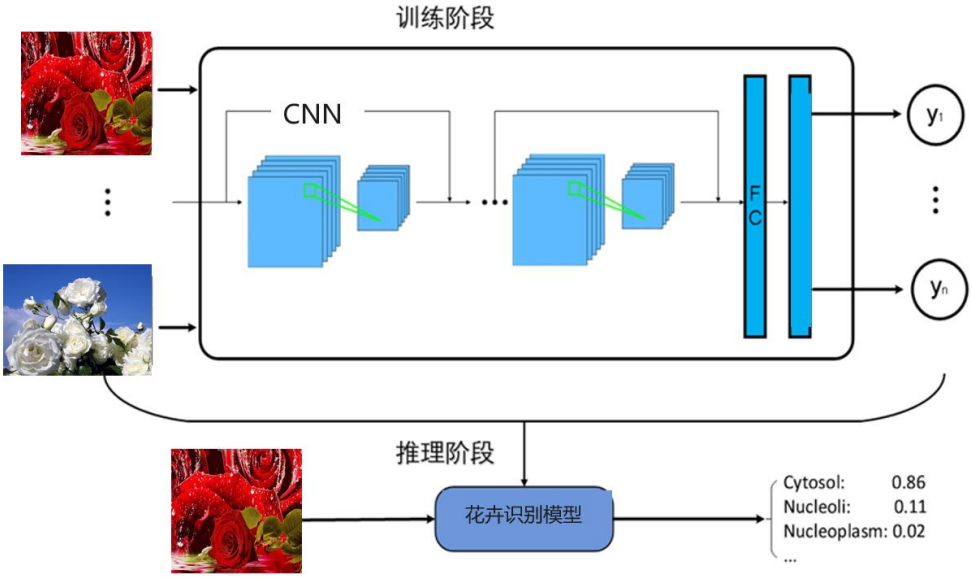


图1-3　实验流程图

1.3 实验详细设计与实现

本节将详细介绍实验的设计与实现。2.3.1 节导入实验环境；2.3.2 节数据准备；2.3.3 节介绍构建花卉识别模型；2.3.4 节介绍如何保存训练后的模型； 2.3.5 节则介绍如何进行模型的测试，以及相应测试结果的展示。

1.3.1 导入实验环境

步骤 1　导入相应的模块并初始化环境

glob 包主要用于查找符合特定规则的文件路径名，跟使用 windows 下的文件搜索相似；

os 模块主要用于处理文件和目录，比如：获取当前目录下文件，删除制定文件，改变目录，查看文件大小等。

```
#easydict 模块用于以属性的方式访问字典的值
from easydict import EasyDict as edict
#glob 模块主要用于查找符合特定规则的文件路径名，类似使用 windows 下的文件搜索
import glob
#os 模块主要用于处理文件和目录
import os

import numpy as np
import matplotlib.pyplot as plt

import mindspore
```

```
#导入 mindspore 框架数据集
import mindspore.dataset as ds
#vision.c_transforms 模块是处理图像增强的高性能模块，用于数据增强图像数据改进训练模型。
import mindspore.dataset.vision.c_transforms as CV
#c_transforms 模块提供常用操作，包括 OneHotOp 和 TypeCast
import mindspore.dataset.transforms.c_transforms as C
from mindspore.common import dtype as mstype
from mindspore import context
#导入模块用于初始化截断正态分布
from mindspore.common.initializer import TruncatedNormal
from mindspore import nn
from mindspore.train import Model
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig, LossMonitor, TimeMonitor
from mindspore.train.serialization import load_checkpoint, load_param_into_net
from mindspore import Tensor
# 设置 MindSpore 的执行模式和设备，此处我们使用的是云平台环境 ModelArts 上的昇腾算力("Ascend")，若在本地执行，device_target 参数设置为"CPU"
context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")
```

步骤 2　定义超参数

```
cfg = edict({
    'data_path': 'flower_photos',
    'data_size':3670,
    'image_width': 100,  # 图片宽度
    'image_height': 100,  # 图片高度
    'batch_size': 32,
    'channel': 3,  # 图片通道数
    'num_class':5,  # 分类类别
    'weight_decay': 0.01,
    'lr':0.0001,  # 学习率
    'dropout_ratio': 0.5,
    'epoch_size': 400,  # 训练次数
    'sigma':0.01,

    'save_checkpoint_steps': 1,  # 多少步保存一次模型
    'keep_checkpoint_max': 1,  # 最多保存多少个模型
    'output_directory': './',  # 保存模型路径
    'output_prefix': "checkpoint_classification"  # 保存模型文件名字
})
```

1.3.2 数据准备

花卉图像识别的数据集为一张张图片，共 5 类图片。MindSpore 读取数据时，通常将数据转换为 dataset 格式，然后传入网络中进行训练和测试。

1.3.2.1 获取数据

这里已经准备好下载链接，运行以下代码就可以获取相应的数据集。

```
# 解压数据集，只需要第一次运行时解压，第二次无需再解压
!wget https://ascend-professional-construction-dataset.obs.myhuaweicloud.com/deep-learning/flower_photos.zip
!unzip flower_photos.zip
```

1.3.2.2 读取数据集并可视化

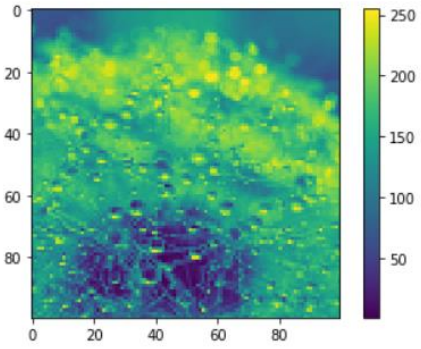
```
#从目录中读取图像的源数据集。
de_dataset = ds.ImageFolderDataset(cfg.data_path,
                                   class_indexing={'daisy':0,'dandelion':1,'roses':2,'sunflowers':3,'tulips':4})
#解码前将输入图像裁剪成任意大小和宽高比。
transform_img = CV.RandomCropDecodeResize([cfg.image_width,cfg.image_height], scale=(0.08, 1.0), ratio=(0.75, 1.333))  #改变尺寸
#转换输入图像；形状（H, W, C）为形状（C, H, W）。
hwc2chw_op = CV.HWC2CHW()
#转换为给定 MindSpore 数据类型的 Tensor 操作。
type_cast_op = C.TypeCast(mstype.float32)
#将操作中的每个操作应用到此数据集。
de_dataset = de_dataset.map(input_columns="image", num_parallel_workers=8, operations=transform_img)
de_dataset = de_dataset.map(input_columns="image", operations=hwc2chw_op, num_parallel_workers=8)
de_dataset = de_dataset.map(input_columns="image", operations=type_cast_op, num_parallel_workers=8)
de_dataset = de_dataset.shuffle(buffer_size=cfg.data_size)
#划分训练集测试集
(de_train,de_test)=de_dataset.split([0.8,0.2])
#设置每个批处理的行数
#drop_remainder 确定是否删除最后一个可能不完整的批（default=False）。
#如果为 True，并且如果可用于生成最后一个批的 batch_size 行小于 batch_size 行，则这些行将被删除，并且不会传播到子节点。
de_train=de_train.batch(cfg.batch_size, drop_remainder=True)
#重复此数据集计数次数。
de_test=de_test.batch(cfg.batch_size, drop_remainder=True)
print('训练数据集数量：',de_train.get_dataset_size()*cfg.batch_size)#get_dataset_size()获取批处理的大小。
print('测试数据集数量：',de_test.get_dataset_size()*cfg.batch_size)
#__next__方法处理后，获取一个 batch 的数据，数据格式为 NCHW，第一维度为 batch 的数量。
data_next=de_dataset.create_dict_iterator(output_numpy=True).__next__()
print('通道数/图像长/宽：', data_next['image'].shape)
```

```
print('一张图像的标签样式：', data_next['label'])  # 一共 5 类，用 0-4 的数字表达类别。
print(data_next['image'][0,...].shape)

plt.figure()
plt.imshow(data_next['image'][1,...])
plt.colorbar()
plt.grid(False)
plt.show()
```

输出结果：

```
训练数据集数量： 2912
测试数据集数量： 704
通道数/图像长/宽： (3, 100, 100)
一张图像的标签样式： 1
(100, 100)
```



1.3.3 构建花卉识别模型

1.3.3.1 网络结构框架介绍

花卉图像数据集准备完成，接下来我们就需要构建训练模型，本实验采用的是 CNN 神经网络算法，所以我们首先需要建立初始化的神经网络。

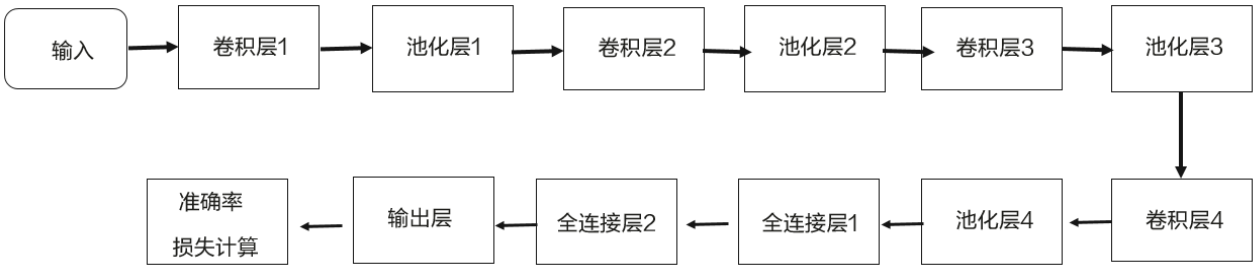


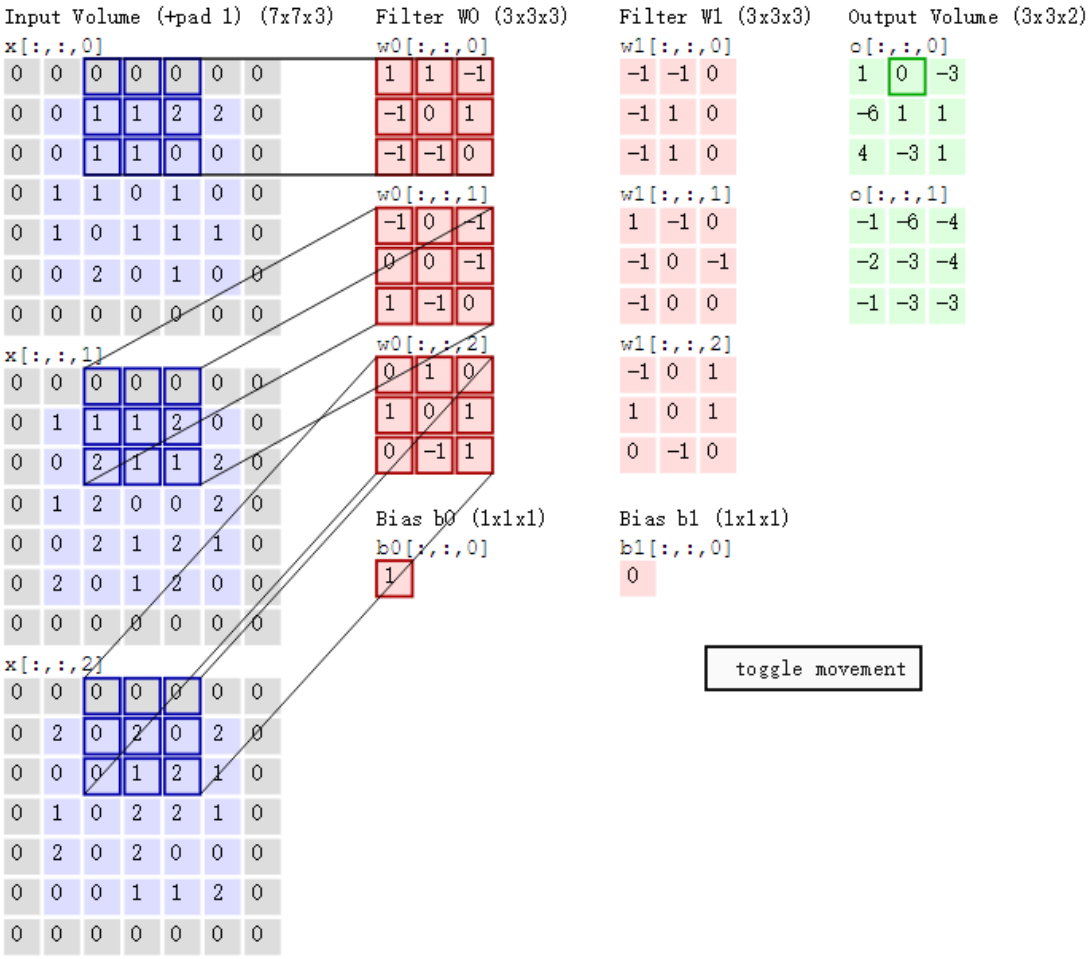
图1-4 神经网络框架图示意图

1.3.3.2 构建 CNN 神经网络

卷积层：

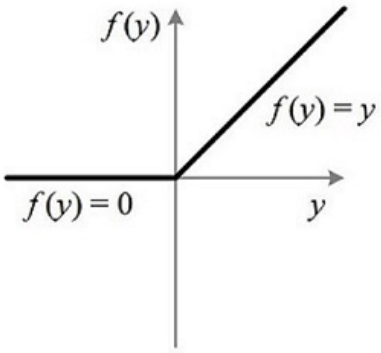
在卷积层计算过程中，输入是一定区域大小(width*height)的数据，和滤波器 filter（带着一组固定权重的神经元）做内积后得到新的二维数据。

具体来说，输入图像，然后就是滤波器 filter（带着一组固定权重的神经元）进行对应点乘，不同的滤波器 filter 会得到不同的输出数据，比如颜色深浅、轮廓。如果想提取图像的不同特征，则用不同的滤波器 filter，提取想要的关于图像的特定信息：颜色深浅或轮廓。



【更多资料】https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

激活函数：

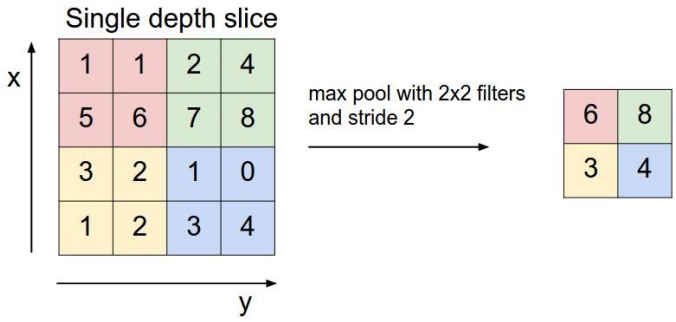


$$ReLU(x) = \begin{cases} x & if x > 0 \\ 0 & if x \le 0 \end{cases}$$

ReLU 函数其实是分段线性函数，把所有的负值都变为 0，而正值不变，这种操作被称为单侧抑制。正因为有了这单侧抑制，才使得神经网络中的神经元具有了稀疏激活性。尤其体现在深度神经网络模型(如 CNN)中，当模型增加 N 层之后，理论上 ReLU 神经元的激活率将降低 2 的 N 次方倍。

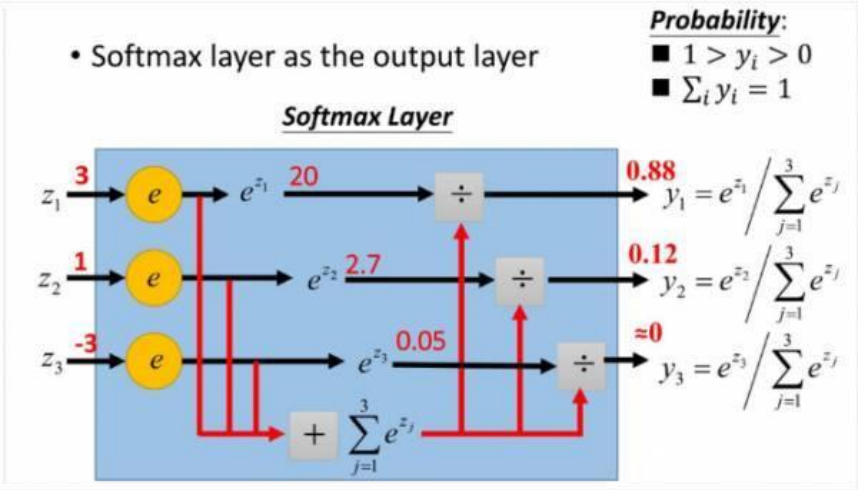
此外，相比于其它激活函数来说，ReLU 有以下优势：对于线性函数而言，ReLU 的表达能力更强，尤其体现在深度网络中；而对于非线性函数而言，ReLU 由于非负区间的梯度为常数 1，因此不存在梯度消失问题(Vanishing Gradient Problem)，使得模型的收敛速度维持在一个稳定状态。这里稍微描述一下什么是梯度消失问题：当梯度小于 1 时，预测值与真实值之间的误差每传播一层会衰减一次，如果在深层模型中使用 sigmoid 作为激活函数，这种现象尤为明显，将导致模型收敛停滞不前。

池化层：



上图所展示的是取区域最大，即上图左边部分中 左上角 2x2 的矩阵中 6 最大，右上角 2x2 的矩阵中 8 最大，左下角 2x2 的矩阵中 3 最大，右下角 2x2 的矩阵中 4 最大，所以得到上图右边部分的结果：6 8 3 4。

全连接层：



softmax 直白来说就是将原来输出是 3,1,-3 通过 softmax 函数作用，就映射成为(0,1)的值， 而这些值的累和为 1（满足概率的性质），那么我们就可以将它理解成概率，在最后选取输出结点的时候，我们就可以选取概率最大（也就是值对应最大的）结点，作为我们的预测目标！

1.3.3.3 思考题

1. Softmax 函数是用于多分类任务的，如果是处理二分类任务我们用什么函数呢？
2. 请结合其他函数的特点，思考下 Relu 函数的优缺点？

【答案】

1、Sigmoid 函数

2、优点：

（1）ReLu 的收敛速度比 sigmoid 和 tanh 快；（梯度不会饱和，解决了梯度消失问题）

（2）计算复杂度低，不需要进行指数运算；

（3）适合用于后向传播。

缺点：

（1）ReLU 的输出不是 zero-centered；

（2）Dead ReLU Problem（神经元坏死现象）：某些神经元可能永远不会被激活，导致相应参数永远不会被更新（在负数部分，梯度为 0）。产生这种现象的两个原因：参数初始化问题；learning rate 太高导致在训练过程中参数更新太大。 解决方法：采用 Xavier 初始化方法，以及避免将 learning rate 设置太大或使用 adagrad 等自动调节 learning rate 的算法。

（3）ReLU 不会对数据做幅度压缩，所以数据的幅度会随着模型层数的增加不断扩张。

1.3.3.4 定义网络

```
# 定义 CNN 图像识别网络
class Identification_Net(nn.Cell):
    def __init__(self, num_class=5,channel=3,dropout_ratio=0.5,trun_sigma=0.01):  # 一共分五类，图片通道数是 3
        super(Identification_Net, self).__init__()
        self.num_class = num_class
        self.channel = channel
        self.dropout_ratio = dropout_ratio
        #设置卷积层
        self.conv1 = nn.Conv2d(self.channel, 32,
                               kernel_size=5, stride=1, padding=0,
                               has_bias=True, pad_mode="same",
                               weight_init=TruncatedNormal(sigma=trun_sigma),bias_init='zeros')
        #设置 ReLU 激活函数
        self.relu = nn.ReLU()
        #设置最大池化层
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2,pad_mode="valid")
        self.conv2 = nn.Conv2d(32, 64,
                               kernel_size=5, stride=1, padding=0,
                               has_bias=True, pad_mode="same",
                               weight_init=TruncatedNormal(sigma=trun_sigma),bias_init='zeros')
        self.conv3 = nn.Conv2d(64, 128,
                               kernel_size=3, stride=1, padding=0,
                               has_bias=True, pad_mode="same",
                               weight_init=TruncatedNormal(sigma=trun_sigma),bias_init='zeros')
        self.conv4 = nn.Conv2d(128, 128,
                               kernel_size=3, stride=1, padding=0,
                               has_bias=True, pad_mode="same",
                               weight_init=TruncatedNormal(sigma=trun_sigma), bias_init='zeros')
        self.flatten = nn.Flatten()
        self.fc1 = nn.Dense(6*6*128, 1024,weight_init =TruncatedNormal(sigma=trun_sigma),bias_init = 0.1)
        self.dropout = nn.Dropout(self.dropout_ratio)
        self.fc2 = nn.Dense(1024, 512, weight_init=TruncatedNormal(sigma=trun_sigma), bias_init=0.1)
        self.fc3 = nn.Dense(512, self.num_class, weight_init=TruncatedNormal(sigma=trun_sigma), bias_init=0.1)
    #构建模型
    def construct(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.max_pool2d(x)
        x = self.conv3(x)
        x = self.max_pool2d(x)
```

```
        x = self.conv4(x)
        x = self.max_pool2d(x)
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

1.3.3.5 构建优化器

定义优化器：使用 adam 优化器。

```
#实例化网络
net=Identification_Net(num_class=cfg.num_class, channel=cfg.channel, dropout_ratio=cfg.dropout_ratio)
#计算 softmax 交叉熵，以此作为损失函数
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
#获取网络中需要更新的参数
fc_weight_params = list(filter(lambda x: 'fc' in x.name and 'weight' in x.name, net.trainable_params()))
other_params=list(filter(lambda x: 'fc' not in x.name or 'weight' not in x.name, net.trainable_params()))
group_params = [{'params': fc_weight_params, 'weight_decay': cfg.weight_decay},
                {'params': other_params},
                {'order_params': net.trainable_params()}]
#设置 Adam 优化器，将前一步设定的需要更新的参数传入
net_opt = nn.Adam(group_params, learning_rate=cfg.lr, weight_decay=0.0)
#net_opt = nn.Adam(params=net.trainable_params(), learning_rate=cfg.lr, weight_decay=0.1)
#编译网络、损失函数和优化器
model = Model(net, loss_fn=net_loss, optimizer=net_opt, metrics={"acc"})
```

1.3.4 训练模型

总共训练 400 个 epoch，每 10 个 epoch 都需要在训练集上进行运行，并打印出相应的 loss 值。

```
#设定 callback 监控指标
loss_cb = LossMonitor(per_print_times=de_train.get_dataset_size()*10)
config_ck = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps,
                             keep_checkpoint_max=cfg.keep_checkpoint_max)
ckpoint_cb = ModelCheckpoint(prefix=cfg.output_prefix, directory=cfg.output_directory, config=config_ck)
print("============== Starting Training ==============")
#开始训练，训练时将 callback 监控指标设定好
model.train(cfg.epoch_size, de_train, callbacks=[loss_cb, ckpoint_cb], dataset_sink_mode=True)
```

```
# 使用测试集评估模型，打印总体准确率
metric = model.eval(de_test)
print(metric)
```

1.3.4.1 思考题

1. Flatten 层用来将输入“压平”，即把多维的输入一维化，我们为什么要做这部分操作呢？

【答案】

1、 将所有图片的像素值转化为一列，映射下一层的神经元。

1.3.5 花卉图像识别模型推理

1.3.5.1 模型验证

上一步的实验中我们已经设定了模型参数的保存路径，这里就直接加载 CKPT 文件然后利用模型进行推理。

```
#加载模型
import os
#获取路径
CKPT = os.path.join(cfg.output_directory,cfg.output_prefix+'-
'+str(cfg.epoch_size)+'_'+str(de_train.get_dataset_size())+'.ckpt')
#实例化网络
net = Identification_Net(num_class=cfg.num_class, channel=cfg.channel, dropout_ratio=cfg.dropout_ratio)
#将参数加载进网络
load_checkpoint(CKPT, net=net)
#编译整个模型
model = Model(net)
# 预测
class_names = {0:'daisy',1:'dandelion',2:'roses',3:'sunflowers',4:'tulips'}
#获取测试集的 batch
test_ = de_test.create_dict_iterator().__next__()
#转换为 tensor
test = Tensor(test_['image'], mindspore.float32)
#预测
predictions = model.predict(test)
predictions = predictions.asnumpy()
true_label = test_['label'].asnumpy()

#显示预测结果
for i in range(9):
    p_np = predictions[i, :]
    pre_label = np.argmax(p_np)
```

```
print('第' + str(i) + '个 sample 预测结果：', class_names[pre_label], '真实结果：',class_names[true_label[i]])
```

模型验证结果：

```
第0个sample预测结果:  sunflowers 真实结果:  sunflowers
第1个sample预测结果:  dandelion 真实结果:  dandelion
第2个sample预测结果:  dandelion 真实结果:  dandelion
第3个sample预测结果:  daisy 真实结果:  daisy
第4个sample预测结果:  dandelion 真实结果:  dandelion
第5个sample预测结果:  dandelion 真实结果:  dandelion
第6个sample预测结果:  tulips 真实结果:  tulips
第7个sample预测结果:  dandelion 真实结果:  dandelion
第8个sample预测结果:  roses 真实结果:  roses
```

图1-5 推理结果

1.3.5.2 思考题

1. 通过结果可以看出图像识别模型出现欠拟合现象，我们如何解决这个问题呢？

【答案】

1、 模型复杂程度、使用合适的激活函数、使用合适的优化器、不使用防止过拟合的方法、检查数据规模

1.4 实验总结

本章提供了一个基于华为自研 MindSpore 框架的图像识别实验。该实验演示了如何使用 MindSpore 框架搭建卷积神经网络，完成图像识别任务。本章对实验做了详尽的剖析。阐明了整个实验功能、结构与流程是如何设计的，详细解释了如何解析数据、如何构建深度学习模型、如何保存模型等内容。

训练保存后的模型在多个类别图片下进行测试，结果表明模型具有较快的推断速度和较好的识别性能。读者可以在该实验实验的基础上开发更有针对性的应用实验。

1.5 思考题-汇总

1、 Softmax 函数是用于多分类任务的，如果是处理二分类任务我们用什么函数？

2、 请结合其他函数的特点，思考下 Relu 函数的优缺点？

3、 Flatten 层将输入“压平”，即把多维的输入一维化，我们为什么要做这部分操作？

4、 通过结果可以看出图像识别模型出现欠拟合现象，我们如何解决这个问题呢？

1.6 开放题（可选）

基于本实验的描述，请使用数据集 CIFAR10 完成图片识别任务并搭建一套自定义的图片识别实验。