《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。

1. **实验内容**
2. 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。
3. 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。
4. 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
5. **实验环境**

Windows11; Anaconda+python3.11; VS Code

1. **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

图片包含 图示

描述已自动生成

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：

1. 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。
2. 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。
3. 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。

图示

描述已自动生成

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：

1. 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。
2. 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。
3. 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。

图形用户界面, 应用程序

描述已自动生成

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。

文本

描述已自动生成

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：，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，每次将特征图大小缩为原来的一半。

文本

描述已自动生成

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，并设置训练超参数为，为，学习率为。训练过程中损失函数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。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。

图表, 折线图

描述已自动生成

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。

文本

描述已自动生成

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：

1. **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。
2. **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。
3. **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。

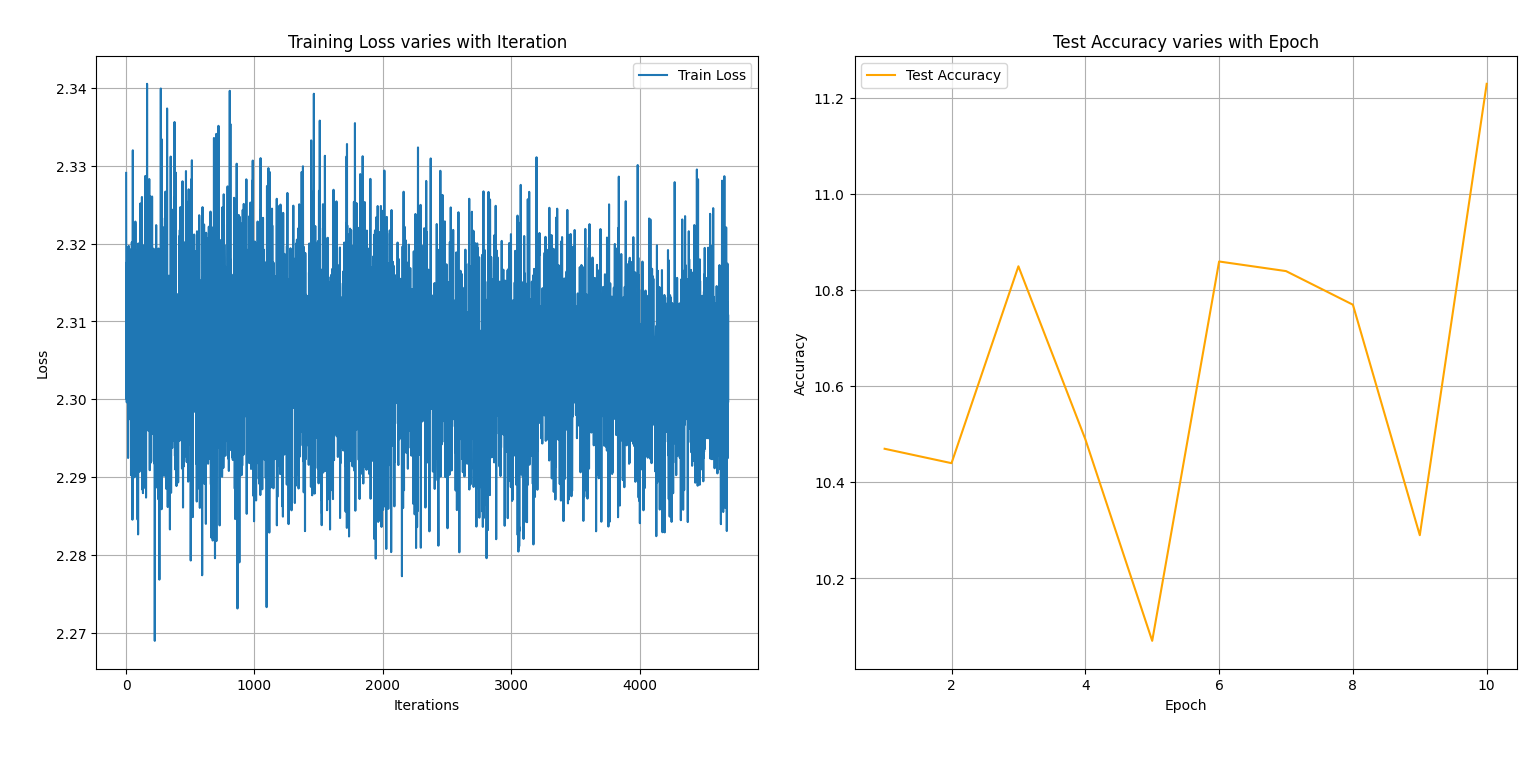


图8 MNIST实验结果（Sigmoid损失函数）

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。

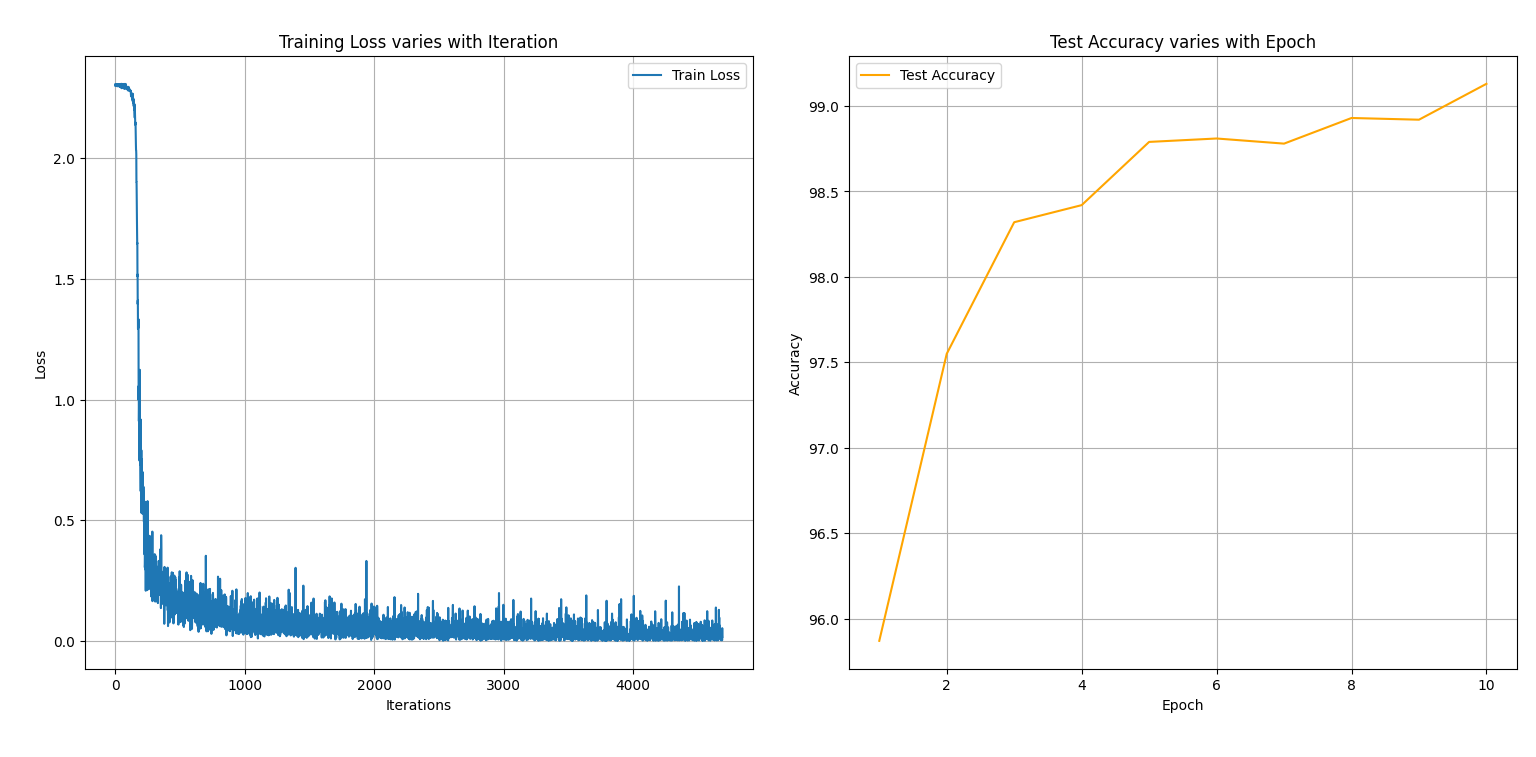


图9 MNIST实验结果（LeakyReLU损失函数）

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。

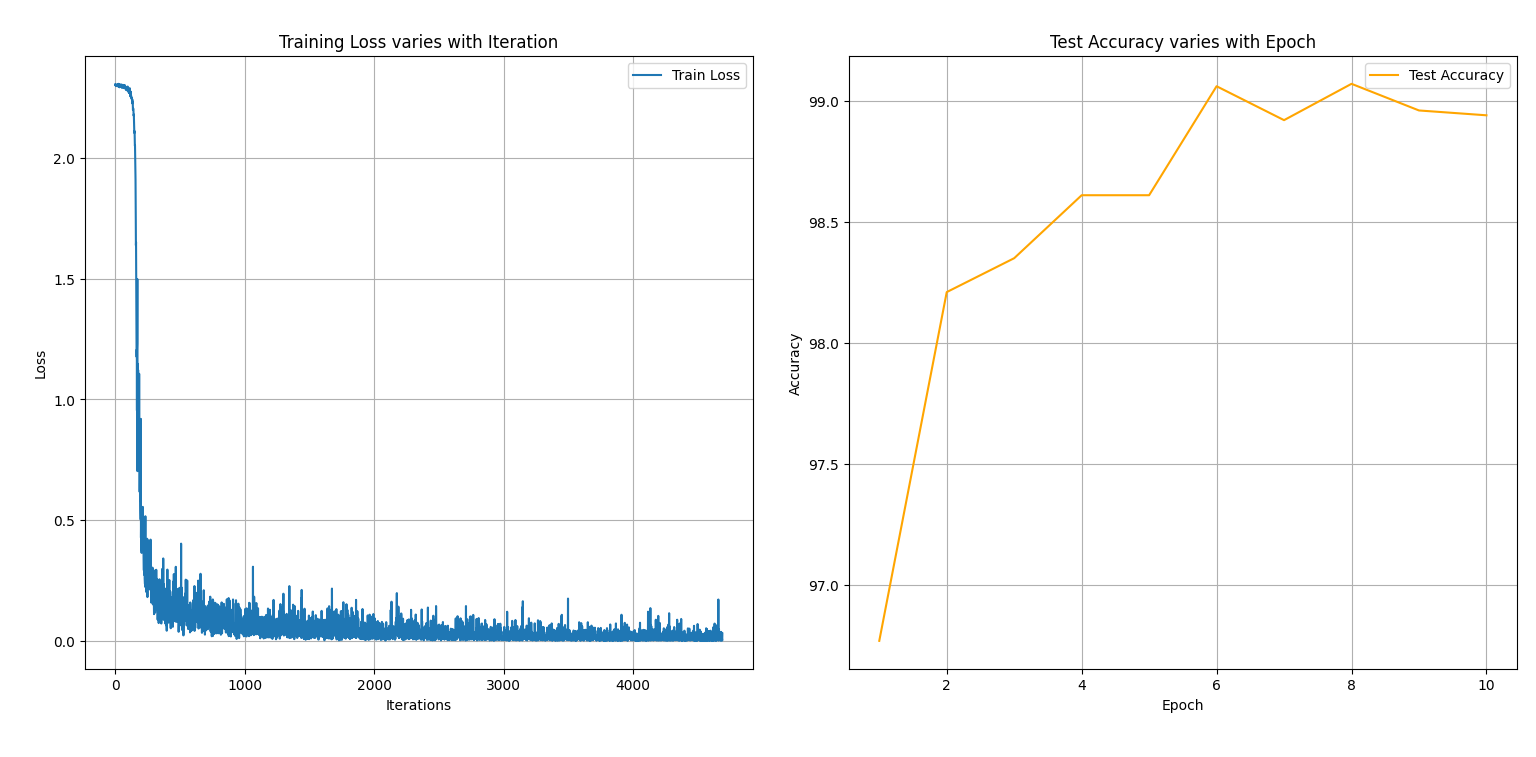


图10 MNIST实验结果（去掉Dropout层）

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为，为，学习率为。训练过程中损失函数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）。

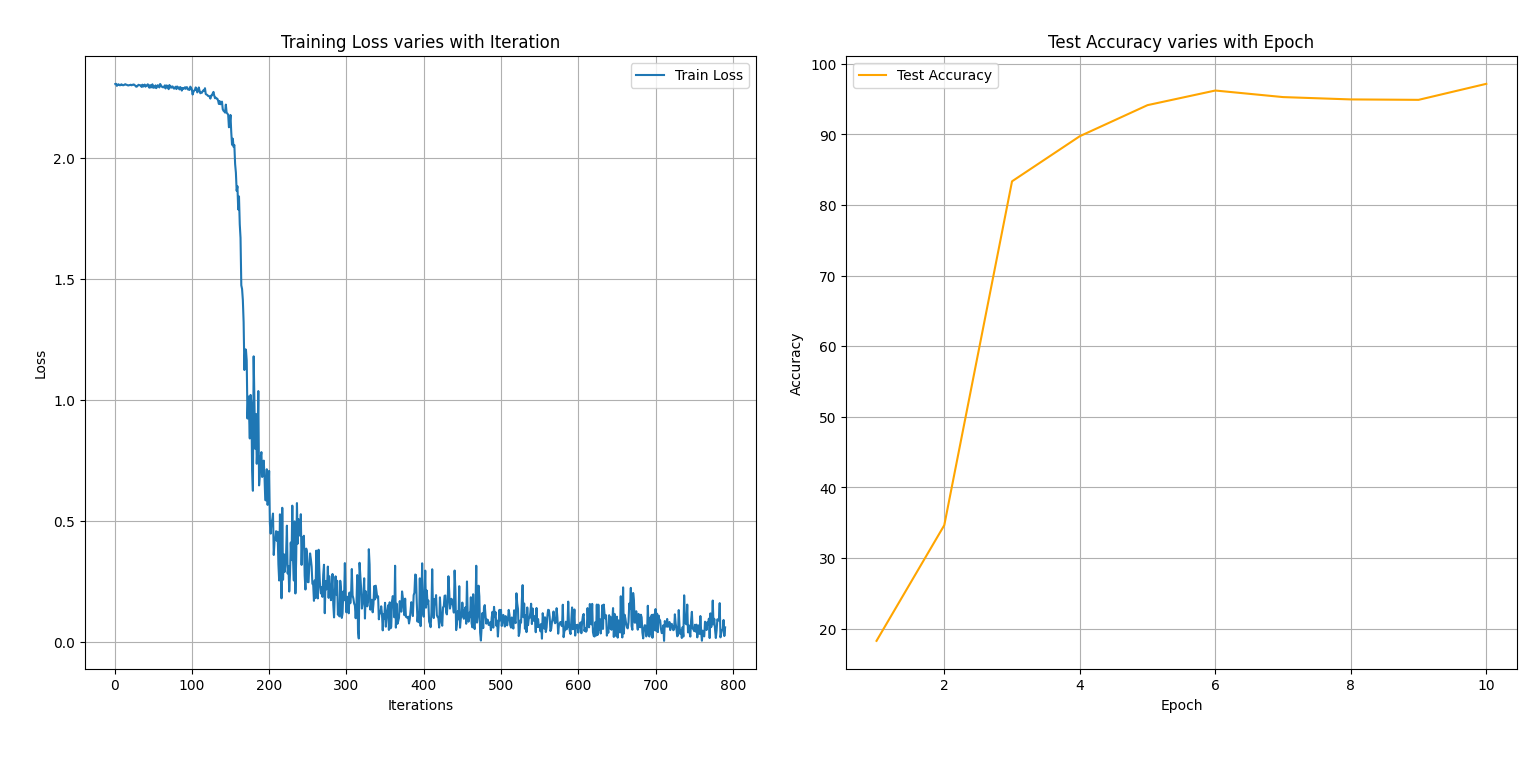


图11 MNIST实验结果（数据量为10000）

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。

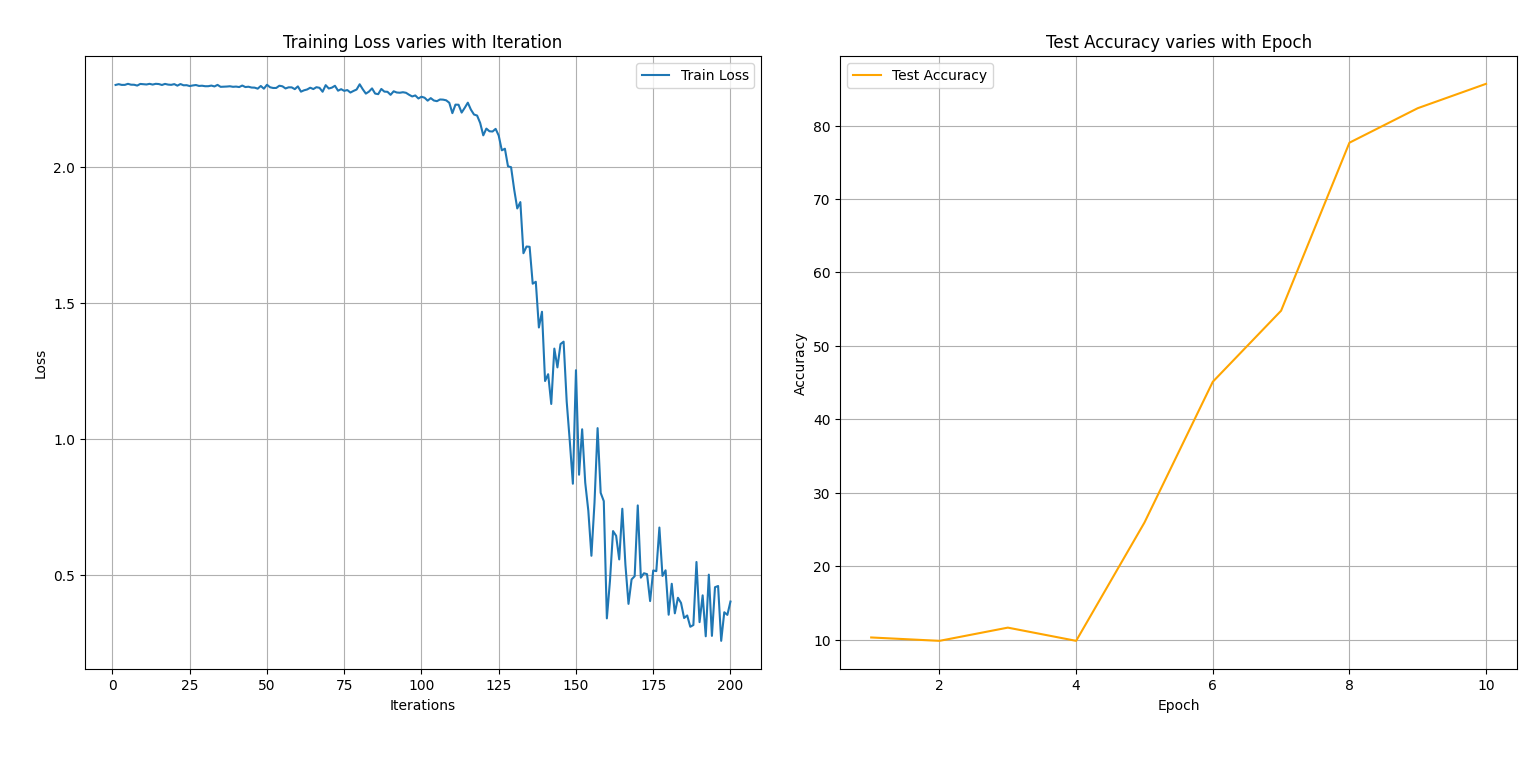
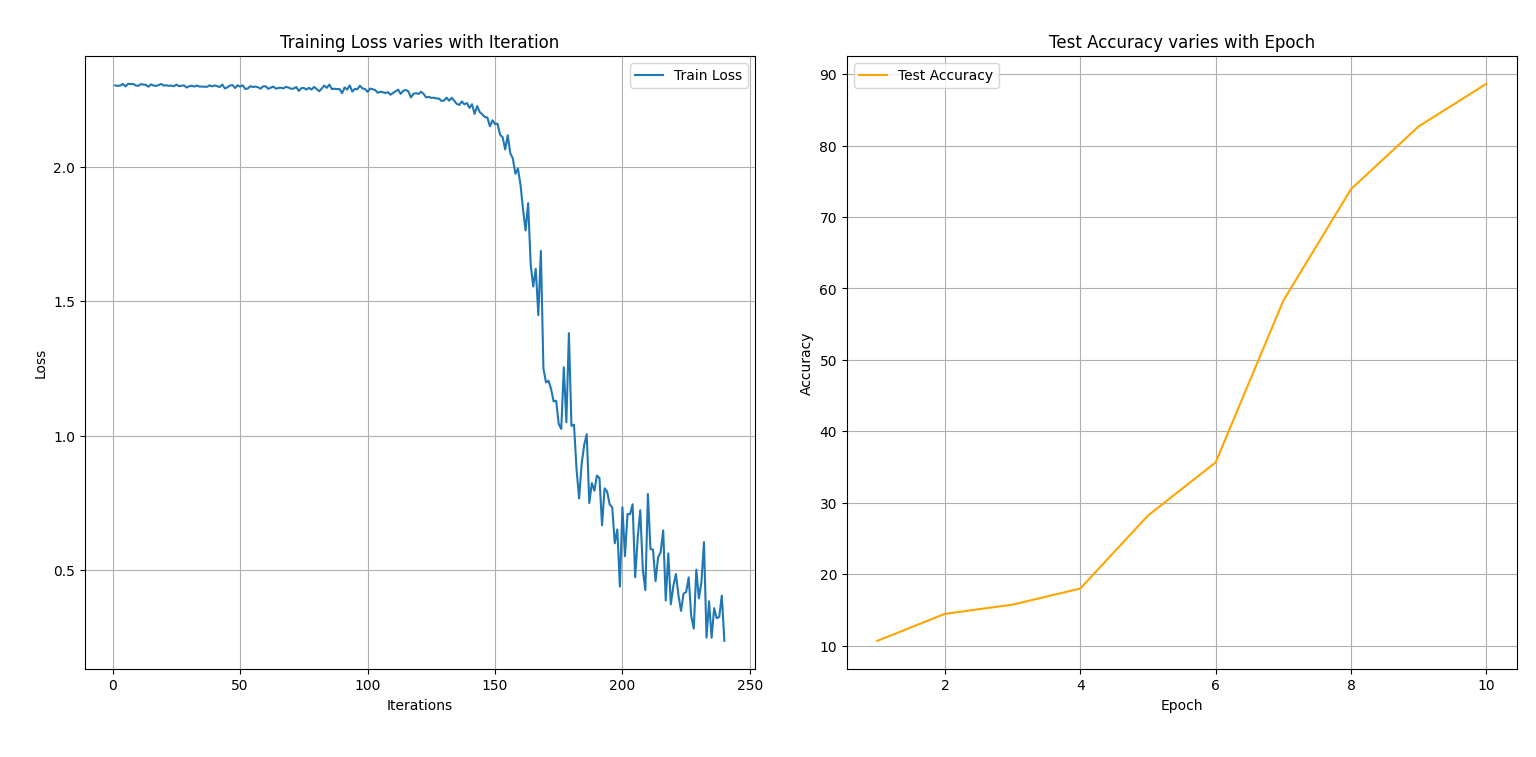


图12 MNIST实验结果（数据量为2500）

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：

1. **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。

文本

描述已自动生成

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。

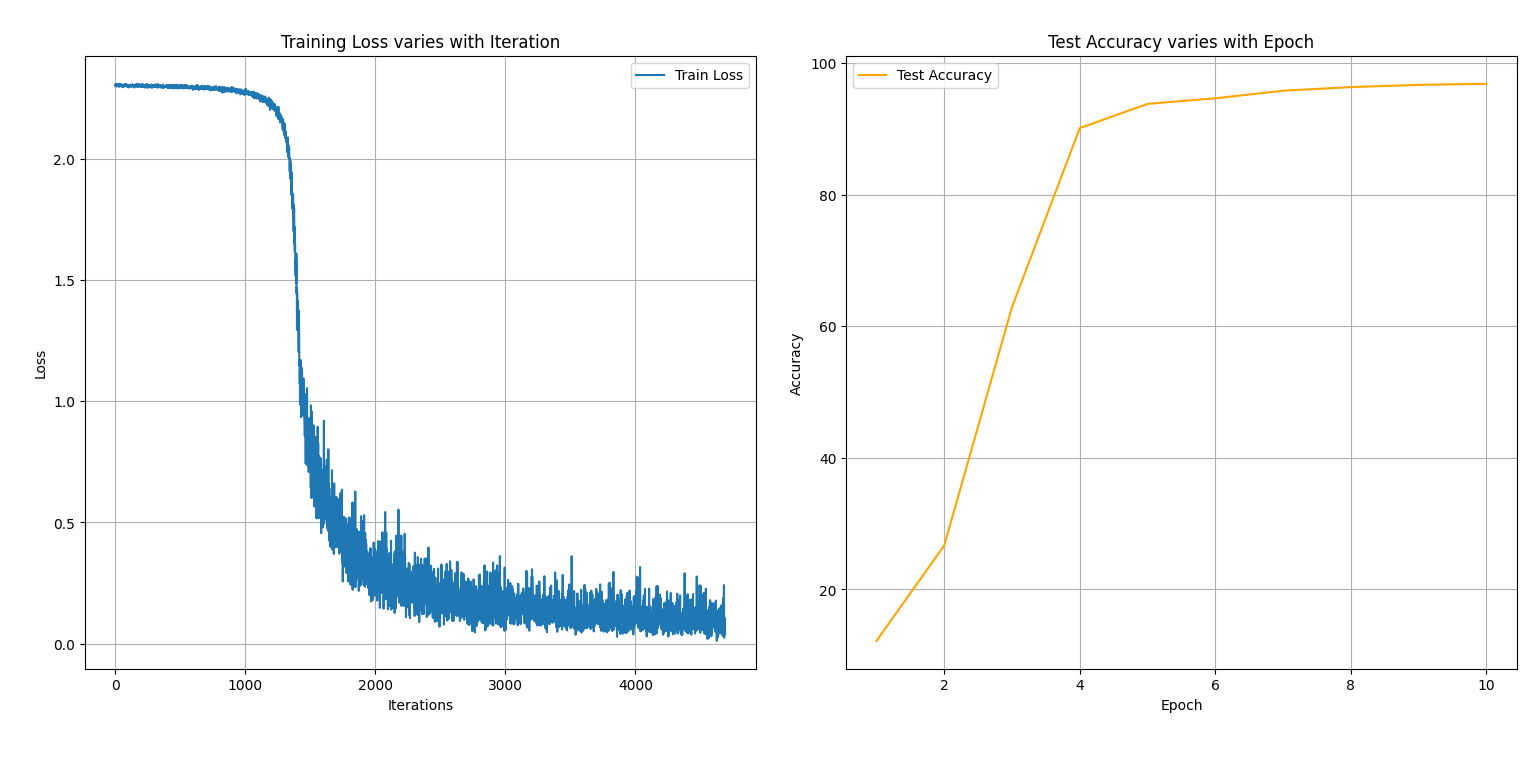


图15 MNIST实验结果（）

1. **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。

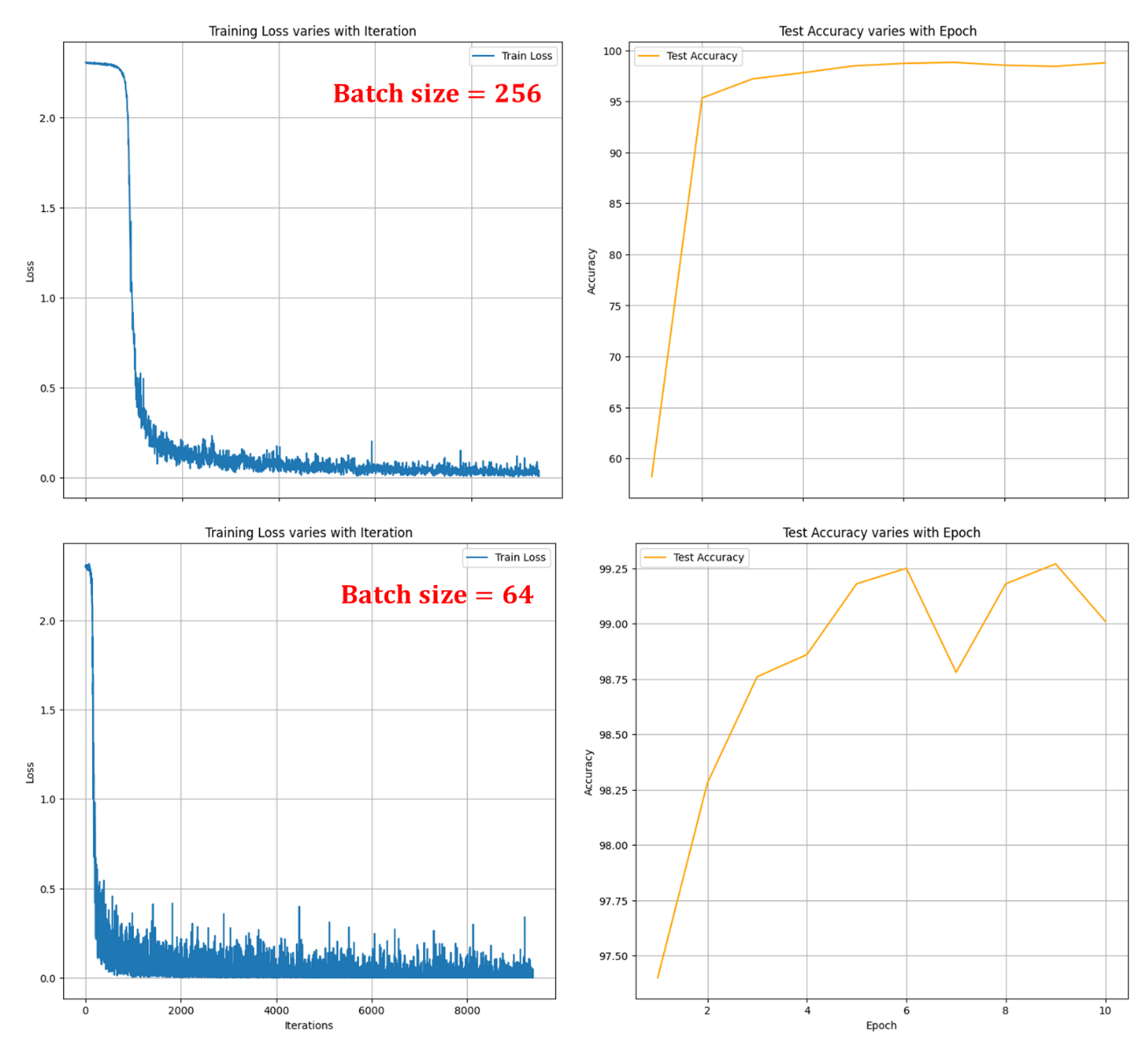


图16 MNIST实验结果（）

图表, 折线图

描述已自动生成

图17 MNIST实验结果（）

1. **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。

1. **完整实验代码**

Alexnet.py

|  |
| --- |
| 1. import matplotlib.pyplot as plt 2. import torch 3. import torch.nn as nn 4. import torch.optim as optim 5. import torchvision 6. import torchvision.transforms as transforms 7. class AlexNet(nn.Module): 8. def \_\_init\_\_(self, width\_mult=1): 9. super(AlexNet, self).\_\_init\_\_() 10. self.layer1 = nn.Sequential( 11. nn.Conv2d(1, 32, kernel\_size=3, padding=1),  *# 32\*28\*28* 12. nn.MaxPool2d(kernel\_size=2, stride=2),  *# 32\*14\*14* 13. nn.ReLU(inplace=True), 14. ) 15. self.layer2 = nn.Sequential( 16. nn.Conv2d(32, 64, kernel\_size=3, padding=1),  *# 64\*14\*14* 17. nn.MaxPool2d(kernel\_size=2, stride=2),  *# 64\*7\*7* 18. nn.ReLU(inplace=True), 19. ) 20. self.layer3 = nn.Conv2d(64, 128, kernel\_size=3, padding=1) *# 128\*7\*7* 21. self.layer4 = nn.Sequential( 22. nn.Conv2d(128, 256, kernel\_size=3, padding=1),  *# 256\*7\*7* 23. ) 24. self.layer5 = nn.Sequential( 25. nn.Conv2d(256, 256, kernel\_size=3, padding=1),  *# 256\*7\*7* 26. nn.MaxPool2d(kernel\_size=3, stride=2),  *# 256\*3\*3* 27. nn.ReLU(inplace=True), 28. ) 29. self.dropout = nn.Dropout(0.5) 30. self.fc1 = nn.Linear(256 \* 3 \* 3, 1024) 31. self.fc2 = nn.Linear(1024, 512) 32. self.fc3 = nn.Linear(512, 10) 33. def forward(self, x): 34. x = self.layer1(x) 35. x = self.layer2(x) 36. x = self.layer3(x) 37. x = self.layer4(x) 38. x = self.layer5(x) 39. x = x.view(-1, 256 \* 3 \* 3) 40. x = self.fc1(x) 41. x = self.dropout(x) 42. x = self.fc2(x) 43. x = self.dropout(x) 44. x = self.fc3(x) 45. return x 46. def data\_processing(str): 47. if str == 'mnist': 48. transform = transforms.ToTensor()  *# 转换为张量* 49. trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform) 50. *# trainset = torch.utils.data.Subset(trainset, range(2500))* 51. trainloader = torch.utils.data.DataLoader(trainset, batch\_size=64, shuffle=True) 52. testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform) 53. testloader = torch.utils.data.DataLoader(testset, batch\_size=64, shuffle=False) 54. return trainloader, testloader 55. elif str == 'cifar': 56. transform = transforms.ToTensor()  *# 转换为张量* 57. trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform) 58. trainloader = torch.utils.data.DataLoader(trainset, batch\_size=128, shuffle=True) 59. testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform) 60. testloader = torch.utils.data.DataLoader(testset, batch\_size=128, shuffle=False) 61. return trainloader, testloader 62. return None, None 63. def eval(model, data): 64. total = 0 65. correct = 0 66. for i, (images, labels) in enumerate(data): 67. images = images.to(device) 68. x = model(images) 69. value, pred = torch.max(x,1) 70. pred = pred.data.cpu() 71. total += x.size(0) 72. correct += torch.sum(pred == labels) 73. return correct\*100./total 74. def train(model, learning\_rate, epochs, trainloader, testloader): 75. criterion = nn.CrossEntropyLoss() 76. optimizer = optim.SGD(model.parameters(), lr=learning\_rate, momentum=0.9) 77. max\_accuracy=0 78. train\_losses = [] 79. test\_accuracies = [] 80. for epoch in range(epochs): 81. for i, (images,labels) in enumerate(trainloader): 82. images = images.to(device) 83. labels = labels.to(device) 84. optimizer.zero\_grad() 85. outputs = model(images) 86. loss = criterion(outputs, labels) 87. loss\_item = loss.item() 88. loss.backward() 89. optimizer.step() 90. train\_losses.append(loss.item()) 91. accuracy = float(eval(model, testloader)) 92. test\_accuracies.append(accuracy) 93. print("Epoch %d accuracy: %f loss: %f" % (epoch, accuracy, loss\_item)) 94. if accuracy > max\_accuracy: 95. best\_model = model 96. max\_accuracy = accuracy 97. print("Saving Best Model with Accuracy: ", accuracy) 98. print('Epoch:', epoch+1, "Accuracy :", accuracy, '%') 99. torch.save(model.state\_dict(), 'checkpoint\_mnist.pt') 100. draw(train\_losses, test\_accuracies) 101. return best\_model 102. def test(testloader): 103. alexnet = AlexNet() 104. alexnet.load\_state\_dict(torch.load('./checkpoint\_mnist.pt')) 105. plt.figure(figsize=(2,5)) 106. for i, (image, label) in enumerate(testloader): 107. predict = torch.argmax(alexnet(image), axis=1) 108. print((predict == label).sum() / label.shape[0]) 109. for j in range(10): 110. plt.subplot(2, 5, j + 1) 111. plt.imshow(image[j, 0], cmap='gray') 112. plt.title(predict[j].item()) 113. plt.axis('off') 114. plt.show() 115. break 116. def draw(train\_losses, test\_accuracies): 117. plt.figure(figsize=(12, 5)) 118. *# 绘制训练损失曲线* 119. plt.subplot(1, 2, 1) 120. plt.plot(range(1, len(train\_losses) + 1), train\_losses, label='Train Loss') 121. plt.xlabel('Iterations') 122. plt.ylabel('Loss') 123. plt.title('Training Loss varies with Iteration') 124. plt.grid(True) 125. plt.legend() 126. *# 绘制测试准确率曲线* 127. plt.subplot(1, 2, 2) 128. plt.plot(range(1, len(test\_accuracies) + 1), test\_accuracies, label='Test Accuracy',color='orange') 129. plt.xlabel('Epoch') 130. plt.ylabel('Accuracy') 131. plt.title('Test Accuracy varies with Epoch') 132. plt.grid(True) 133. plt.legend() 134. plt.tight\_layout() 135. plt.show() 136. if \_\_name\_\_  == '\_\_main\_\_': 137. device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu') 138. model = AlexNet().to(device) 139. trainloader, testloader = data\_processing('mnist') 140. best\_model = train(model=model, learning\_rate=0.01, epochs=10, trainloader=trainloader, testloader=testloader) 141. test(testloader) |

1. **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.