中山大学计算机学院

人工智能

本科生实验报告

课程名称：Artificial Intelligence

学号	22336326	姓名	朱禹溪

一、 实验题目

深度学习

中药图片分类任务

利用pytorch框架搭建神经网络实现中药图片分类，其中中药图片数据分为训练集`train`和测试集`test`，训练集仅用于网络训练阶段，测试集仅用于模型的性能测试阶段。训练集和测试集均包含五种不同类型的中药图片：`baihe`、`dangshen`、`gouqi`、`huaihua`、`jinyinhua`。请合理设计神经网络架构，利用训练集完成网络训练，统计网络模型的训练准确率和测试准确率，画出模型的训练过程的loss曲线、准确率曲线。

提示

1. 最后提交的代码只需包含性能最好的实现方法和参数设置. 只需提交一个代码文件, 请不要提交其他文件.
2. 本次作业可以使用`pytorch`相关的库、`numpy`库、`matplotlib`库以及python标准库.
3. 数据集可以在Github上下载。
4. 模型的训练性能以及测试性能均作为本次作业的评分标准。
5. 测试集不可用于模型训练。

二、 实验内容

1. 算法原理

代码使用卷积神经网络（CNN）进行训练图像分类模型，并对其在训练和测试过程中的性能进行评估。

模型结构:

该 CNN 包含多个卷积层（nn.Conv2d）和池化层（nn.MaxPool2d），随后是全连接层（nn.Linear）。

在每个卷积层后，使用 ReLU 激活函数来引入非线性。

池化层通过减小特征图的空间维度来减少计算量并提取不变特征。

最后的全连接层将提取的特征映射到不同类别上。

训练器类:

Trainer 类在训练集和测试集上进行模型训练和评估。

训练过程使用了交叉熵损失和 Adam 优化器（optim.Adam）来更新模型参数。

数据加载和预处理:

使用 torchvision 中的 datasets.ImageFolder 创建训练和测试数据集，并将数据加载进数据加载器中。

使用 transforms 对图像进行预处理，包括调整大小、转换为张量和标准化。

训练过程:

训练模型时，遍历每个周期（epoch），通过前向传播、反向传播和参数更新来最小化损失函数。

在每个周期结束后，计算训练损失和准确率，并在测试集上计算测试损失和准确率。

2. 关键代码展示（可选）

```python
# 定义卷积神经网络模型
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # 定义卷积层
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)  # 输入通道为3，输出通道为32，卷积核大小为3x3
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)  # 输入通道为32，输出通道为64，卷积核大小为3x3
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)  # 输入通道为64，输出通道为128，卷积核大小为3x3
        # 定义池化层
        self.pool = nn.MaxPool2d(2, 2)  # 最大池化，核大小为2x2
        # 定义全连接层
        self.fc1 = nn.Linear(128 * 28 * 28, 512)  # 输入特征数为128*28*28，输出特征数为512
        self.fc2 = nn.Linear(512, 5)  # 输出层，5个中药类别

    def forward(self, x):
        # 前向传播
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = x.view(-1, 128 * 28 * 28)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
class Trainer:
    def __init__(self, model, train_loader, test_loader, criterion, optimizer, num_epochs):
        self.model = model.to(device)
        self.train_loader = train_loader
        self.test_loader = test_loader
        self.criterion = criterion
        self.optimizer = optimizer
        self.num_epochs = num_epochs
        self.train_losses = []  # 保存训练损失
        self.train_accuracies = []  # 保存训练准确率
        self.test_losses = []  # 保存测试损失
        self.test_accuracies = []  # 保存测试准确率

    def train(self):
        for epoch in range(self.num_epochs):
            train_loss, train_acc = self._train_epoch()  # 训练模型
            test_loss, test_acc = self._test_epoch()  # 测试模型
            self.train_losses.append(train_loss)
            self.train_accuracies.append(train_acc)
            self.test_losses.append(test_loss)
            self.test_accuracies.append(test_acc)
            print(f'Epoch [{epoch+1}/{self.num_epochs}], Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, Test Loss: {test_loss:.4f}, Test Acc: {test_a

    def _train_epoch(self):
        self.model.train()  # 训练模式
        running_loss = 0.0
        correct = 0
        total = 0
```

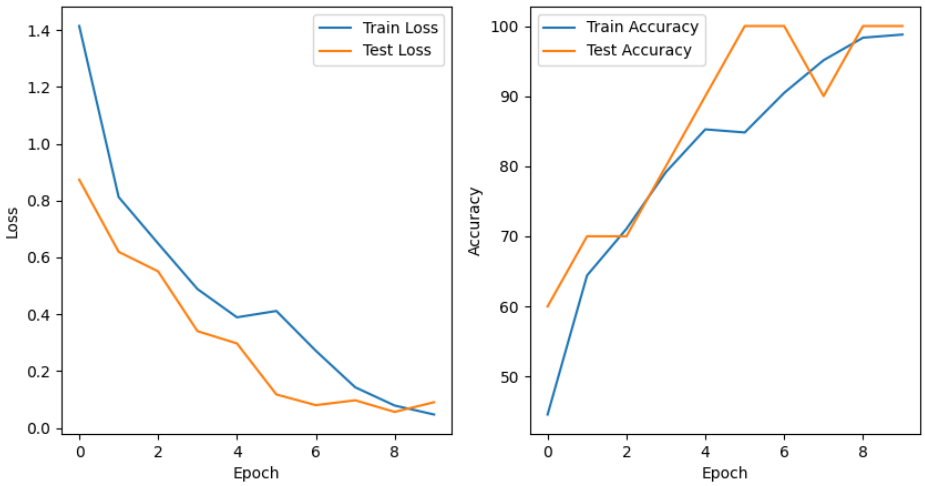
```
        for inputs, labels in self.train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            self.optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = self.criterion(outputs, labels)
            loss.backward()
            self.optimizer.step()
            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()
        train_loss = running_loss / len(self.train_loader)
        train_acc = 100. * correct / total
        return train_loss, train_acc

    def _test_epoch(self):
        self.model.eval()  # 测试模式
        running_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, labels in self.test_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = self.model(inputs)
                loss = self.criterion(outputs, labels)
                running_loss += loss.item()
                _, predicted = outputs.max(1)
                total += labels.size(0)
                correct += predicted.eq(labels).sum().item()
        test_loss = running_loss / len(self.test_loader)
        test_acc = 100. * correct / total
        return test_loss, test_acc
```

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

学习效率 0.001，批次大小 32，迭代 10 次

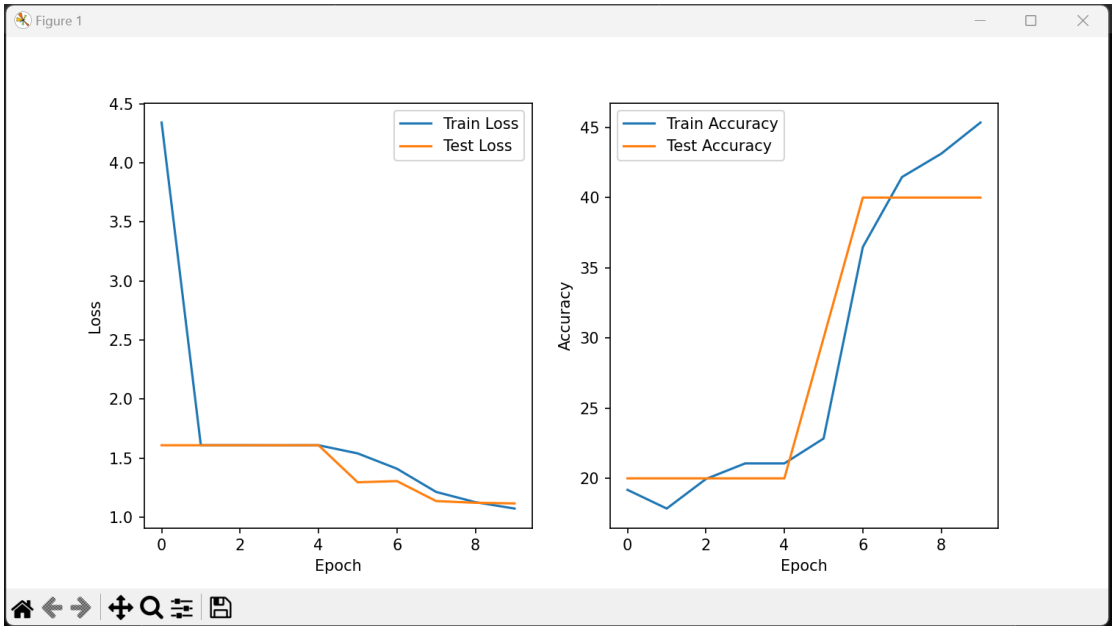
```
Epoch [1/10], Train Loss: 1.4143, Train Acc: 44.57%, Test Loss: 0.8737, Test Acc: 60.00%
Epoch [2/10], Train Loss: 0.8124, Train Acc: 64.41%, Test Loss: 0.6197, Test Acc: 70.00%
Epoch [3/10], Train Loss: 0.6492, Train Acc: 71.06%, Test Loss: 0.5509, Test Acc: 70.00%
Epoch [4/10], Train Loss: 0.4886, Train Acc: 79.16%, Test Loss: 0.3405, Test Acc: 80.00%
Epoch [5/10], Train Loss: 0.3893, Train Acc: 85.25%, Test Loss: 0.2973, Test Acc: 90.00%
Epoch [6/10], Train Loss: 0.4115, Train Acc: 84.81%, Test Loss: 0.1178, Test Acc: 100.00%
Epoch [7/10], Train Loss: 0.2716, Train Acc: 90.47%, Test Loss: 0.0801, Test Acc: 100.00%
Epoch [8/10], Train Loss: 0.1430, Train Acc: 95.12%, Test Loss: 0.0972, Test Acc: 90.00%
Epoch [9/10], Train Loss: 0.0788, Train Acc: 98.34%, Test Loss: 0.0564, Test Acc: 100.00%
Epoch [10/10], Train Loss: 0.0472, Train Acc: 98.78%, Test Loss: 0.0904, Test Acc: 100.00%
```



3. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

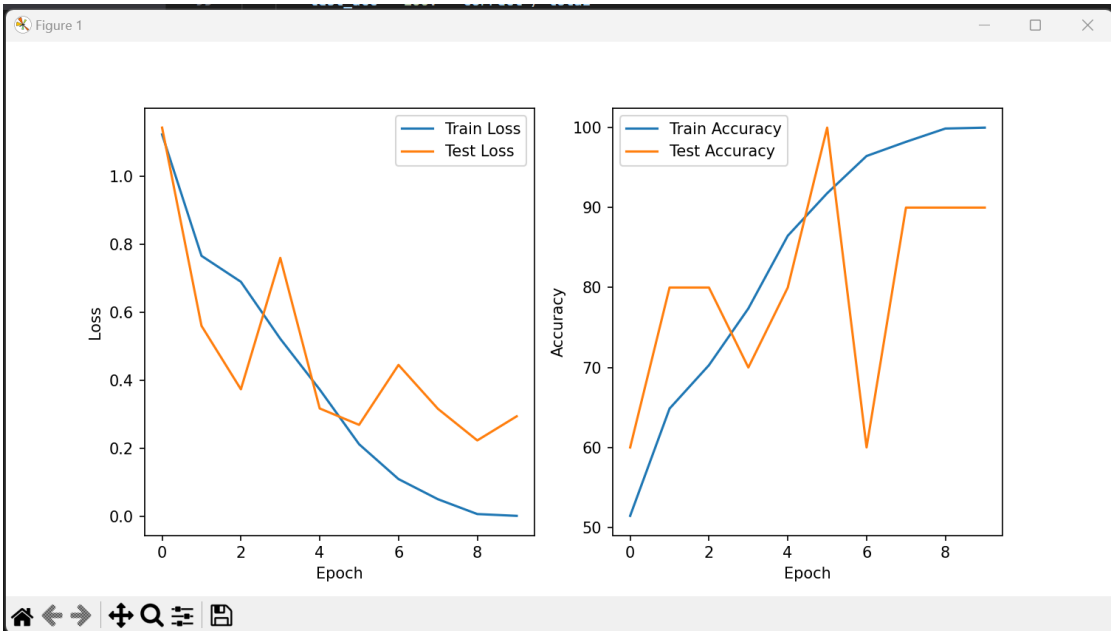
当学习率由 0.001 增加到 0.005 时发现 test accurancy 的波动概率变大并且结果变差。下面是 0.005 的最差的一幅图像

```
Epoch [1/10], Train Loss: 4.3399, Train Acc: 19.18%, Test Loss: 1.6098, Test Acc: 20.00%
Epoch [2/10], Train Loss: 1.6101, Train Acc: 17.85%, Test Loss: 1.6097, Test Acc: 20.00%
Epoch [3/10], Train Loss: 1.6102, Train Acc: 19.96%, Test Loss: 1.6097, Test Acc: 20.00%
Epoch [4/10], Train Loss: 1.6094, Train Acc: 21.06%, Test Loss: 1.6096, Test Acc: 20.00%
Epoch [5/10], Train Loss: 1.6099, Train Acc: 21.06%, Test Loss: 1.6101, Test Acc: 20.00%
Epoch [6/10], Train Loss: 1.5408, Train Acc: 22.84%, Test Loss: 1.2963, Test Acc: 30.00%
Epoch [7/10], Train Loss: 1.4114, Train Acc: 36.47%, Test Loss: 1.3067, Test Acc: 40.00%
Epoch [8/10], Train Loss: 1.2148, Train Acc: 41.46%, Test Loss: 1.1374, Test Acc: 40.00%
Epoch [9/10], Train Loss: 1.1282, Train Acc: 43.13%, Test Loss: 1.1233, Test Acc: 40.00%
Epoch [10/10], Train Loss: 1.0742, Train Acc: 45.34%, Test Loss: 1.1174, Test Acc: 40.00%
```

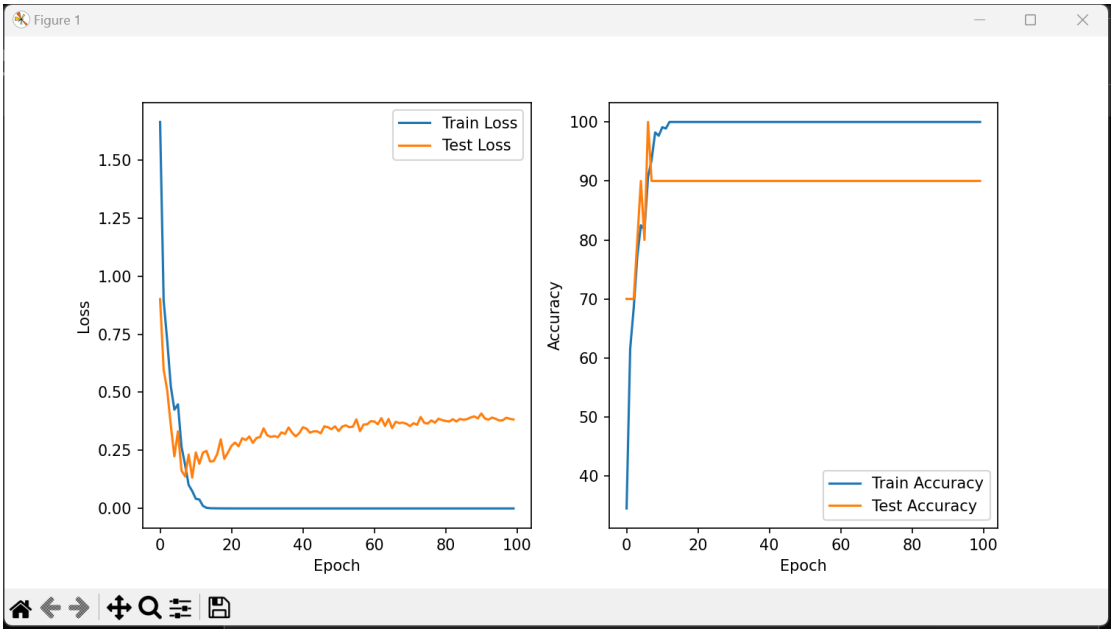


当批次大小由 32 减小到 8 时计算效率下降由原先的 55s 上下变为 70s 上下，图像波动也较为剧烈

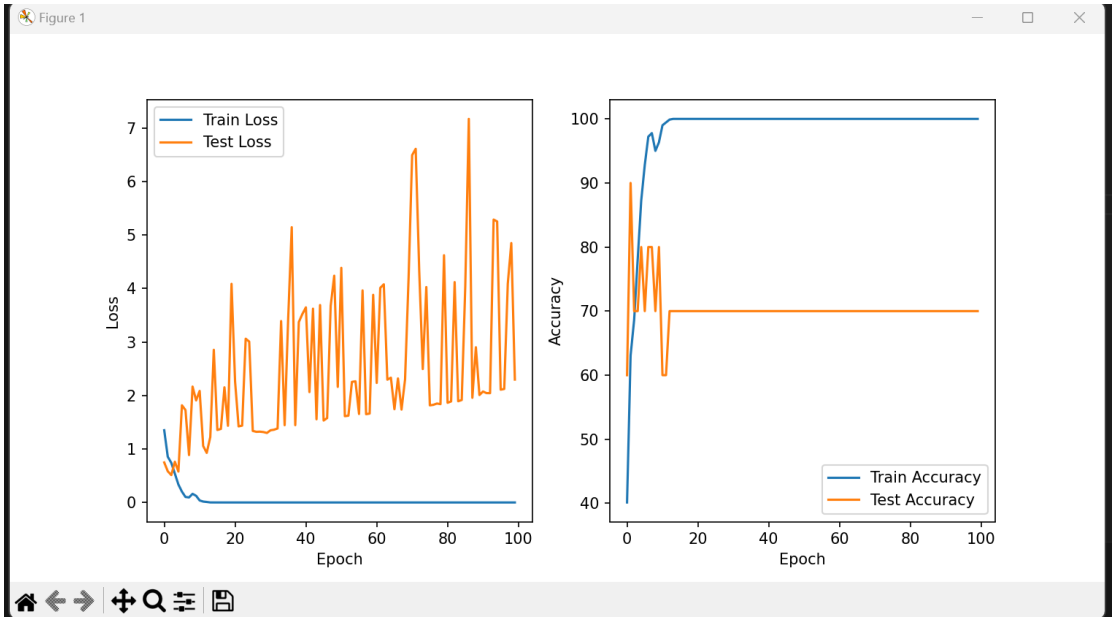
```
Epoch [1/10], Train Loss: 1.1230, Train Acc: 51.44%, Test Loss: 1.1429, Test Acc: 60.00%
Epoch [2/10], Train Loss: 0.7659, Train Acc: 64.86%, Test Loss: 0.5601, Test Acc: 80.00%
Epoch [3/10], Train Loss: 0.6896, Train Acc: 70.29%, Test Loss: 0.3734, Test Acc: 80.00%
Epoch [4/10], Train Loss: 0.5220, Train Acc: 77.38%, Test Loss: 0.7598, Test Acc: 70.00%
Epoch [5/10], Train Loss: 0.3729, Train Acc: 86.47%, Test Loss: 0.3172, Test Acc: 80.00%
Epoch [6/10], Train Loss: 0.2119, Train Acc: 91.80%, Test Loss: 0.2689, Test Acc: 100.00%
Epoch [7/10], Train Loss: 0.1095, Train Acc: 96.45%, Test Loss: 0.4451, Test Acc: 60.00%
Epoch [8/10], Train Loss: 0.0502, Train Acc: 98.23%, Test Loss: 0.3162, Test Acc: 90.00%
Epoch [9/10], Train Loss: 0.0065, Train Acc: 99.89%, Test Loss: 0.2231, Test Acc: 90.00%
Epoch [10/10], Train Loss: 0.0013, Train Acc: 100.00%, Test Loss: 0.2940, Test Acc: 90.00%
Total running time: 73.26 seconds
```



当批次为 32 迭代次数达到 100 时发现 test accurance 收敛于 90%



而批次为 8 迭代次数达到 100 时发现收敛于 70% 故批次还是不宜较小



对主要部分进行时间复杂度分析：

卷积神经网络模型的前向传播:

三个卷积层和池化层的时间复杂度都取决于输入特征图的大小，通常记为 O(n^2) ，其中 n 是输入特征图的大小。

全连接层的计算复杂度与输入特征数和输出特征数成正比，通常记为 O(m*n)，其中 m 是输入特征数，n 是输出特征数。

训练过程:

在每个周期（epoch）中，对于训练集和测试集的遍历，时间复杂度通常为 O(n)，其中 n 是数据集的大小。

在每次遍历中，前向传播和反向传播的时间复杂度与网络层的数量和输入大小有关。

优化器更新:

优化器更新参数的时间复杂度通常取决于参数数量，一般为 O(p)，其中 p 是参数的数量。

综合考虑，整个代码片段的时间复杂度主要取决于卷积神经网络模型的前向传播、训练过程中的遍历和优化器更新。如果考虑实际运行时长，还需考虑具体数据集大小和网络结构参数量的影响。

四、 参考资料

ppt