东北大学秦皇岛分校

计算机与通信工程学院

人工智能导论大作业

设计题目　基于 CNN 的水果识别模型和 WEB 应用程序

专业名称	计算机科学与技术
班级学号	180235 20188068
学生姓名	**孔天欣**
指导教师	徐长明
设计时间	**2020 年 6 月 5 日**

目　录

人工智能导论大作业实验报告

一、实验背景

水果在现实生活中无处不在，品类繁多，形状各异。一个能够实现水果智能识别的系统，在现实生产和生活中可以得到广泛的应用。本实验报告旨在训练出一个分类器，它是一个可以较好地识别各种水果的深层神经网络模型。本次实验采用的数据集名为 Fruit-360，来自 kaggle[1] Oltean, M.,Muresan, H.. Fruits 360 dataset on kaggle .Online, 2019.9.22。它是一个包含超过 10 万张各类水果图片的数据集。

在本实验报告中，会首先讲述水果识别深层神经网络模型的运作原理，再详细介绍从代码编写、训练、部署上线、构建 web 应用并上线的过程，最后，将回顾实验过程，做出未来的展望。

可通过访问 http://112.124.21.96:8000/fruit/，体验和测试该实验报告中训练完毕的模型。鉴于服务器性能原因，网页加载完成速度较慢，若无法进入页面，请多加刷新，建议使用 chrome 浏览器以获得最佳体验。

二、实验原理

2.1 CNN

卷积神经网络（Convolutional Neural Network,CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于图像处理有出色表现。它包括卷积层(convolutional layer)，池化层(pooling layer)、全连接层(fullyconnected-layer)和损失函数层(loss layer)。在一个典型的 CNN 架构中，每个卷积层后面都有一个线性整流层(ReLU)，然后是一个池化层，然后是一个或多个相似的卷积层，最后是一个或多个全连接层。

2.1.1 卷积层

卷积层是 CNN 的主要层，其中每个神经元都连接到输入区域的某个区域，这个区域被称为感受野(Receptive Field)。用于对输入的图像结构进行特征提取。它通过在输入图像上滑动不同的卷积核并运行一定的运算而组成。此外，在每一个滑动的位置上，卷积核与输入图像之间会运行一个元素对应乘积并求和的运算以将感受野内的信息投影到特征图中的一个元素。卷积核的尺寸要比输入图像小得多，且重叠或平行地作用于输入图像中，一张特征图中的所有元素都是通过一个卷积核计算得出的，也即一张特征图共享了相同的权重和偏置项。

2.1.2 池化层

池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种非线性形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。

2.1.3 完全连接层

在经过几个卷积和最大池化层之后，神经网络中的最终推理通过完全连接层来完成。完全连接层中的神经元与前一层中的所有激活都有联系。因此，它们的激活可以作为仿射变换来计算。

2.1.4 Relu

在深度学习模型中，直线单元是最常用的激活函数。如果接收到任何负的输入，函数返回 0，但是如果接收到任何正的输入，函数返回该值。它可以写成 f(x)=max(0,x)。

2.1.5 损失函数层

损失函数层（loss layer）用于决定训练过程如何来“惩罚”网络的预测结果和真实结果之间的差异，它通常是网络的最后一层。各种不同的损失函数适用于不

同类型的任务。例如，Softmax 交叉熵损失函数常常被用于在 K 个类别中选出一个，而 Sigmoid 交叉熵损失函数常常用于多个独立的二分类问题。

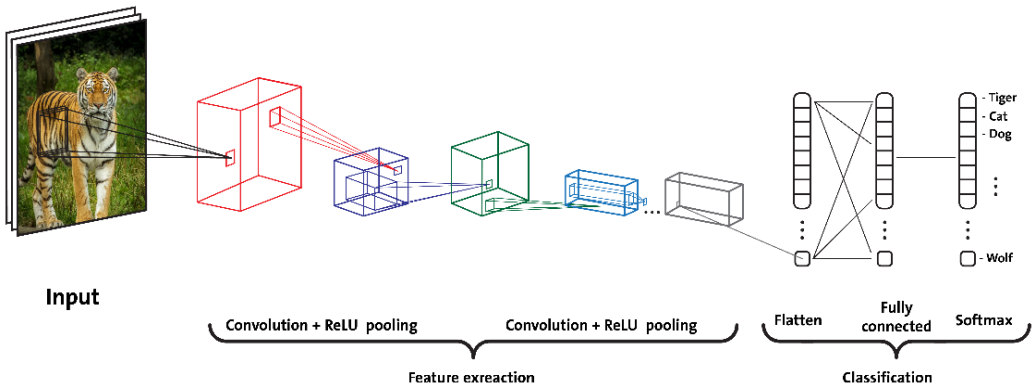

图 2.1.1 CNN 卷积神经网络

2.2 pytorch

PyTorch 是一个开源的 Python 机器学习库，基于 Torch，底层由 C++实现，应用于人工智能领域，如自然语言处理。它最初由 Facebook 的人工智能研究团队开发。

2.3 华为云-AI 开发平台 ModelArts

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 集成 Jupyter Notebook，可提供在线的交互式开发调试工具。用户可以通过创建开发环境，自行开发、调测、训练模型。于在线环境安装常用的机器学习引擎和软件库，实现即开即用。

在 ModelArts 上训练好的模型，通过模型管理和服务部署功能，可快速发布成在线推理服务，实现高吞吐、低延时，支持多模型灰度发布。

2.4 关于数据集 Fruits-360

版本：2019.09.22

训练集大小:67692 张图片(每张图片一个水果或蔬菜)。

测试集大小:22688 张图片(每张图片一个水果或蔬菜)。

类数:131(水果和蔬菜)。

图像大小:100x100 像素。

2.5 web 开发框架 Django

Django 是一个开放源代码的 Web 应用框架，由 Python 写成。

三、实验环境

Python 版本：3.7.4

深度学习包：Pytorch-1.3.1, numpy-1.16.4, sklearn-0.21.2, matplotlib-3.1.0 等。

Web 应用包：Django-3.0.6

云服务器实例一台。

四、实验过程

4.1 编写代码

4.1.1 构建模型

构建模型代码如下：

```
1.
   # in_channels = 3,out channels = 16,kernel_size = 5
2.          # output_size = (100-5+1)*(100-5+1)*16
3.          self.conv1 = nn.Conv2d(in_channels=3, out_channel
   s=16, kernel_size=5)
4.          self.bn1 = nn.BatchNorm2d(16)
```

```
        # output_size = ((96-2)/2+1)*((96-2)/2+1)*16
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        # in_channels = 16,out channels = 32,kernel_size = 5
        # output_size = (48-5+1)*(48-5+1)*32
        self.conv2 = nn.Conv2d(16, 32, 5)
        self.bn2 = nn.BatchNorm2d(32)
        # output_size = ((44-2)/2+1)*((44-2)/2+1)*32
        self.pool2 = nn.MaxPool2d(2, 2)
        # in_channels = 32,out channels = 64,kernel_size = 5
        # output_size = (22-5+1)*(22-5+1)*64
        self.conv3 = nn.Conv2d(32, 64, 5)
        self.bn3 = nn.BatchNorm2d(64)
        # output_size = ((18-2)/2+1)*((18-2)/2+1)*64
        self.pool3 = nn.MaxPool2d(2, 2)
        # in_channels = 64,out channels = 128,kernel_size = 5
        # output_size = (8-5+1)*(8-5+1)*64
        self.conv4 = nn.Conv2d(64, 128, 5)
        self.bn4 = nn.BatchNorm2d(128)
        # output_size = ((4-2)/2+1)*((4-2)/2+1)*128
        self.pool4 = nn.MaxPool2d(2, 2)

        self.linear1 = nn.Linear(2 * 2 * 128, 256)
        self.linear2 = nn.Linear(256, 131)

    def forward(self, x):
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))

        x = self.pool2(F.relu(self.bn2(self.conv2(x))))

        x = self.pool3(F.relu(self.bn3(self.conv3(x))))

        x = self.pool4(F.relu(self.bn4(self.conv4(x))))

        x = x.view(x.size(0), -1)
        x = F.relu(self.linear1(x))
        x = F.log_softmax(self.linear2(x), dim=1)

        return x
```

因此，该深层神经网络模型定义如下表所示：

表 1 神经网络模型各层定义

层类型	输入参数	输出大小
Convolutional	in_channels=3, out_channels=16, kernel_size=5	96 x 96 x 16
BatchNorm2d	16	-
Max pooling	kernel_size=2, Stride=2	48 x 48 x16
Convolutional	in_channels=16, out_channels=32, kernel_size=5	42 x 42 x 32
BatchNorm2d	32	-
Max pooling	kernel_size=2, Stride=2	22 x 22 x 32
Convolutional	in_channels=32, out_channels=64, kernel_size=5	18 x 18 x 64
BatchNorm2d	64	-
Max pooling	kernel_size=2, Stride=2	9 x 9 x 64
Convolutional	in_channels=64, out_channels=128, kernel_size=5	5 x 5 x 128
BatchNorm2d	128	-
Max pooling	kernel_size=3, Stride=2	2 x 2 x 128
Fully connected	512	256

Fully connected	256	131

该模型使用了 4 个卷积层，4 个 BatchNorm 层，以及 4 个池化层和 2 个全连接层。Batch Normalization 将数据拉回到均值为 0，方差为 1 的正态分布上，一方面使得数据分布一致，另一方面避免梯度消失，有利于提高最终训练准确率。该模型原先并无 BatchNorm 层，测试集准确率徘徊在 80%~90%之间。加上后能够提升到 93%+。各层的激活函数使用了 Relu，最后一层全连接层使用了 log_softmax 函数，logSoftmax 实质上是对 softmax 的 log，这样既可以加快运算速度，也可以保持数值的稳定性。

4.1.2 训练模块

训练模块代码定义如下：

```
def train_network(dataloader_train):
    net = FruitNet()
    net = net.to(device)
    # 随机梯度下降法
    optimizer = torch.optim.SGD(net.parameters(), lr=0.003, momentum=0.9)
    criterion = nn.CrossEntropyLoss()
    net.train()

    for epoch in range(epochs):
        print(f"开始第 {epoch + 1} 批")
        for i_batch, (images, labels) in enumerate(dataloader_train):
            images, labels = images.to(device), labels.to(device)
            x = images.float()
            y = labels.long()
            optimizer.zero_grad()
            y_pred = net(x)
            correct = y_pred.max(1)[1].eq(y).sum()
            print("INFO : 预测正确的个数是 : {}".format(correct.item()))
            loss = criterion(y_pred, y)
            print("Loss : {}".format(loss.item()))
            loss.backward()
            optimizer.step()
```

```
23.
24.     # 保存训练模型参数
25.     torch.save(net.state_dict(), "./model/fruit_model_st
   ate_dict.pth")
26.     print("INFO: 总共完成了 {} 批".format(epochs + 1))
27.
28.     return net
```

训练代码部分采用了随机梯度下降法，学习率最初设置是 0.1，但效果不佳。经过多次训练，得到最终的设置为 0.003，可以得到最高的测试准确率。动量的设置是 0.9，有利于减小局部最优解的可能性。计算损失函数采用了 CrossEntropyLoss（交叉熵损失），它在做分类训练的时候可以提高分类精确率。for 循环部分的内容是根据训练图片集和对应的标签集，再根据模型对该训练图片集的预测集合和标签集进行比对，计算准确率，再进行反向传播和梯度优化。

此外，设定的训练批次（epoch）为 10 次，每一个 batch 训练 64 张图片。测试集一轮测试，每一个 batch 也是训练 64 张图片。训练情况见图 4.2.4 和图 4.2.5.

4.1.2 测试模块

利用训练好的模型，对其使用测试集，评估模型的准确率，并为模型的进一步优化提供依据。测试模块的代码定义如下：

```
1. def test_network(net, dataloader_test):
2.     net.eval()
3.     criterion = nn.CrossEntropyLoss()
4.     accuracies = []
5.     with torch.no_grad():
6.         for feature, label in dataloader_test:
7.             feature = feature.to(device)
8.             label = label.to(device).long()
9.             pred = net(feature)
10.             # 要在 cpu 上完成
11.             accuracy = accuracy_score(label.cpu().data.n
   umpy(), pred.max(1)[1].cpu().data.numpy()) * 100
12.             print(f"准确率: {round(accuracy, 3)} %")
13.             loss = criterion(pred, label).item()
14.             print(f"loss : {loss}")
15.             accuracies.append(accuracy)
```

```
16.
17.     # 计算平均正确率
18.     total = 0.0
19.     for j in range(len(accuracies)):
20.         total = total + accuracies[j]
21.     avg_acc = total / len(accuracies)
22.     print(f"TOTAL : 经测试得出的识别准确率
   是: {round(avg_acc, 3)} %")
```

测试时对模型启用 eval 模式，通过计算每次测试准确率的总和除以测试次数长度，即可得出平均识别准确率。

4.1.3 数据集预处理

通过对训练集图片进行预处理和数据增强，有利于提高最后测试集的准确性，减少过拟合的可能性。训练集和测试集预处理代码定义如下：

```
1. data_transform_train = transforms.Compose(
2.         [
3.             transforms.RandomGrayscale(p=0.1),
4.             transforms.RandomRotation(20),
5.             transforms.RandomHorizontalFlip(),
6.             transforms.ToTensor(),
7.             transforms.Normalize(mean=[0.5, 0.5, 0.5],
8.                                  std=[0.5, 0.5, 0.5])])
9.
10.     data_transform_test = transforms.Compose(
11.         [
12.             transforms.ToTensor(),
13.             transforms.Normalize(mean=[0.5, 0.5, 0.5],
14.                                  std=[0.5, 0.5, 0.5])])
```

训练集和测试集的数据处理有所不同。首先，训练集采用了随机灰度（RandomGrayscale）、随机旋转（RandomRotation，20°），随机水平翻转（RandomHorizontalFlip），提高了训练集图片的多样性。同时，对图片进行转为 Tensor 张量和标准化处理（Normalize，各通道均值各为 0.5，标准差亦同）。测试集无需类似处理，只需要转 Tensor 和标准化处理即可。

4.1.4 数据集 npy 化存储

将训练集和测试集的图片集和标签集分别存储到对应的.npy 中，方便读取和使用，训练集存储为.npy 文件的代码如下：

```
train_data = []
train_labels = []
for fruit in fruit_names:

    folder_path = os.path.join(image_path, "Training", fruit)
    images = os.listdir(folder_path)

    for i in range(len(images)):
        final_path = os.path.join(folder_path, images[i])

        img = plt.imread(final_path)
        train_data.append(img)
        train_labels.append(fruit_names.index(fruit))
    print(fruit, " 完毕")

train_data = np.array(train_data)
train_labels = np.array(train_labels)

print("INFO : 正在保存到.npy 文件中。。。")

# 保存数据文件.
np.save('./npy/train/train_data.npy', train_data)
np.save('./npy/train/train_labels.npy', train_labels)

print("INFO : 训练集保存完毕。")
```

测试集存储为.npy 文件的代码结构和上述是相似的，具体实现请见附录：深度学习部分。

4.1.5 创建接口读取.npy 图片和标签

通过创建数据集读取接口，可以写出一个基于.npy 文件数据集读取方式的 DataSet 的子类。

```
1. class Fruit(data.Dataset):
2.
3.     def __init__(self, root_dir, train=True, transform=None):
4.         # 获取数据集资源目录
5.         self.root_dir = os.path.abspath(root_dir)
6.         print(self.root_dir)
7.         self.transform = transform
8.         self.train = train
9.
10.        if (self.train):
11.            self.data = np.load(os.path.join(self.root_dir, "train
   ", "train_data.npy"), "r+")
12.            self.labels = np.load(os.path.join(self.root_dir, "tra
   in", "train_labels.npy"))
13.        else:
14.            self.data = np.load(os.path.join(self.root_dir, "test"
   , "test_data.npy"), "r+")
15.            self.labels = np.load(os.path.join(self.root_dir, "tes
   t", "test_labels.npy"))
16.
17.
18.    def __getitem__(self, index):
19.        img, target = self.data[index], self.labels[index]
20.        img = Image.fromarray(img)
21.
22.        if self.transform is not None:
23.            img = self.transform(img)
24.            img = np.array(img)
25.        return img, target
26.        ……
```

首先在构造函数内选择根据训练集或者测试集读取不同的.npy 文件，此外，在__getitem__方法中，为了进行数据增强，如果 transfrom 存在，则先对图像进行数据增强，再返回 array。具体实现请见附录：深度学习部分。

4.2 模型训练

4.2.1 训练准备

鉴于数据集容量较大（超过 1G），因此.npy 文件无法一次性上传文件至

notebook 当中，必须首先将训练集和测试集的.npy 文件分别上传到 obs 桶中。obs-brower 应用程序可以进行快速的上传操作，上传完毕后，文件存放如下：

图 4.2.1 上传数据集至 obs 桶

之后在 notebook 运行相关代码，远程连接 obs 桶，建立文件传输，将训练集和测试集.npy 文件导入到 notebook 当中。相关代码如下所示：

```
In [ ]: import os
print (os.getcwd())

from modelarts.session import Session

session = Session()
session. download_data(bucket_path="/aibook/projectX/npy/train/train_labels.npy", path="./npy/train/train_labels.npy")
session. download_data(bucket_path="/aibook/projectX/npy/train/train_data.npy", path="./npy/train/train_data.npy")
session. download_data(bucket_path="/aibook/projectX/npy/train/test_data.npy", path="./npy/test/test_data.npy")
session. download_data(bucket_path="/aibook/projectX/npy/train/test_labels.npy", path="./npy/test/test_labels.npy")
```

图 4.2.2 和 obs 桶文件传输至 notebook 上

在本地完成代码编写后，需要进行上传，将整个项目的代码上传至 notebook 当中。上传完成后，notebook 上的深度学习项目目录结构如下所示：

图 4.2.2 notebook 上的项目目录结构

在华为云的 ModelArts 平台，进行模型的训练。鉴于训练作业模块操作繁琐，准备复杂，因此直接在 notebook 开发环境上进行训练。训练过程如下所示：

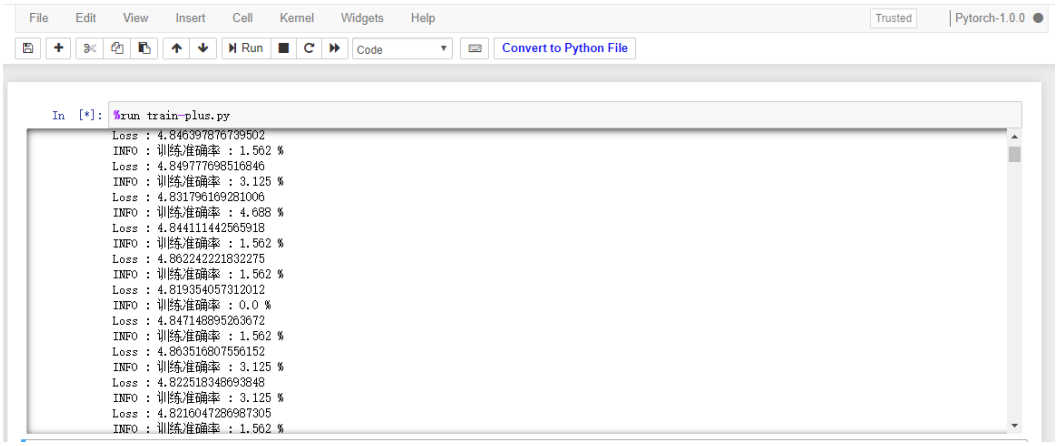

图 4.2.3 notebook 训练过程

4.2.2 训练效果

训练完成后会导出日志文件和训练完成的模型至项目目录下。此时将训练数据通过 matplotlib 绘图，得到训练集准确率百分比-损失函数值图：

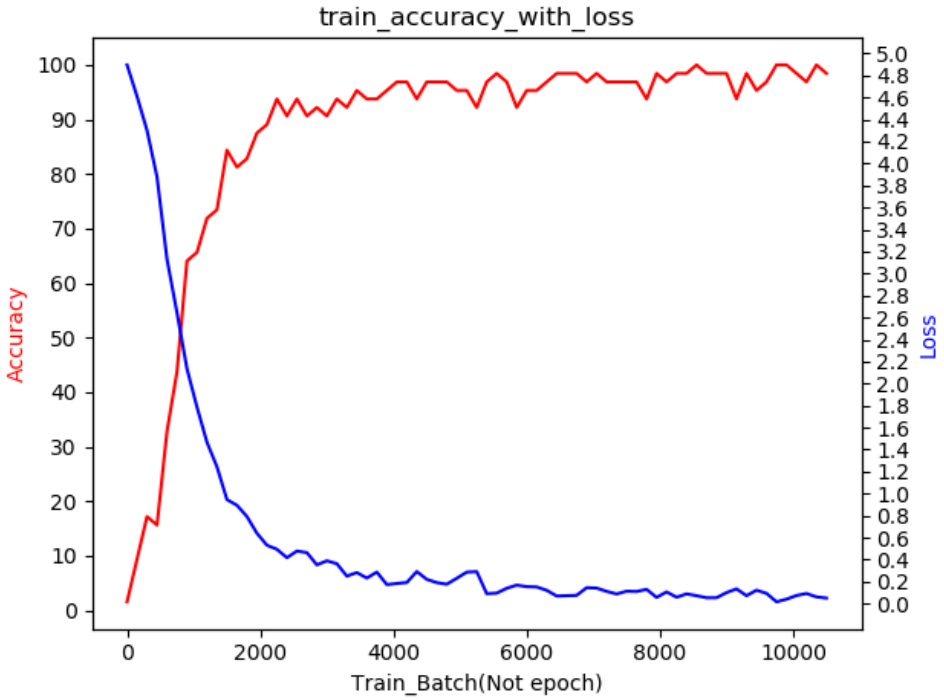

图 4.2.4 训练集 loss-accuray 图

训练完成的模型，在测试集的平均准确率是 97.245%，测试情况见图 4.2.5。

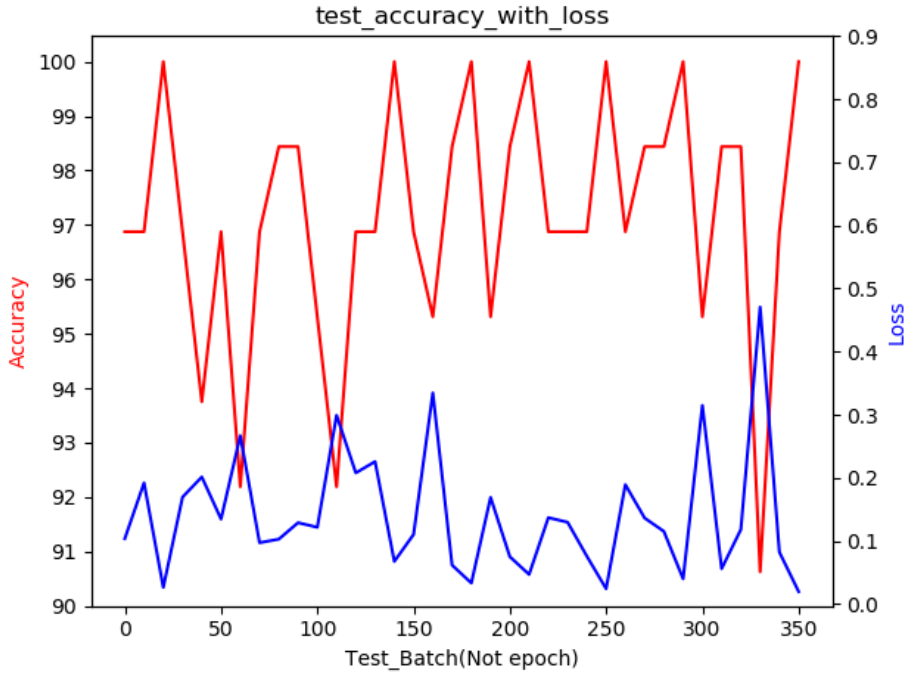

图 4.2.5 测试集 loss-accuray 图

4.3 部署模型上线

训练模型完毕后，应当将模型部署上线，因此需要自行编写后端代码部分以完成模型训练结果数据（json 格式）的响应。华为云官方已经封装完成相关的包，我们只需要编写模型的处理部分即可。此外，还需要准备配置文件 config.json 和训练完成的.pth 模型。

4.3.1 后端代码处理

收到前端请求后，首先应当初始化该后端处理类：

```
1. class PTVisionService(PTServingBaseService):
2.
3.     def __init__(self, model_name, model_path):
4.         # 调用父类构造方法
5.         super(PTVisionService, self).__init__(model_name,
   model_path)
6.         logger.info('model_name: ', model_name)
7.         logger.info('model_path: ', model_path)
8.         # 调用自定义函数加载模型
9.         self.model_name = model_name
```

```
        self.model = get_model(model_path)
        # 加载标签
        self.label = [...]
```

“...”的内容由于所占篇幅太大，故省略不计，只需知道是为标签集对应的数组即可。在构造方法中，首先设置模型的名称，接着调用 get_model 函数加载自定义的水果识别模型：

```
def get_model(model_path, **kwargs):
    # 生成网络
    model = FruitNet()
    # 加载模型
    device = torch.device('cpu')
    model.load_state_dict(torch.load(model_path))
    model.to(device)
    torch.no_grad()
    # 声明为推理模式
     model.eval()
     logger.info("get_model OK")

     return model
```

可以看到，首先初始化了 FruitNet 水果识别模型对象，然后对该模型进行了一系列配置，load_state_dict 函数将项目文件夹下的训练完成的模型数据导入该模型，最后将其声明为推理模式，然后返回这个模型。其中 FruitNet 为自定义的神经网络模型类，该类的详细内容请见前文的 4.1.1。

初始化完毕后，需要解析前端请求，因此重写预处理函数_preprocess：

```
# 定义模型预处理
data_transforms = transforms.Compose(
    [
        transforms.Resize((100, 100)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])

def _preprocess(self, data):
     logger.info("_preprocess begin")
```

```
    preprocessed_data = {}
    for k, v in data.items():
        for file_name, file_content in v.items():
            image = Image.open(file_content)
            img = data_transforms(image)
            logger.info("transforms OK")
            preprocessed_data[k] = img.unsqueeze(0)
            logger.info(preprocessed_data[k])
    logger.info("_preprocess OK")
    return preprocessed_data
```

在该函数中，可以提取出图片的键值对，将图片经过 data_transforms 预处理，例如标准化归一化，图片尺寸按比例缩小成 100*100 像素，以及转为 Tensor。预处理完成后，再将图片进行 unsqueeze 降维，最后将图片键值对送入_inference 函数中处理：

```
def _inference(self, data):
    logger.info("_inference begin")
    for k, v in data.items():
        logger.info("k is :" + k)
        pred = self.model(v)
        pred_sorted_label = torch.sort(pred, descending=True)[1].data.numpy()[0]
        result = self.label[pred_sorted_label[0]]
        res = {'prediction': "I think it maybe :" + result}
        logger.info("_inference OK")
         return res

def _postprocess(self, data):
    logger.info("_postprocess begin")
    return data
```

在重写的_inference 函数中，使用 self.model 函数将图片送入模型进行预测，pred_sorted_label 为通过模型预测后的标签可能性数组，经过降序排列后，位列首位的标签的可能性最大，由于 pred_sorted_label 数组存储的是 labels 数组的下标集合，这些集合按照预测可能性排序，因此可以从 label 数组中根据前几个 labels 下标取标签名（可能性从大到小）。获得标签名后，把结果存入键值对 res 中，为

简便起见，此处只放入可能性最大的键值对，该段代码的后续扩展应用请见 4.4：构建 web 应用部分。

重写的_postprocess 函数直接返回 data，也就是之前的 res 键值对，作为响应即可。

4.3.2 部署文件准备

将即将部署上线的文件放入 obs 桶内，并在同一个文件夹中，文件夹名为 model。如下图所示，其中 state_dict 为训练好的模型，config.json 定义了接口请求和响应数据格式规范以及部署本项目需要的 python 运行环境。关于 config.json 的详细内容，请见附录的深度学习部分。

图 4.3.1 部署文件在 obs 桶中的存放

4.3.3 导入模型和项目上线

在华为云 ModelArts 控制台的“模型”选项中，选择将模型导入，导入内容便是图 4.3.1 所示的文件夹。导入完成后，会出现一个模型，将该模型部署即可，部署完成后如图所示：

图 4.3.2 模型部署成功

4.4 构建 web 应用

部署完成后，ModelArts 会为该部署成功的项目提供一个 API 接口地址。由于华为云官方提供的 token 并无法使用，故无法调用该 API 接口。因此本人自行搭建了一套 web 应用系统，该系统内置了训练完成的模型，以取代远程调用 API 接口。图像识别时，后端调用该模型即可完成预测。

本套水果识别 web 应用系统在前端页面构建完成后，利用 ajax 进行异步发送请求至后端，该请求携带了用户上传的识别图片，后端接受到图片后，调用模型进行识别，然后以预测结果作为响应返回至页面，完成识别的整套流程。由于前端页面的设计和实现和本实验报告关联不大，故不再详细叙述，

该 web 应用程序的后端部分基于 django 实现,项目结构如下：

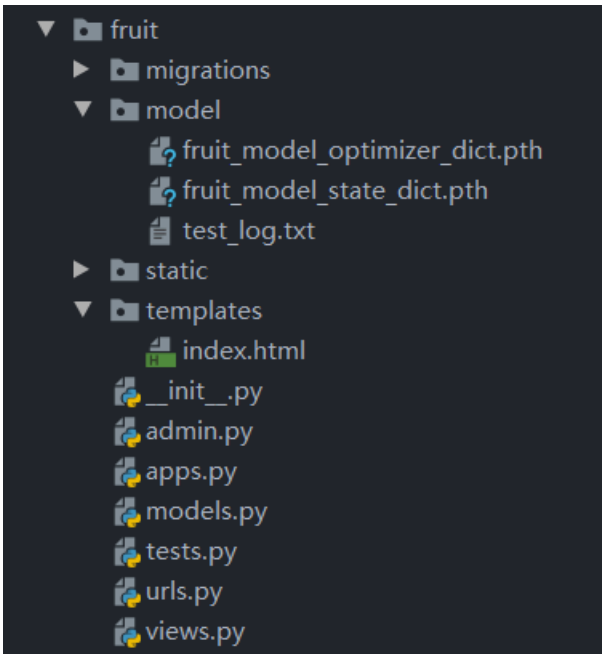

图 4.4.1 web 应用目录树

其中，model 文件夹内存放训练模型，views.py 作为前端控制器，对前端请求进行调用模型和业务处理，图像识别请求部分代码如下：

```
1. def classify(request):
2.     path = './fruit/static/fruit/file/'
3.     dicts = {}
4.     if request.method == "POST":
5.         img = request.FILES.get("image")
6.         file = open(path + "image.jpg", 'wb+')
7.         for chunk in img.chunks():
8.             file.write(chunk)
9.         file.close()
10.         dicts = fruit_model.predict(path)
11.         return JsonResponse(dicts)
12.     else:
13.         return HttpResponse("Method Should Be POST !")
```

predict 方法中调用了模型并完成识别，具体实现详见附件：web 应用部分。

水果识别 Web 应用程序构建完成后，即可在服务器上部署。

4.5 应用上线

通过购买弹性云服务器，在云服务器上部署 web 应用程序，便可以通过

URL 访问水果识别页面和调用模型。

首先启动云服务器，在云服务器上安装 python 环境，然后通过 pip3 安装各个水果识别 web 应用程序所需要的包，安装完毕后如下图所示：

```
[root@izbp1icdyevcrfsbmd4woaz ~]# pip3 list
Package         Version
--------------- -------
asgiref         3.2.7
cycler          0.10.0
Django          2.1.8
future          0.18.2
joblib          0.15.1
kiwisolver      1.2.0
matplotlib      3.2.1
numpy           1.18.4
Pillow          7.1.2
pip             19.0.3
pyparsing       2.4.7
python-dateutil 2.8.1
pytz            2020.1
scikit-learn    0.23.1
scipy           1.4.1
setuptools      40.8.0
six             1.15.0
sqlparse        0.3.1
threadpoolctl   2.0.0
torch           1.5.0
torchvision     0.6.0
uWSGI           2.0.18
```

图 4.5.1 已安装包清单

安装完毕后各大包后，通过 xftp 运输项目文件，如图所示，项目已经导入到系统当中：

```
az /]# ls www/FruitNetWebSite/fruit/
.py  migrations  model  models.py  __pycache__  static  templates  tests.py  urls.py  views.py
az /]#
```

图 4.5.2 查看项目下文件

通过指令运行项目，并设定端口号为 8000，成功启动 web 应用程序。

```
[03/Jun/2020 12:07:14] "GET /fruit HTTP/1.1" 301 0
[03/Jun/2020 12:07:14] "GET /fruit/ HTTP/1.1" 200 13493
[03/Jun/2020 12:07:15] "GET /static/fruit/images/noimage.png HTTP/1.1" 200 1537
[03/Jun/2020 12:07:15] "GET /static/fruit/js/bootstrap-fileinput.js HTTP/1.1" 200 5653
[03/Jun/2020 12:07:15] "GET /static/fruit/js/bootstrap.min.js HTTP/1.1" 200 36874
[03/Jun/2020 12:07:15] "GET /static/fruit/js/jquery.easing.1.3.min.js HTTP/1.1" 200 7039
[03/Jun/2020 12:07:15] "GET /static/fruit/js/jquery.min.js HTTP/1.1" 200 93106
[03/Jun/2020 12:07:15] "GET /static/fruit/js/appear.js HTTP/1.1" 200 6274
[03/Jun/2020 12:07:15] "GET /static/fruit/js/bootstrap-progressbar.min.js HTTP/1.1" 200 2333
[03/Jun/2020 12:07:16] "GET /static/fruit/img/preloader.gif HTTP/1.1" 200 13999
[03/Jun/2020 12:07:16] "GET /static/fruit/js/owl.carousel.min.js HTTP/1.1" 200 23936
[03/Jun/2020 12:07:16] "GET /static/fruit/js/jquery.mixitup.min.js HTTP/1.1" 200 28948
[03/Jun/2020 12:07:16] "GET /static/fruit/js/waypoints.min.js HTTP/1.1" 200 8051
[03/Jun/2020 12:07:16] "GET /static/fruit/js/jquery.counterup.min.js HTTP/1.1" 200 1074
[03/Jun/2020 12:07:16] "GET /static/fruit/js/animated-jquery.js HTTP/1.1" 200 5532
[03/Jun/2020 12:07:16] "GET /static/fruit/js/wow.min.js HTTP/1.1" 200 4777
[03/Jun/2020 12:07:19] "GET /static/fruit/js/custom.js HTTP/1.1" 200 4962
[03/Jun/2020 12:07:20] "GET /static/fruit/file/image.jpg HTTP/1.1" 200 9580
[03/Jun/2020 12:07:22] "GET /static/fruit/js/jquery-1.11.1.min.js HTTP/1.1" 200 95790
[03/Jun/2020 12:07:22] "GET /static/fruit/img/2.jpg HTTP/1.1" 200 217292
[03/Jun/2020 12:07:25] "GET /static/fruit/img/3.jpg HTTP/1.1" 200 418398
[03/Jun/2020 12:07:25] "GET /static/fruit/img/1.jpg HTTP/1.1" 200 295202
```

图 4.5.3 web 应用程序正在运行

运行成功后，开放 8000 端口的安全组，即可通过浏览器访问该 web 应用程序，如图所示：

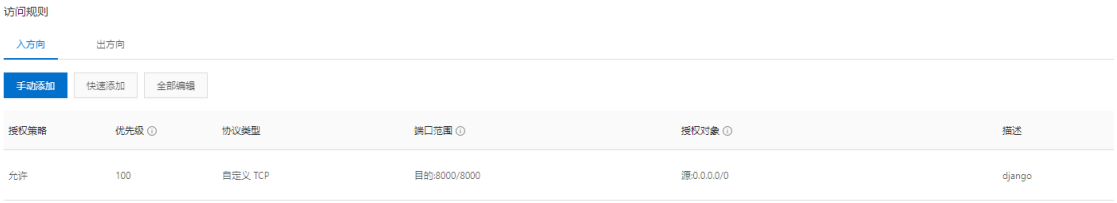

图 4.5.4 设置安全组规则

设置完毕后，在浏览器访问 http://112.124.21.96:8000/fruit/ ，用户即可访问水果识别 web 应用程序的实现，并能够上传图片和返回预测结果。

五、实验效果

5.1 在 ModelArts 中的部署效果

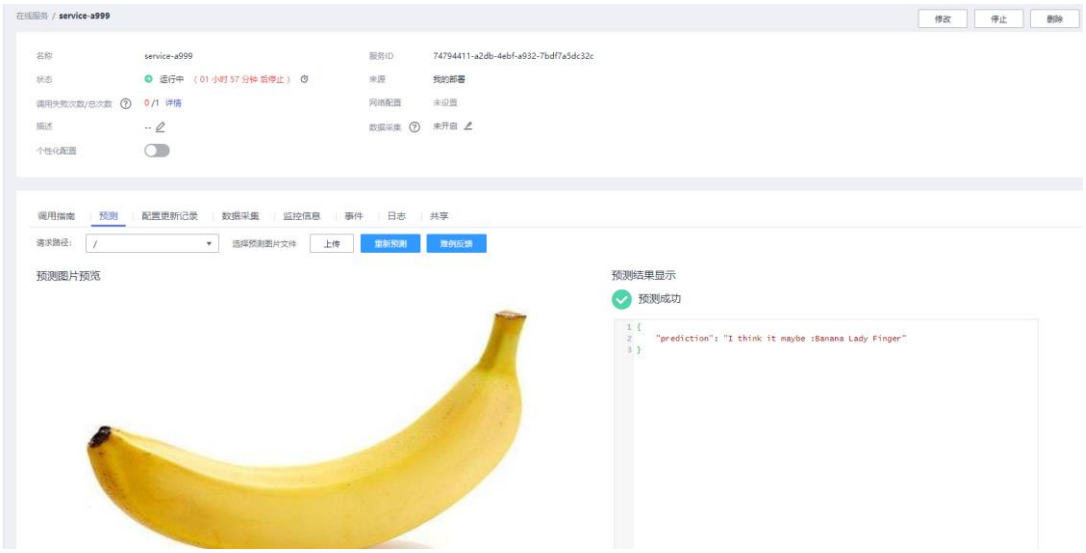

图 5.1.1 ModelArts 部署的预测(1)

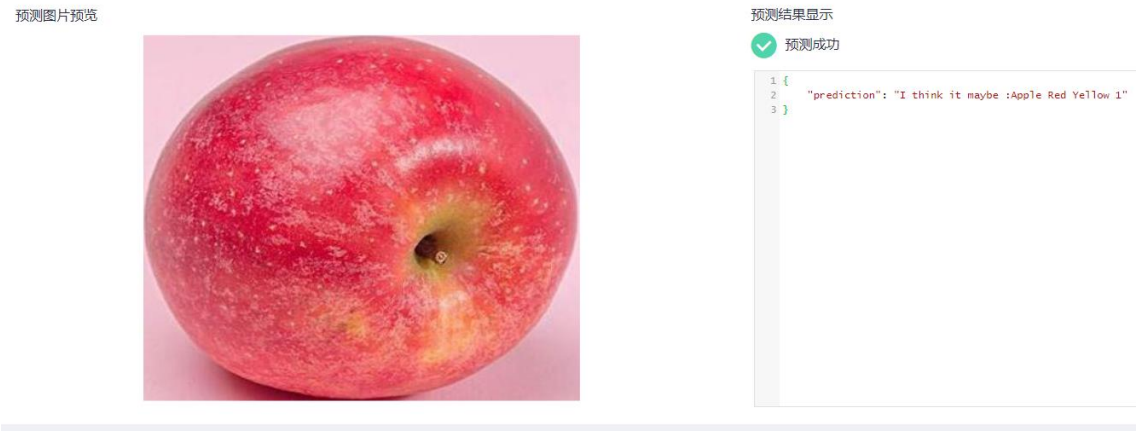

图 5.1.2 ModelArts 部署的预测(2)

5.2 在云服务器 WEB 应用程序中的部署效果

图 5.2.1 web 应用程序展示(1)

图 5.2.2 web 应用程序展示(2)

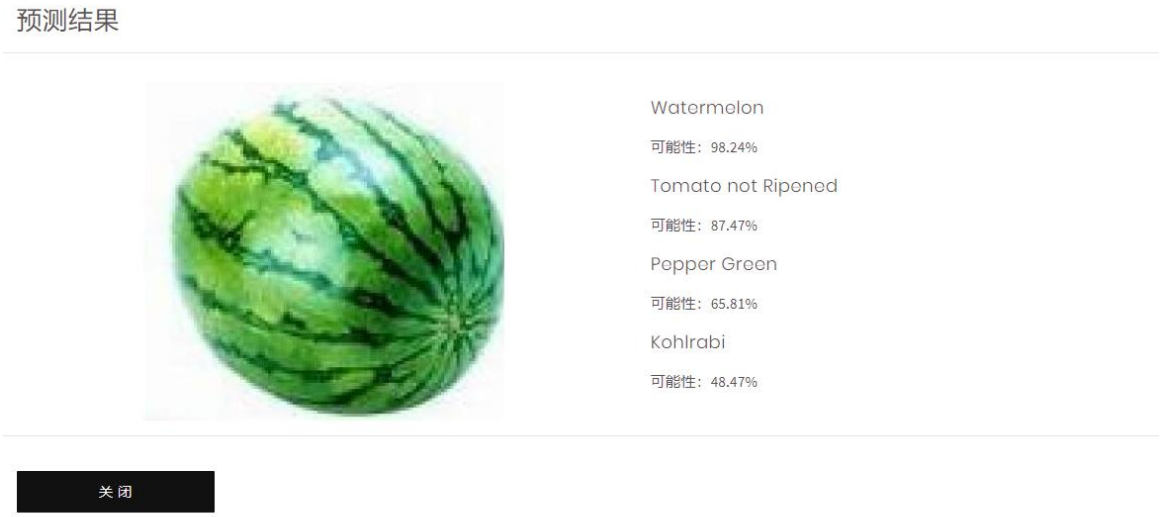

图 5.2.3 web 应用程序展示(3)

图 5.2.4 web 应用程序展示(4)

六、结束语

通过一学期的人工智能导论学习，本人初步掌握了一些人工智能的基本理论及其应用，同时也独立完成了大作业代码的编写、训练、部署、web 应用过程，完成了前后端开发以及训练人工智能模型一整套流程。大作业布置不久以后，刚开始编写代码时，常常充满迷惑和不解，因为 python 和 pytorch 等工具都未曾学过，而且深度学习理论知识学得不多。后来在徐长明老师深度学习直播课程的指导下，本人通过观看推荐的视频，并结合代码，逐渐初步掌握了 python 语法、深度学习的基本数学原理和工程应用开发。最后能够较为顺利地完成大作业。

由于本人在本学期才第一次接触人工智能知识，学习时间不长，而且之前选定的方向是服务端开发，因此在人工智能领域的知识水平往往有所不足。本次训练的模型虽然在测试集准确率较高，但实际应用中往往表现不与之相符，且模型对图像的展现形式也有一定需求，这也说明了本人在人工智能领域的能力亟待提高。鉴于本人有一定的 web 应用开发经验，因此临时学习 django，并结合模型制作了一个简易的识别网站，用户可以方便地通过网站进行各类水果的识别。

本学期的人工智能导论课程学习，本人收获颇丰，一是了解了算法和数学在

计算机程序中的重要性，同时先前数据结构课程和各类高等数学的部分理论在本课程得到了拓展；二是学习和掌握各类基础的人工智能算法，能够拿来解决一些常见的问题，例如搜索算法；三是了解代码程序能够通过自我学习进行更新，并达到人工智能的效果，以及深度学习的理论。同时，本人也意识到计算机应用领域不止前后端 web 开发，还有人工智能，两者结合可以展现出更新颖的效果。综上，本门课程带着我了解了人工智能领域的基础知识，对本人未来的发展富有意义。希望未来有机会接触到更多人工智能的相关应用。

参考资料

[1] Oltean, M.,Muresan, H.. Fruits 360 dataset on kaggle .Online, 2019.9.22

[2] Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Acta Univ. Sapientiae, Informatica Vol. 10, Issue 1, pp. 26-42, 2018.

[3] Stuart Russell,《人工智能：一种现代的方法（第 3 版）》. 人民邮电出版社,2010

附录

源代码：深度学习部分

深度学习水果识别项目目录树如下：

```
├──dataset 数据集
├──logs 日志
├──model 模型
│       fruit_model_state_dict.pth
├──npy .npy 文件
│   ├──test
│   │       test_data.npy
│   │       test_labels.npy
│   │
```

```
|   └──train
|           train_data.npy
|           train_labels.npy
├──TestImage   测试用例
|   config.json   配置文件
|   controller.py
|   fruit_data.py
|   load_dataset.py
|   logger.py
|   paint.py   绘图用
|   predict.py
|   train.py
```

1. Load_data.py 本文件用于读取数据集，并制作成.npy 文件。

```python
import os
import numpy as np
from matplotlib import pyplot as plt

image_path = "dataset"

folder_paths = os.path.join(image_path, "Training")
fruit_names = os.listdir(folder_paths)
print(fruit_names)
print(f"INFO:  训练集图片的路径是 : {image_path}")
print(f"INFO: 测试的标签种类有 {len(fruit_names)} 个")

print("INFO : 正在加载训练集。。。")
# 加载数据集
train_data = []
train_labels = []
for fruit in fruit_names:

    folder_path = os.path.join(image_path, "Training", fruit)
    images = os.listdir(folder_path)

```

```
    for i in range(len(images)):
        final_path = os.path.join(folder_path, images[i])
        img = plt.imread(final_path)
        train_data.append(img)
        train_labels.append(fruit_names.index(fruit))
    print(fruit, " 完毕")

train_data = np.array(train_data)
train_labels = np.array(train_labels)

print("INFO : 正在保存到.npy 文件中。。。")

# 保存数据文件.
np.save('./npy/train/train_data.npy', train_data)
np.save('./npy/train/train_labels.npy', train_labels)

print("INFO : 训练集保存完毕。")

print("INFO : 正在加载测试集。。。")
# 读取测试集
test_data = []
test_labels = []
for fruit in fruit_names:
    print(fruit)
    folder_path = os.path.join(image_path, "Test", fruit)
    images = os.listdir(folder_path)

    for image in images:
        final_path = os.path.join(folder_path, image)
        if not os.path.isfile(final_path):
            print(f"This path doesn't exist : {final_path}")
            continue
        img = plt.imread(final_path)
        test_data.append(img)
        test_labels.append(fruit_names.index(fruit))

test_data = np.array(test_data)
test_labels = np.array(test_labels)

print("INFO : 正在保存到.npy 文件中。。。")

# 保存数据文件.
np.save('./npy/test/test_data.npy', test_data)
```

```
65.np.save('./npy/test/test_labels.npy', test_labels)
66.
67.print("INFO : 测试集保存完毕。")
```

2. fruit_data.py 本文件的 Fruit 类继承了 DataSet 类，自定义提取数据集信息的方式，负责提取.npy 文件中的图片和标签。

```
1. import os
2. import numpy as np
3. from PIL import Image
4. from matplotlib import pyplot as plt
5.
6. import torch.utils.data as data
7.
8.
9. class Fruit(data.Dataset):
10.
11.     def __init__(self, root_dir, train=True, transform=None):
12.         # 获取数据集资源目录
13.         self.root_dir = os.path.abspath(root_dir)
14.         print(self.root_dir)
15.         self.transform = transform
16.         self.train = train
17.
18.         if (self.train):
19.             self.data = np.load(os.path.join(self.root_dir, "train
    ", "train_data.npy"), "r+")
20.             self.labels = np.load(os.path.join(self.root_dir, "tra
    in", "train_labels.npy"))
21.         else:
22.             self.data = np.load(os.path.join(self.root_dir, "test"
    , "test_data.npy"), "r+")
23.             self.labels = np.load(os.path.join(self.root_dir, "tes
    t", "test_labels.npy"))
24.
25.
26.     def __getitem__(self, index):
27.         img, target = self.data[index], self.labels[index]
28.         img = Image.fromarray(img)
29.
30.         if self.transform is not None:
31.             img = self.transform(img)
```

```
            img = np.array(img)
        return img, target

    def __len__(self):
        return (len(self.data))
```

3. train.py 本文件的内容是水果识别的深度学习模型，以及训练模型和测试模型的代码。

```
import torch
import torch.nn as nn
from torchvision import transforms
from torch.utils.data import DataLoader
import fruit_data
import torch.nn.functional as F
from sklearn.metrics import accuracy_score
from PIL import Image
import os
import logger
import sys

device = torch.device("cuda") if (torch.cuda.is_available()) else torch.device("cpu")


# 水果识别模型
class FruitNet(nn.Module):
    def __init__(self):
        super(FruitNet, self).__init__()

        # N=(W-F+2P)/S+1
        # in_channels = 3,out channels = 16,kernel_size = 5
        # output_size = (100-5+1)*(100-5+1)*16 =
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5)
        self.bn1 = nn.BatchNorm2d(16)
        # output_size = ((96-2)/2+1)*((96-2)/2+1)*16
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        # in_channels = 16,out channels = 32,kernel_size = 5
        # output_size = (48-5+1)*(48-5+1)*32
```

```
30.         self.conv2 = nn.Conv2d(16, 32, 5)
31.         self.bn2 = nn.BatchNorm2d(32)
32.         # output_size = ((44-2)/2+1)*((44-2)/2+1)*32
33.         self.pool2 = nn.MaxPool2d(2, 2)
34.         # in_channels = 32,out channels = 64,kernel_size = 5
35.         # output_size = (22-5+1)*(22-5+1)*64
36.         self.conv3 = nn.Conv2d(32, 64, 5)
37.         self.bn3 = nn.BatchNorm2d(64)
38.         # output_size = ((18-2)/2+1)*((18-2)/2+1)*64
39.         self.pool3 = nn.MaxPool2d(2, 2)
40.         # in_channels = 64,out channels = 128,kernel_size = 5
41.         # output_size = (9-5+1)*(9-5+1)*64
42.         self.conv4 = nn.Conv2d(64, 128, 5)
43.         self.bn4 = nn.BatchNorm2d(128)
44.         # output_size = ((5-3)/2+1)*((5-3)/2+1)*128
45.         self.pool4 = nn.MaxPool2d(3, 2)
46.
47.         self.linear1 = nn.Linear(2 * 2 * 128, 256)
48.         self.linear2 = nn.Linear(256, 131)
49.
50.     def forward(self, x):
51.         x = self.pool1(F.relu(self.bn1(self.conv1(x))))
52.         x = self.pool2(F.relu(self.bn2(self.conv2(x))))
53.         x = self.pool3(F.relu(self.bn3(self.conv3(x))))
54.         x = self.pool4(F.relu(self.bn4(self.conv4(x))))
55.         x = x.view(x.size(0), -1)
56.         x = F.relu(self.linear1(x))
57.         x = F.log_softmax(self.linear2(x), dim=1)
58.
59.         return x
60.
61.
62.# 训练模型
63.def train_network(dataloader_train):
64.     net = FruitNet()
65.     net = net.to(device)
66.     # 随机梯度下降法
67.     optimizer = torch.optim.SGD(net.parameters(), lr=0.03, momentu
   m=0.9)
68.     criterion = nn.CrossEntropyLoss()
69.     net.train()
70.
71.     for epoch in range(epochs):
```

```
72.         print(f"开始第 {epoch + 1} 批")
73.         for i_batch, (images, labels) in enumerate(dataloader_trai
    n):
74.             images, labels = images.to(device), labels.to(device)
75.             x = images.float()
76.             y = labels.long()
77.             optimizer.zero_grad()
78.             y_pred = net(x)
79.             # 取正确率
80.             correct = y_pred.max(1)[1].eq(y).sum()
81.             print("INFO : 训练准确
    率 : {} %".format(round(correct.item() / len(labels) * 100, 3)))
82.             loss = criterion(y_pred, y)
83.             print("Loss : {}".format(loss.item()))
84.             loss.backward()
85.             optimizer.step()
86.
87.     # 保存训练模型参数
88.     torch.save(net.state_dict(), "./model/fruit_model_state_dict.p
    th")
89.     print("INFO: 总共完成了 {} 批".format(epochs + 1))
90.
91.     return net
92.
93.
94.# 测试模型
95.def test_network(net, dataloader_test):
96.     net.eval()
97.     criterion = nn.CrossEntropyLoss()
98.     accuracies = []
99.     with torch.no_grad():
100.         for feature, label in dataloader_test:
101.             feature = feature.to(device)
102.             label = label.to(device).long()
103.             pred = net(feature)
104.             # 要在 cpu 上完成
105.             accuracy = accuracy_score(label.cpu().data.numpy(), p
    red.max(1)[1].cpu().data.numpy()) * 100
106.             print(f"正确率: {round(accuracy, 3)} %")
107.             loss = criterion(pred, label).item()
108.             print(f"loss : {loss}")
109.             accuracies.append(accuracy)
110.
```

```
111.     # 计算平均正确率
112.     total = 0.0
113.     for j in range(len(accuracies)):
114.         total = total + accuracies[j]
115.     avg_acc = total / len(accuracies)
116.     print(f"TOTAL : 经测试得出的识别准确率
    是: {round(avg_acc, 3)} %")
117.
118.
119.# 主函数
120.def main():
121.     # 数据增强
122.     data_transform_train = transforms.Compose(
123.         [
124.             transforms.RandomGrayscale(p=0.1),
125.             transforms.RandomRotation(20),
126.             transforms.RandomHorizontalFlip(),
127.             transforms.ToTensor(),
128.             transforms.Normalize(mean=[0.5, 0.5, 0.5],
129.                                  std=[0.5, 0.5, 0.5])])
130.
131.     data_transform_test = transforms.Compose(
132.         [
133.             transforms.ToTensor(),
134.             transforms.Normalize(mean=[0.5, 0.5, 0.5],
135.                                  std=[0.5, 0.5, 0.5])])
136.
137.     transformed_dataset = fruit_data.Fruit(data_dir,
138.                                            train=True,
139.                                            transform=data_transfo
    rm_train)
140.
141.     dataloader_train = DataLoader(transformed_dataset,
142.                                   batch_size=64,
143.                                   shuffle=True)
144.
145.     transformed_test_dataset = fruit_data.Fruit(data_dir,
146.                                                 train=False,
147.                                                 transform=data_tr
    ansform_test)
148.
149.     dataloader_test = DataLoader(transformed_test_dataset,
150.                                  batch_size=64,
```

```
                                    shuffle=True)

    data_iter = iter(dataloader_train)
    images, labels = next(data_iter)
    print("INFO: 图像的尺寸是 {}".format(images.shape))
    print("INFO: 图像 Tensor 的类型是 : {}".format(images.type()))
    print("INFO: 标签的尺寸是 : {}".format(labels.shape))
    print("INFO: 标签 Tensor 的类型是 : {}".format(labels.type()))

    # 训练模型
    net = train_network(dataloader_train)
    # 加载模型
    # net = FruitNet()
    # net.load_state_dict(torch.load("./model/fruit_model_state_dict.pth"))
    # 测试模型
    test_network(net, dataloader_test)


if __name__ == '__main__':
    size = 100
    training_dataset_path = "./dataset/Training"
    data_dir = "./npy"
    epochs = 10
    sys.stdout = logger.Logger("./logs/test_log.txt")
    main()
```

4. predict.py 本文件是利用已经训练好的模型，对实际的一些图片进行分类预测。这些图片放在项目目录的 TestImages 目录下。

```
import torch
import torch.nn as nn
from torchvision import transforms
from torch.utils.data import DataLoader
import fruit_data
import torch.nn.functional as F
from sklearn.metrics import accuracy_score
from PIL import Image
import os
import logger
import sys
import train_plus_plus
```

```
13.
14.device = torch.device("cuda") if (torch.cuda.is_available()) else
   torch.device("cpu")
15.
16.
17.def predict(img_folder_path):
18.     img_files = os.listdir(img_folder_path)
19.     data_transforms = transforms.Compose(
20.         [
21.             transforms.Resize((size, size)),
22.             transforms.ToTensor(),
23.             transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0
   .5, 0.5])
24.         ])
25.     net = train_plus_plus.FruitNet()
26.     net.load_state_dict(torch.load("./model/fruit_model_state_dict
   .pth"))
27.     net = net.to(device)
28.     torch.no_grad()
29.     net.eval()
30.     for img_file in img_files:
31.         print("-----------------------")
32.         print(f"文件名是: {img_file}")
33.         print("-----------------------")
34.         img_path = os.path.join(img_folder_path, img_file)
35.         img = Image.open(img_path)
36.         img = data_transforms(img).unsqueeze(0)
37.         # print(type(img))
38.         img = img.to(device)
39.
40.         pred = net(img)
41.         pred_sorted_like = torch.sort(pred, descending=True)[0].da
   ta.numpy()[0]
42.         pred_sorted_label = torch.sort(pred, descending=True)[1].d
   ata.numpy()[0]
43.         sum = 0
44.         for i in range(4):
45.             sum = sum + pred_sorted_like[i]
46.         for i in range(4):
47.             print('我觉得这个图片可能
   是 :', labels[pred_sorted_label[i]])
```

```
48.             print(f'可能性
    是 :{round((sum - pred_sorted_like[i]) / sum * 100, 2)} % ')
49.
50.
51.if __name__ == '__main__':
52.     size = 100
53.     image_path = "dataset"
54.
55.     folder_paths = os.path.join(image_path, "Training")
56.     labels = os.listdir(folder_paths)
57.     predict("./TestImage")
```

5. controller.py 本文件是华为云 Model-Arts 部署部分的推理代码。

```
1. from PIL import Image
2. from model_service.pytorch_model_service import PTServingBaseServi
   ce
3. import torch.nn.functional as F
4. import torch.nn as nn
5. import torch
6. import json
7. import numpy as np
8. import torchvision.transforms as transforms
9. import os
10.import logging
11.
12.logger = logging.getLogger(__name__)
13.# 定义模型预处理
14.data_transforms = transforms.Compose(
15.     [
16.         transforms.Resize((100, 100)),
17.         transforms.ToTensor(),
18.         transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5,
    0.5])
19.     ])
20.
21.
22.class PTVisionService(PTServingBaseService):
23.
24.     def __init__(self, model_name, model_path):
25.         # 调用父类构造方法
```

```
        super(PTVisionService, self).__init__(model_name, model_path)
        logger.info('model_name: ', model_name)
        logger.info('model_path: ', model_path)
        # 调用自定义函数加载模型
        self.model_name = model_name
        self.model = get_model(model_path)
        # 加载标签
        self.label = [……]
]

    def _preprocess(self, data):
        logger.info("_preprocess begin")
        preprocessed_data = {}
        for k, v in data.items():
            for file_name, file_content in v.items():
                image = Image.open(file_content)
                img = data_transforms(image)
                logger.info("transforms OK")
                preprocessed_data[k] = img.unsqueeze(0)
                logger.info(preprocessed_data[k])
        logger.info("_preprocess OK")
        return preprocessed_data

    def _inference(self, data):
        logger.info("_inference begin")
        for k, v in data.items():
            logger.info("k is :" + k)
            pred = self.model(v)
            pred_sorted_label = torch.sort(pred, descending=True)[1].data.numpy()[0]
            result = self.label[pred_sorted_label[0]]
            res = {'prediction': "I think it maybe :" + result}
            logger.info("_inference OK")
            return res

    def _postprocess(self, data):
        logger.info("_postprocess begin")
        return data


# 水果识别模型
class FruitNet(nn.Module):
```

```
67.     def __init__(self):
68.         super(FruitNet, self).__init__()
69.
70.         # N=(W-F+2P)/S+1
71.         # in_channels = 3,out channels = 16,kernel_size = 5
72.         # output_size = (100-5+1)*(100-5+1)*16 =
73.         self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, ker
    nel_size=5)
74.         self.bn1 = nn.BatchNorm2d(16)
75.         # output_size = ((96-2)/2+1)*((96-2)/2+1)*16
76.         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
77.         # in_channels = 16,out channels = 32,kernel_size = 5
78.         # output_size = (48-5+1)*(48-5+1)*32
79.         self.conv2 = nn.Conv2d(16, 32, 5)
80.         self.bn2 = nn.BatchNorm2d(32)
81.         # output_size = ((44-2)/2+1)*((44-2)/2+1)*32
82.         self.pool2 = nn.MaxPool2d(2, 2)
83.         # in_channels = 32,out channels = 64,kernel_size = 5
84.         # output_size = (22-5+1)*(22-5+1)*64
85.         self.conv3 = nn.Conv2d(32, 64, 5)
86.         self.bn3 = nn.BatchNorm2d(64)
87.         # output_size = ((18-2)/2+1)*((18-2)/2+1)*64
88.         self.pool3 = nn.MaxPool2d(2, 2)
89.         # in_channels = 64,out channels = 128,kernel_size = 5
90.         # output_size = (9-5+1)*(9-5+1)*64
91.         self.conv4 = nn.Conv2d(64, 128, 5)
92.         self.bn4 = nn.BatchNorm2d(128)
93.         # output_size = ((5-3)/2+1)*((5-3)/2+1)*128
94.         self.pool4 = nn.MaxPool2d(3, 2)
95.
96.         self.linear1 = nn.Linear(2 * 2 * 128, 256)
97.         self.linear2 = nn.Linear(256, 131)
98.
99.     def forward(self, x):
100.         x = self.pool1(F.relu(self.bn1(self.conv1(x))))
101.         x = self.pool2(F.relu(self.bn2(self.conv2(x))))
102.         x = self.pool3(F.relu(self.bn3(self.conv3(x))))
103.         x = self.pool4(F.relu(self.bn4(self.conv4(x))))
104.         x = x.view(x.size(0), -1)
105.         x = F.relu(self.linear1(x))
106.         x = F.log_softmax(self.linear2(x), dim=1)
107.
108.         return x
```

```
109.
110.
111.def get_model(model_path, **kwargs):
112.    # 生成网络
113.    model = FruitNet()
114.    # 加载模型
115.    device = torch.device('cpu')
116.    model.load_state_dict(torch.load(model_path))
117.    model.to(device)
118.    torch.no_grad()
119.    # 声明为推理模式
120.    model.eval()
121.    logger.info("get_model OK")
122.
123.    return model
```

6. config.json 本文件是华为云 ModelArts 平台模型部署的配置文件。

```
1. {
2.     "model_type": "PyTorch",
3.     "model_algorithm": "image_classification",
4.     "runtime":"python3.7",
5.     "metrics": {
6.         "f1": 0.345294,
7.         "accuracy": 0.462963,
8.         "precision": 0.338977,
9.         "recall": 0.351852
10.    },
11.    "apis": [{
12.        "protocol": "https",
13.        "url": "/",
14.        "method": "post",
15.        "request": {
16.            "Content-type": "multipart/form-data",
17.            "data": {
18.                "type": "object",
19.                "properties": {
20.                    "images": {
21.                        "type": "file"
22.                    }
23.                }
24.            }
```

```
        },
        "response": {
            "Content-type": "multipart/form-data",
            "data": {
                "type": "object",
                "properties": {
                    "predicted_label": {
                        "type": "string"
                    },
                    "scores": {
                        "type": "array",
                        "items": [{
                            "type": "array",
                            "minItems": 2,
                            "maxItems": 2,
                            "items": [{
                                    "type": "string"
                                },
                                {
                                    "type": "number"
                                }
                            ]
                        }]
                    }
                }
            }
        }
    }],
    "dependencies": [{
        "installer": "pip",
        "packages": [
            {
                "restraint": "ATLEAST",
                "package_version": "1.0.0",
                "package_name": "torch"
            },
            {
                "restraint": "ATLEAST",
                "package_version": "1.16.0",
                "package_name": "numpy"
            }
        ]
    }]
```

```
68.}
```

源代码：Web 应用部分

1.views.py 该文件是 django 的前端控制器，因此负责处理前端送来的请求，在这里，前端发送了待识别的图片，后端返回了预测结果。

```
1. from django.shortcuts import render
2.
3.
4. from django.http import HttpResponse, JsonResponse
5. import os
6. from PIL import Image
7. from fruit import fruit_model
8.
9.
10.def index(request):
11.    return render(request, "index.html")
12.
13.
14.def classify(request):
15.    path = './fruit/static/fruit/file/'
16.    dicts = {}
17.    if request.method == "POST":
18.        img = request.FILES.get("image")
19.        file = open(path + "image.jpg", 'wb+')
20.        for chunk in img.chunks():
21.            file.write(chunk)
22.        file.close()
23.        dicts = fruit_model.predict(path)
24.        return JsonResponse(dicts)
25.    else:
26.        return HttpResponse("Method Should Be POST !")
```

2. fruit_model.py 该文件存放了水果识别模型代码和预测代码。

```
1. import torch
2. import torch.nn as nn
```

```
from torchvision import transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
import os
from PIL import Image

device = torch.device("cuda") if (torch.cuda.is_available()) else torch.device("cpu")

# 水果识别模型
class FruitNet(nn.Module):
    def __init__(self):
        super(FruitNet, self).__init__()

        # N=(W-F+2P)/S+1
        # in_channels = 3,out channels = 16,kernel_size = 5
        # output_size = (100-5+1)*(100-5+1)*16 =
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5)
        self.bn1 = nn.BatchNorm2d(16)
        # output_size = ((96-2)/2+1)*((96-2)/2+1)*16
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        # in_channels = 16,out channels = 32,kernel_size = 5
        # output_size = (48-5+1)*(48-5+1)*32
        self.conv2 = nn.Conv2d(16, 32, 5)
        self.bn2 = nn.BatchNorm2d(32)
        # output_size = ((44-2)/2+1)*((44-2)/2+1)*32
        self.pool2 = nn.MaxPool2d(2, 2)
        # in_channels = 32,out channels = 64,kernel_size = 5
        # output_size = (22-5+1)*(22-5+1)*64
        self.conv3 = nn.Conv2d(32, 64, 5)
        self.bn3 = nn.BatchNorm2d(64)
        # output_size = ((18-2)/2+1)*((18-2)/2+1)*64
        self.pool3 = nn.MaxPool2d(2, 2)
        # in_channels = 64,out channels = 128,kernel_size = 5
        # output_size = (9-5+1)*(9-5+1)*64
        self.conv4 = nn.Conv2d(64, 128, 5)
        self.bn4 = nn.BatchNorm2d(128)
        # output_size = ((5-3)/2+1)*((5-3)/2+1)*128
        self.pool4 = nn.MaxPool2d(3, 2)

        self.linear1 = nn.Linear(2 * 2 * 128, 256)
        self.linear2 = nn.Linear(256, 131)
```

```
44.
45.     def forward(self, x):
46.         x = self.pool1(F.relu(self.bn1(self.conv1(x))))
47.         x = self.pool2(F.relu(self.bn2(self.conv2(x))))
48.         x = self.pool3(F.relu(self.bn3(self.conv3(x))))
49.         x = self.pool4(F.relu(self.bn4(self.conv4(x))))
50.         x = x.view(x.size(0), -1)
51.         x = F.relu(self.linear1(x))
52.         x = F.log_softmax(self.linear2(x), dim=1)
53.
54.         return x
55.
56.# 预测代码
57.def predict(img_folder_path):
58.     labels = [......]
59.     size = 100
60.     img_files = os.listdir(img_folder_path)
61.     data_transforms = transforms.Compose(
62.         [
63.             transforms.Resize((size, size)),
64.             transforms.ToTensor(),
65.             transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0
   .5, 0.5])
66.         ])
67.     net = FruitNet()
68.     net.load_state_dict(torch.load("./fruit/model/fruit_model_stat
   e_dict.pth"))
69.     net = net.to(device)
70.     torch.no_grad()
71.     net.eval()
72.     for img_file in img_files:
73.         print("----------------------")
74.         print(f"FILE_NAME: {img_file}")
75.         print("----------------------")
76.         img_path = os.path.join(img_folder_path, img_file)
77.         img = Image.open(img_path)
78.         img = data_transforms(img).unsqueeze(0)
79.         img = img.to(device)
80.
81.         pred = net(img)
82.         pred_sorted_like = torch.sort(pred, descending=True)[0].da
   ta.numpy()[0]
```

```
83.         pred_sorted_label = torch.sort(pred, descending=True)[1].d
   ata.numpy()[0]
84.
85.         sum = 0
86.         dicts = {}
87.         for i in range(4):
88.             sum = sum + pred_sorted_like[i]
89.         for i in range(4):
90.             dicts[labels[pred_sorted_label[i]]] = str(round((sum -
    pred_sorted_like[i]) / sum * 100, 2))
91.
92.         return dicts
```

注：前端网页部分与报告内容无关，且篇幅过大，故不在此附上。