深度学习 实验2 卷积神经网络图像分类

PB21081601 张芷苒

实验要求

使用pytorch 或者tensorflow 实现卷积神经网络CNN，在CIFAR-10 数据集上进行图片分类。研究**dropout、normalization、learning rate decay** 等模型训练技巧，以及**卷积核大小、网络深度**等超参数对分类性能的影响。

实验步骤

网络框架

pytorch, GPU.

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

数据集导入与划分

1. 预处理，提升数据质量，增加数据的多样性

```
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # 数据增强：图像随机水平翻转
    transforms.RandomCrop(32, padding=4),  # 数据增强：随机裁剪
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) # 归一化
])
```

2. 加载原训练集以及测试集

```
data_root = './data'
train_data = datasets.CIFAR10(root=data_root, train=True,
transform=transform, download=False) # 训练集
test_data = datasets.CIFAR10(root=data_root, train=False,
transform=transform, download=False) # 测试集
```

3. 按照实验要求把原训练集划分为互不相交的训练集和验证集

```
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)
```

CNN 模型搭建

搭建一个基本的CNN神经网络：

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(256 * 4 * 4, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 10)
        self.dropout = nn.Dropout(0.3)  # Dropout 概率 0.3

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = x.view(-1, 256 * 4 * 4)
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.dropout(F.relu(self.fc2(x)))
        x = self.fc3(x)
        return x
```

卷积层

3个卷积层`(conv1, conv2, conv3)`，输入通道分别为3，64，128，卷积核大小 3x3。填充(padding): 1 (在输入数据的边界添加一圈0值，使得输出特征图的空间尺寸不变)

池化层

池化: 使用最大池化层（MaxPooling）来减小特征图的空间尺寸（即宽度和高度），有助于减少计算量和控制过拟合，同时保持特征的主要部分。池化窗口为2x2，步长也为2。

全连接层

3个全连接层，`(fc1, fc2, fc3)`,第一个全连接层，将展平后的特征图连接到1024个神经元。`fc2`:第二个全连接层，将1024个神经元降维到512个神经元。`fc3`:最后一个全连接层，将512个神经元输出到10个神经元，每个对应一个类别的得分。

前向传播（`forward`方法）

数据在网络中的流动顺序为：

1. 通过第一卷积层及其批标准化和激活函数（ReLU），接着经过池化。
2. 经过第二卷积层的处理后，同样进行批标准化、ReLU激活和池化。
3. 第三卷积层处理后同样的流程。
4. 展平处理后的特征图（用于从卷积层到全连接层的转换）。
5. 通过全连接层，并在中间使用Dropout和ReLU激活。.
6. 最后一个全连接层输出最终的类别得分。

参数设置

对于网络深度、卷积核大小、normalization这几个参数，考虑到问题规模，由于不想增加网络复杂度，所以先挑选了常见的设置试一下。如果对dropout和学习率衰减的调整能满足实验要求，则不再对这三个量做改动。具体参数设置和调整详见“调参分析”。

调参分析

调参分析主要针对实验要求中的五个超参量：dropout, learning rate decay, noramalization, 网络深度，卷积核大小。

部分调试log可见代码最后的注释。

dropout

设置为x的dropout比率，意味着在训练过程中有x的神经元随机失活。这是一个正则化手段，用来防止模型过拟合。代码详见cnn网络构造部分。第一次训练时设置了0.5的dropout, 发现这可能会导致学习不足，再考虑到防止过拟合的需求，最后调整为0.3附近。

在0.3附近控制其他变量不变进行了dropout=0.28, 0.3, 0.32的3轮测试，发现预测效果0.32略逊，其他两组差不多，但是理论上说0.3的防止过拟合效果更佳。最后选择dropout = 0.3.

learning rate decay

第一次训练时故意把学习率衰减设置为每20个步长衰减0.8，而实验只跑了10个周期，因此这个学习率衰减是无效的。观察结果，发现在前3-4个 epoch 预测准确率都能有显著的提升（每次6%-7%左右），然而在 epoch 4-7 时会出现loss下降，accuracy停滞的现

象，再往后的几个epoch, accuracy 甚至会下降。这表明在后期出现了过拟合的现象。因此决定设置每3 epoch学习率衰减0.8进行测试。为了更好的观察学习率衰减产生的影响，将实验延长至14个epoch，这样可以进行4次衰减，并且第四次衰减后还可以再观察两个周期。

当设置`scheduler = StepLR(optimizer, step_size=3, gamma=0.8)`时，发现大体可以满足在准确率停滞或下降出现之前及时降低学习率、避免过拟合。再进行一些调整，换step_size为4，换gamma为0.78，0.82等，得到的效果都不如这种设置好。因此选择`scheduler = StepLR(optimizer, step_size=3, gamma=0.8)`。

网络深度 卷积核大小

按照问题规模设置为常见的3，3×3，并保持这个设置进行实验。在进行前两项参数的调整后，这个设置可以满足实验要求。为了不增加网络复杂程度，保持这一组设置。

normalization

`transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))`负责对图像进行归一化处理。它将每个颜色通道的像素值（在转换为张量后的0-1范围内）标准化，使其具有约为0的均值和0.5的标准差。
前面的`(0.5, 0.5, 0.5)`是均值参数，用于每个颜色通道（红、绿、蓝），后面的`(0.5, 0.5, 0.5)`是标准差（std）参数，每个颜色通道的像素值将被减去0.5并除以0.5。

这依然是一个比较常见的设置，在微调之后发现并不能对整体训练产生任何太大的提升，因此不予修改。

实验结果

确定最佳参数 **dropout = 0.3, learning rate decay = 0.8/3 epoch, noramalization = ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), 网络深度 = 3 ， 卷积核大小 =** 3×3。

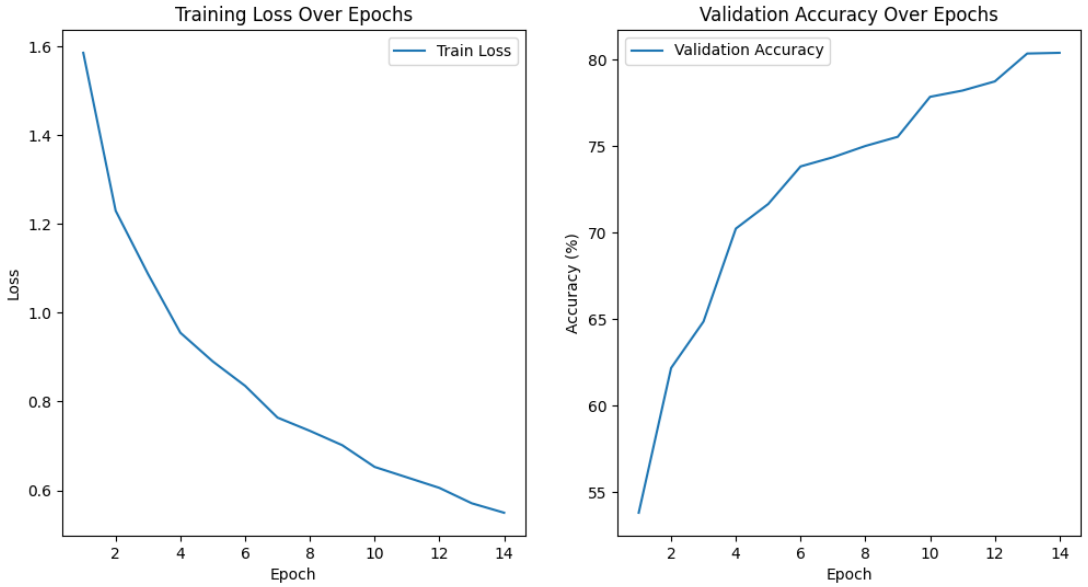
确定了这一组参数之后，加上测试集部分：

```
def test_model(model, test_loader, device):
    model.eval()  # 设置模型为评估模式
    total_correct = 0
    total_images = 0
    total_loss = 0.0
    criterion = nn.CrossEntropyLoss()  # 重新声明损失函数，用于计算测试损失

    with torch.no_grad():  # 不追踪梯度
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
```

```
            outputs = model(images)
            loss = criterion(outputs, labels)
            total_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total_images += labels.size(0)
            total_correct += (predicted == labels).sum().item()

    avg_loss = total_loss / len(test_loader)
    accuracy = (total_correct / total_images) * 100
    print(f'Test Loss: {avg_loss:.4f}, Test Accuracy: {accuracy:.2f}%')
    return avg_loss, accuracy
```

再次进行训练，得到以下结果：

```
Epoch 1/14 - Loss: 1.5857, Validation Accuracy: 53.82%
Epoch 2/14 - Loss: 1.2299, Validation Accuracy: 62.18%
Epoch 3/14 - Loss: 1.0875, Validation Accuracy: 64.86%
Epoch 4/14 - Loss: 0.9551, Validation Accuracy: 70.23%
Epoch 5/14 - Loss: 0.8905, Validation Accuracy: 71.65%
Epoch 6/14 - Loss: 0.8355, Validation Accuracy: 73.82%
Epoch 7/14 - Loss: 0.7639, Validation Accuracy: 74.35%
Epoch 8/14 - Loss: 0.7341, Validation Accuracy: 75.00%
Epoch 9/14 - Loss: 0.7018, Validation Accuracy: 75.53%
Epoch 10/14 - Loss: 0.6528, Validation Accuracy: 77.84%
Epoch 11/14 - Loss: 0.6293, Validation Accuracy: 78.20%
Epoch 12/14 - Loss: 0.6056, Validation Accuracy: 78.73%
Epoch 13/14 - Loss: 0.5709, Validation Accuracy: 80.34%
Epoch 14/14 - Loss: 0.5496, Validation Accuracy: 80.38%
Test Loss: 0.5767, Test Accuracy: 80.04%
Final Test Loss: 0.5767, Final Test Accuracy: 80.04%
```



可知，在14轮训练结束后，验证集和测试集都能达到80%以上的预测准确率。并且14轮训练过程中，没有出现accuracy震荡现象。这完成了实验要求。