《模式识别与机器学习 A》实验报告

实验题目：卷积神经网络

学　　号：2021112845

姓　　名：张智雄

实验报告内容

1、 实验目的

采用任意一种课程中介绍过的或者其它卷积神经网络模型（例如 LeNet-5、AlexNet 等）用于解决某种媒体类型的模式识别问题。

2、 实验内容

1) 卷积神经网络可以基于现有框架如 TensorFlow、Pytorch 或者 Mindspore 等构建，也可以自行设计实现。

2) 数据集可以使用手写体数字图像标准数据集，也可以自行构建。预测问题可以包括分类或者回归等。实验工作还需要对激活函数的选择、dropout 等技巧的使用做实验分析。必要时上网查找有关参考文献。

3) 用不同数据量，不同超参数，比较实验效果，并给出截图和分析

3、 实验环境

Windows11; Anaconda+python3.11; VS Code

4、 实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）

4.1 算法理论支撑

4.1.1 卷积神经网络(CNN)的基本原理

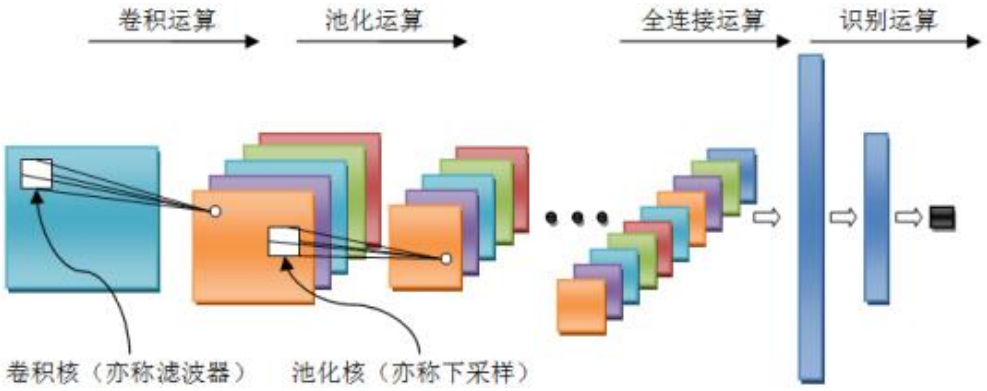

图 1 卷积神经网络模型结构

卷积神经网络(Convolutional Neural Network，CNN)是一种深度学习模型，特别设计用于处理和分析具有网格结构的数据，如图像和视频。它能够自动学习图像中的特征并进行高效的图像分类、对象检测、图像生成和分割等任务，其模型结构主要包含以下部分：

a) 卷积层: 卷积层负责从图像中提取特征，如边缘和纹理。它们通过应用过滤器来捕捉这些特征，逐渐形成更复杂的视觉模式。

b) 池化层: 池化层在保留基本信息的同时减小了特征图的大小。最常见的方法是最大池化，它有助于缩小图像，同时保持关键特征并增强鲁棒性。

c) 全连接层: 全连接层结合从前一层提取的特征进行分类和决策。他们将这些特征映射到不同的类别，识别图像中的内容。

4.1.2 AlexNet 的基本结构

AlexNet 网络结构相对简单，使用了 8 层卷积神经网络，前 5 层是卷积层，剩下的 3 层是全连接层，具体如下图 2 所示。

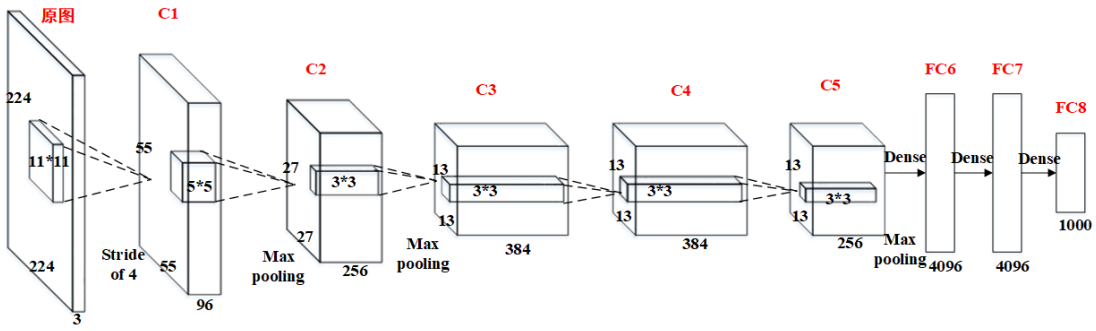

图 2 AlexNet 判别器结构

与原始的 LeNet 相比，AlexNet 网络结构更深，同时还包括以下特点：

a) ReLU 激活函数的引入：采用修正线性单元(ReLU)的深度卷积神经网络能够大幅提高训练速度，同时能够有效防止过拟合现象的出现。

b) 层叠池化操作：AlexNet 中池化层采用了层叠池化操作，即池化大小>步长，这种类卷积操作可以使相邻像素间产生信息交互和保留必要的联系。

c) Dropout 操作：Dropout 操作会将概率小于 0.5 的每个隐层神经元的输出设为 0，即去掉一些神经节点，能够有效防止过拟合现象的出现。

4.2 实验设计

4.2.1 实验数据集及数据预处理

MNIST 数据集(Mixed National Institute of Standards and Technology database)是美国国家标准与技术研究院收集整理的大型手写数字数据集，包含 60,000 个样本的训练集以及 10,000 个样本的测试集。其中包括 0 到 9 的数字。

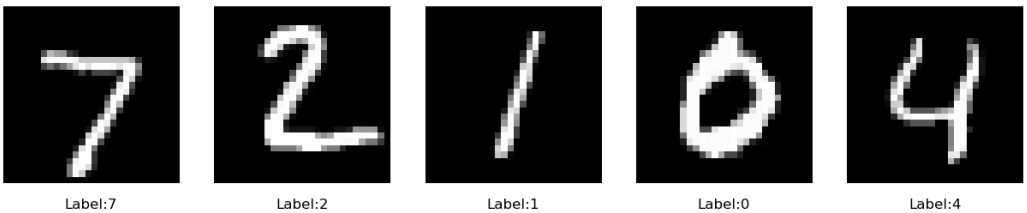

图 3 MNIST 数据集

在本实验中，使用 torchvision 自带的数据集加载 MNIST 和 CIFAR-10 数据集，并使用 transforms.ToTensor 方法加载为 Tensor 张量，最后通过 DataLoader 加载进 GPU 进行运算。

```python
def data_processing(str):
    if str == 'mnist':
        transform = transforms.ToTensor()  # 转换为张量
        trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
        trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

        testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
        testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False)
        return trainloader, testloader
    elif str == 'cifar':
        transform = transforms.ToTensor()  # 转换为张量
        trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
        trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True)

        testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
        testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False)
        return trainloader, testloader
    return None, None
```

图 4 数据预处理代码截图

4.2.2 模型设计

在本次实验中，仿照 AlexNet，实现了包含五个卷积层和三个全连接层构建一个深度卷积神经网络，网络的定义是重写 nn.Module 实现的，卷积层和全连接层之间将数据通过 view 拉平，同时可选择加入 Dropout 层防止数据过拟合。

Feature map 数变化：$1 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256$，卷积核 kernel size 均为 3，同时在边缘填充单位长度的 0，步幅均为 1。

MaxPooling 核大小为2×2，每次将特征图大小缩为原来的一半。

```python
class AlexNet(nn.Module):
    def __init__(self, width_mult=1):
        super(AlexNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),  # 32*28*28
            nn.MaxPool2d(kernel_size=2, stride=2),  # 32*14*14
            nn.ReLU(inplace=True),
        )
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, padding=1),  # 64*14*14
            nn.MaxPool2d(kernel_size=2, stride=2),  # 64*7*7
            nn.ReLU(inplace=True),
        )
        self.layer3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)# 128*7*7
        self.layer4 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, padding=1),  # 256*7*7
        )
        self.layer5 = nn.Sequential(
            nn.Conv2d(256, 256, kernel_size=3, padding=1),  # 256*7*7
            nn.MaxPool2d(kernel_size=3, stride=2),  # 256*3*3
            nn.ReLU(inplace=True),
        )
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(256 * 3 * 3, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 10)
```

图 5 AlexNet 模型结构代码

4.3 实验结果及分析

4.3.1 实验结果

在本次实验中，使用交叉熵损失函数和 SGD 优化器，激活函数采用 ReLU，将模型输入通道根据数据集设为1，并设置训练超参数epoch为10，batch size为128，学习率learning rate为0.01。训练过程中损失函数*loss*的值和在测试集上的准确率变化如下图所示。

实验发现，随训练过程的进行，损失函数不断降低，在测试集上准确率逐渐升高，最终测试正确率最高能够达到约98.94%。损失函数和测试准确率在训练最后阶段呈现波动态，可能原因是在局部最优点附近振荡。

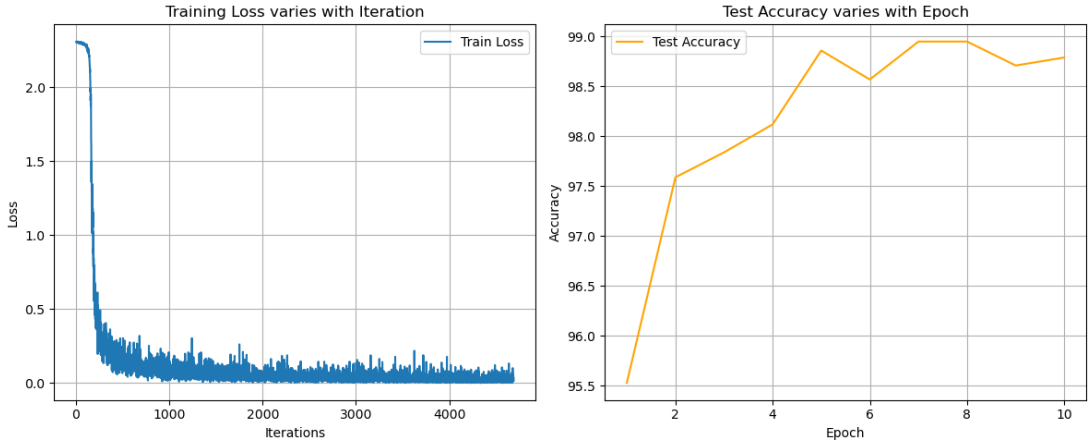

图 6 MNIST 实验结果（*左为损失变化，右为测试集上准确率*）

而后通过 torch.load 方法加载模型对测试集进行直观展示，模型能够对手写数字作出较为准确的分类，具有一定的泛化能力。

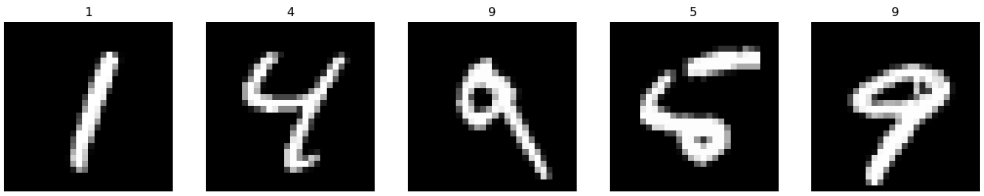

图 7 测试集上分类效果

4.3.2 不同激活函数的比较

将所有激活函数换为 Sigmoid 函数，发现结果很差，损失函数强烈震荡，几乎毫无效果。分析原因可能为：

a) **梯度消失：**Sigmoid 在输入极值附近的梯度接近于零，这可能导致梯度消失问题，特别是在深层网络中。这可能会影响网络的训练效率和能力。

b) **输出偏移：**Sigmoid 函数的输出在 0 到 1 之间，这意味着它倾向于产生偏向于 0 或 1 的输出，这可能在梯度下降过程中导致网络权重的不稳定更新。

c) **非稀疏性：**与 ReLU 不同，Sigmoid 的输出不稀疏，因为它在整个输入范围内都有非零输出。这可能导致网络的表示能力受到限制。

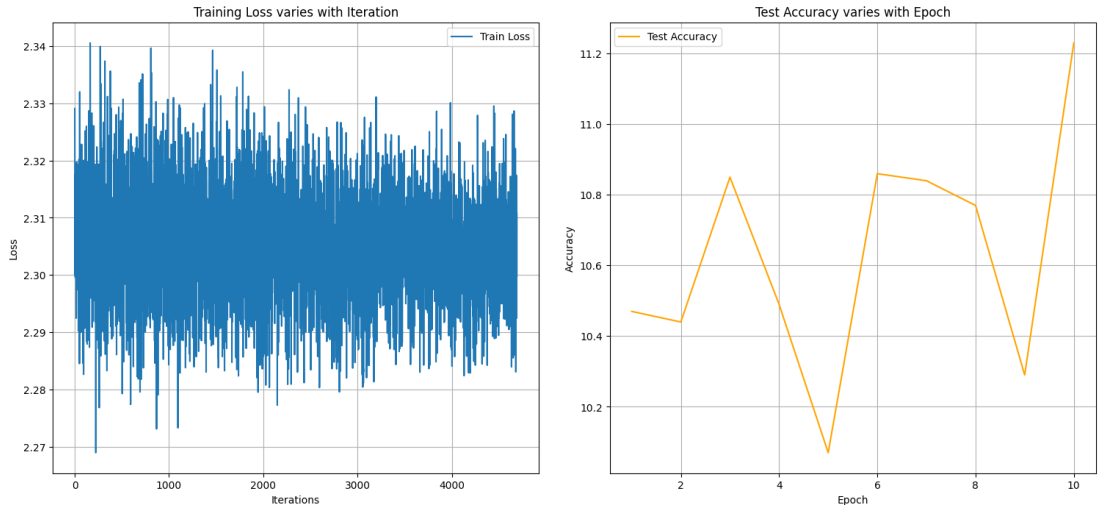

图 8 MNIST 实验结果（Sigmoid 损失函数）

而将所有激活函数换为 LeakyReLU 函数，发现结果有一定提升，最高能够达到99.13%左右，且收敛速度较快，原因可能为：

传统的 ReLU 在负数输入时输出为零，这可能导致梯度在训练过程中变得非常小或者为零，称为梯度消失。Leaky ReLU 引入了一个小的负数斜率，使得梯度在负数输入时仍然存在，从而导致更均匀的梯度分布，可以减少训练过程中的梯度爆炸问题，并使权重更新更加平滑。

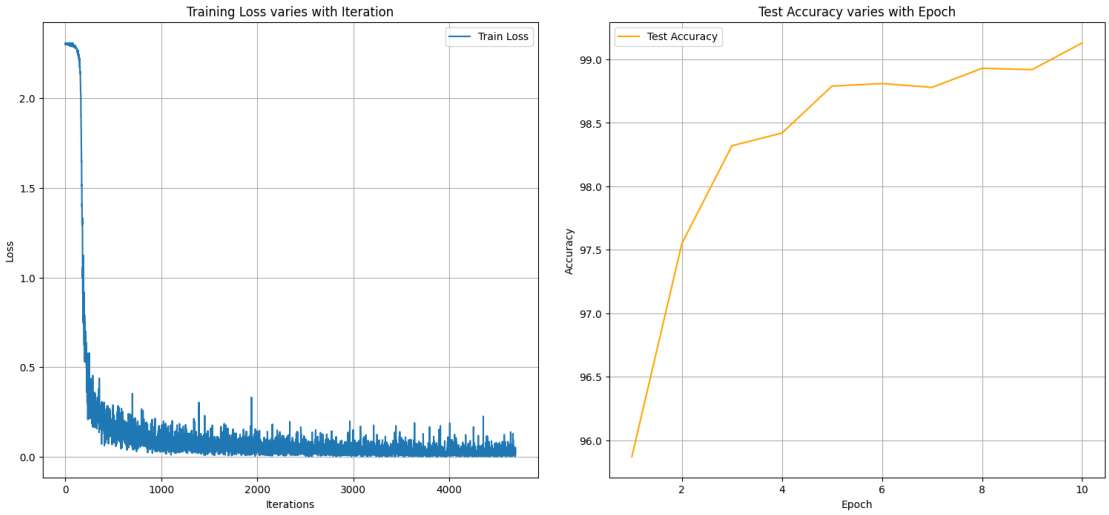

图 9 MNIST 实验结果（LeakyReLU 损失函数）

4. 3. 3 Dropout 操作的比较

上述 4.3.1 节中在最后的三层线性连接层间加入了两次 Dropout 操作，去掉模型传播过程中的 Dropout 操作，实验结果如下：

Dropout 会随机选择一部分神经元进行“丢弃”，目的是降低模型对训练数据的过于依赖，使模型更具泛化能力。但在 MNIST 数据集上意义不大的原因可能为，数据量足够充足且具有一定的多样性，本身就具有较强的泛化能力，从而不容易过拟合。

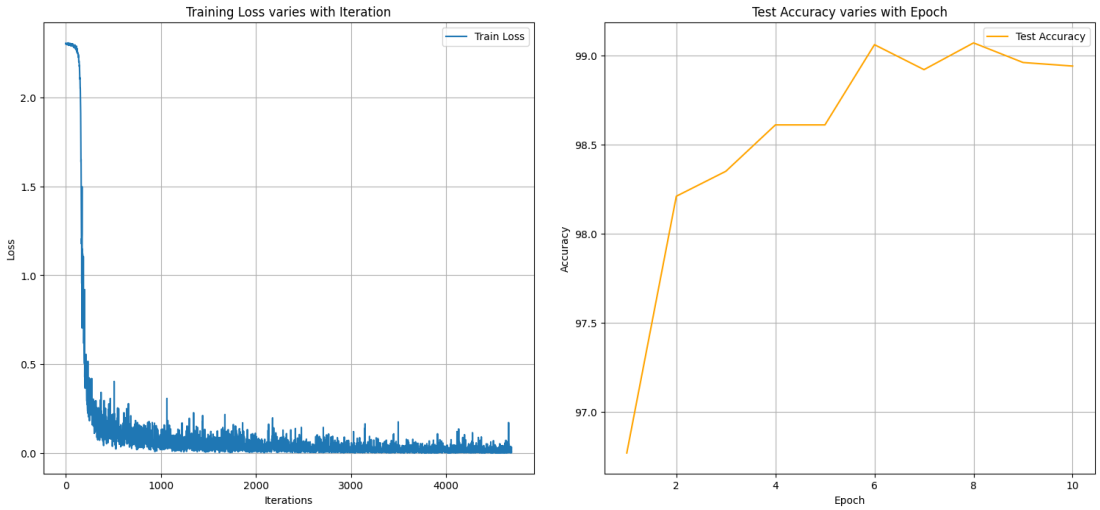

图 10 MNIST 实验结果（去掉 Dropout 层）

4.3.4 不同数据量的比较

减少数据集为 10000 组，同样设置训练超参数epoch为10，batch size为128，学习率learning rate为0.01。训练过程中损失函数*loss*的值和在测试集上的准确率变化如下图所示：

此时正确率相较于 60000 组有所下降，同时收敛速度也较为下降（达到较高正确率所需 epoch 较大）。

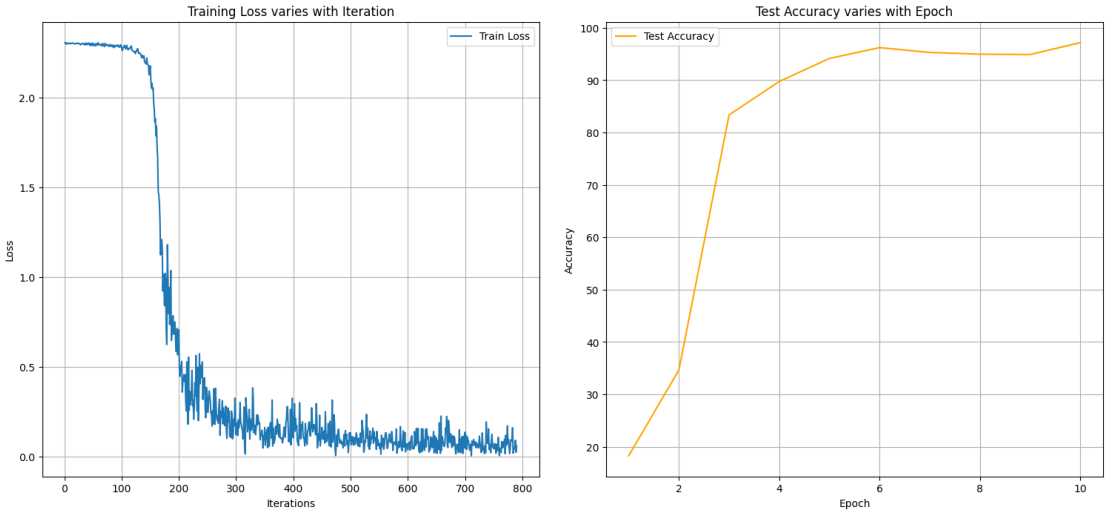

图 11 MNIST 实验结果（数据量为 10000）

而当数据量减少至 2500 时，此时正确率进一步下降。此时加入 Dropout 层也没有提升，推测原因可能是未收敛，或者 MNIST 数据集本身较为简单，训练集和测试集相似度高，一般不会出现过拟合现象。

但是，反复试验发现，加入 Dropout 层的训练更为平滑，loss 函数和 accuracy 函数变化曲线震荡都有一定程度的减弱，这也许时加入 Dropout 层后，网络不会过于依赖特定的神经元，从而减少了网络对训练数据的记忆和复杂的共适应。

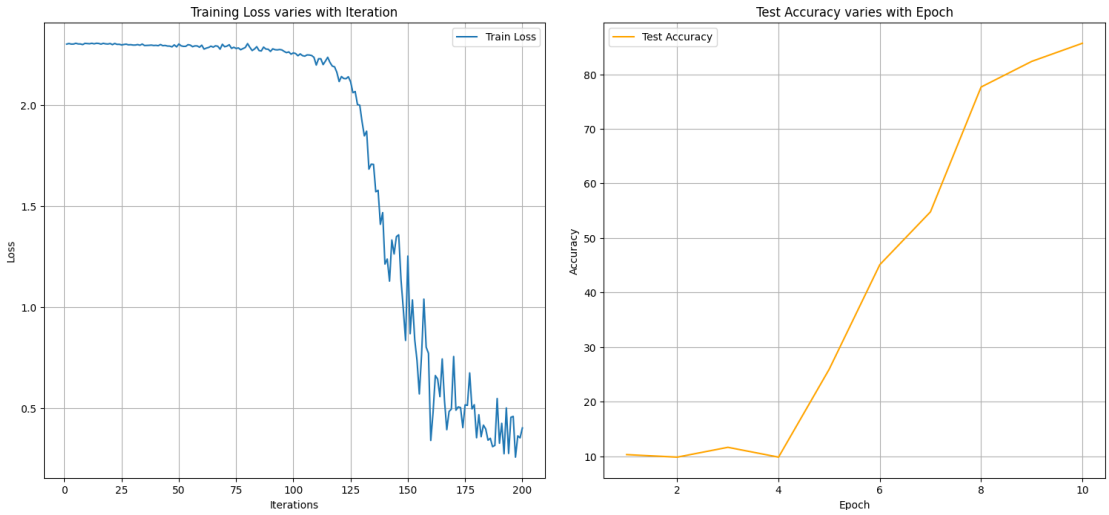

图 12 MNIST 实验结果（数据量为 2500）

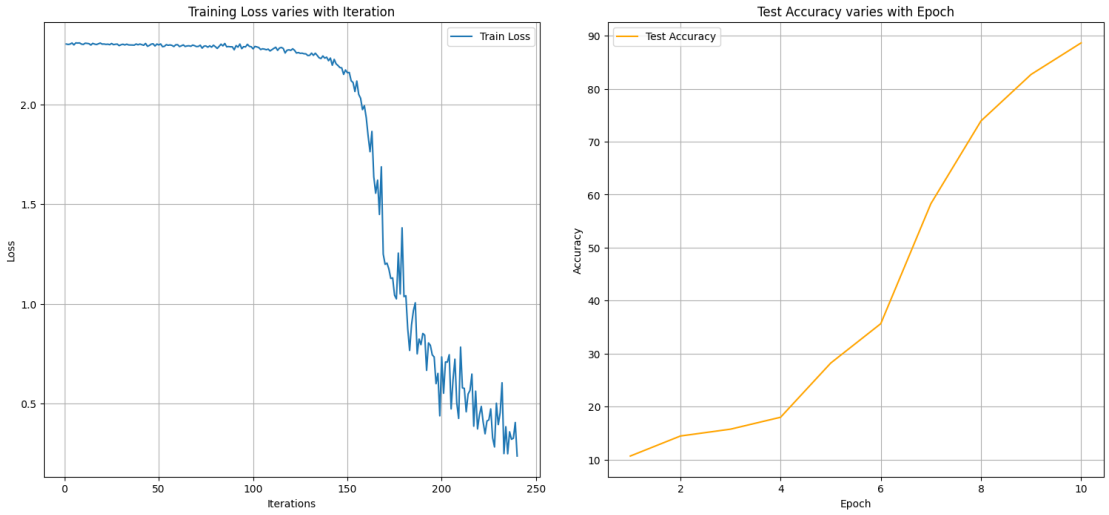

图 13 MNIST 实验结果（数据量为 2500，加入 Dropout 层）

4.3.5 不同超参数的比较

针对 Alexnet，分别修改其**学习率、批处理大小**等超参数，训练轮数由于已经体现在上述实验结果中不再比较，观察实验结果如下：

a) **学习率**

设置学习率为 0.1，发现模型损失直接爆掉了 nan，效果很差，这是因为学习率设置过大，模型在更新参数时可能会跳过最优值，导致梯度爆炸而无法收敛到全局最优解。

```
Epoch 0 accuracy: 9.800000 loss: nan
Saving Best Model with Accuracy:  9.800000190734863
Epoch: 1 Accuracy : 9.800000190734863 %
Epoch 1 accuracy: 9.800000 loss: nan
Epoch: 2 Accuracy : 9.800000190734863 %
Epoch 2 accuracy: 9.800000 loss: nan
Epoch: 3 Accuracy : 9.800000190734863 %
```

图 14 梯度爆炸

设置学习率为 0.001，模型在性能上差距不大，但收敛速度大幅下降，这是因为每次迭代更新步长较小，参数逼近到最优解较慢。

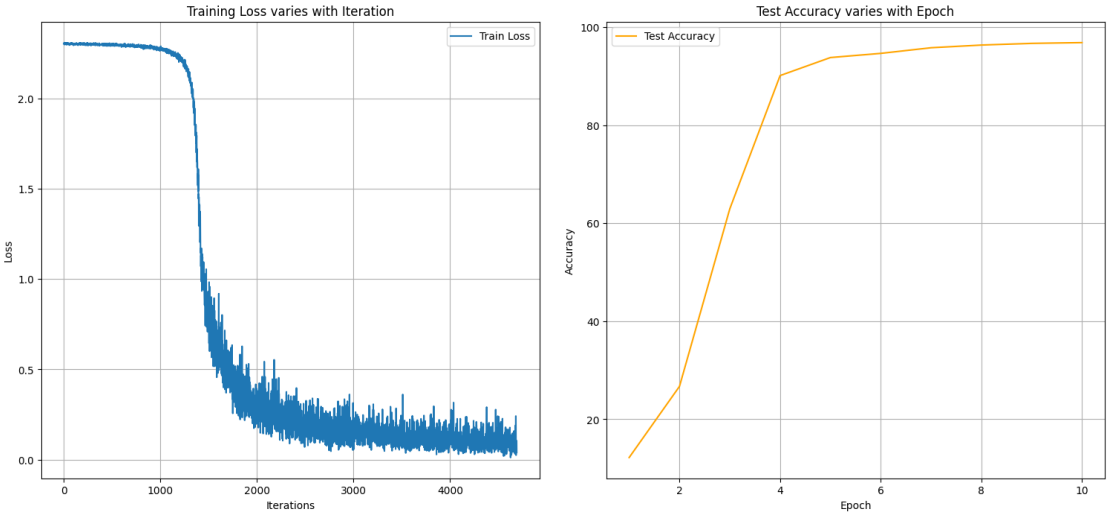

图 15 MNIST 实验结果（lr = 0.001）

b) **批处理大小**

分别设置 batch size 为 256 和 64 与上述实验结果进行比较：

实验发现，使用 batch size 为 64 时，准确率略有提高，能够达到 99.27%左右，训练时间相对增加；而使用 batch size 为 256 时，准确率略微下降，但训练时间更短。

这是因为，较小的 batch size 可能导致模型更快地收敛，因为它在每个 batch 上进行了更多次的参数更新，一定程度上能够跳出局部最优解，但可能引入更多的随机性，使模型更容易受到噪声的干扰。

而较大的 batch size 能更快地处理数据，减少训练时间，具有更好的稳定性，但需要更多的数据来估计梯度，可能导致模型收敛速度较慢。此外，较大的 batch size 由于更新次数较少，可能使模型更容易陷入局部最小值，而不容易跳出这些局部最小值。

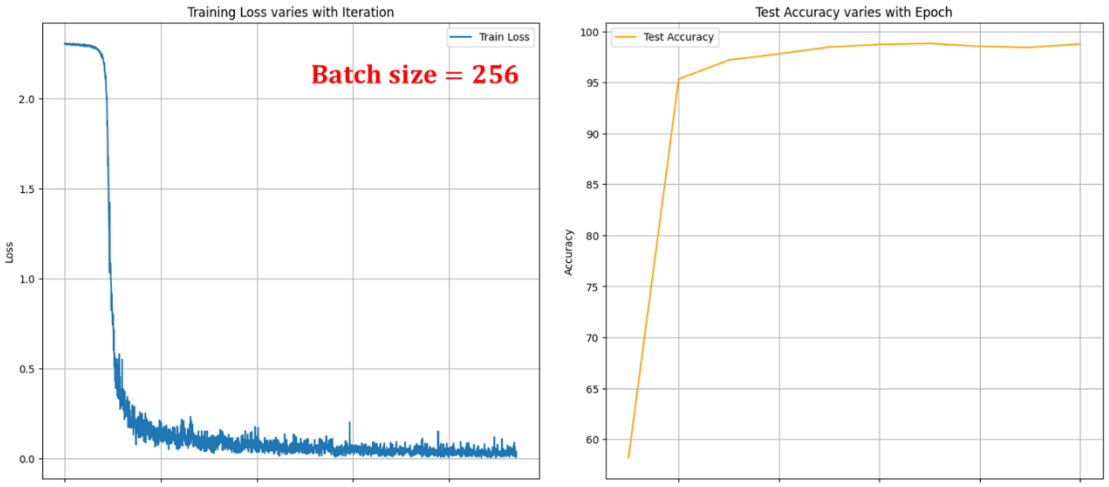

图 16 MNIST 实验结果（Batch size = 256）

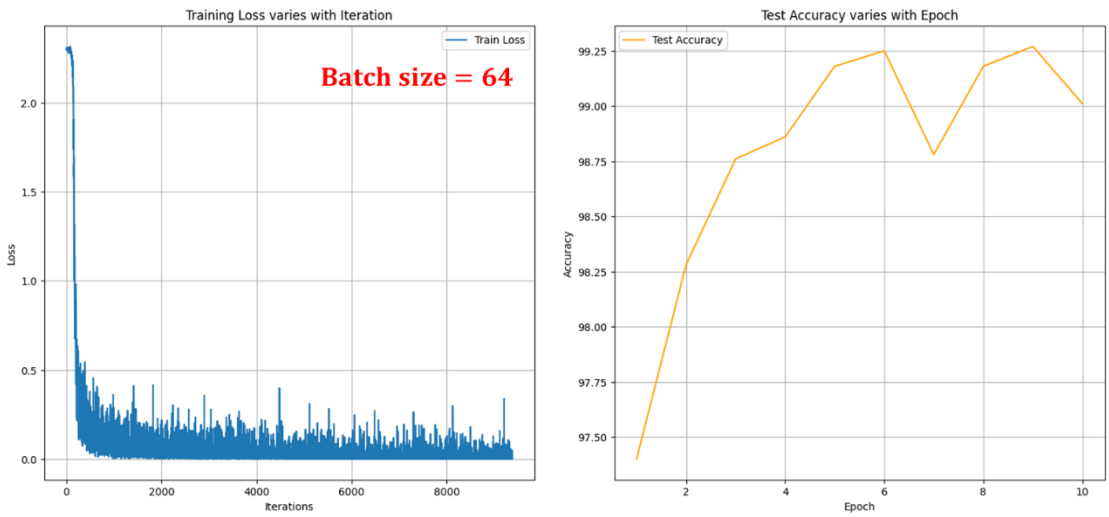

图 17 MNIST 实验结果（Batch size = 64）

5、 实验结论

卷积神经网络使用卷积操作，相较于全连接，其网络层与层之间的连接是稀疏的。其次同一层的卷积的参数是共享的，且每一层的卷积运算可以并行处理，具有较快的学习和推理速度，同时也具有较强的表示和学习能力，在图像分类领域具有较为广泛的应用。

同时，需要针对数据集和具体任务选择合理的超参数，采用合适的权重初始化方法，能够有效提高模型的性能。同时，适时的引入 Dropout 操作，可以通过随机断开神经元的连接，使模型更具鲁棒性，降低模型过拟合风险。

此外，CNN 还可用作其他任务的基础模型，如生成对抗网络（GAN），作为其 backbone 模型来辅助生成高质量的图像。

6、 完整实验代码

Alexnet.py

```
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms

class AlexNet(nn.Module):
    def __init__(self, width_mult=1):
        super(AlexNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),  # 32*28*28
            nn.MaxPool2d(kernel_size=2, stride=2),  # 32*14*14
```

```
14.                 nn.ReLU(inplace=True),
15.             )
16.             self.layer2 = nn.Sequential(
17.                 nn.Conv2d(32, 64, kernel_size=3, padding=1),  # 64*
    14*14
18.                 nn.MaxPool2d(kernel_size=2, stride=2),  # 64*7*7
19.                 nn.ReLU(inplace=True),
20.             )
21.             self.layer3 = nn.Conv2d(64, 128, kernel_size=3, padding
    =1) # 128*7*7
22.             self.layer4 = nn.Sequential(
23.                 nn.Conv2d(128, 256, kernel_size=3, padding=1),  # 2
    56*7*7
24.             )
25.             self.layer5 = nn.Sequential(
26.                 nn.Conv2d(256, 256, kernel_size=3, padding=1),  # 2
    56*7*7
27.                 nn.MaxPool2d(kernel_size=3, stride=2),  # 256*3*3
28.                 nn.ReLU(inplace=True),
29.             )
30.             self.dropout = nn.Dropout(0.5)
31.             self.fc1 = nn.Linear(256 * 3 * 3, 1024)
32.             self.fc2 = nn.Linear(1024, 512)
33.             self.fc3 = nn.Linear(512, 10)
34.
35.
36.         def forward(self, x):
37.             x = self.layer1(x)
38.             x = self.layer2(x)
39.             x = self.layer3(x)
40.             x = self.layer4(x)
41.             x = self.layer5(x)
42.             x = x.view(-1, 256 * 3 * 3)
43.             x = self.fc1(x)
44.             x = self.dropout(x)
45.             x = self.fc2(x)
46.             x = self.dropout(x)
47.             x = self.fc3(x)
48.             return x
49.
50.     def data_processing(str):
51.         if str == 'mnist':
52.             transform = transforms.ToTensor()  # 转换为张量
```

```
53.             trainset = torchvision.datasets.MNIST(root='./data', tr
    ain=True, download=True, transform=transform)
54.             # trainset = torch.utils.data.Subset(trainset, range(25
    00))
55.             trainloader = torch.utils.data.DataLoader(trainset, bat
    ch_size=64, shuffle=True)
56.
57.             testset = torchvision.datasets.MNIST(root='./data', tra
    in=False, download=True, transform=transform)
58.             testloader = torch.utils.data.DataLoader(testset, batch
    _size=64, shuffle=False)
59.             return trainloader, testloader
60.         elif str == 'cifar':
61.             transform = transforms.ToTensor()  # 转换为张量
62.             trainset = torchvision.datasets.CIFAR10(root='./data',
    train=True, download=True, transform=transform)
63.             trainloader = torch.utils.data.DataLoader(trainset, bat
    ch_size=128, shuffle=True)
64.
65.             testset = torchvision.datasets.CIFAR10(root='./data', t
    rain=False, download=True, transform=transform)
66.             testloader = torch.utils.data.DataLoader(testset, batch
    _size=128, shuffle=False)
67.             return trainloader, testloader
68.         return None, None
69.
70.     def eval(model, data):
71.         total = 0
72.         correct = 0
73.         for i, (images, labels) in enumerate(data):
74.             images = images.to(device)
75.             x = model(images)
76.             value, pred = torch.max(x,1)
77.             pred = pred.data.cpu()
78.             total += x.size(0)
79.             correct += torch.sum(pred == labels)
80.
81.         return correct*100./total
82.
83.     def train(model, learning_rate, epochs, trainloader, testloader
    ):
84.         criterion = nn.CrossEntropyLoss()
85.         optimizer = optim.SGD(model.parameters(), lr=learning_rate,
     momentum=0.9)
```

```
86.         max_accuracy=0
87.
88.         train_losses = []
89.         test_accuracies = []
90.
91.         for epoch in range(epochs):
92.             for i, (images,labels) in enumerate(trainloader):
93.                 images = images.to(device)
94.                 labels = labels.to(device)
95.                 optimizer.zero_grad()
96.                 outputs = model(images)
97.                 loss = criterion(outputs, labels)
98.                 loss_item = loss.item()
99.                 loss.backward()
100.                optimizer.step()
101.                train_losses.append(loss.item())
102.
103.            accuracy = float(eval(model, testloader))
104.            test_accuracies.append(accuracy)
105.            print("Epoch %d accuracy: %f loss: %f" % (epoch, accura
    cy, loss_item))
106.            if accuracy > max_accuracy:
107.                best_model = model
108.                max_accuracy = accuracy
109.                print("Saving Best Model with Accuracy: ", accuracy
    )
110.            print('Epoch:', epoch+1, "Accuracy :", accuracy, '%')
111.        torch.save(model.state_dict(), 'checkpoint_mnist.pt')
112.        draw(train_losses, test_accuracies)
113.
114.        return best_model
115.
116.    def test(testloader):
117.        alexnet = AlexNet()
118.        alexnet.load_state_dict(torch.load('./checkpoint_mnist.pt')
    )
119.        plt.figure(figsize=(2,5))
120.        for i, (image, label) in enumerate(testloader):
121.            predict = torch.argmax(alexnet(image), axis=1)
122.            print((predict == label).sum() / label.shape[0])
123.            for j in range(10):
124.                plt.subplot(2, 5, j + 1)
125.                plt.imshow(image[j, 0], cmap='gray')
126.                plt.title(predict[j].item())
```

```
127.                 plt.axis('off')
128.             plt.show()
129.             break
130.
131.     def draw(train_losses, test_accuracies):
132.         plt.figure(figsize=(12, 5))
133.
134.         # 绘制训练损失曲线
135.         plt.subplot(1, 2, 1)
136.         plt.plot(range(1, len(train_losses) + 1), train_losses, lab
    el='Train Loss')
137.         plt.xlabel('Iterations')
138.         plt.ylabel('Loss')
139.         plt.title('Training Loss varies with Iteration')
140.         plt.grid(True)
141.         plt.legend()
142.
143.         # 绘制测试准确率曲线
144.         plt.subplot(1, 2, 2)
145.         plt.plot(range(1, len(test_accuracies) + 1), test_accuracie
    s, label='Test Accuracy',color='orange')
146.         plt.xlabel('Epoch')
147.         plt.ylabel('Accuracy')
148.         plt.title('Test Accuracy varies with Epoch')
149.         plt.grid(True)
150.         plt.legend()
151.
152.         plt.tight_layout()
153.         plt.show()
154.
155.     if __name__  == '__main__':
156.         device = torch.device('cuda' if torch.cuda.is_available() e
    lse 'cpu')
157.         model = AlexNet().to(device)
158.         trainloader, testloader = data_processing('mnist')
159.         best_model = train(model=model, learning_rate=0.01, epochs=
    10, trainloader=trainloader, testloader=testloader)
160.         test(testloader)
```

7、 参考文献

[1] 李航. 统计学习方法[M]. 清华大学出版社, 2012.

[2] 周志华. 机器学习[M]. 清华大学出版社, 2016.

[3] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolute onal neural networks[J]. Advances in neural information processing systems, 2012, 25.