

哈尔滨工业大学(深圳)

《人工智能》实验报告

实验二

深度学习实现花卉识别

学 院: 计算机科学与技术

班 级: 1

姓 名: 林先一/谢俊安/杨航/向柳媛

学 号: 190110128/15/21/30

学 期: 2022 春

实验地点:

实验教师:

目录

1	实验内容	2
2	实验记录	2
2.1.	实验环境	2
2.2.	数据集处理	3
2.3.	模型构建与训练	6
2.3.1.	CNN 模型 190110121-杨航	6
(1)	模型定义	6
(2)	模型训练	9
2.3.2.	CNN 模型_tensorflow 框架 (190110130_向柳媛)	10
(1)	模型定义	10
(2)	模型训练	13
2.3.3.	CNN 模型_MindSpore [190110128_林先一]	15
(3)	模型定义	15
(4)	模型训练	15
2.3.4.	利用 VGG16 网络架构进行的迁移学习 (by 谢俊安)	18
(1)	模型定义	18
(2)	模型训练	19
2.4.	实验结果	24
2.5.	深度学习框架对比	29
3	总结	32
3.1.	问题及解决方法	32
3.2.	实验的启发、总结及建议	34
4	参考文献	36

实验报告内容包含但不限于以下内容，如有补充请用红色*标注。

注意标注每部分的作者，作者可标注在小标题上。

1 实验内容

简要概述本次实验的内容，用到哪些框架。

基于给定的数据集，在本地分别用 TensorFlow、MindSpore、Pytorch 框架实现花卉识别。须自己一层层实现模型的定义。

2 实验记录

2.1. 实验环境

阐述实验用到的环境，包括操作系统、开发软件、用到的库及版本号。

杨航：

实验环境：Python3

操作系统：windows

开发软件：Pycharm2021

库：torch、numpy、sklearn

向柳媛：

实验环境：Python3

操作系统：windows

开发软件：Pycharm2021

库：tensorflow、numpy、sklearn

林先一：

- 实验环境：Python3.7.5
- 操作系统：windows+macos

- 开发软件：Visual Studio Code
- 库：MindSpore, numpy

谢俊安：

- 操作系统：MacOS Monterey12.3.1
- 开发软件：PyCharm
- 库及版本号：
 - Tensorflow 2.6.2
 - Cv2
 - Keras 2.6.0
 - numpy 1.19.5
 - matplotlib 3.3.4
 - scikit-learn 0.24.2

2.2. 数据集处理

阐述对数据集进行的预处理，包括但不限于数据集划分、数据扩充、数据重采样、数据增强等，写明处理的方法、理由及对实验实验结果的影响。

杨航：

数据集采取随机划分的方式，使用 `sklearn` 库，按照 8: 2 的比例将数据集划分为训练集和测试集。同时为了提高模型的收敛速度，对数据进行的标准化处理。

对所确定模型，尝试了三个不同的种子进行三次随机划分，将最后得到的在测试集上的表现取了平均，得到了最终的结果。三次随机划分所得到的准确率都较相似，表明模型具有一定稳定性。

向柳媛：

用 `sklearn` 库里的 `train_test_split` 函数划分数据，测试集占 10%，并设置随机

数种子:

```
seed = 109 # 设置随机数种子, 即seed值
np.random.seed(seed) # 保证生成的随机数具有可预测性, 即相同的种子 (seed值) 所产生的随机数是相同的

# 拆分数据集一部分为训练集一部分为验证集, 拆分比例可调整
(x_train, x_val, y_train, y_val) = train_test_split(data, label, test_size=0.10, random_state=seed)
```

用 keras 库里的 preprocessing 将数据标准化处理:

```
# 第1层: 输入层: 归一化
ke.layers.experimental.preprocessing.Rescaling(1. / 255, input_shape=(h, w, 3)),
```

处理后加快了模型的速度, 且随即划分的结果都比较相似, 表明模型具有一定稳定性。

林先一:

刚开始只是一味对原数据集进行训练, 调整学习率或者 epoch, 但是总是要么在验证集上本身就预测效果极差, 要么就是在验证集上预测过度完美, 即过拟合的情况, 然后在测试集上表现很差。

鉴于数据集本身规模较小, 且不同分类之间的规模差异较大, 加上训练过程中还需要进一步的划分训练集和验证集, 故考虑做图像数据的预处理。

预处理部分: 通常情况的图像分类中的预处理, 为了是模型的泛化能力更强, 更能应对不同环境条件来源的图像, 会采用在图片中随机裁剪/旋转/遮挡等, 这里参考了谷歌在 2018 年提出通过 AutoML 来自动搜索数据增强策略, 即 AutoAugment。搜索方法采用强化学习, 在搜索空间里, 一个 policy 包含 5 个 sub-policies, 每个 sub-policy 包含两个串行的图像增强操作, 每个增强操作有两个超参数: 进行该操作的概率和图像增强的幅度 (magnitude, 这个表示数据增强的强度, 比如对于旋转, 旋转的角度就是增强幅度, 旋转角度越大, 增强越大)。每个 policy 在执行时, 首先随机从 5 个策略中随机选择一个 sub-policy, 然后序列执行两个图像操作。考虑到[一方面 AutoAugment 本身需要训练网络, 第二方面就是本人电脑实在是有点老旧, 无法在本地完成, 可是使用华为云移植是又实在无法处理文件路径等方面出现的 bug, 所以放弃了。

最终选择了 AutoAugment 的儿童版本, 解决了前者存在“搜索空间巨大”的难点, 即谷歌于 2019 年提出的一个更简单的数据增强策略: RandAugment。

通过概率随机的形式，将不同的 policy 变换应用在小规模的数据集上。具体实现中，使用了 GitHub/CoinCheung 复现的基于 opencv 的 RandAugment。

具体的代码实现可以参考源码中被注释掉的部分，截图仅为配置与一些简单的处理结果：首先定义 RandAugment 处理流程，N=2 为每次对给定图像做两种类型的变换，M=9 为变换的强度，NM 参数的设置参考了相关介绍以及论文中的实验结果，没有过多的时间和资源进行调整；首先对 6 个分类的数据集都做了一

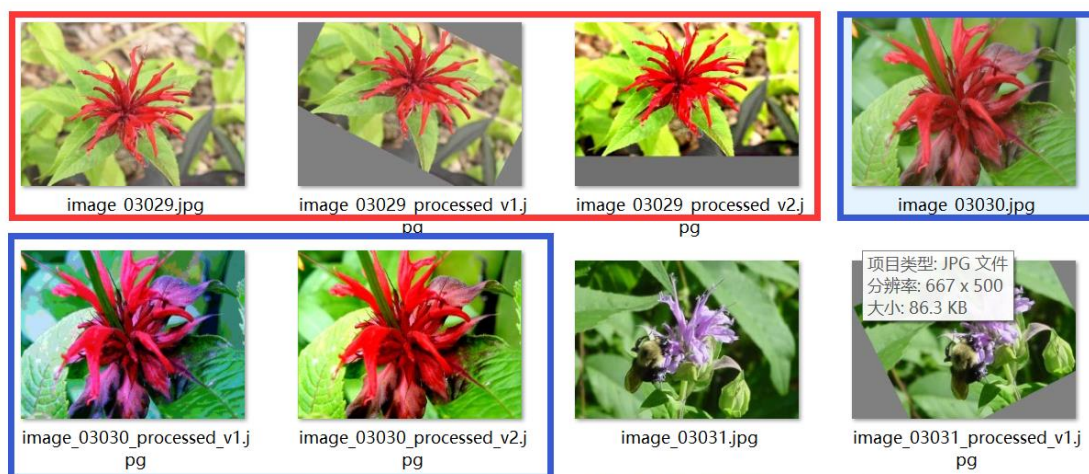
```
# 用于图像数据集增强
import cv2 as cv
from AutoAugment_opencv.AA_classification import AutoAugment, RandomAugment

# 2. 数据集预处理：应用RandomAugment进行数据集扩充，同时减少模型训练过程中的过拟合问题
# 源数据集-路径
| #old_dataset_path = 'C:/Users/Charles/Desktop/mindspore-classification/flower_photos_before'
old_dataset_path = '../flower_photos_before'
# 扩充及其他防过拟合处理后的数据集-路径
new_dataset_path = '../flower_photos_processed'

flower_kinds = os.listdir(old_dataset_path)
print(flower_kinds)

RA = RandomAugment(N=2,M=9) # 定义randomAugment随机图像数据增强操作
```

次扩充，然后对 4 个数据规模较小的额外处理了一次，平衡了一下数据集规模之间的差异，处理后的图片效果展示如下：



谢俊安：

数据集划分为训练集:测试集=8:2，相对是比较合理的划分方法，既给予了模型足够的训练样本，也获得了充足的测试样例来验证模型的正确性。

2.3. 模型构建与训练

以下部分以模型为单位写，每种模型均需要包括模型定义和模型训练部分。

2.3.1. CNN 模型 190110121-杨航

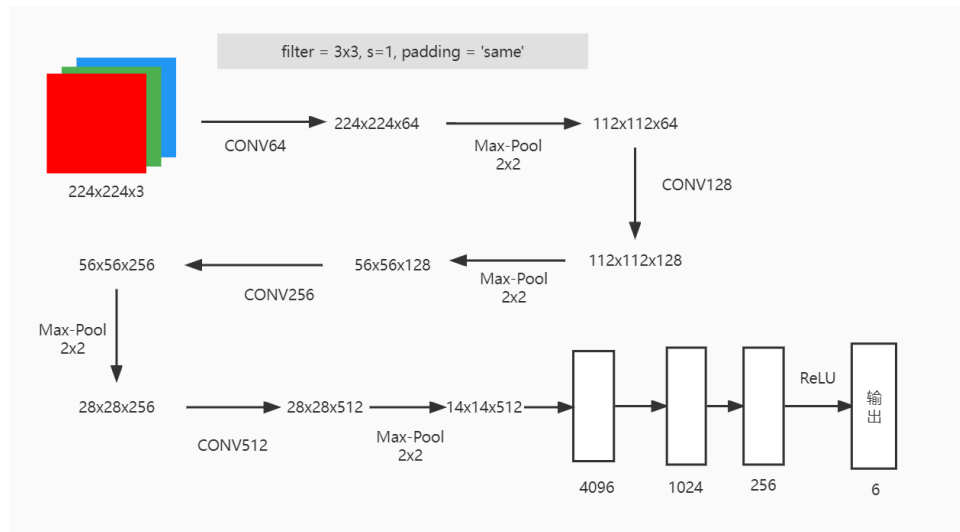
(1) 模型定义

阐述本次实验使用的模型，详细说明模型结构及选型，包含但不限于卷积层、池化层、全连接层、损失函数等主要参数及选型理由，提供完整的模型结构图，截图模型定义代码。

使用框架：pytorch

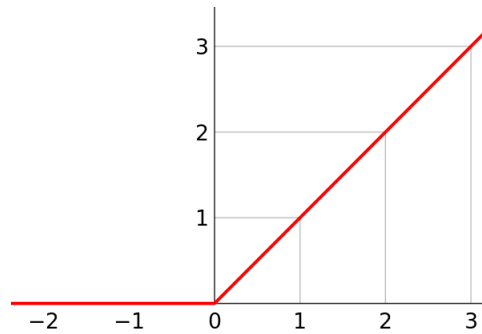
模型参考：VGG16

模型结构：



优化器：Adam

激活函数：ReLU



激活函数优势：

- 仿生物学原理
- 更有效率的梯度下降及反向传播，避免了梯度消失和梯度爆炸
- 简化了计算过程

防止过拟合：在全连接层进行随机 dropout, dropout 率为 0.5

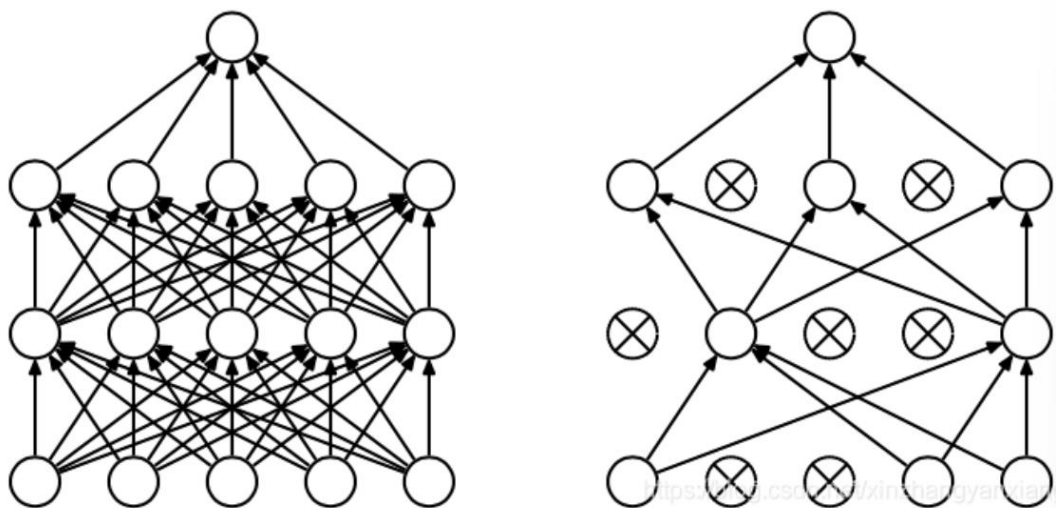
Dropout 的优势及原理：

优势：

- 找到更“瘦”的网络，减少训练时间
- 防止过拟合

原理：

进行 dropout 前的网络如下图左边所示，dropout 后的如右边所示：



对于一个有 N 个节点的神经网络，有了 dropout 后，就可以看做是 2^n 个模型的集合了，但此时要训练的参数数目却是不变的，这就解脱了费时的问题。

经过交叉验证，隐含节点 dropout 率等于 0.5 的时候效果最好，原因是 0.5 的时候 dropout 随机生成的网络结构最多。

选型理由：VGG16 模型能够很好的适用于分类的任务，但由于本次实验的数据量较小，因此在网络原本的基础上删除掉了一部分卷积层，将网络结构简化了许多。在选择 VGG16 前，还尝试过 LeNet5，但由于网络较浅，在同样训练次数的表现上，效果不如 VGG16。

定义代码：

本地：

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # 网络层定义
        # 输入[3*128*128]
        ## 第一层卷积
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=3, # 输入图片的通道
                out_channels=64, # 第一层的filter, 数量不能太
                kernel_size=3,
                stride=1,
                padding='same'
            ),
            # 经过卷积层, 输出[64, 224, 224], 传入池化层
            nn.MaxPool2d(kernel_size=2), # 经过池化, 输出[64, 1
        )
        ## 第二层卷积
        self.conv2 = nn.Sequential(
            nn.Conv2d(
                in_channels=64,
                out_channels=128,
                kernel_size=3,
                stride=1,
                padding='same'
            ),
            # 经过卷积层, 输出[128, 112, 112]
            nn.MaxPool2d(kernel_size=2), # 输出[128, 56, 56]
            # nn.Dropout(p=0.5),
            # nn.ReLU()
        )
        ## 第三层卷积
        self.conv3 = nn.Sequential(
            nn.Conv2d(
                in_channels=128,
                out_channels=256,
                kernel_size=3,
                stride=1,
                padding='same'
            ),
            # 经过卷积层, 输出[256, 56, 56]
            # nn.ReLU(),
            nn.MaxPool2d(kernel_size=2), # 输出[256, 28, 28]
        )
        ## 第四层卷积
        self.conv4 = nn.Sequential(
            nn.Conv2d(
                in_channels=256,
                out_channels=512,
                kernel_size=3,
                stride=1,
                padding='same'
            ),
            # 经过卷积层, 输出[512, 28, 28]
            # nn.Dropout(p=0.5),
            # nn.ReLU(),
            nn.MaxPool2d(kernel_size=2) # 输出[512, 14, 14]
        )
        ## 线性输出层
        # 6种分类, 因此out_features为6
        self.linear1 = nn.Sequential(
            nn.Linear(in_features=512 * 14 * 14, out_features=4096),
            nn.Linear(in_features=4096, out_features=1024),
            nn.Linear(in_features=1024, out_features=256),
            nn.ReLU()
        )
        self.output = nn.Linear(in_features=256, out_features=6)
```

云上训练的模型线形层：

```
self.linear1 = nn.Sequential(
    nn.Flatten(),
    nn.Linear(in_features=512 * 14 * 14, out_features=4096),
    nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(in_features=4096, out_features=1000),
    # nn.Linear(in_features=1024, out_features=256),
    nn.ReLU(),
    # nn.Dropout(p=0.3)
)
self.output = nn.Linear(in_features=1000, out_features=6)
```

前推函数：

```
def forward(self, x):  
    x = self.conv1(x)  
    x = self.conv2(x)  
    x = self.conv3(x)  
    x = self.conv4(x)  
    x = x.view(x.size(0), -1) # 保留batch, 将后面的乘到一起  
    x = self.linear1(x)  
    output = self.output(x) # 输出[batch,10]  
    return output
```

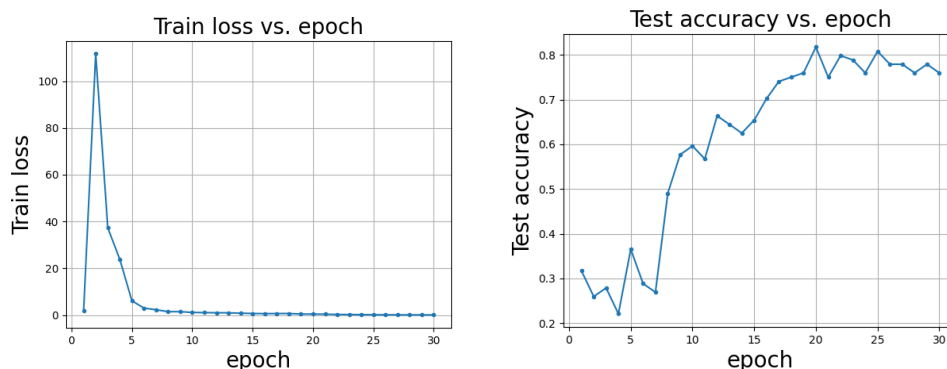
(2) 模型训练

详细说明模型在训练集上的训练速度、收敛速度、收敛精度、超参数设置与调整、优化算法选择等，提供训练日志或 loss 曲线的截图。

在 lr 固定为 0.001, batch_size 为 64 时，训练回合为 30 时得到的其中一次结果如下：

```
第24次epoch:  
Epoch: 24 | train loss: 0.1339 | test accuracy: 0.81  
第25次epoch:  
Epoch: 25 | train loss: 0.1067 | test accuracy: 0.78  
第26次epoch:  
Epoch: 26 | train loss: 0.0957 | test accuracy: 0.78  
第27次epoch:  
Epoch: 27 | train loss: 0.0705 | test accuracy: 0.76  
第28次epoch:  
Epoch: 28 | train loss: 0.0809 | test accuracy: 0.78  
第29次epoch:  
Epoch: 29 | train loss: 0.0565 | test accuracy: 0.76  
range(1, 31)  
range(1, 31)  
y_pred [5 3 5 4 5]  
y_val tensor([5, 3, 0, 4, 5])  
第 1 朵花预测:bee  
第 2 朵花预测:blackberry  
第 3 朵花预测:blanket  
第 4 朵花预测:bee  
第 5 朵花预测:bromelia  
第 6 朵花预测:foxglove
```

其中 loss、accuracy 随着 epoch 的变化如下：



可以见到在大约 20 回合后，准确率便开始在 0.8 左右波动，分析可能为学习速率偏高。

当降低学习率/减小 batch_size:

```
Epoch: 28 | train loss: 0.6156 | test accuracy: 0.72
第29次epoch:
Epoch: 29 | train loss: 0.5931 | test accuracy: 0.70
y_pred [5 3 5 4 5]
y_val tensor([5, 3, 0, 4, 5])
shape of data: torch.Size([6, 3, 64, 64])
6
第 1 朵花预测:bromelia
第 2 朵花预测:blackberry
第 3 朵花预测:blackberry
第 4 朵花预测:foxglove
第 5 朵花预测:bougainvillea
第 6 朵花预测:foxglove
```

最后在测试的六张图片上表现不佳，且准确率也低于上一表现。

同时，由于网络模型较大，运行速度偏慢，因此尝试了将前一次训练的模型参数保存下来，在下一次训练时先 load 进网络，调低学习率继续训练的方法。这样单次训练时间较短，并且能够通过结果实时调整参数观察结果，再继续修订模型。这样的方法每次训练回合数较低，但结果优于单次训练。下图为部分结果：

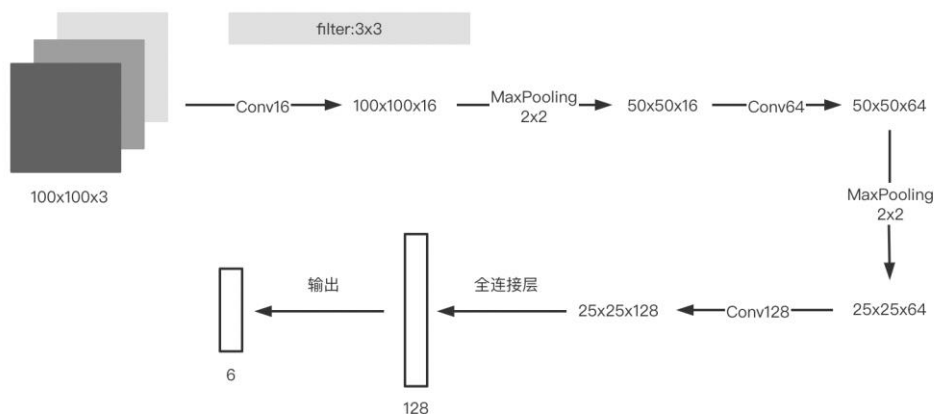
```
Lab2_flower_classify x
第4次epoch:
Epoch: 4 | train loss: 0.2039 | test accuracy: 0.77
第5次epoch:
Epoch: 5 | train loss: 0.1858 | test accuracy: 0.77
第6次epoch:
Epoch: 6 | train loss: 0.1481 | test accuracy: 0.80
第7次epoch:
Epoch: 7 | train loss: 0.1302 | test accuracy: 0.78
第8次epoch:
Epoch: 8 | train loss: 0.1351 | test accuracy: 0.79
第9次epoch:
Epoch: 9 | train loss: 0.1109 | test accuracy: 0.85
y_pred [5 3 5 4 5]
y_val tensor([5, 3, 0, 4, 5])
shape of data: torch.Size([6, 3, 64, 64])
6
第 1 朵花预测:bromelia
第 2 朵花预测:blackberry
第 3 朵花预测:blanket
第 4 朵花预测:bougainvillea
第 5 朵花预测:bromelia
第 6 朵花预测:foxglove
```

但由于参数文件较大，无法上传到提交平台，因此最终未采取这种方法。

2.3.2. CNN 模型_tensorflow 框架（190110130_向柳媛）

（1）模型定义

- 模型：卷积神经网络（CNN）
- 模型结构图：



● 模型定义代码:

本地:

```
# Todo 自行实现模型结构
model = ke.Sequential([
    # 第1层: 输入层: 归一化
    ke.layers.experimental.preprocessing.Rescaling(1. / 255, input_shape=(h, w, 3)),
    # 第2-3层: 卷积层, 池化层
    ke.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(h, w, 3)), # 卷积核数
    ke.layers.MaxPooling2D((2, 2)), # 最大池化, 降低维度
    # 第4-5层: 卷积层, 池化层
    ke.layers.Conv2D(64, (3, 3), activation='relu'),
    ke.layers.MaxPooling2D((2, 2)),
    # 第6层: 卷积层
    ke.layers.Conv2D(128, (3, 3), activation='relu'),
    # 第7层: Flatten层: 连接卷积层与全连接层, 把多维的输入一维化
    ke.layers.Flatten(),
    # 第8层: 全连接层: 输出维度为128, 激活函数为relu
    ke.layers.Dense(128, activation='relu'),
    # 第9层: 输出层: 输出预期结果 (参数为类别数: 6)
    ke.layers.Dense(6)
])
```

云上训练的模型:

```

# Todo 自行实现模型结构
x_train = x_train / 255 # 训练集图片标准化
x_val = x_val / 255 # 测试集图片标准化
model = ke.Sequential([
    # 第1-2层: 卷积层,池化层
    ke.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(h, w, 3)),
    ke.layers.MaxPooling2D((2, 2)), # 最大池化, 降低维度
    # 第3-4层: 卷积层,池化层
    ke.layers.Conv2D(64, (3, 3), activation='relu'),
    ke.layers.MaxPooling2D((2, 2)),
    # 第5层: 卷积层
    ke.layers.Conv2D(128, (3, 3), activation='relu'),
    # 第6层: Flatten层: 连接卷积层与全连接层, 把多维的输入一维化
    ke.layers.Flatten(),
    # 第7层: 全连接层: 输出维度为128,, 激活函数为relu
    ke.layers.Dense(128, activation='relu'),
    # 第8层: 输出层: 输出预期结果 (参数为类别数: 6)
    ke.layers.Dense(6)
])

```

● 模型选型:

■ 第 1 层: 输入层

将数据归一化

■ 第 2-3 层: 卷积层, 池化层

卷积层: 提取输入的不同特征。

卷积核数目为 16 (决定输出的 depth 厚度), 卷积核尺寸为 3*3, 激活函数为 relu (加入非线性因素, 提高对模型的表达能力);

池化层: 降低维度, 缩减模型大小, 提高计算速度。

池化窗口尺寸: 2*2。

■ 第 4-5 层: 卷积层, 池化层

卷积核数目为 64, 卷积核尺寸为 3*3, 激活函数为 relu

池化窗口尺寸: 2*2

■ 第 6 层: 卷积层

卷积核数目为 128, 卷积核尺寸为 3*3, 激活函数为 relu

■ 第 7 层: Flatten 层

连接卷积层与全连接层

■ 第 8 层: 全连接层

转化成一维的向量, 输出维度为 128, 激活函数为 relu

■ 第 9 层: 输出层

输出预期结果，参数为花卉类别数：6

(2) 模型训练

优化器选择的是“adam”，因为经过多次实验发现 adam 的效果最好；

损失函数选择的是“SparseCategoricalCrossentropy”：

```
# 编译模型以供训练
# TODO可选择其他损失函数
model.compile(optimizer='adam',
              loss=ke.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

epochs（训练回合）为 20 时得到的其中一次结果如下：

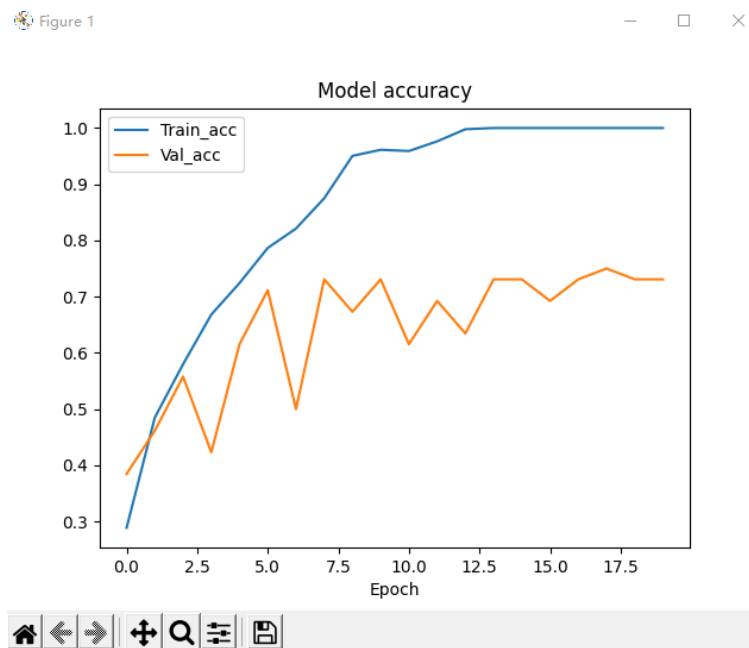
```
lab2_flower_classify x
15/15 - 2s - loss: 6.7693e-04 - accuracy: 1.0000 - val_loss: 1.3374 - val_accuracy: 0.7115
Epoch 17/20
15/15 - 2s - loss: 5.5241e-04 - accuracy: 1.0000 - val_loss: 1.3539 - val_accuracy: 0.6731
Epoch 18/20
15/15 - 2s - loss: 4.7675e-04 - accuracy: 1.0000 - val_loss: 1.3532 - val_accuracy: 0.6923
Epoch 19/20
15/15 - 2s - loss: 4.2002e-04 - accuracy: 1.0000 - val_loss: 1.3638 - val_accuracy: 0.7115
Epoch 20/20
15/15 - 2s - loss: 3.6990e-04 - accuracy: 1.0000 - val_loss: 1.3781 - val_accuracy: 0.7115
Model: "sequential"
```

```
lab2_flower_classify x
Trainable params: 7,309,830
Non-trainable params: 0
-----
2/2 - 0s - loss: 1.2972 - accuracy: 0.7115 - 72ms/epoch - 36ms/step
shape of data: (6, 100, 100, 3)
第 1 朵花预测:bee
第 2 朵花预测:blackberry
第 3 朵花预测:blanket
第 4 朵花预测:bougainvillea
第 5 朵花预测:bromelia
第 6 朵花预测:foxglove
进程已结束,退出代码0
```

可以看到 20 回合后，准确率达到 71%，损失值约为 1.3。

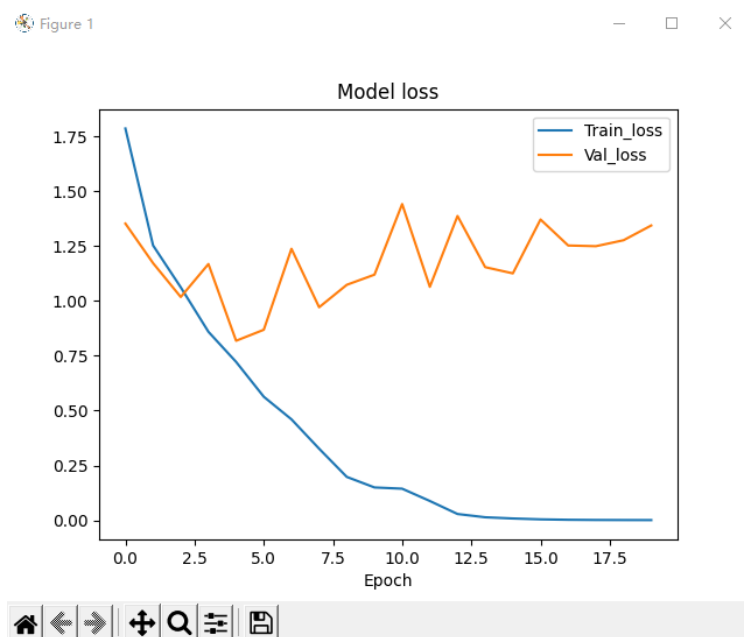
最后测试的六张图片全部正确，结果比较好。

Accuracy 曲线截图：



随着回合数增加，训练集准确率升高至 1，验证集总体趋势为上升，最终约为 70%。

Loss 曲线截图：



随着回合数增加，训练集丢失值降低至 0，验证集丢失值起伏不定，最终约为 1.3。

2.3.3. CNN 模型_MindSpore 框架 [190110128_林先一]

(1) 模型定义

- 模型：卷积神经网络 CNN
- 模型结构：

```
#构建模型
def construct(self, x):
    x = self.conv1(x)
    #print(x.shape)
    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)
    #print(x.shape)
    x = self.dropout(x)
    x = self.fc2(x)
    x = self.relu(x)
    x = self.dropout(x)
    x = self.fc3(x)
```

- 卷积部分，一共使用了 4 层卷积+池化操作，通过 CNN 网络隐层神经元的局部感知野，提取图像中特定细节的信息，通过池化层对信息进行进一步抽象和特征提取，减少参数的数量；RELU 激活层通过将特征映射到非线性空间中，为特征引入了非线性的信息，增强了模型的感受特征空间和表达能力。
 - Flatten 层对多层卷积后提取到的特征做展平操作，使得特征数据的规模形状能适配全连接层
 - 全连接层共 3 层，其中前两层全连接层后，紧跟着 RELU 与 DROPOUT 层，RELU 同前述部分作用相同，DROPOUT 层在模型训练时，随机的让网络中一些节点输出置 0，即本次训练过程中不参与前向传播，自然也不参与反向的权重更新，相当于随机的忽略掉了一部分特征，有效地减轻了过拟合的发生，增强了泛化和鲁棒性
- 具体网络层描述：

■ 1~3 卷积层，激活，池化层

卷积核数目为 32——尺寸为 5×5 ，步长 stride 为 1，边缘 padding 为 0；

激活函数为 relu；

池化层：池化核规模 2×2 ，步长 2

■ 4~6 层：卷积层，激活，池化层

卷积核数目为 64——尺寸为 5×5 ，步长 stride 为 1，边缘 padding 为 0；

激活以及池化层与前序一致

■ 7~10 层：卷积+池化

卷积核进一步提高到 128，但是缩减了尺寸为 3×3 ，步长及边缘 padding 保持不变；

池化层与前序一致

■ 第 11 层：Flatten 展平层

调整网络规模，为全连接层输入准备

■ 第 12~14 层：全连接层+激活/Dropout

全连接层线性分类，激活层为输出引入非线性特性，随后通过 Dropout 层对输出做随机灭活处理

■ 第 15 层：仅全连接层

对所有的特征输出做最后的线性输出，作为模型的最后输出，输出为分类类型对应的标签序号，在训练阶段，其输出作为损失函数以及权值更新的依据，在预测阶段，其输出即为预测结果。

(2) 模型训练

详细说明模型在训练集上的训练速度、收敛速度、收敛精度、超参数设置与调整、优化算法选择等，提供训练日志或 loss 曲线的截图。

优化器选择的是 Adam

损失函数选择的是 Softmax 交叉熵

基本参数都与教程范例中一致

随着 epoch 训练次数的逐步提高,模型在验证集上的损失函数指标几乎稳定在 0.2 左右

Mindspore 框架真的参考太少,官方文档也很烂,训练速度也很慢,有限时间内只针对了训练 epoch, batch, 学习率, 网络层的部署做了相应的调整,最终结果与官方教程代码基本一致甚至效果更差

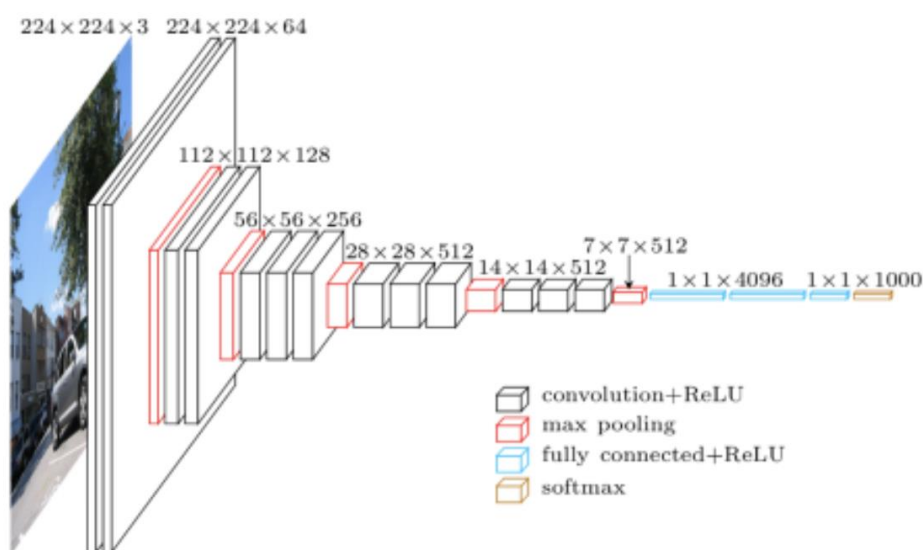
```
epoch: 10 step: 31, loss is 1.344336
epoch: 20 step: 31, loss is 1.0384951
epoch: 30 step: 31, loss is 1.3756701
epoch: 40 step: 31, loss is 0.8379159
epoch: 50 step: 31, loss is 0.65440804
epoch: 60 step: 31, loss is 0.8422977
epoch: 70 step: 31, loss is 0.3575681
epoch: 80 step: 31, loss is 0.5150201
epoch: 90 step: 31, loss is 0.43078065
epoch: 100 step: 31, loss is 0.3742515
epoch: 110 step: 31, loss is 0.2505471
epoch: 120 step: 31, loss is 0.31889352
epoch: 130 step: 31, loss is 0.1621328
epoch: 140 step: 31, loss is 0.45156235
epoch: 150 step: 31, loss is 0.38065234
epoch: 160 step: 31, loss is 0.5162813
epoch: 170 step: 31, loss is 0.11255997
epoch: 180 step: 31, loss is 0.09184313
epoch: 190 step: 31, loss is 0.34747216
epoch: 200 step: 31, loss is 0.2758978
epoch: 210 step: 31, loss is 0.25206363
epoch: 220 step: 31, loss is 0.17159666
epoch: 230 step: 31, loss is 0.3619469
epoch: 240 step: 31, loss is 0.262833
epoch: 250 step: 31, loss is 0.11514902
epoch: 260 step: 31, loss is 0.16668873
epoch: 270 step: 31, loss is 0.08247546
epoch: 280 step: 31, loss is 0.24402684
epoch: 290 step: 31, loss is 0.36856043
epoch: 300 step: 31, loss is 0.17280626
epoch: 310 step: 31, loss is 0.062829584
epoch: 320 step: 31, loss is 0.27224907
epoch: 330 step: 31, loss is 0.047582164
epoch: 340 step: 31, loss is 0.14305995
epoch: 350 step: 31, loss is 0.0806947
epoch: 360 step: 31, loss is 0.5212231
epoch: 370 step: 31, loss is 0.13224845
epoch: 380 step: 31, loss is 0.22475566
epoch: 390 step: 31, loss is 0.13458505
epoch: 400 step: 31, loss is 0.19325736
```

2.3.4. 利用 VGG16 网络架构进行的迁移学习 (by 谢俊安)

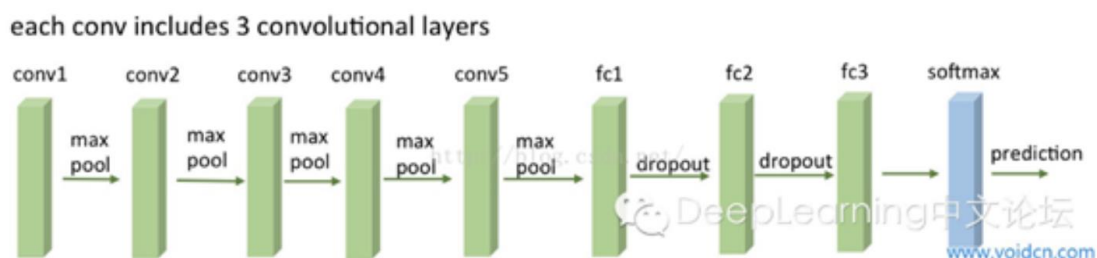
(1) 模型定义

本模块选用了 VGG16 网络架构和 ImageNet 预训练网络进行训练并与本小组自写的模型进行对比参照，VGG 卷积神经网络模型在 2014ImageNet 图像分类与定位挑战赛 ILSVRC-2014 中取得在分类任务第二，定位任务第一的优异成绩。

选用的变种是 VGG16，这是一个拥有 16 层的模型，模型结构图如下：



其中具体包括 5 个卷积层和 3 个全连接层，VGG16 的第 3、4、5 块：256、512、512 个 3×3 滤波器依次用来提取复杂的特征。本实验将顶层的 3 个全连接网络删除，并进行微调以适应本任务。模型的权重采用 ImageNet 数据集上预训练的权重。模型的默认输入尺寸是 224×224 ，本任务调整为 100×100 以在保持训练精度的情况下降低训练成本。



特别地，VGG 的优越性是通过依次采用多个 3×3 卷积，模仿出更大的感受野效果。模型定义如下：

```
# 实现VGG16模型
model_vgg16 = VGG16(weights='imagenet',
                      include_top=False,
                      input_shape=(100, 100, 3))
```

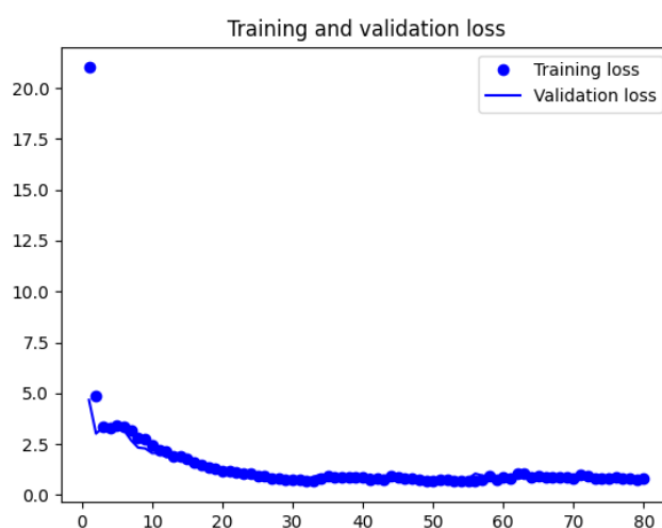
(2) 模型训练

a) 训练速度:

```
Total params: 17,107,718
Trainable params: 2,393,030
Non-trainable params: 14,714,688
-----
train time: 660.197685956955
```

由图可知，模型对精度为 100*100 的图片训练 80 个 epoch 共耗费 660s，训练速度较快。

b) 模型基本在 epochs=30 时已经收敛



c) 超参数:

- i. 最终选择：图片像素为 100*100、epochs=80、batch_size=200、优化器 Adam 学习率 learning_rate=0.01、添加 l2 正则化。

ii. 调整过程:

(1) 图片像素过低导致的精度过低:

```
3/3 - 1s - loss: 2.3126 - accuracy: 0.3252 - val_loss: 1.4804 - val_accuracy: 0.3846  
Epoch 29/30  
3/3 - 1s - loss: 2.2328 - accuracy: 0.3083 - val_loss: 1.4758 - val_accuracy: 0.4038  
Epoch 30/30  
3/3 - 1s - loss: 1.8669 - accuracy: 0.3155 - val_loss: 1.4720 - val_accuracy: 0.4038  
Model: "sequential"
```

分析: 如图显示的为 $32*32$ 像素的图片样本, 个人猜想原因是图片像素被压缩时, 物件本身的特征也被平坦化了, 导致模型的训练效果不佳。尝试将输入图片像素调整为 $224*224$ 。

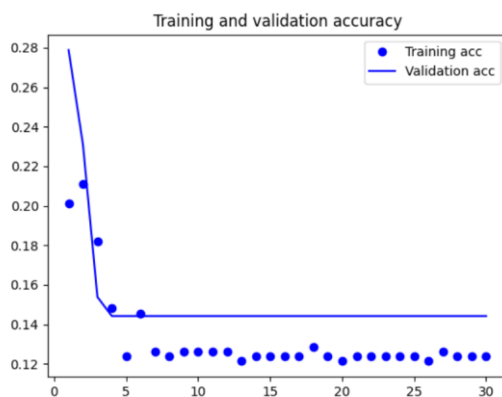
(2) 对原有模型的理解不通透

具体描述: 上述问题经调整图片像素之后有了小幅改进, 但结果仍不理想 (如下图)

```
3/3 - 48s - loss: 2.2014 - accuracy: 0.6044 - val_loss: 0.9232 - val_accuracy: 0.5769  
Epoch 27/30  
3/3 - 44s - loss: 1.4812 - accuracy: 0.6626 - val_loss: 1.0620 - val_accuracy: 0.7500  
Epoch 28/30  
3/3 - 46s - loss: 1.2458 - accuracy: 0.6553 - val_loss: 1.1111 - val_accuracy: 0.7885  
Epoch 29/30  
3/3 - 44s - loss: 1.1696 - accuracy: 0.6359 - val_loss: 1.1968 - val_accuracy: 0.6923  
Epoch 30/30  
3/3 - 47s - loss: 1.1329 - accuracy: 0.6092 - val_loss: 1.1159 - val_accuracy: 0.5865  
Model: "sequential"
```

分析原因: 没有理解使用的 VGG16 模型的特性, 原模型的最后全链接层使用的激活函数是 softmax 而不是 rula

(3) 越训练效果越差

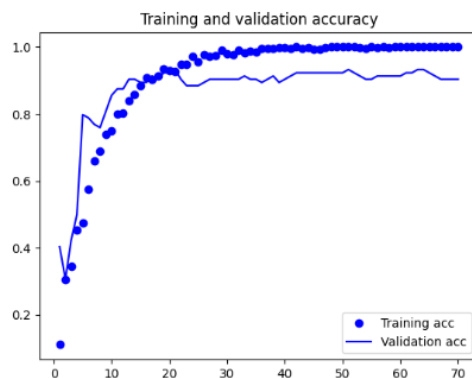


原因及解决办法: 样本像素太低, 花种的特征已经消失。提

高图片像素有很大提升。

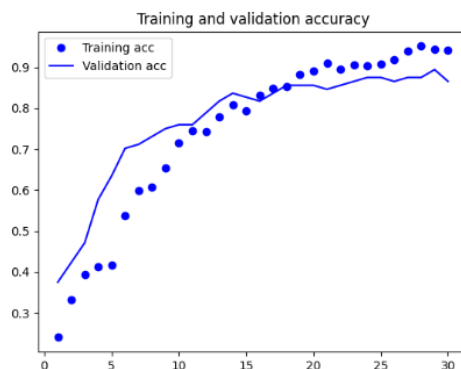
```
3/3 - 57s - loss: 1.4376 - accuracy: 0.5850 - val_loss: 1.2578 - val_accuracy: 0.7212  
Epoch 29/30  
3/3 - 61s - loss: 1.6149 - accuracy: 0.5850 - val_loss: 1.1422 - val_accuracy: 0.7019  
Epoch 30/30  
3/3 - 64s - loss: 1.6707 - accuracy: 0.5874 - val_loss: 1.0456 - val_accuracy: 0.7019  
Model: "sequential"
```

(4) 训练次数过高



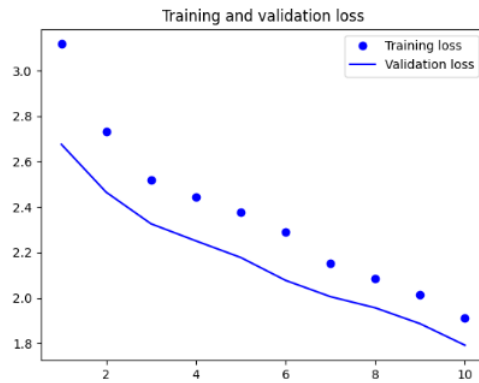
分析：实际上但 `epochs==30` 时模型精度已经收敛，趋于稳定，没必要再抬高训练次数，反而造成资源浪费。

(5) 出现模型过拟合的现象



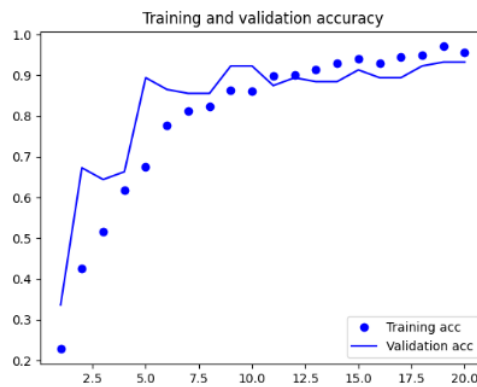
分析原因：学习率过低导致的模型泛化水平差，尝试提升模型学习率 `0.00001->0.001`。

(6) 模型未收敛



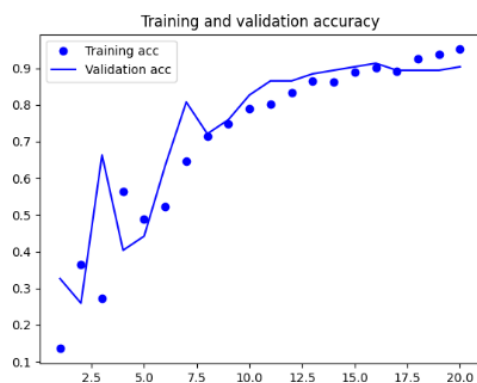
解决办法：提高学习率/增加训练次数

(7) 模型收敛很不稳定



分析原因&解决办法：增加 **batchsize** 为原来的 **N** 倍时，按照线性缩放规则，学习率应该增加为原来的 **N**。所以应该提升 **batchsize**。

(8) 模型仍旧过拟合



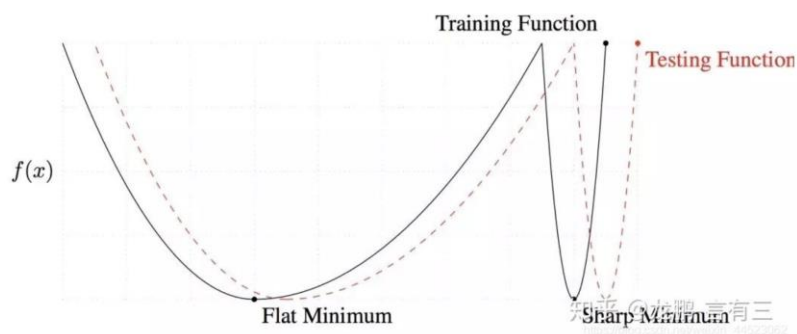
分析原因：

在一定范围内，一般来说 **Batch_Size** 越大，其确定的下

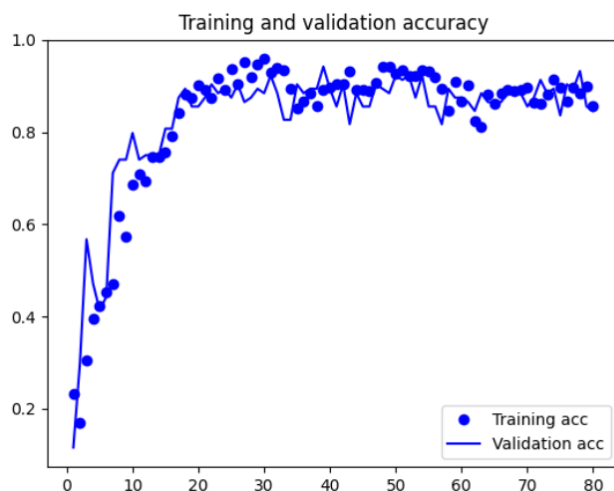
降方向越准，引起训练震荡越小。但是本次训练中 `batchsize` 调整到 2000 显然有些超出了“一定范围”，而 `batchsize` 过大会导致模型收敛到 sharp minimum，比 `batchsize` 小的模型泛化能力更差。

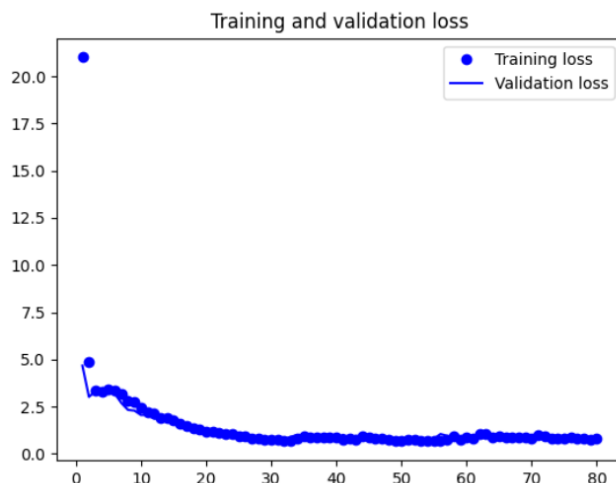
模型训练过度。

解决办法：尝试调整 `batchsize` 为 200；添加 L2 正则化防止过拟合；进行 dropout 率为 0.5 的 dropout 防止过拟合。



(9) 拟合效果良好





2.4. 实验结果

阐述模型测试与评估结果，包括各种模型在本地预测的效果截图与说明、[ModelArts](#) 在线部署的预测效果截图与说明。

190110121-杨航：

本地预测效果截图：

LeNet5：

```
Epoch: 24 | train loss: 0.7796 | test accuracy: 0.58
第25次epoch:
Epoch: 25 | train loss: 0.7656 | test accuracy: 0.59
第26次epoch:
Epoch: 26 | train loss: 0.7525 | test accuracy: 0.57
第27次epoch:
Epoch: 27 | train loss: 0.7405 | test accuracy: 0.59
第28次epoch:
Epoch: 28 | train loss: 0.7298 | test accuracy: 0.56
第29次epoch:
Epoch: 29 | train loss: 0.7203 | test accuracy: 0.56
y_pred [5 2 5 4 3]
y_val tensor([5, 3, 0, 4, 5])
shape of data: torch.Size([6, 3, 32, 32])
6
第 1 朵花预测:bee
第 2 朵花预测:bougainvillea
第 3 朵花预测:blanket
第 4 朵花预测:bougainvillea
第 5 朵花预测:bougainvillea
第 6 朵花预测:foxglove
```

最后的 loss 还较大，可能是学习率较小或者训练回合不够的原因，导致结果表现不佳。

VGG-16 简化版：

更改 seed 值对数据集进行了三次不同的划分，得到了三个结果，分别如下。

```

第24次epoch:
Epoch: 24 | train loss: 0.1339 | test accuracy: 0.81
第25次epoch:
Epoch: 25 | train loss: 0.1067 | test accuracy: 0.78
第26次epoch:
Epoch: 26 | train loss: 0.0957 | test accuracy: 0.78
第27次epoch:
Epoch: 27 | train loss: 0.0705 | test accuracy: 0.76
第28次epoch:
Epoch: 28 | train loss: 0.0809 | test accuracy: 0.78
第29次epoch:
Epoch: 29 | train loss: 0.0565 | test accuracy: 0.76
range(1, 31)
y_pred [5 3 5 4 5]
y_val tensor([5, 3, 0, 4, 5])
第 1 朵花预测:bee
第 2 朵花预测:blackberry
第 3 朵花预测:blanket
第 4 朵花预测:bee
第 5 朵花预测:bromelia
第 6 朵花预测:foxglove

Epoch: 26 | train loss: 0.0941 | test accuracy: 0.80
第27次epoch:
Epoch: 27 | train loss: 0.0719 | test accuracy: 0.81
第28次epoch:
Epoch: 28 | train loss: 0.0610 | test accuracy: 0.81
第29次epoch:
Epoch: 29 | train loss: 0.0484 | test accuracy: 0.81
range(1, 31)
range(1, 31)
y_pred [0 2 5 3 3]
y_val tensor([0, 2, 5, 3, 3])

epoch: 27 | train loss: 0.0925 | test accuracy: 0.78
第28次epoch:
Epoch: 28 | train loss: 0.0982 | test accuracy: 0.75
第29次epoch:
Epoch: 29 | train loss: 0.0709 | test accuracy: 0.77
range(1, 31)
range(1, 31)
y_pred [3 3 2 3 5]
y_val tensor([3, 5, 2, 3, 5])
reading the images:../TestImages/test1.jpg
reading the images:../TestImages/test2.jpg
reading the images:../TestImages/test3.jpg
reading the images:../TestImages/test4.jpg
reading the images:../TestImages/test5.jpg
reading the images:../TestImages/test6.jpg
第 1 朵花预测:bee
第 2 朵花预测:blackberry
第 3 朵花预测:blanket
第 4 朵花预测:bee
第 5 朵花预测:bromelia
第 6 朵花预测:foxglove

```

将几次测试结果取平均值得到该模型在训练集上的表现。

平均准确率：78%

ModelArts 上训练结果：

（epoch 为 60，网络模型也比本地的更大一些）

```

388 Epoch: 55 | train loss: 0.0632 | test accuracy: 0.80
389 第56次epoch:
390 Epoch: 56 | train loss: 0.0569 | test accuracy: 0.79
391 第57次epoch:
392 Epoch: 57 | train loss: 0.1602 | test accuracy: 0.76
393 第58次epoch:
394 Epoch: 58 | train loss: 0.0652 | test accuracy: 0.80
395 第59次epoch:
396 Epoch: 59 | train loss: 0.0646 | test accuracy: 0.80

```

虽然最后在训练集上的 loss 较小，但在测试集上的表现并不是很好。可能是由于网络层数不够，对特征的提取不够充分。

190110130-向柳媛：

1) 本地预测效果截图（tensorflow）：

其中两次结果：

a)

```
lab2_flower_classify x
15/15 - 2s - loss: 6.7693e-04 - accuracy: 1.0000 - val_loss: 1.3374 - val_accuracy: 0.7115
Epoch 17/20
15/15 - 2s - loss: 5.5241e-04 - accuracy: 1.0000 - val_loss: 1.3539 - val_accuracy: 0.6731
Epoch 18/20
15/15 - 2s - loss: 4.7675e-04 - accuracy: 1.0000 - val_loss: 1.3532 - val_accuracy: 0.6923
Epoch 19/20
15/15 - 2s - loss: 4.2002e-04 - accuracy: 1.0000 - val_loss: 1.3638 - val_accuracy: 0.7115
Epoch 20/20
15/15 - 2s - loss: 3.6990e-04 - accuracy: 1.0000 - val_loss: 1.3781 - val_accuracy: 0.7115
Model: "sequential"
```

```
lab2_flower_classify x
Trainable params: 7,309,830
Non-trainable params: 0
-----
2/2 - 0s - loss: 1.2972 - accuracy: 0.7115 - 72ms/epoch - 36ms/step
shape of data: (6, 100, 100, 3)
第 1 朵花预测:bee
第 2 朵花预测:blackberry
第 3 朵花预测:blanket
第 4 朵花预测:bougainvillea
第 5 朵花预测:bromelia
第 6 朵花预测:foxglove
进程已结束,退出代码0
```

b)

```
lab2_flower_classify x
Epoch 16/20
15/15 - 2s - loss: 0.2304 - accuracy: 0.9397 - val_loss: 1.5970 - val_accuracy: 0.6154 -
Epoch 17/20
15/15 - 2s - loss: 0.1149 - accuracy: 0.9655 - val_loss: 1.5209 - val_accuracy: 0.6923 -
Epoch 18/20
15/15 - 2s - loss: 0.0234 - accuracy: 0.9978 - val_loss: 1.2961 - val_accuracy: 0.6538 -
Epoch 19/20
15/15 - 2s - loss: 0.0114 - accuracy: 1.0000 - val_loss: 1.0253 - val_accuracy: 0.7115 -
Epoch 20/20
15/15 - 2s - loss: 0.0050 - accuracy: 1.0000 - val_loss: 1.2226 - val_accuracy: 0.7308 -
Model: "sequential"
```

```
lab2_flower_classify x
-----
2/2 - 0s - loss: 1.2226 - accuracy: 0.7308 - 75ms/epoch - 37ms/step
shape of data: (6, 100, 100, 3)
第 1 朵花预测:bee
第 2 朵花预测:blackberry
第 3 朵花预测:blanket
第 4 朵花预测:bougainvillea
第 5 朵花预测:bromelia
第 6 朵花预测:foxglove
进程已结束,退出代码0
```

平均准确率达到 70%左右, loss 值 1.3 左右

2) ModelArts 上训练结果 (epoch 为 40):



最后的 accuracy 准确率有所提高，但是 loss 损失值表现不佳。

190110115-谢俊安

1. 本地预测效果:

```
3/3 - 9s - loss: 0.8010 - accuracy: 0.8835 - val_loss: 0.7098 - val_accuracy: 0.9327
Epoch 79/80
3/3 - 9s - loss: 0.7396 - accuracy: 0.9005 - val_loss: 0.8326 - val_accuracy: 0.8558
Epoch 80/80
3/3 - 9s - loss: 0.8370 - accuracy: 0.8568 - val_loss: 0.8826 - val_accuracy: 0.8462
Model: \"sequential\"
```

本地预测结果	正确应为
第 1 朵花预测:foxglove	▪ foxglove
第 2 朵花预测:bromelia	▪ bromelia
第 3 朵花预测:bougainvillea	▪ bougainvillea
第 4 朵花预测:bromelia	▪ bee
第 5 朵花预测:blanket	▪ blanket
第 6 朵花预测:blackberry	▪ blackberry

预测结果显示错误的是 bee 和 bromelia 花种，初步估计可能是由于训练集太小导致的模型特征提取不够充分，预测效果不太明显。

2. ModelArts 在线部署的预测结果:

```

shape of data: (6, 100, 100, 3)
第 1 朵花预测:bee
第 2 朵花预测:bee
第 3 朵花预测:bee
第 4 朵花预测:bee
第 5 朵花预测:bee
第 6 朵花预测:bee

```

可以看到效果非常差，原因是因为 ModelArts 平台不能访问外网的数据，因此无法在调用模型的时候加载 imagenet 权重，因此训练效果比本地差。

[190110128-林先一-MindSpore:]

没有进行本地部署的训练和预测，尝试过 MacOS 以及 Windows，模型的在 MacOS 端系统版本下没有支持版本，在 Windows 下仅支持 CPU，且训练过程出现了无法解决的报错

故只在本地进行编程以及代码正确性验证，训练为 ModelArts 平台完成，最终在本地利用训练得到的模型文件进行预测：

- ModelArts 平台训练结果：最终的 accuracy 在 0.96 左右[这里的数据集和验证集均来自于已做扩充处理的源数据集]

```

260 epoch: 360 step: 31, loss is 0.5212231
261 epoch: 370 step: 31, loss is 0.13224845
262 epoch: 380 step: 31, loss is 0.22475566
263 epoch: 390 step: 31, loss is 0.13458505
264 epoch: 400 step: 31, loss is 0.19325736
265 [WARNING] ME(316:140329314461504,MainProcess):2022-04-28-22:22:20.7
    [mindspore/train/model.py:774] CPU cannot support dataset sink mode
    evaluating process will be performed with dataset non-sink mode.
266 {'acc': 0.9508928571428571}
267 [WARNING] ME(316:140329314461504,MainProcess):2022-04-28-22:22:22.4
    [mindspore/train/model.py:774] CPU cannot support dataset sink mode
    evaluating process will be performed with dataset non-sink mode.
268 {'acc': 0.9596774193548387}
269 [2022-04-28T22:22:32+08:00][ModelArts Service Log]exiting...
270 [2022-04-28T22:22:32+08:00][ModelArts Service Log]exit with 0

```

- 通过训练得到的模型文件，在本地预测 TestImage 中 6 张图片得到的

结果完全正确

```

xe c:/Users/Charles/Desktop/mindspore/flowers_classification.py
[WARNING] ME(12492:19540,MainProcess):2022-04-29-00:17:46.877.492 [mindspore\dataset\engine\datasets.py:1
079] Dataset is shuffled before split.
训练数据集数量: 384
测试数据集数量: 96
通道数/图像长/宽: (3, 100, 100)
一张图像的标签样式: 5
第0个sample预测结果: bromelia 真实结果: bromelia
第1个sample预测结果: bougainvillea 真实结果: bougainvillea
第2个sample预测结果: bee 真实结果: bee
第3个sample预测结果: blanket 真实结果: blanket
第4个sample预测结果: foxglove 真实结果: foxglove
第5个sample预测结果: blackberry 真实结果: blackberry

```

- 结合验证机以及小样本测试集的结果, 可得整体模型的预测效果较好

2.5. 深度学习框架对比

1. 本地用到的几种深度学习框架的优缺点, 在相同模型结构、超参数等情况下, 不同框架在开发效率、训练速度及精度、推理速度及精度等方面的区别。

190110115-谢俊安

框架	优点	缺点	开发效率	训练速度
TensorFlow	支持其他机器学习算法, 支持强化学习, 拥有更加完善的 AI 体系; 适合大规模部署, 跨平台和嵌入式部署;	版本分裂严重, 不同版本几乎不可通用; 安装环境要考虑 cuda/cudn 版本; 不适合可变长度序列, 缺少符号循环功能。	中等	较快
PyTorch	熟悉 Python 和常见的深度学习概念会非常容易上手; 安装不用考虑 cuda/cudnn 版本	做大规模分布式并行时需要定制性修改大量的代码;	快	中等
MindSpore	支持昇腾芯片、GPU、CPU 等异构平台; 昇腾+MindSpore 组合下性能遥遥领先; 接口易用性能有保障;	社区比较小; 文档友好型差; API 接口混乱;	慢	中等

	在分布式集群和华为全栈下性能提升很大；			
--	---------------------	--	--	--

2. 云端不同框架的区别。

190110115-谢俊安

框架	在线部署	本地运行
PyTorch	训练时必须采用 state_dict 的保存方式	-
TensorFlow	使用 model.save 接口保存模型	-
MindSpore	在 ModelArts 上部署更复杂一些； 只支持昇腾 310 芯片。	-

3. 本地、云端同一框架之间的区别

190110121-杨航

框架	优点	缺点
Pytorch	上手简单，代码简洁；定义网络方便；能够随时修改网络结构，方便调整参数，动态观察模型效果。	没有自带的可视化工具，动态编程导致模型效率较低，训练速度慢。
ModelArts	训练速度快，无需占用本地资源与算力，搭建好算法后能够对多个数据集进行多次训练；观察训练日志更加方便	对代码的修改不如本地方便，需要在本地修改后重新新建算法；可能会有版本号与本地不匹配的问题

190110130-向柳媛

框架	优点	缺点
Tensorflow	理解简单，网上学习资料很多，框架已经搭建好，绘制 accuracy 和 loss 图非常方便。	训练速度慢
ModelArts	无需搭建环境，只需上传数据集和代码即可运行，十分方便；训练日志清楚记录的历史训练，便于查找	每次修改完代码之后需要重新上传再重建、训练，稍显麻烦

190110128-林先一 [3. MindSpore 框架——本地&云端]

框架	优点	缺点
MindSpore	集成封装了很多功能库，一次安装成功后，可以使用其集成的许多工具完成具体功能，无需繁琐的去引入定义其他众多的包，整体代码简洁； “辩证地”提升了易用性和开发门槛； 定义网络结构时清晰明了；	官方文档以及官方的问题解答教程较为敷衍，同时网路上的参考或者教程资源相对还是比较匮乏，出错以后只能到论坛中搜索或者提问； 本地安装无法一步到位，需要到官网按固定格式语法安装； 训练速度相较于其他两模型来说降低了很多； 框架整体设计的初衷是尽可能的封装为用户带来便利，但是过多或者过高层次的封装，降低了很多可用性，同时也带来了相对较多的学习成本（从其他框架迁移过来时）
MS on ModelArts	在华为云使用 MindSpore 框架可以申请使用自家爸爸的昇腾 Ai 处理器资源池作为硬件支持；	运行出错或者训练失败后需要修改代码再重新上传，代码修改繁琐； 操作流程繁琐；

	平台训练速度快，存储资源容量大，摆脱本机的硬件的限制，算法配置成功后，可以对多个数据集进行多次训练；同时还有部署等功能	不具备调试的功能； 数据读取及存储繁琐； 虽然不占用本地但是要钱的.....
--	---	--

190110115-谢俊安 [1. TensorFlow 框架——本地&云端]

框架	优点	缺点
TensorFlow	每次训练时从外网加载预训练的模型，利用 imagenet 权重能得到较好的效果； 上手简单，网上开源资料多。	训练速度相比较慢。
ModelArts	不需按照麻烦的 cuda 就可以用 GPU 跑程序，速度快； 不占本地的空间；	要钱； 云端上传文件有 100 的上限限制，当上传的数据集较大的时候很不方便； 不可访问外网的数据，因而需要自行下载数据；

3 总结

3.1. 问题及解决方法

190110121-杨航

问题：在进行网络搭建的时候，对应该搭建一个什么结构的网络比较迷茫。不知道卷积层、全连接层和池化层应该各自放在什么位置能够有比较好的表现。

解决方法：去查询近年得到广泛使用的模型的搭建方法及各层之间的连接，

通过在现有模型上增删网络层、修改参数以达到好的效果。

190110130-向柳媛

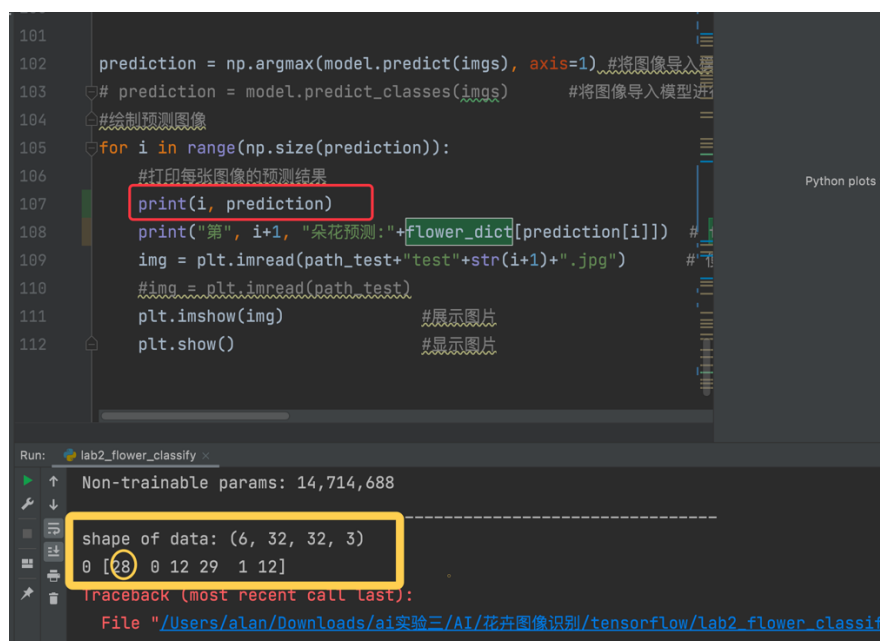
问题：首先安装环境就花了我不少时间。因为 tensorflow 安装错了版本，keras 一直用不了。改了其他版本后，keras 里面的包又用不了。上网查了很多资料，最后换了一种写法解决了。在实现模型结构时，对一些参数不知道怎么调整才能使结果更好，一长段时间怎么调都没效果。

解决方法：上网查找资料弄清 CNN 的原理，还有框架里各种函数里面参数的含义。之前一直没考虑修改 model.compile 的参数提高效果，后来尝试修改后发现其实损失函数的选择也会对结果有比较大的影响。

190110115-谢俊安

(1) 对深度学习神经网络的原理不熟悉。

进一步导致预测结果出现越界的异常（预测出本来不应该存在的标签，如下图）



```
101
102 prediction = np.argmax(model.predict(imgs), axis=1) #将图像导入模型
103 # prediction = model.predict_classes(imgs) #将图像导入模型进行预测
104 #绘制预测图像
105 for i in range(np.size(prediction)):
106     #打印每张图像的预测结果
107     print(i, prediction)
108     print("第", i+1, "朵花预测："+flower_dict[prediction[i]]) #
109     img = plt.imread(path_test+"test"+str(i+1)+".jpg") #
110     #img = plt.imread(path_test)
111     plt.imshow(img) #展示图片
112     plt.show() #展示图片
```

Run: lab2_flower_classify

Non-trainable params: 14,714,688

shape of data: (6, 32, 32, 3)

0 [28 0 12 29 1 12]

Traceback (most recent call last):

File "/Users/alan/Downloads/ai实验三/AI/花卉图像识别/tensorflow/lab2_flower_classif

解决办法：不要急于求成，先去理解基本原理和代码的逻辑思路，再去写代码。以上 BUG 是因为全链接层的书写错误，未写最后的输出特征导致的。

(2) 对深度学习的不熟悉，急功近利地想要完成实验。

分析及解决办法：

这是第一次正式地接触深度学习，最开始对于卷积层、全连接层、池化层等概念一直处于一知半解的状态，对于 `batchsize`、`lr`、激活函数等对模型能力的影响更是完全不知道。并且对个人任务预估的目标太高（完成三个平台成熟模型的搭建和对比）以至于项目初期没有沉下心去理解模型的原理和内涵，只是盲目地调整参数查看训练结果。

此外遇到的困难还包括：刚入手深度学习，对于如何选型、成熟的优秀的模型是哪些、如何找等问题很茫然，没有框架的意识，导致前期浪费了很多时间。

最后发现这样像无头苍蝇一样乱撞乱碰运气的测试方法不仅杯水车薪，而且对于自身学习并没有好处。于是终于沉下心选择学习 `tensorflow` 平台上深度学习开发。

190110128-林先一-MindSpore:

首先，`mindspore` 无论是 `pip` 安装或者是直接包编译安装都及其麻烦，一是对不同平台的适配不够，二是过程繁琐；安装过程官方文档也近乎敷衍，几乎没有给出任何对应报错的解决方法，网络资源也相应的很匮乏

其次，可供参考学习的代码少之又少，几乎只有官方的那个依赖于旧版本 `Mindspore` 编写的教程，没有博客论文引导，只能通过晦涩难懂且稍显简陋的官方文档自己摸索，加上实验还需要进行代码的移植，整个过程中太多有坑的地方，只能跟同为 `mindspore` 的同学相依为命，有些共同都无法结局的问题只好向源码低头。

3.2. 实验的启发、总结及建议

190110121-杨航

在这次实验之前已经尝试过很多次系统地学习深度学习的知识，但最多也只是停留在调取已经成熟的库、模型上，虽然学习了网络搭建的皮毛知识，但是逐层定义地搭建网络还是第一次。在之前调取像 `VGG16`、`VGG19` 等这样的网络模型

的时候，由于网络内部结构已经确定，最多需要调整的参数只有 `batch` 大小和学习速率，但如果要有较好的训练效果，模型本身才是更重要的。要对模型有优化需要对内部有更深入的了解，所以个人认为这一次实验是很有意义的。

190110130-向柳媛

这次实验借机让我学习了卷积神经网络的原理和搭建，感慨神经网络的精妙。在调整代码时，遇到瓶颈要积极上网查找资料，思考原因，而不是盲目地一个个随便尝试。

190110115-谢俊安

前期盲目地乱套模型+调试参数浪费了大量的时间，但一直处于一知半解的状态。最后去阅读了所选用模型的源码，阅读了相关的论文。对模型本身有了一定的认知，再查看与理解其他项目的调用别的模型方式，才对整个迁移学习的过程有了一定的了解。

做完之后才觉得项目的难度其实也没有很高，需要操作的代码量也没有很大，但是最开始毫不了解的时候确实觉得非常难以下手。哪怕是调用模型的任务，因为网上其他项目的深度学习任务代码框架和老师给的不一样，所以想要真正理解并且完成此次任务还是需要去用心理解透彻之后，再回到框架自己认真书写。实际阅读并理解了一些 VGG16 的源码之后，感觉其实自己的工作量也并不小。

这门课是深度学习入门的第一堂课，本次实验之后学到了很多很多！

190110128-林先一：

虽然最终模型的准确性指标和预测结果都很满意，但是说实话从实验中学学习到的仅仅是 CNN 网络框架以及一些数据预处理的部分，其他部分丝毫没有时间深入了解。过程中 Mindspore 相关报错以及 Mindspore 移植到 ModelArts，又或者是最最终的本地预测，每一步都是无比困难，浪费掉了几乎 80%的时间，最后可能还无法解决。同学光做 Mindspore 的代码编写和预测就已经是困难重重，自己因为源码的可操作的地方少之又少，想尝试一下数据集处理，然后到了移植和本地

预测的阶段，又是另一个深渊。两三周的时间下来，除了前期对网络以及图像分类相关的了解感觉有所收获，后面长线的处理各类异常或者人类未解之谜，真的花费了大量的时间，还是一些无关紧要的问题，至少我的部分我认为是折磨大于收获，过程中是对肉体和精神，包括对浏览器搜索引擎的摧残。

希望以后没有 MindSpore 这部分，或者实验指导书以及相关资源能够支持的情况下可以尝试，或许对于一些长期在机器学习人工智能方面有实践的人来说，可能是一种尝试国货或者拓展技能的过程，但是对于想要入门或者进一步构建人工智能相关知识框架的我们来说，过程中的收获少之又少，相比于 Pytorch 或者 Tensorflow 这类有大量对于的文献模型的复现或者时间可供学习的框架来说，MindSpore 感觉对于初级阶段来说太没有必要了，更何况，实验还是进行在多门课都在结课且有其他实验的情况下，同时还有返校等一堆繁琐的事情，还有 ModelArts 这个及其陌生的云平台需要部署了解，还是在有限的两三周的时间内，MindSpore 更没有必要了感觉。

4 参考文献

[1] 理解 Dropout

<https://blog.csdn.net/stdcoutzyx/article/details/49022443>

[2] 深度学习中 Batch size 对训练效果的影响

<https://blog.csdn.net/u010417185/article/details/79533539>

[3] nn.Conv2d 和其中的 padding 策略

<https://blog.csdn.net/q11d111/article/details/82665265>

190110115-谢俊安:

[1] VGG in Tensorflow (<https://www.cs.toronto.edu/~frossard/post/vgg16/>)

[2] Usage of VGG16 (<https://github.com/machrisaa/tensorflow-vgg>)

[3] Very Deep Convolutional NetWorks for Large-Scale Image Recognition

([https://kns.cnki.net/kns8/Detail?sfield=fn&QueryID=4&CurRec=1&recid=&FileName=XTYY202109045&DbName=CJFDLAST2021&DbCode=CJFD&yx=&pr=&URLID=\)](https://kns.cnki.net/kns8/Detail?sfield=fn&QueryID=4&CurRec=1&recid=&FileName=XTYY202109045&DbName=CJFDLAST2021&DbCode=CJFD&yx=&pr=&URLID=))