华中科技大学

《基于 Python 的数据分析与机器学习》课程实验报告

项目名称	CIFAR-10 多分类问题
所在学院	电气与电子工程学院
团队成员	杜文杰 U202112476 罗士勇 U202112487 吴泽汝 U202112498
完成时间	2024 年 12 月 30 日

	成员 1	成员 2	成员 3
总 成 绩			
教师签名			

1. 研究综述

实验 CIFAR-10 数据集一共 10 类图片，每一类有 6000 张图片，每张图片的尺寸是 32x32，图片是彩色图。

本次实验将利用 torch.nn 搭建神经网络模型，利用上述数据集进行训练与测试，分析实验结果并绘制训练数据集和测试集的 Loss 曲线，网络结构采用卷积神经网络。

对比的实验内容包括：

- 对比 3 组超参数
- 使用三种不同的激活函数
- 实现一种解决过拟合的方法

2. 团队成员及分工

本团队包含 3 名成员，具体分工是：

杜文杰同学负责基础的卷积神经网络搭建与训练实验内容，完成报告编程语言、机器学习/深度学习核心算法、团队成员及分工部分内容;

罗士勇同学负责算法改建优化，进行不同激活函数、超参数对比的实验内容，完成报告数据预处理、实验过程部分内容;

吴泽汝同学负责数据预处理和结果分析实验内容，完成报告研究综述、结果展示、实验总结部分内容。

3. 编程语言

本实验采用 Python 编程语言，将调用 torch 库、torchvision 库、malplotlib 库、numpy 库、datetime 库。

其中，机器学习核心算法用到了 torch 库和 torchvision 库。绘图用到了 matplotlib 库和 numpy 库。计算训练时间用到了 datetime 库。

4. 机器学习/深度学习核心算法

本实验将采用卷积神经网络算法，其核心思想是卷积、池化、全连接等。架构如图 1 所示。

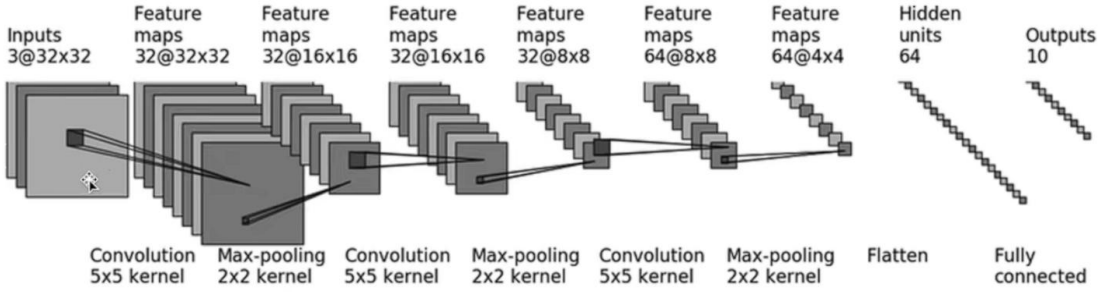

图 1 CIFAR-10 分类问题算法架构

网络架构：

```
# 构建模型
class Net(nn.Module):
    # """
    # 使用 sequential 构建网络，Sequential()函数的功能是将网络的层组合到一起
    # """
    def __init__(self):
        super(Net, self).__init__()
        self.model = nn.Sequential(
            #lenet-5
            nn.Conv2d(3, 32, 5, 1, 2),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 32, 5, 1, 2),
            nn.MaxPool2d(2),
            nn.Conv2d(32, 64, 5, 1, 2),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(64*4*4, 64),
            # 激活函数
            nn.ReLU(True),
            #新增的隐藏层
            # nn.Linear(256, 64),
            # 激活函数
            # nn.ReLU(True),
            nn.Linear(64, 10),
            nn.ReLU(True),
        )
    def forward(self, x):
        x = self.model(x)
        return x

#创建模型实例
```

```
model = Net().to(device)
#  损失函数
criterion = nn.CrossEntropyLoss().to(device)
#  优化器
# optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate,
momentum=momentum)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, betas=(0.9,
0.99))
```

训练算法；

```
#  训练模型并绘制 Loss 函数曲线
print('start training')
iterations = 0
#记录数据
losses = []
acces = []
eval_losses = []
eval_acces = []

#开始计时
start = datetime.datetime.now()

for epoch in range(num_epoches):
    train_loss = 0
    train_acc = 0
    model.train()
    #动态修改学习率
    if epoch%5==0:
        optimizer.param_groups[0]['lr']*=0.9
        print('lr is set to {}'.format(optimizer.param_groups[0]['lr']))
    for img,label in train_loader:
        img,label = img.to(device),label.to(device)
        #前向传播
        out = model(img)
        #计算损失
        loss = criterion(out,label)
        #反向传播
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        #记录误差
        train_loss += loss.item()
        #计算准确率
```

```
            _,pred = torch.max(out,1)
            num_correct = (pred == label).sum().item()
            acc = num_correct/img.shape[0]
            train_acc += acc
        losses.append(train_loss/len(train_loader))
        acces.append(train_acc/len(train_loader))
        #在测试集上测试
        eval_loss = 0
        eval_acc = 0

        model.eval()
        for img,label in test_loader:
            img,label = img.to(device),label.to(device)
            out = model(img)
            loss = criterion(out,label)
            #记录误差
            eval_loss += loss.item()
            #计算准确率
            _,pred = torch.max(out,1)
            num_correct = (pred == label).sum().item()
            acc = num_correct/img.shape[0]
            eval_acc += acc
        eval_losses.append(eval_loss/len(test_loader))
        eval_acces.append(eval_acc/len(test_loader))
        print('epoch: {}, Train Loss: {:.6f}, Train Acc: {:.6f}, Test Loss: {:.6f}, Test
Acc: {:.6f}'
                .format(epoch, train_loss/len(train_loader), train_acc/len(train_loader),
                        eval_loss/len(test_loader), eval_acc/len(test_loader)))
#结束计时
end = datetime.datetime.now()
# 绘制损失函数曲线
plt.title('train loss')
plt.plot(np.arange(len(losses)), losses, "blue", label="Training Loss")
plt.plot(np.arange(len(eval_losses)), eval_losses, "red", label="Test Loss")
plt.xlim([0,epoch])
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend(['Train Loss', 'Test Loss'], loc='upper right')
plt.savefig('./loss_adam.png')
# 计算训练时间
print('time used: ', end-start)
torch.save(model,'./model/model_momentum.ckpt')
```

5. 数据预处理

本实验数据集从 torch 下载，训练集包含 60000 张图片，测试集包含 10000 张图片

预处理代码：

```
# 定义预处理函数
transform = transforms.Compose([
    #将图像数据转换为张量
    transforms.ToTensor(),
    #将图像数据归一化到[-1,1]
    #transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    # #随机水平翻转
    # transforms.RandomHorizontalFlip(),
    # #随机遮挡
    # transforms.RandomErasing(scale=(0.04,0.2), ratio=(0.5,2)),
    # #随机裁剪
    # transforms.RandomCrop(32, padding=4)
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
])
#下载数据，并对数据进行预处理
mnist_train= torchvision.datasets.CIFAR10(root='./data', train=True,
transform=transform,download=True)
mnist_test= torchvision.datasets.CIFAR10(root='./data', train=False,
transform=transform_test,download=True)
```

6. 实验过程

库调用部分：torch 库与 torchvision 库用于实现机器学习核心算法；matplotlib 库与 numpy 库用于绘制所需图像；datetime 库用于记录训练时间。

超参数定义部分：**默认状况下**，训练批次大小与测试批次大小设置为 64；学习率初始值设置为 0.01；迭代次数设置为 100；动量设置为 0；丢弃率设置为 0.5。

数据预处理部分：定义预处理函数，将图像数据转换为张量后归一化至[-1,1]；下载数据后，对数据进行预处理，并得到生成器。

模型构建部分：**默认状况下**，采用 sequential 构建网络；设置激活函数与隐藏层数，激活函数为 ReLU 函数；选取损失函数与优化器算法。

模型训练部分：动态修改学习率；记录误差与准确率等。

模型测试部分：测试模型在一个或数个 batch 的分类情况，并输出图像及对应预测名称，打印正确率。

可视化数据部分：显示测试集图像；绘制 Loss 函数曲线；绘制准确率曲线。

通过修改各个部分参数，完成实验要求，结果展示在第 7 小节。

7. 结果展示

7.1 超参数对比

7.1.1 增加隐藏层层数： 增加一层隐藏层，其他相比默认状况不变。

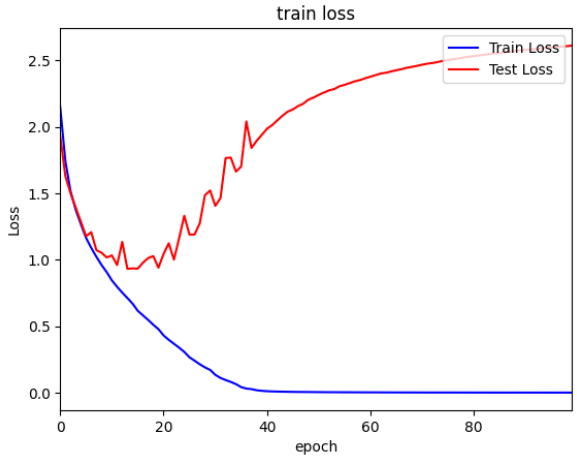

图 2

图 3

Test Loss: 2.609220, Test Acc: 0.701500

7.1.2 改变批量大小：由 64 增加为 128，其他相比默认状况不变。

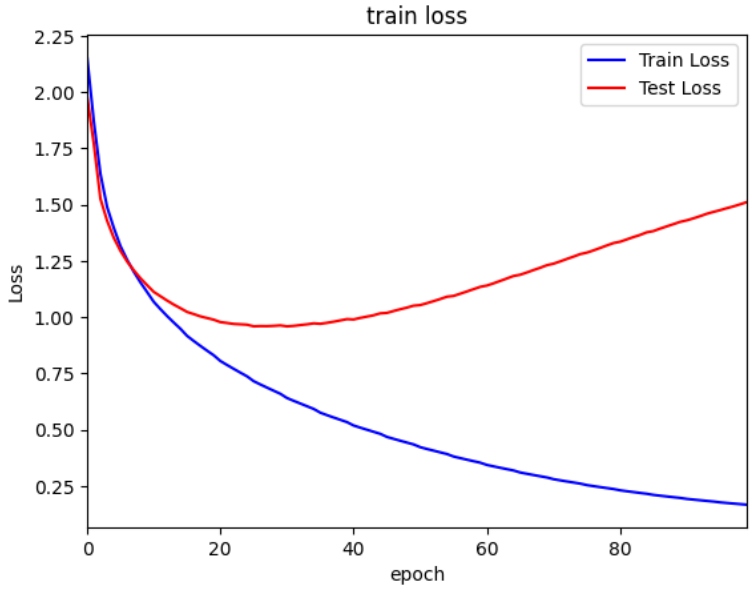

图 4

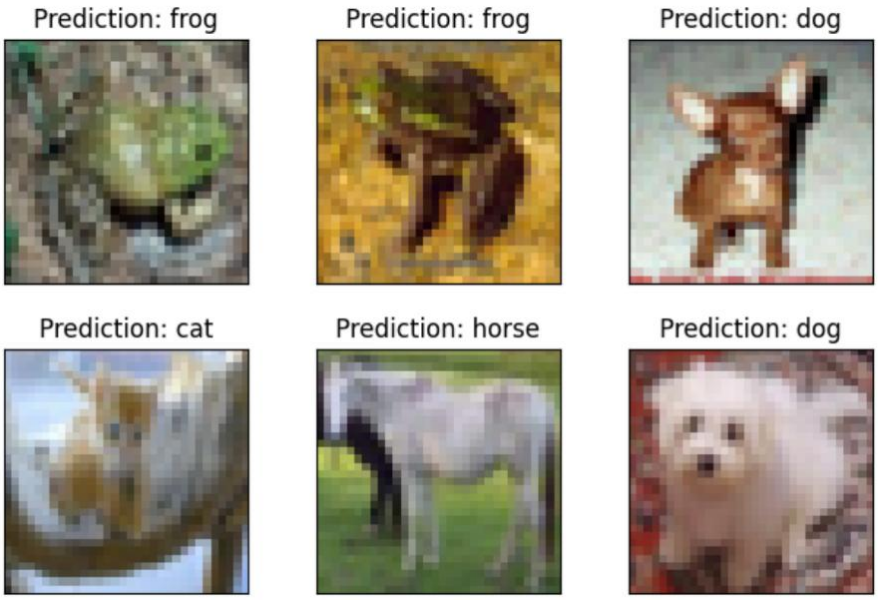

图 5

Test Loss: 1.513551, Test Acc: 0.674000

7.1.3 改变动量：由 0 改为 0.5，其他相比默认状况不变。

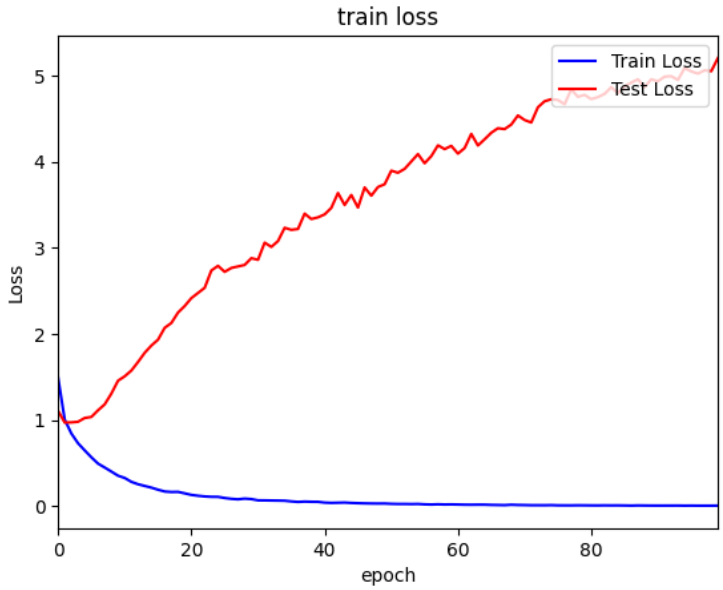

图 6

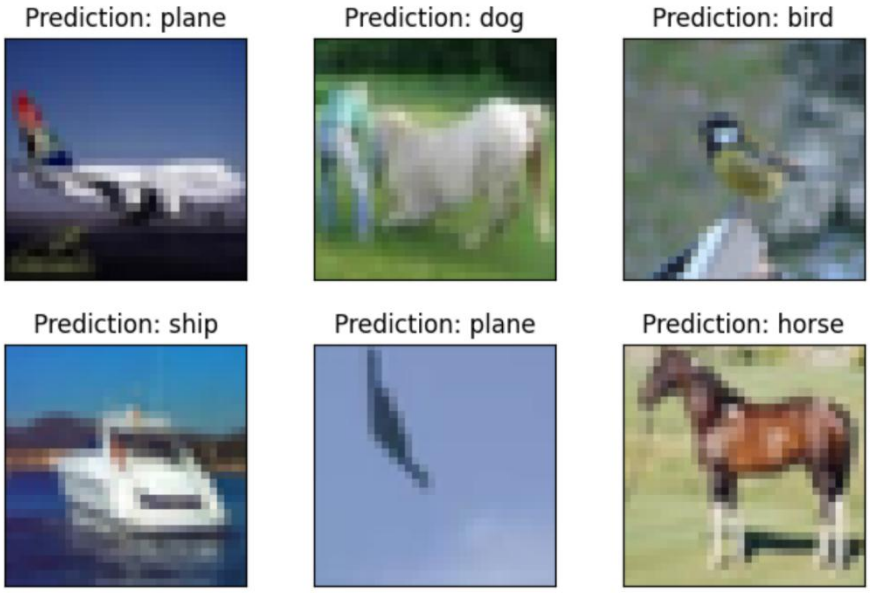

图 7

Test Loss: 5.204598, Test Acc: 0.682200

7.2 激活函数对比

7.2.1 采用 Sigmoid 函数，其他相比默认状况不变。

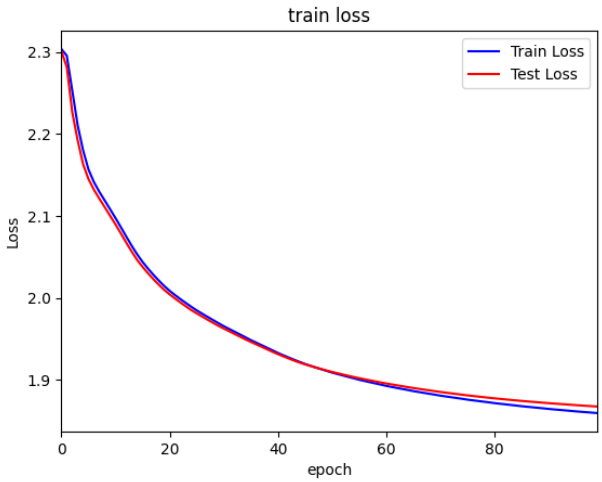

图 8

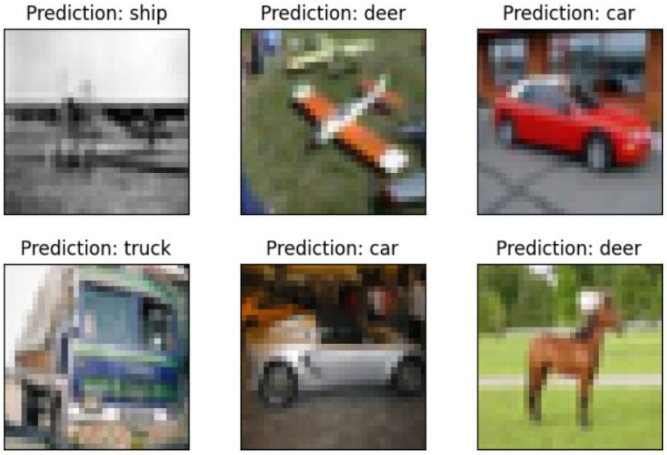

图 9

Test Loss: 1.867585, Test Acc: 0.505500

7.2.2 采用 ReLU 函数，为默认状况。

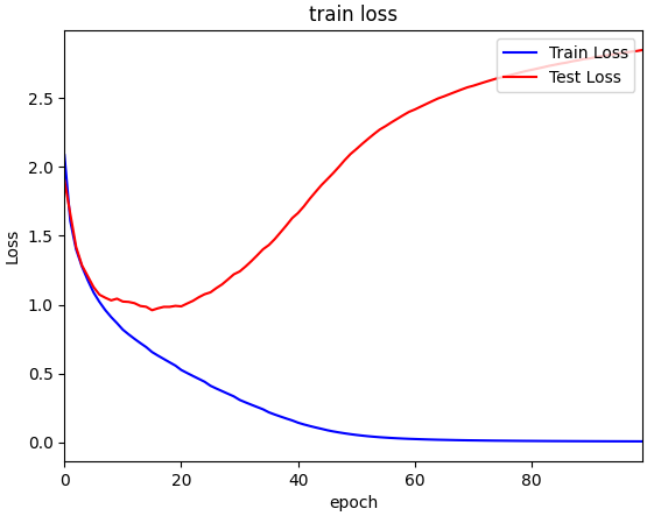

图 10

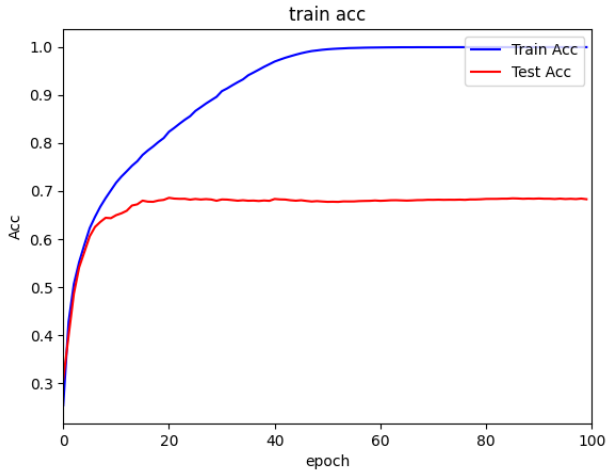

图 11

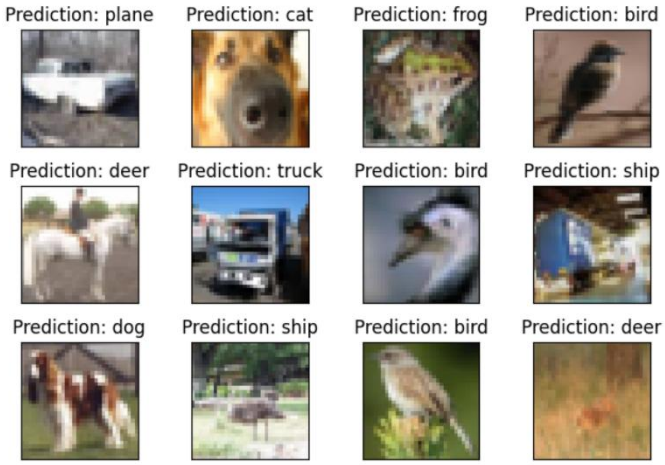

图 12

Test Loss: 2.849063, Test Acc: 0.682900

7.2.3 采用 leakyReLU 函数，其他相比默认状况不变。

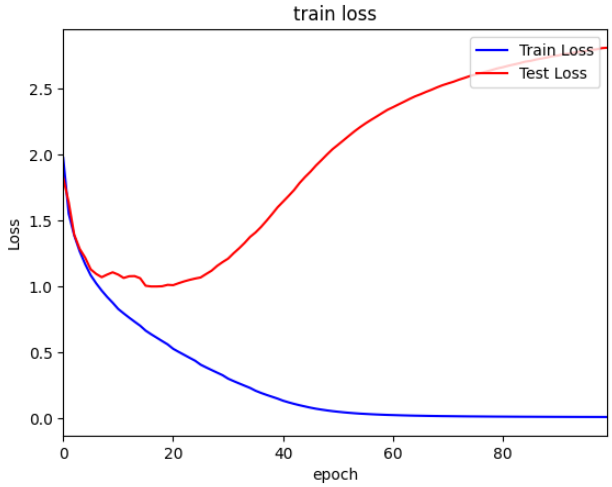

图 13

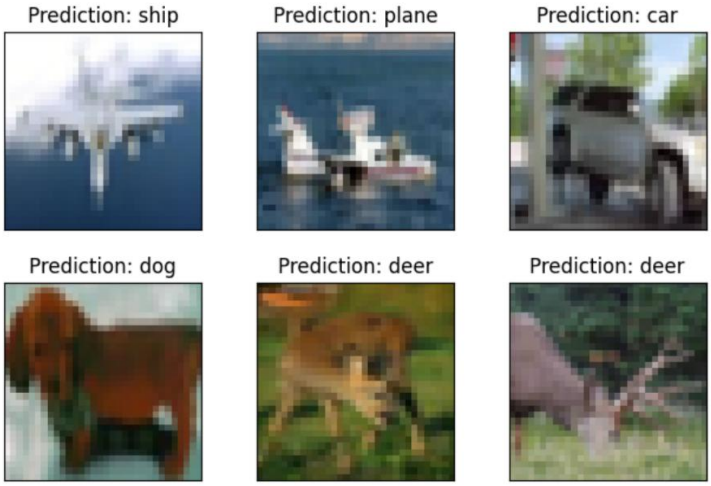

图 14

Test Loss: 2.798215, Test Acc: 0.685400

7.3 解决过拟合：采用 Dropout 方法，其他相比默认状况不变。

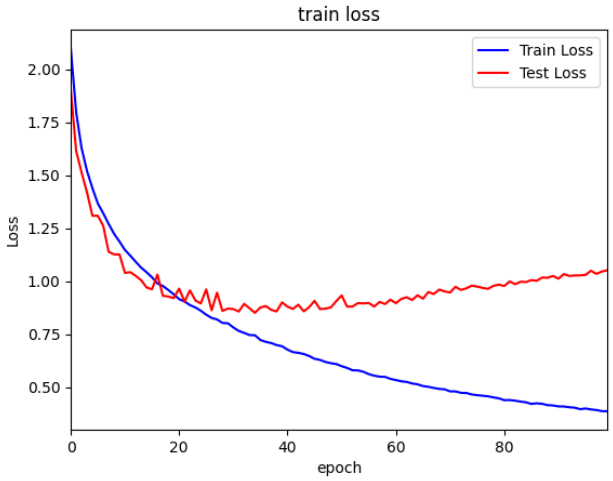

图 15

图 16

Test Loss: 1.048531, Test Acc: 0.720300

8. 实验总结

通过本次实验，我们基本完成了基于卷积神经网络的 CIFAR-10 多分类问题。通过实验，我们获得了以下主要收获与经验：

- 模型优化与对比分析：

在网络结构上，我们尝试了增加隐藏层的深度，优化了网络的表示能力，但也发现层数的增加对模型收敛速度与计算资源需求有显著影响。

通过对比不同批量大小，验证了较小批量适合细粒度的权重更新，而较大批量有助于稳定训练。

动量的引入对梯度下降过程的优化效果显著，通过调整动量参数，我们平衡了收敛速度与震荡程度。

- 激活函数的影响：

激活函数对模型的性能有重要作用，ReLU 函数在我们的实验中表现最佳，既能够加速模型训练，又有效缓解梯度消失问题。

相较于 Sigmoid 和 leakyReLU，ReLU 的稀疏激活特性使其在深度神经网络中具有优势。

- 过拟合问题的处理：

Dropout 作为一种简单有效的正则化方法，在抑制过拟合方面效果显著。通过随机丢弃神经元，我们有效提升了模型在测试集上的泛化能力。

通过以上实验与总结，我们不仅掌握了卷积神经网络的基本原理和应用方法，还了解了超参数调整和模型优化的重要性，为今后更复杂的深度学习任务打下了坚实的基础。