LeNet 小作业

211098325 毛丁

2023 年 10 月 12 日

题目 1. 基于 Pytorch 实现 LeNet-5 ，并完成 CIFAR10 识别。可以尝试使用一些图像预处理技术（去噪，归一化，分割等），再使用神经网络进行特征提取。同时可以对训练过程进行可视化处理，分析训练趋势。需要提交的内容包括但不限于:

1. 可运行代码（LeNet.py），关键部分代码需要注释；
2. PDF 报告，报告中要有明确的实验过程说明、精度截图以及实验数据分析等。

解答. LeNet-5 是一个经典的深度卷积神经网络，由 Yann LeCun 在 1998 年提出，旨在解决手写数字识别问题。其输入为 32X32 的图像，构成主要为两个卷积层，两个池化层，以及三个全连接层组成。我以 LeNet 原始网络为骨架，使用 CIFAR-10 数据集，在 LeNet 原始骨架的基础上，并加以些许改进，最终在测试集达到了约 65% 的正确率。

实验步骤:

首先定义 LeNet 的网络结构，主要在第一次卷积和前两次全连接层间插入了一层 ReLU 激活函数

```python
class Lenet(nn.Module):
    def __init__(self):
        super(Lenet, self).__init__()
        self.model = nn.Sequential(
            nn.Conv2d(3, 6, 5), nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 16, 5),
            nn.MaxPool2d(2, 2),
            nn.Flatten(),
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10)
        )

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

图 1: 网络结构

然后获取 CIFAR-10 数据集，主要的数据预处理步骤为对图像做了归一化处理：

```python
# 下载与预处理数据集
trans = transforms.Compose(
    [transforms.Resize((32, 32)), transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
train_set = torchvision.datasets.CIFAR10(root="./data", train=True, download=True, transform=trans)
test_set = torchvision.datasets.CIFAR10(root="./data", train=False, download=True, transform=trans)
```

图 2: 获取并预处理数据集

损失函数选择交叉熵，优化器选择随机梯度下降，经过多次试验，采用的学习率为 0.01 以及动量为 0.9, 相关参数如下：

```
# 损失函数
loss_fn = nn.CrossEntropyLoss().to(device)
# 优化器
learning_rate = 0.01
optimizer = torch.optim.SGD(lenet.parameters(), lr=learning_rate, momentum=0.9)
```

图 3: 损失函数与优化器

采用训练轮数 epoch 为 10，batchsize 为 64 开始训练，使用 tensorboard 绘图，最终的训练结果的可视化如下：

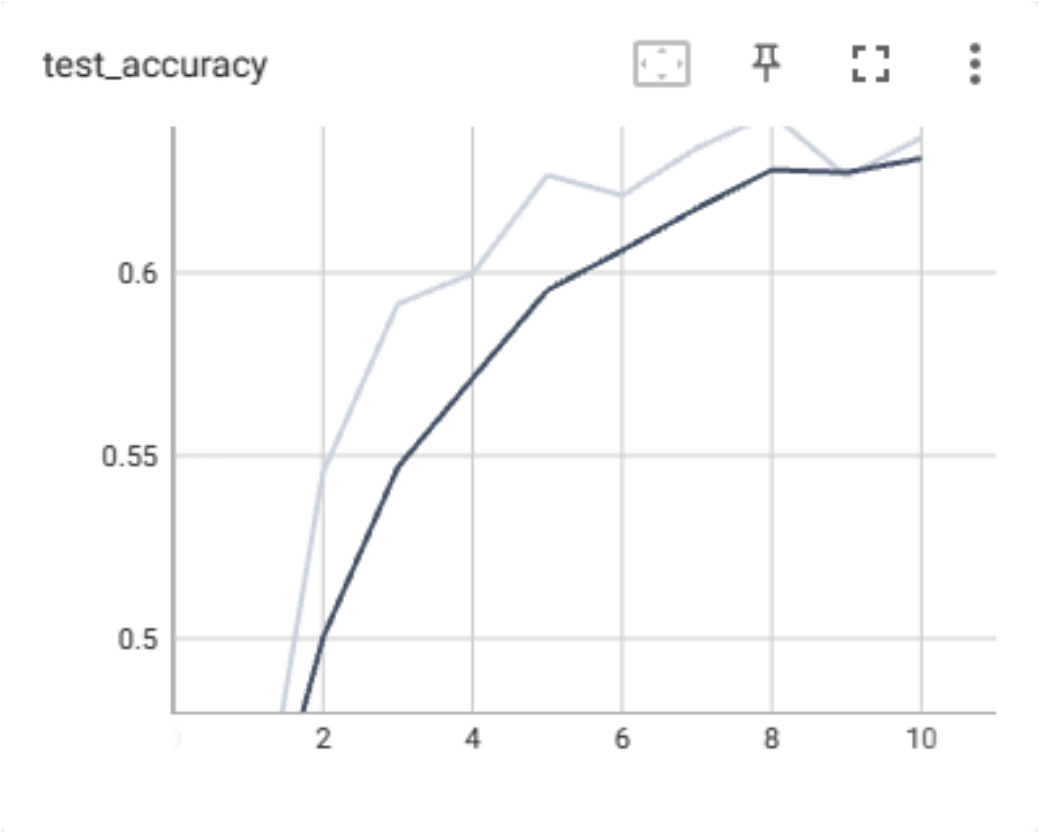

图 4: 测试集上的准确率

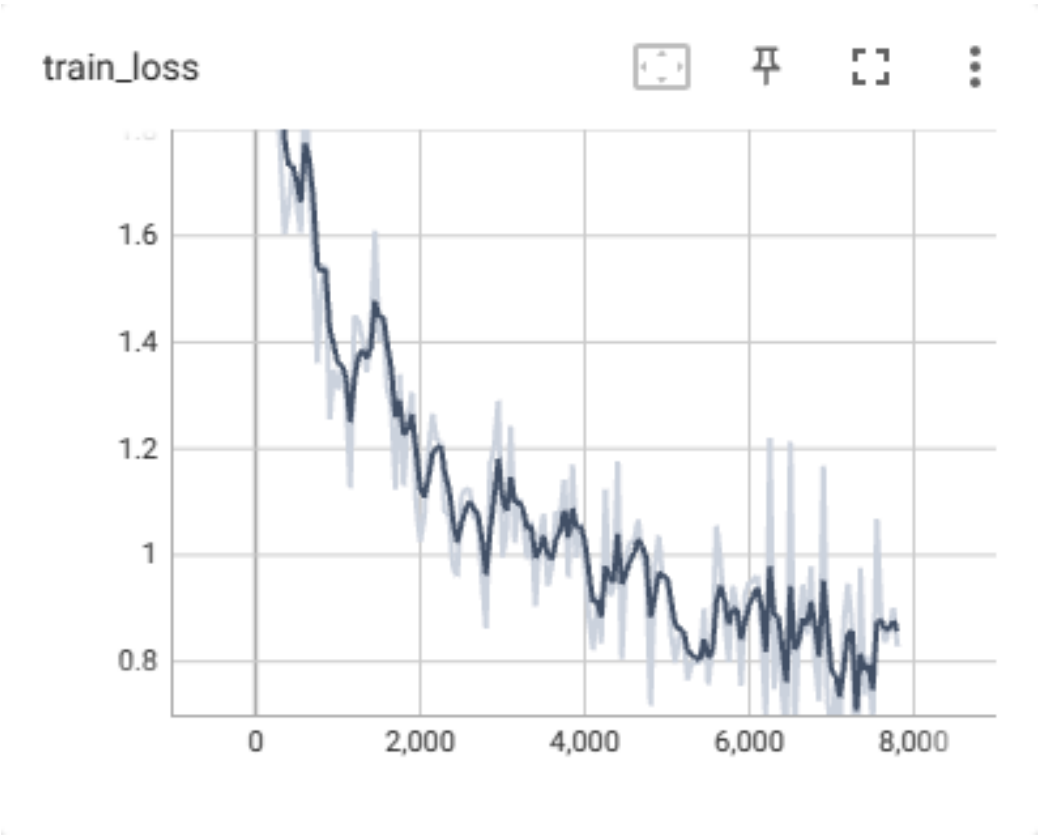

图 5: 每 50 次训练 step 的损失

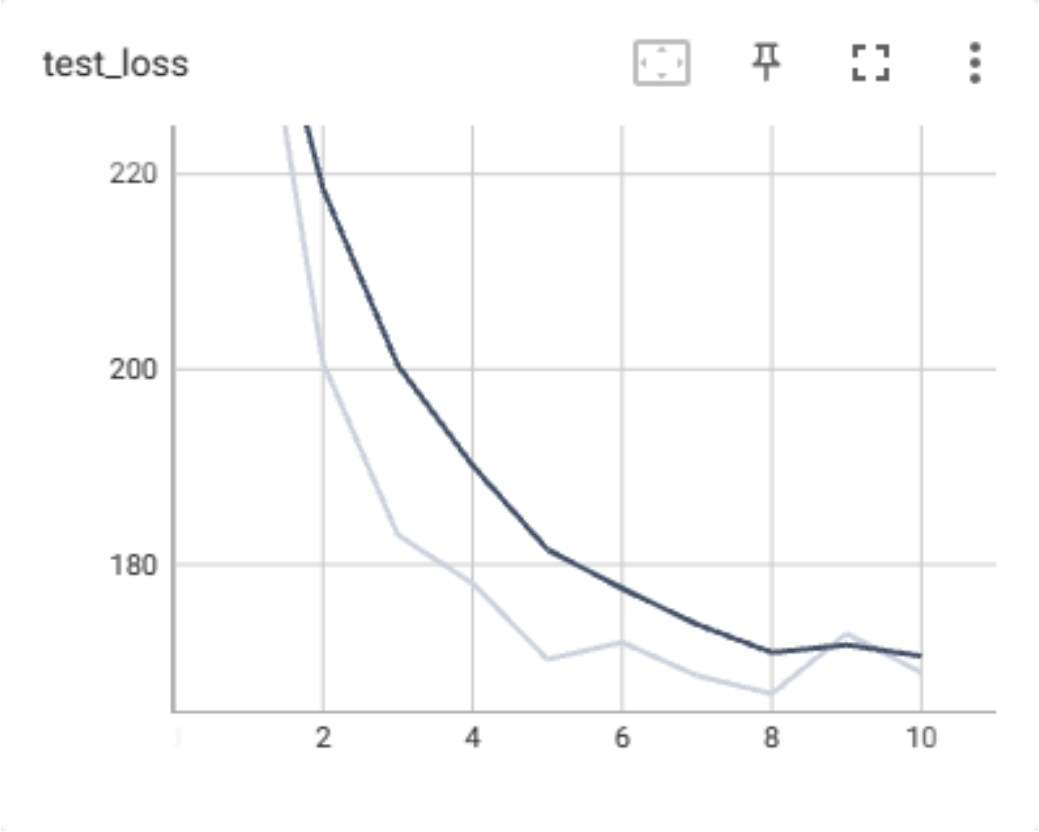

图 6: 每轮训练在测试集上的总 loss

实验数据分析:

通过 tensorboard 的图像可以看到，训练过程大概在第八轮左右得到收敛，由于 lenet 是一个较为简单的分类网络，因此我在调整学习率，调整训练轮数 (epoch) 等其他参数时未有明显的精度提升，反而还可能导致精度下降或不收敛的情况。后通过网上查找相关精度提升策略，在网络结构中加入 Relu 激活函数以及在 SGD 优化器中加入动量为 0.9 的设置，使训练精度在原始的 LeNet 模型精度下提升了 10% 左右。