实验二：卷积神经网络实现

120L021716 蔡泽栋

一、实验目的：

基于 PyTorch 实现 AlexNet 结构
在 Caltech101 数据集上进行验证
使用 tensorboard 进行训练数据可视化

二、实验环境：

Python 3.9.13; torch =2.0.0+cu117; torchvision=0.15.0+cu117; GPU
图片输入 resize 后为 224x224

```
python                    3.9.13              h6244533_2
python-dateutil           2.8.2                  pypi_0
pytorch-mutex             1.0                      cpu

pytz                      2023.3                 pypi_0
regex                     2023.3.23              pypi_0
requests                  2.28.1          py39haa95532_1
scikit-learn              1.2.2                  pypi_0
scipy                     1.10.1                 pypi_0
setuptools                65.6.3          py39haa95532_0
six                       1.16.0             pyhd3eb1b0_1
sqlite                    3.41.1              h2bbff1b_0
sympy                     1.11.1          py39haa95532_0
threadpoolctl             3.1.0                  pypi_0
tk                        8.6.12              h2bbff1b_0
torch                     2.0.0+cu117            pypi_0
torchvision               0.15.0+cu117           pypi_0
tqdm                      4.65.0                 pypi_0
```

三、实验内容：

1. 初始化模型超参数并构建一个 AlexNet：

```
# 设置超参数
num_epochs = 30
batch_size = 64
learning_rate = 0.0001
```

AlexNet 网络包含 5 个卷积层和 3 个全连接层。

层数	定义
C1	64 个 11x11x8 卷积核，步长 2，填充 2
C2	192 个 5x5x64 卷积核，步长 1，填充 2
C3	384 个 3x3x192 卷积核，步长 1，填充 1
C4	256 个 3x3x384 卷积核，步长 1，填充 1
C5	256 个 3x3x256 卷积核，步长 1，填充 1
FC6	卷积核 6x6x6x256，4096 个神经元
FC7	全连接层，4096 个神经元
Output	全连接层，101 个神经元

```python
class AlexNet(nn.Module):
    def __init__(self, num_classes=101):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

2. 数据读取：

```python
def get_data_loaders(data_dir, batch_size):
    # 定义数据预处理方式，包括裁剪、缩放和标准化
    transform = transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])

    # 加载数据集
    full_dataset = datasets.ImageFolder(os.path.join(data_dir), transform=transform)

    # 将数据集按照数量划分为训练集、验证集和测试集
    num_samples = len(full_dataset)
    num_train = int(num_samples * 0.8)
    num_valid = int(num_samples * 0.1)
    num_test = num_samples - num_train - num_valid

    train_dataset, valid_dataset, test_dataset = torch.utils.data.random_split(
        full_dataset, [num_train, num_valid, num_test])
    print(len(train_dataset), len(valid_dataset), len(test_dataset))
    # 创建DataLoader对象
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4)
    valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_size=batch_size, shuffle=False, num_workers=4)
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=4)

    return train_loader, valid_loader, test_loader
```

按照 8：1：1 的比例划分 train，valid，test 集进行训练、验证和测试。caltech-101/101_ObjectCategories 的子文件夹存放了各个类别的图片，所以在构造的过程中采用 data.random_split()函数将总的数据集按设定好的比例随机划分，使得实验数据更泛化

3. 定义优化器并训练

```python
for epoch in range(num_epochs):
    train_loss = 0.0
    train_acc = 0.0
    valid_loss = 0.0
    valid_acc = 0.0

    # 训练模型
    for inputs, labels in train_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * inputs.size(0)
        _, preds = torch.max(outputs, 1)
        train_acc += torch.sum(preds == labels.data)
```

使用交叉熵作为损失函数，Adam 作为优化器，训练模型：

optimizer.zero_grad()：清除优化器中之前保存的梯度。在反向传播计算梯度之前，需要先将梯度清零，否则之前计算的梯度会累加到当前的梯度中，导致计算出错。

outputs = model(images)：使用模型进行前向传播，生成输出结果。

loss = criterion(outputs, labels)：计算模型预测结果和真实结果之间的损失。

loss.backward()：计算损失函数关于模型参数的梯度。梯度的计算是通过自动微分机制实现的。

optimizer.step()：更新模型参数。根据优化算法的规则，以梯度下降的方式更新模型的参数，以尽可能减小损失函数的值。

每次执行一次可以让模型的参数更新一次，从而逐渐提升模型的准确性。反复执行这个训练步骤，直到模型收敛或达到指定的训练轮数。

在每次进行过一轮 epoch 后，放入 valid 集检验正确率。然后将正确率最高的模型作为模型参数进行保存。

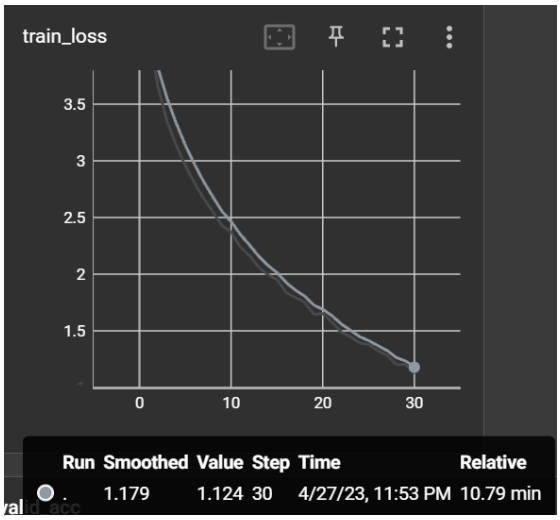
```
# 如果当前模型在验证集上的准确率更好，则保存模型参数
if valid_acc > best_valid_acc:
    best_valid_acc = valid_acc
    best_model_params = model.state_dict()
```

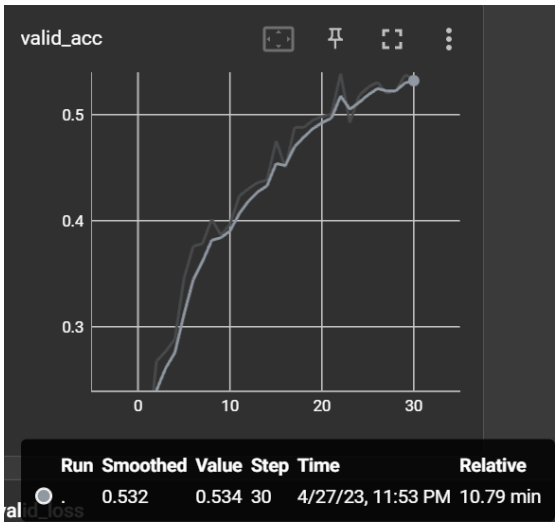
四、实验结果分析：

```
def test(test_loader, criterion, writer):
    new_model = AlexNet(num_classes=101)
    # 加载保存的模型参数
    checkpoint = torch.load('alexnet.pth')
    new_model.load_state_dict(checkpoint)

    # 设置模型为评估模式
    new_model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = new_model(data)
            test_loss += criterion(output, target).item()  # 将一批的损失相加
            pred = output.argmax(dim=1, keepdim=True)  # 找到概率最大的下标
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    test_accuracy = 100. * correct / len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset), test_accuracy))
    writer.add_scalar('Testing Loss', test_loss)
    writer.add_scalar('Testing Accuracy', test_accuracy)
```

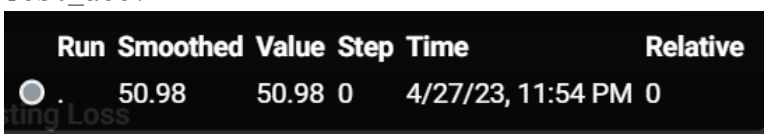
测试过程不需要 train 训练，而是将之前保存好的最优模型 load_state 并在 test 集上测试即可

对于 tensorboard 可视化，本次实验只将 train、valid 的 loss 以及正确率和 test 集的正确率写入了文件。最后在终端运行 tensorboard --logdir=路径名即可得到结果。





Test_acc:



对比实验结果可以得出：

1. epochs 的提升对模型训练集准确率提升较大，但是在 20 轮以后在验证集和测试集效果很微小并且有过拟合的倾向。测试集在 20 轮以后的准确率稳定在 50%左右
2. 过大的学习率(0.01)很容易出现已经达到了最低点但是跨出去的情况，j鉴于本实验收敛轮次较大，本实验采用减小学习率(0.0001)提升训练轮次达到收敛效果。