哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2022 年春季

课程名称：人工智能

实验名称：深度学习实现花卉识别

实验性质：课内实验

实验时间：2022.4.16 地点：/

学生专业：计算机科学与技术

学生班级：1901105

学生学号：190110509

学生姓名：王铭

同组成员：肖力炜、郑羿恺、陈柏江

授课教师：郑海刚

报告成绩：

一、实验内容

本实验是以小组的形式，利用当前流行的三种深度学习框架，对给定的花卉图片数据集进行训练，构造将输入图片分为六种类别的神经网络，并在给定的测试集上进行预测分类。在小组作业中，我主要负责的是基于 pytorch 框架在本地实现神经网络并完成分类任务，完成 pytorch 在云端训练和本地预测以及 TensorFlow 的 ModelArts 在线部署。

二、算法与模型介绍

本模型采用了四层卷积+池化后连接三层全连接层的神经网络结构，完整的模型结构图如图 2.1 所示。

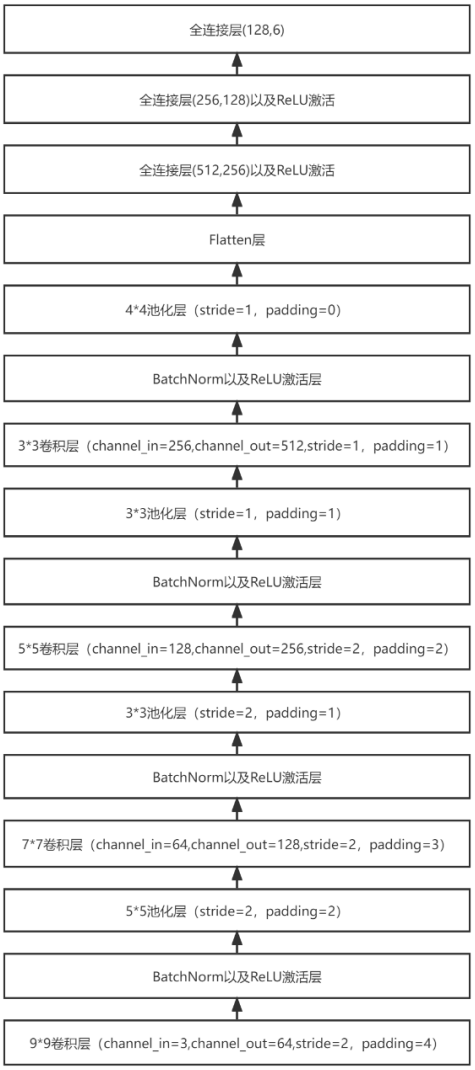


图 2.1

输入图片为 RGB 三个通道，长、宽均为 128，第一个卷积层所用卷积核为 9，步

长为 2，填充为 4，可以根据公式输出维度=$\frac{\text{（输入维度-kernel\_size+2*padding）}}{stride}+1$

计算出输出维度为 64*64，输入通道数为 3，输出通道数为 64，即用了 64 个卷积核，则经过第一层卷积后输出维度为(64,64,64)。然后经过 BatchNorm 层进行规范化，选择 ReLU 激活函数对结果进行激活。经过第一个池化层，采用最大池化，由步长和填充大小可以算出，池化后的输出维度为(64,32,32)。同理经过第二、三、四层卷积+池化后，维度分别为(128,8,8)、(256,4,4)、(512,1,1)。卷积池化操作后，经过 Flatten 展平层可以将输入转化为长度为 512 的一维数据，之后连接 3 层全连接层便可以进行分类。可以发现，相较于将输入图片直接连接全连接层进行计算，通过卷积池化后，在保存了输入图片的特征的条件下，大大降低了输入数据的维度，从而能有效降低全连接层模型的参数总数。

本模型选择的损失函数为较为常见的交叉熵损失函数 CrossEntropyLoss，相较于均方损失函数，采用交叉熵损失函数可以避免损失值较小时梯度较小的问题。

本模型选择的优化器为 Adam 优化器即采用自适应梯度来更新神经网络的参数，相较于随机梯度下降 SGD 算法来说，Adam 的实现更加简单高效，且能够自动调整学习率，便于训练时更好地收敛，因而选用 Adam 优化器。

三、算法实现

3.1 实验环境

操作系统：Windows10，CUDA 版本：10.2

编程语言：采用 Python3.7+pytorch1.9.0，训练过程在 GPU 上完成

3.2 数据集处理

数据读取部分，采用 openCV，从训练集中读入的图片，并将读入的图片统一缩放为 128*128，按照读入图片的种类顺序将标签与类别序号对应，并将图片和标签转换成 numpy 的 ndarray，其中图片采用浮点数据类型，标签采用整数型。代码实现如图 3.1 所示：

```
flower_dict = {}
def read_img(path, flower):
    cate=[path+x for x in os.listdir(path) if os.path.isdir(path+x)]
    imgs=[]
    labels=[]
    for idx,folder in enumerate(cate):
        flower[idx] = folder.split('/')[-1]
        for im in glob.glob(folder+'/*.jpg'):
            # print('reading the images:%s'%(im))
            img = cv2.imread(im)
            img = cv2.resize(img,(w,h))
            imgs.append(img)
            labels.append(idx)
    return np.asarray(imgs, np.float32), np.asarray(labels, np.int32)
```

图 3.1

按照常用的 8：2 的比例，将读入的训练图片划分成训练集和验证集，并将图片进行标准化处理，由于输入图片的维度为 128*128*3，为使用 pytorch，需要将其转变为 3*128*128，同时将数据转换成 torch 的类型，实现代码如图 3.2 所示：

```
(x_train, x_val, y_train, y_val) = train_test_split(data, label, test_size=0.20, random_state=seed)
x_train = x_train / 255
x_val = x_val / 255
x_train = x_train.transpose(0, 3, 1, 2)
x_val = x_val.transpose(0, 3, 1, 2)
train_iter = torch.from_numpy(x_train)
train_iter_y = torch.from_numpy(y_train)
test_iter = torch.from_numpy(x_val)
test_iter_y = torch.from_numpy(y_val)
train_iter_y = train_iter_y.to(torch.int64)
test_iter_y = test_iter_y.to(torch.int64)
```

图 3.2

3.3 模型构建与训练

模型定义代码如图 3.3 所示：

```
model = nn.Sequential(
    # 卷积+池化层一
    nn.Conv2d(3, 64, kernel_size=9, stride=2, padding=4),# 64*64
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=5, stride=2, padding=2),# 32*32
    # 卷积+池化层二
    nn.Conv2d(64, 128, kernel_size=7, stride=2, padding=3),  # 16*16
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2, padding=1),  # 8*8
    # 卷积+池化层三
    nn.Conv2d(128, 256, kernel_size=5, stride=2, padding=2),  # 4*4
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=1, padding=1),  # 4*4
    # 卷积+池化层四
    nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),  # 4*4
    nn.BatchNorm2d(512),
    nn.ReLU(),
```

```
    nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1),  # 4*4
    nn.BatchNorm2d(512),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=4),  # 1*1

    FlattenLayer(),
    nn.Linear(512, 256),
    nn.ReLU(),
    nn.Linear(256, 128),
    nn.ReLU(),
    nn.Linear(128, 6)
)
```

图 3.3

学习率 lr=0.0001，训练次数 nums_epoch=16 以及 Adam 优化器的设置代码如图 3.4 所示：

```
lr, num_epochs = 0.0001, 16
batch_size = 64
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
train(model, train_iter, train_iter_y, test_iter, test_iter_y, batch_size, optimizer, device, num_epochs)
```

图 3.4

采用 pytorch 框架的 backword 和 step 可以完成参数的训练更新，采用的交叉熵损失函数以及训练过程的具体实现代码如图 3.5 所示：

```
def train(net, train_iter, train_iter_y, test_iter, test_iter_y, batch_size, optimizer, device, num_epoch
    net = net.to(device)
    print("training on", device)
    loss = torch.nn.CrossEntropyLoss()# 定义损失函数为交叉熵
    batch_count = 0
    for epoch in range(num_epochs):
        train_l_sum, train_acc_sum, n, start = 0.0, 0.0, 0, time.time()
        for i in range(train_iter.shape[0] // batch_size + 1):
            X = train_iter[i * batch_size: (i + 1) * batch_size]
            y = train_iter_y[i * batch_size: (i + 1) * batch_size]
            X = X.to(device)
            y = y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
            train_l_sum += l.cpu().item()
            train_acc_sum += (y_hat.argmax(dim=1) == y).sum().cpu().item()
            n += y.shape[0]
            batch_count += 1
            n += y.shape[0]
            batch_count += 1
        test_acc = evaluate(test_iter, test_iter_y, net)
        print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f, time %.1f sec'
              % (epoch + 1, train_l_sum / batch_count, train_acc_sum / n, test_acc, time.time() - start))
```

图 3.5

3.4 实验结果

训练采用分类成功精确度指标判断模型好坏，训练过程的输出如图 3.6 所示：

```
epoch 1, loss 1.5524, train acc 0.398, test acc 0.240, time 2.2 sec
epoch 2, loss 0.6134, train acc 0.575, test acc 0.240, time 1.2 sec
epoch 3, loss 0.3336, train acc 0.689, test acc 0.240, time 1.2 sec
epoch 4, loss 0.1920, train acc 0.837, test acc 0.240, time 1.2 sec
epoch 5, loss 0.1072, train acc 0.930, test acc 0.250, time 1.2 sec
epoch 6, loss 0.0552, train acc 0.973, test acc 0.625, time 1.2 sec
epoch 7, loss 0.0255, train acc 0.995, test acc 0.788, time 1.2 sec
epoch 8, loss 0.0111, train acc 1.000, test acc 0.673, time 1.2 sec
epoch 9, loss 0.0053, train acc 1.000, test acc 0.875, time 1.2 sec
epoch 10, loss 0.0026, train acc 1.000, test acc 0.856, time 1.2 sec
epoch 11, loss 0.0015, train acc 1.000, test acc 0.894, time 1.2 sec
epoch 12, loss 0.0009, train acc 1.000, test acc 0.846, time 1.2 sec
epoch 13, loss 0.0006, train acc 1.000, test acc 0.846, time 1.2 sec
epoch 14, loss 0.0004, train acc 1.000, test acc 0.875, time 1.2 sec
epoch 15, loss 0.0003, train acc 1.000, test acc 0.875, time 1.1 sec
epoch 16, loss 0.0003, train acc 1.000, test acc 0.885, time 1.2 sec
```

图 3.6

可以将训练过程的损失值 loss，训练集的精确度 train_acc 以及验证集上的精确度 test_acc 画图表示：

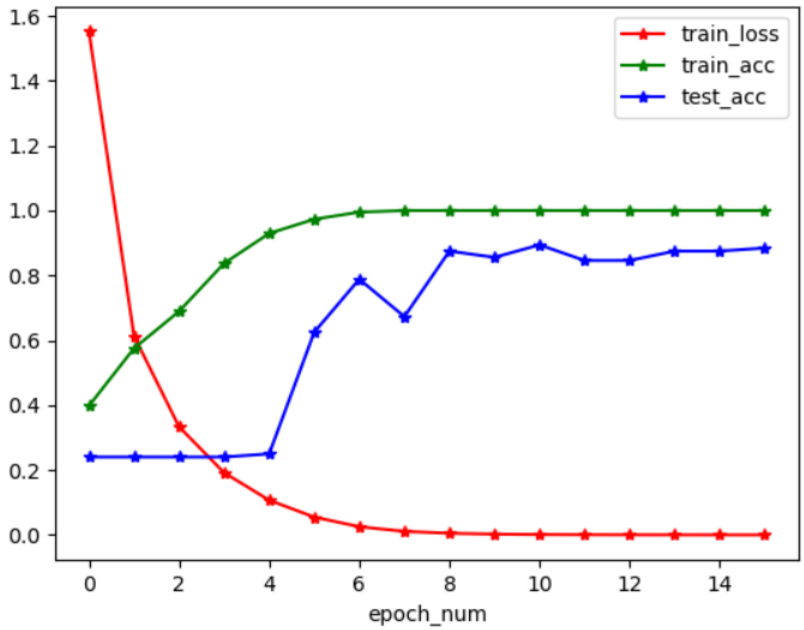


图 3.7

由图 3.7 可知，随着训练轮次增加，训练的损失值逐渐降低，并最终趋于 0.验证集和测试集的准确度均逐渐收敛，在训练集上的准确率趋近于 1，测试集上的准确率可以达到 88.5%，由于输入数据集较小，故收敛速度较快，在训练了 10 轮之后逐渐收敛。

在本地进行预测结果如图 3.8 所示：

```
第 1 朵花预测:bee_balm
第 2 朵花预测:blackberry_lily
第 3 朵花预测:blanket_flower
第 4 朵花预测:bougainvillea
第 5 朵花预测:bromelia
第 6 朵花预测:foxglove
```

图 3.8

对 TestImage 中的 6 张图片进行预测的结果正确率达到 100%，因而可以认为该模型能够较好的完成本实验的花卉识别分类任务。

在 modelarts 训练、预测结果如图 3.9，图 3.10 所示，

```
{0: 'blanket_flower', 1: 'bougainvillea', 2: 'foxglove', 3: 'bee_balm', 4: 'blackberry_lily', 5: 'bromelia'}
training on  cuda
epoch 1, loss 1.5644, train acc 0.350, test acc 0.260, time 0.4 sec
epoch 2, loss 0.6386, train acc 0.471, test acc 0.260, time 0.3 sec
epoch 3, loss 0.3580, train acc 0.597, test acc 0.260, time 0.3 sec
epoch 4, loss 0.2158, train acc 0.740, test acc 0.260, time 0.3 sec
epoch 5, loss 0.1280, train acc 0.874, test acc 0.260, time 0.3 sec
epoch 6, loss 0.0724, train acc 0.971, test acc 0.337, time 0.3 sec
epoch 7, loss 0.0378, train acc 0.995, test acc 0.683, time 0.3 sec
epoch 8, loss 0.0186, train acc 1.000, test acc 0.846, time 0.3 sec
epoch 9, loss 0.0088, train acc 1.000, test acc 0.817, time 0.3 sec
epoch 10, loss 0.0046, train acc 1.000, test acc 0.865, time 0.3 sec
epoch 11, loss 0.0024, train acc 1.000, test acc 0.894, time 0.3 sec
epoch 12, loss 0.0014, train acc 1.000, test acc 0.885, time 0.3 sec
epoch 13, loss 0.0010, train acc 1.000, test acc 0.875, time 0.3 sec
epoch 14, loss 0.0007, train acc 1.000, test acc 0.894, time 0.3 sec
epoch 15, loss 0.0005, train acc 1.000, test acc 0.875, time 0.3 sec
epoch 16, loss 0.0004, train acc 1.000, test acc 0.875, time 0.3 sec
```

图 3.9

```
test1.jpg 预测为:bee_balm
test2.jpg 预测为:blackberry_lily
test4.jpg 预测为:bougainvillea
test3.jpg 预测为:blanket_flower
test6.jpg 预测为:foxglove
test5.jpg 预测为:bromelia
```

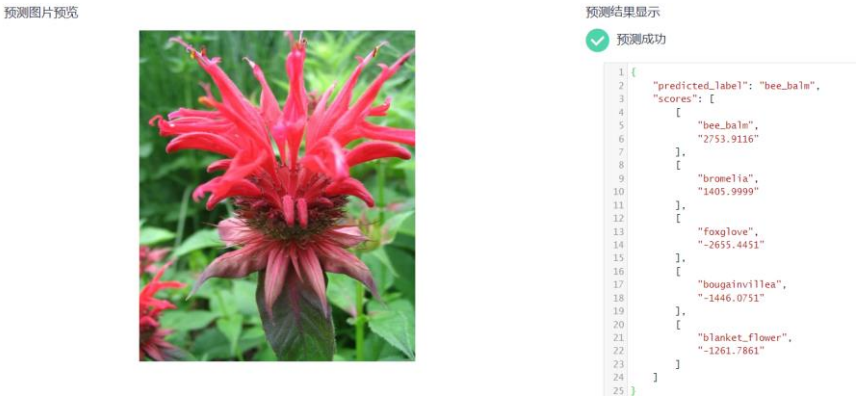
图 3.10

在云端训练的结果均正确，可以将训练后保存的模型下载到本地训练，结果如图 3.11 所示，由于云端是乱序读入，故根据图 3.9 打印的映射关系可知，预测正确。


```
第 1 朵花预测: 3
第 2 朵花预测: 4
第 3 朵花预测: 0
第 4 朵花预测: 1
第 5 朵花预测: 5
第 6 朵花预测: 2
```

图 3.11

在 ModelArts 完成在线部署，最终的预测结果如图所示：



预测图片预览

预测结果显示

预测成功


```
{
    "predicted_label": "blackberry_lily",
    "scores": [
        [
            "blackberry_lily",
            "836.2618"
        ],
        [
            "bougainvillea",
            "8.1273"
        ],
        [
            "blanket_flower",
            "-769.3613"
        ],
        [
            "bee_balm",
            "-762.1447"
        ],
        [
            "foxglove",
            "-392.0389"
        ]
    ]
}
```


预测图片预览

预测结果显示

预测成功

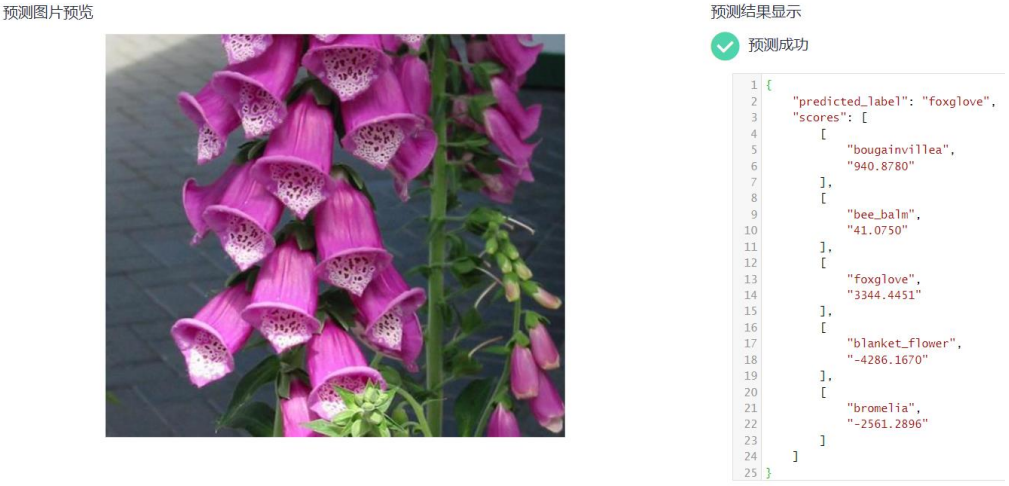

```
{
    "predicted_label": "blanket_flower",
    "scores": [
        [
            "blanket_flower",
            "2515.7017"
        ],
        [
            "bougainvillea",
            "-652.2163"
        ],
        [
            "bromelia",
            "-2502.3704"
        ],
        [
            "bee_balm",
            "-2125.4524"
        ],
        [
            "foxglove",
            "-1726.1395"
        ]
    ]
}
```


预测图片预览

预测结果显示

预测成功


```
{
    "predicted_label": "bougainvillea",
    "scores": [
        [
            "foxglove",
            "210.0067"
        ],
        [
            "bougainvillea",
            "1977.9742"
        ],
        [
            "blackberry_lily",
            "-826.5481"
        ],
        [
            "bee_balm",
            "-591.2386"
        ],
        [
            "blanket_flower",
            "-2532.6035"
        ]
    ]
}
```



由图知，预测结果均正确。

3.5 深度学习框架总结

在完成实验过程中，相同模型的结构、超参、损失函数以及优化器在不同框架下训练的效果不同，将我的 pytorch 网络模型在 tensorflow、mindspore 中实现时，效果均不理想，可能与损失函数或优化器的具体实现上的不同有关。相同的 pytorch 模型可以通过 modelarts，在云端训练好并保存模型的参数，再下载到本地进行训练，在训练过程中，显然云端的 GPU 性能更优，花费的时间更少，在进行一些大规模数据训练时采用云端训练模型会更节省时间。

四、总结

4.1 碰到的问题及解决方法

（1）问题：输入数据的维度与定义的模型不匹配
解决方法：查阅资料知，读入图片维度最后一维为通道数，而模型输入时，需要将通道维度置于第一维（batch 维之后），利用 transpose 可以实现。

```
x_train = x_train.transpose(0, 3, 1, 2)
x_val = x_val.transpose(0, 3, 1, 2)
```

（2）卷积后连接全连接层前未进行展平操作，报错如下：

```
RuntimeError: CUDA error: CUBLAS_STATUS_INVALID_VALUE when calling `cublasSgemm( handle, opa, opb, m, n, k, &alpha, a, lda, b,
  ldb, &beta, c, ldc)`
** On entry to SGEMM  parameter number 10 had an illegal value
```

解决方法：在卷积池化操作完成之后，先添加展平层 FlattenLayer 将数据展平再与全连接层进行对接。

```
# 卷积后展开再进行全连接
class FlattenLayer(nn.Module):
    def __init__(self):
        super(FlattenLayer, self).__init__()
    def forward(self, x): # x shape: (batch, *, *, ...)
        return x.view(x.shape[0], -1)
```

（3）在云端在线部署时，由于本版本的 pytorch 实现所用是 opencv 读入图片并进行训练，在推理代码的 preproce 中用 PIL 读取导致预测效果极差，但改用 opencv 时，出现错误。

```
from .cv2 import *
ImportError: libGL.so.1: cannot open shared object file: No such file or directory
```

解决方法：

修改 config.json 文件 pip 安装 opencv 库，如图所示：

```
"dependencies": [{
    "installer": "pip",
    "packages": [{
            "restraint": "EXACT",
            "package_version": "1.15.0",
            "package_name": "numpy"
        },
        {
            "restraint": "EXACT",
            "package_version": "5.2.0",
            "package_name": "Pillow"
        },
        {
            "restraint": "EXACT",
            "package_version": "3.4.2.17",
            "package_name": "opencv-python-headless"
        }
    ]
}]
```

在预处理时，先用 PIL 处理图片，再转换为 opencv 的格式即可，代码如图所示：

```
def _preprocess(self, data):
    # 读取图片的二进制流转成数组，注意resize大小要跟训练时保持一致
    images = []
    for k, v in data.items():
        for file_name, file_content in v.items():
            image1 = Image.open(file_content)
            img = cv2.cvtColor(np.asarray(image1),cv2.COLOR_RGB2BGR)
            img=cv2.resize(img,(256,256))
            images.append(img)
    images = np.asarray(images,np.float32)
    images = tf.convert_to_tensor(images)
    preprocessed_data = images
    return preprocessed_data
```

4.2 实验的启发、总结、建议

通过本次小组实验，我了解了当前常用的三种深度学习框架的基本使用以及他们之间的区别，加深了我对理论学习的认识同时，为后续可能用到深度学习的学习打下了基础，同时加强了小组合作意识。

建议：本次实验中所用框架的环境搭建起来较为麻烦，且对于基础较为薄弱的同学上手较难，实验过程更偏向于调参，希望后续可以改进实验环境的配置，降低上手难度，扩展实验内容。