2023-2024 学年

第 1 学期

《深度学习》

课程实验报告

姓名：李嘉政　　学号：1120211695

实验 1：基于全连接网络和 CNN 的图像分类任务

一、 实验目的

本次实验通过使用 Pytorch 完成 CIFAR-10 图像分类任务，使同学们进一步熟悉 Pytorch 训练框架，掌握 Pytorch 中的数据加载、数据增强实现、卷积神经网络搭建、分类网络训练、分类模型测试以及模型优化等内容。利用具体的图像分类任务来帮助大家理解课程中的相关内容。

二、 实验内容

1.掌握 Pytorch 中加载数据的方法，根据提供的图片和标签，加载 CIFAR-10 数据，自定义 Pytorch 中的 Dataset 和 DataLoader。

2.掌握 Pytorch 中卷积神经网络的构建方法，搭建合适的网络结构用于 CIFAR-10 数据集图像分类任务。

3.掌握 Pytorch 中损失函数和优化器的定义方法，并探索不同损失函数和优化方法对于模型效果的影响，针对图像分类任务进行合适的选择。

4.掌握 pytorch 中模型训练的步骤，并探索不同超参数对于模型效果的影响，学会调整参数来提高模型的训练效果。

三、 实验原理

本实验中使用的 CIFAR-10 数据集，它包含十个类别：‘airplane’，‘automobile’，‘bird’，‘cat’，‘deer’，‘dog’，‘frog’，‘horse’，‘ship’，‘truck’。CIFAR-10 中的图像尺寸为 3×32×32，也就是 RGB 的 3 层颜色通道，每层通道内的尺寸为 32×32。在 Pytorch 中训练一个图像分类器需如下步骤：加载 CIFAR-10 的训练和测试数据集、定义一个卷积神经网络、定义一个损失函数、在训练样本数据上训练网络、在测试样本数据上测试网络。

四、 实验过程

1. Dataset

```
class CifarDataset(Dataset):
    def __init__(self, mode):
        with open("Dataset/" + mode + ".txt", "r") as f:
            self.data = f.readlines()
            if mode == "trainset" or mode == "validset":
                self.label = [int(line.split(" ")[1]) for line in self.data]
                self.data = [line.split(" ")[0] for line in self.data]
            else:
                self.label = None
                self.data = [line.strip() for line in self.data]
        if mode == "trainset":
            self.transform = transforms.Compose([
                transforms.Resize((32, 32)),
                transforms.RandomChoice(transforms=[
                    transforms.TrivialAugmentWide(),
                    transforms.Lambda(lambda x: x)],
                    p=[0.95, 0.05]),
                transforms.ToTensor(),
            ])
        else:
            self.transform = transforms.Compose([
                transforms.Resize((32, 32)),
                transforms.ToTensor(),
            ])
        self.data = [self.transform(Image.open("Dataset/image/" + data)) for data in self.data]
```

使用 torchvision.transforms 对数据进行增强，随机选择了部分加强方法来加强。而面对不同模型时输入图像大小被修改，具体修改在后续架构介绍时会提到。

2. Train, Validation, Test

```python
def train(net, criterion, optimizer):
    epoch_num = 100
    val_num = 1
    train_acc = []
    val_acc = []
    max_acc = 0

    for epoch in range(epoch_num):
        print("Epoch:", epoch + 1)
        correct = 0
        total = 0
        net.train()
        for data in tqdm.tqdm(train_loader):
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            # print(images, labels)
            outputs = net(images)
            loss = criterion(outputs, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            correct += (torch.max(outputs, 1)[1] == labels).sum().item()
            total += labels.size(0)
```

```python
        print("Train Accuracy:", correct / total)
        train_acc.append(correct / total)

        net.eval()
        if epoch % val_num == 0:
            acc = validation(net)
            val_acc.append(acc)
            print("Validation Accuracy:", acc)
            if acc > max_acc:
                max_acc = acc
                checkpoint = {
                    "net": net.state_dict(),
                    "optimizer": optimizer.state_dict(),
                }
                torch.save(checkpoint, "checkpoint_den.pth")
                print("Model Saved!")

    print('Finished Training!')

    plt.plot(range(1, epoch_num + 1), train_acc, label="Train")
    plt.plot(range(1, epoch_num + 1), val_acc, label="Val")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.savefig("result_den.png")
```

```python
def validation(net):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in tqdm.tqdm(val_loader):
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    return correct / total
```

```python
def test(net):
    with open("result_den.txt", "w") as f:
        for data in tqdm.tqdm(test_loader):
            images, labels = data
            images = images.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            for label in predicted:
                f.write(str(label.item()) + "\n")
```

Train, Validation, Test 过程差不多。区别在于 Validation 和 Test 不会去反向传播。Train 的时候每一轮都会进行 Validation，并存储 Accurary 绘制表图方便观察模型情况。同时每次利用 Validation 的结果，取最优的 Validation 来作为后续 Test 模型。

3. 网络架构

3.1 全连接神经网络 (FNN)

使用较为简单的四层 Linear 层，输入图像使用正常的 (32, 32, 3)。

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(32 * 32 * 3, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)
        self.fc4 = nn.Linear(128, 10)

    def forward(self, x):
        x = x.view(-1, 32 * 32 * 3)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

3.2 简单卷积神经网络 (AlexNet)

参照 AlexNet 网络架构，由于 Cifar10 数据集输入图像过小，所以将 Cifar10 的输入图像从 (32, 32, 3) 修改成了 (70, 70, 3)，以保证卷积之后不会让图像缩小到 (0, 0)。

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.size = 70
        self.conv1 = nn.Conv2d(3, 96, kernel_size=11, stride=2, padding=1)
        self.size = (self.size - 11 + 2 * 1) // 2 + 1
        self.maxpool1 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.size = (self.size - 3) // 2 + 1
        self.conv2 = nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2)
        self.size = (self.size - 5 + 2 * 2) // 1 + 1
        self.maxpool2 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.size = (self.size - 3) // 2 + 1
        self.conv3 = nn.Conv2d(256, 384, kernel_size=3, stride=1, padding=1)
        self.size = (self.size - 3 + 2 * 1) // 1 + 1
        self.conv4 = nn.Conv2d(384, 384, kernel_size=3, stride=1, padding=1)
        self.size = (self.size - 3 + 2 * 1) // 1 + 1
        self.conv5 = nn.Conv2d(384, 256, kernel_size=3, stride=1, padding=1)
        self.size = (self.size - 3 + 2 * 1) // 1 + 1
        self.maxpool3 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.size = (self.size - 3) // 2 + 1
        self.fc1 = nn.Linear(256 * self.size * self.size, 4096)
        self.dropout1 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(4096, 4096)
        self.dropout2 = nn.Dropout(0.5)
        self.fc3 = nn.Linear(4096, 10)
```

```
    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.maxpool2(x)
        x = self.conv3(x)
        x = F.relu(x)
        x = self.conv4(x)
        x = F.relu(x)
        x = self.conv5(x)
        x = F.relu(x)
        x = self.maxpool3(x)
        x = x.view(-1, 256 * 6 * 6)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x
```

3.3 稠密连接网络 (DenseNet)

使用了 torchvision.models 里的 densenet201 架构，并在已经

pretrained 好的模型上 fine-tuning，为了让输入图像能更好地被捕捉特征，将输入图像从 (32, 32, 3) 修改成了 (224, 224, 3)。

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.model = densenet201(pretrained=False)
        self.model.classifier = nn.Linear(1920, 10)

    def forward(self, x):
        return self.model(x)
```

五、 实验结果与分析

运行成功结果:

```
Epoch: 1
 55%|                                                  | 171/313 [00:01<00:01, 134.48it/s]
```

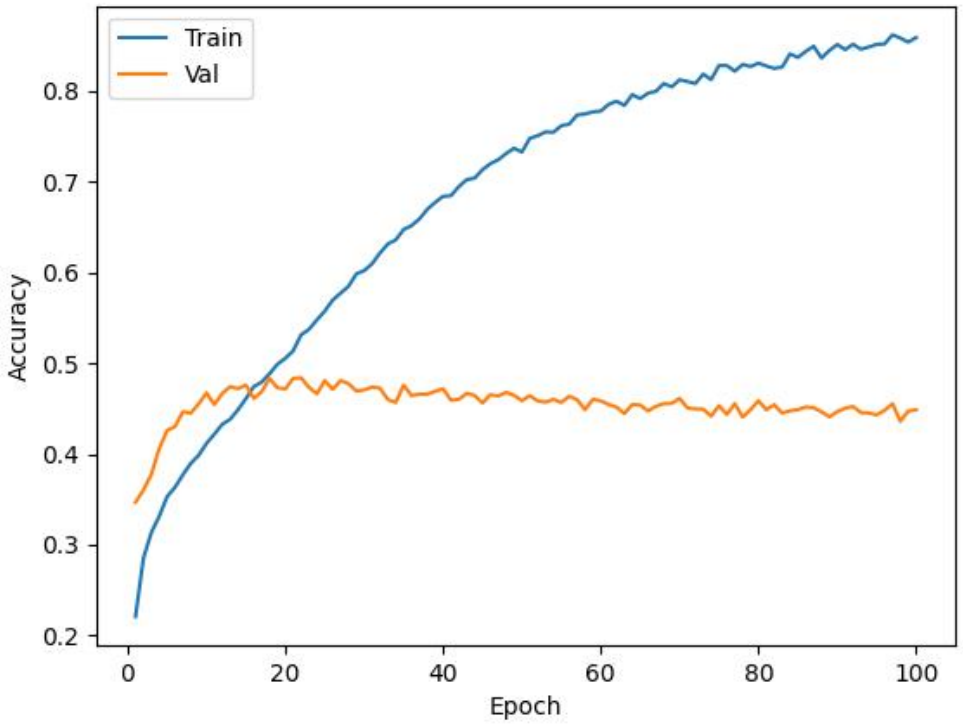

图 1 全连接神经网络

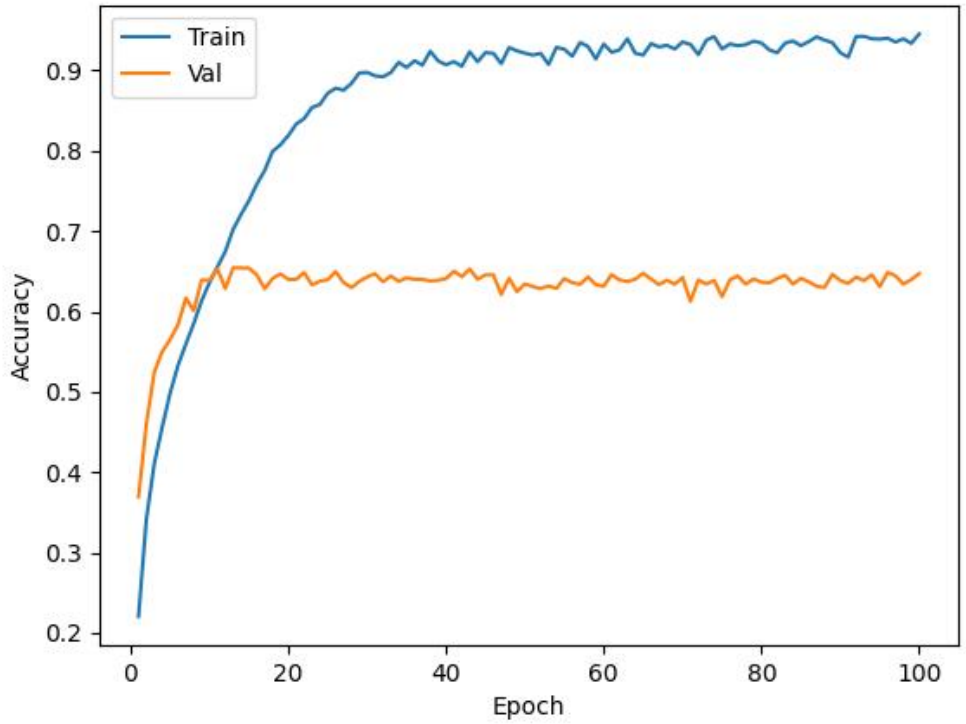

图 2 简单卷积神经网络（AlexNet）

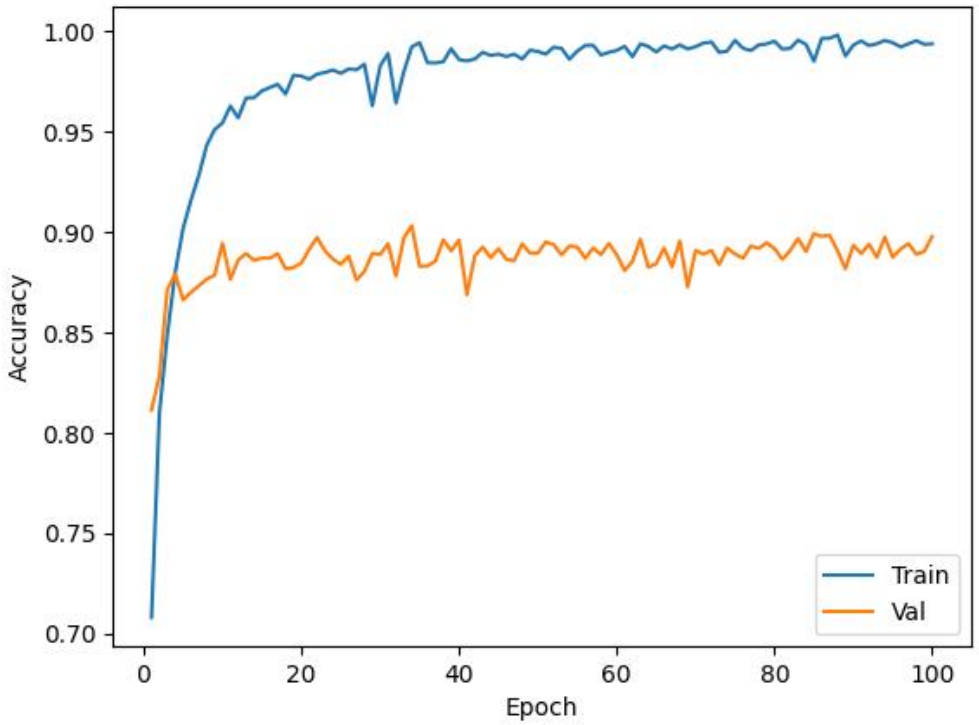

图 3 稠密卷积神经网络（DenseNet）

从三张图中可以看出，在 Cifar10 数据上，三种网络都比较容易过拟合，基本都在十轮训练以内就发生过拟合了。在效果上，全连接的验证集最后基本稳定在 45% 左右，AlexNet 基本稳定在 65% 左右，而 DenseNet 基本稳定在 90% 左右。训练上，三者的 Acc 都几乎能到 100%。

六、 心得体会

本次实验中，我分别使用了三种古典的神经网络架构进行了训练与测试。可以看出，面对 Cifar10 这样一种比较小的数据时，不基于现代思路的网络也能表现得很好，尤其是 DenseNet。那么如果用更现代的，比如 ViT 或者 CLIP 等模型，容易获得更为成功的结果，这值得继续研究。