《模式识别与机器学习 A》实验报告

实验题目：卷积神经网络实验

学号：2021110683

姓名：徐柯炎

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（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

Windows10; python3.9;PyCharm 2021.2.2

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）

(1) 卷积神经网络的原理

卷积神经网络（Convolutional Neural Networks，简称 CNN）是一种具有局部连接、权值共享等特点的深层前馈神经网络（Feedforward Neural Networks），是深度学习（deep learning）的代表算法之一，擅长处理图像特别是图像识别等相关机器学习问题，比如图像分类、目标检测、图像分割等各种视觉任务中都有显著的提升效果，是目前应用最广泛的模型之一。

卷积神经网络由以下几个部分构成：

卷积层（Convolutional Layer）：

卷积操作是 CNN 的核心部分，它通过滤波器（卷积核）对输入图像进行卷积运算，以提取特征。每个卷积操作都会产生一个特征映射（Feature Map），如下所示：

$$\text{Feature Map}_{i,j} = \sum_{m} \sum_{n} (\text{Input Image}_{i+m,j+n} \cdot \text{Filter}_{m,n})$$

其中，$Feature\ Map_{i,j}$ 是特征映射的像素值，$Input\ Image_{i+m,j+n}$ 是输入图像的像素值，$Filter_{m,n}$是滤波器的权重。卷积操作是通过滑动滤波器在输入图像上进行的。

池化层（Pooling Layer）：

池化操作用于降低特征映射的空间维度，减少计算复杂度，并增强特征的不变性。最常见的池化操作是最大池化（Max Pooling）和平均池化（Average Pooling）。

全连接层（Fully Connected Layer）：

全连接层将卷积和池化层提取的特征映射连接起来，然后通过激活函数进行非线性变换，最终输出网络的预测结果。全连接层的公式如下：

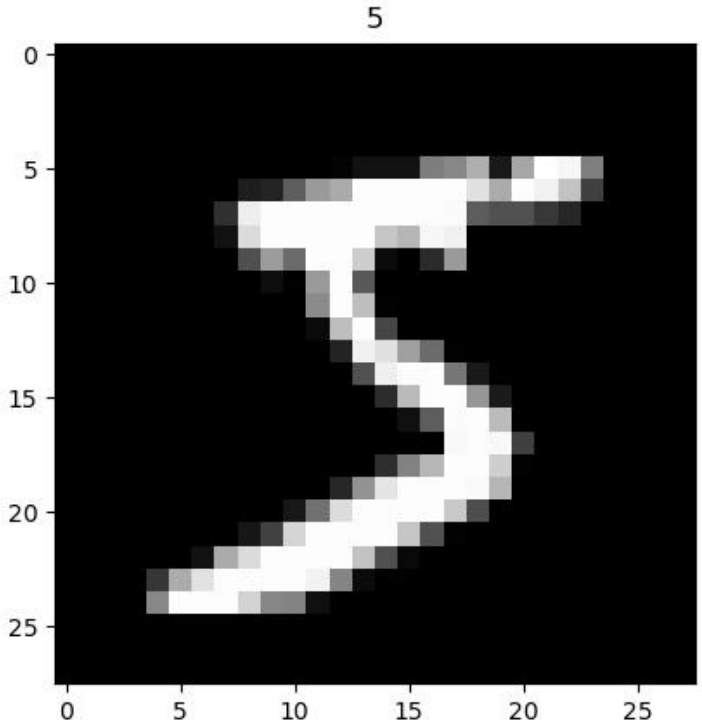
$$\text{Output}_k = f(\sum_i \sum_j (\text{Fully Connected Layer}_{i,j} \cdot \text{Feature Map}_{i,j}) + \text{Bias}_k)$$

其中，$Output_k$是网络的第 k 个输出，$Fully\ Connected\ Layer_{i,j}$ 是全连接层的权重，$Feature\ Map_{i,j}$是特征映射的像素值，$Bias_k$ 是偏置项，$f()$ 是激活函数，通常是 ReLU（Rectified Linear Unit）或 Sigmoid 函数。

这些是卷积神经网络的基本原理，它们允许 CNN 自动学习图像中的特征，并在图像分类、目标检测和其他计算机视觉任务中取得出色的性能。

(2) 实验过程

本实验采用 pytorch 库，数据集采用 mnist 数据集，mnist 数据集的图像如下图所示：



本实验采用的神经网络为 lenet 模型，代码如下图所示：

```python
class LeNet(nn.Module):
    def __init__(self, activate='relu', p=0.8):
        super().__init__()
        self.conv1 = nn.Conv2d( in_channels: 1,  out_channels: 8,  kernel_size: 5)
        self.conv2 = nn.Conv2d( in_channels: 8,  out_channels: 16,  kernel_size: 5)
        self.pool = nn.MaxPool2d(2)
        self.linear1 = nn.Linear(4 * 4 * 16,  out_features: 120)
        self.linear2 = nn.Linear( in_features: 120,  out_features: 84)
        self.linear3 = nn.Linear( in_features: 84,  out_features: 10)
        print(f'dropout参数p为{p}')
        self.dropout = nn.Dropout(p=p)

        if activate == 'relu':
            print('使用relu函数作为激活函数')
            self.activate = nn.ReLU()
        elif activate == 'sigmoid':
            print('使用sigmoid函数作为激活函数')
            self.activate = nn.Sigmoid()
        elif activate == 'tanh':
            print('使用tanh函数作为激活函数')
            self.activate = nn.Tanh()
        elif activate == 'leaky_relu':
            print('使用leaky_relu函数作为激活函数')
            self.activate = nn.LeakyReLU()
        else:
            print('使用relu函数作为激活函数')
            self.activate = nn.ReLU()

```

Lenet 的前向传播函数如图所示：

```python
def forward(self, x):
    x = self.activate(self.conv1(x))
    x = self.pool(x)
    x = self.activate(self.conv2(x))
    x = self.pool(x)
    x = torch.flatten(x, start_dim=1, end_dim=3)
    x = self.activate(self.linear1(x))
    x = self.dropout(x)
    x = self.activate(self.linear2(x))
    x = self.dropout(x)
    x = self.linear3(x)
    return x
```

接着用 train 函数来训练模型：

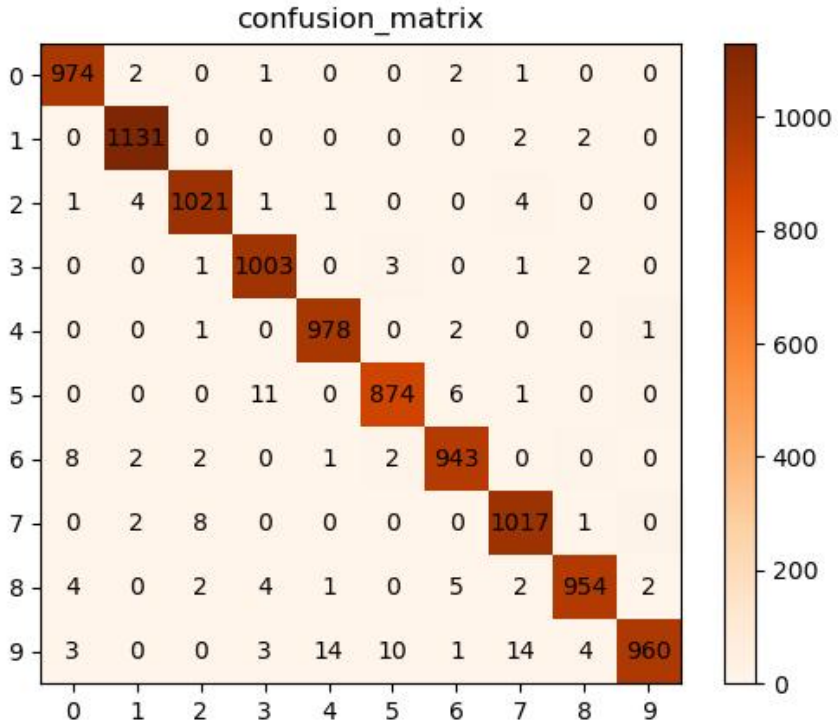
```python
def train_net():
    global run_loss
    net.train()
    print(f'epoch: {e}')
    for i, data in enumerate(train_loader):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        run_loss += loss.item()
        if i % print_num == print_num - 1:
            run_loss /= print_num
            loss_list.append(run_loss)
            print(f'[{e + 1}, {(i + 1) * batch_size}:{train_num}] \tloss: {round(run_loss, 5)}')
            run_loss = 0.0
```

最后用 test 函数来测试模型：

```python
def test_net():
    net.eval()
    correct = 0
    confusion_matrix = torch.zeros(10, 10, dtype=torch.int32)
    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data
            outputs = net(inputs)
            _, predicts = torch.max(outputs.detach(), 1)
            correct += (predicts == labels).sum().item()
            for p, l in zip(predicts, labels):
                confusion_matrix[p, l] += 1
    return confusion_matrix
```

(3) 实验测试

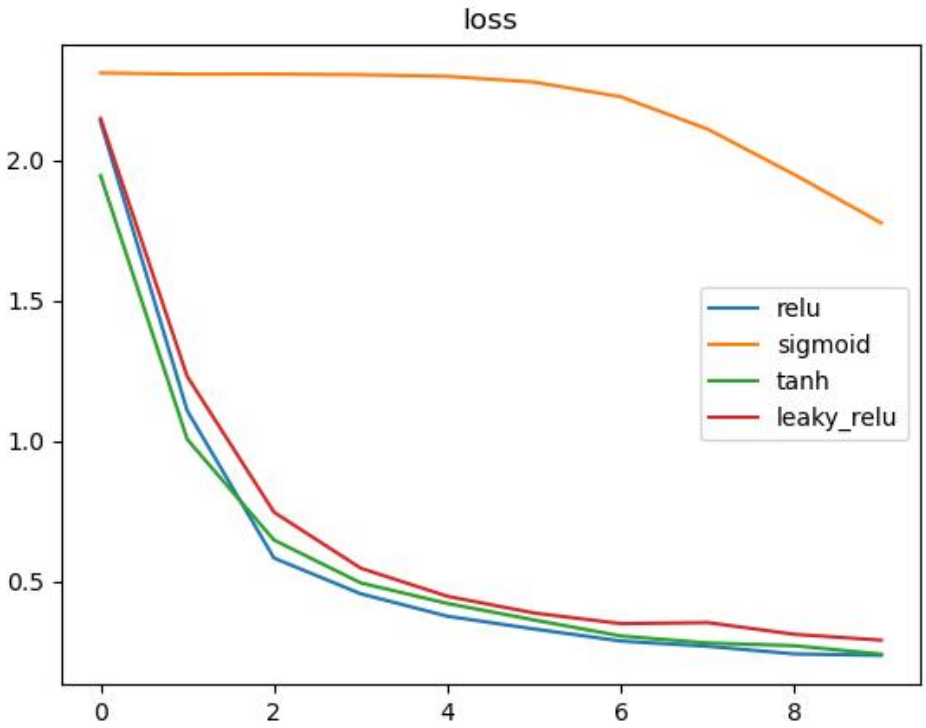
本实验从以下几个方面来测试：loss 的下降速率、准确率（accuracy）、精确率（precision）、召回率（recall）、f1 分数，训练用时，同时采用混淆矩阵绘制图像（由于太占空间这里就放一张，如下图所示）。



a) 激活函数的选择：

本实验对常用的激活函数进行测试，具体包括以下几个函数：sigmoid 函数、relu 函数，tanh 函数、leaky relu 函数。测试结果如下：

下图为 loss 下降的过程：



下图对模型进行评价：

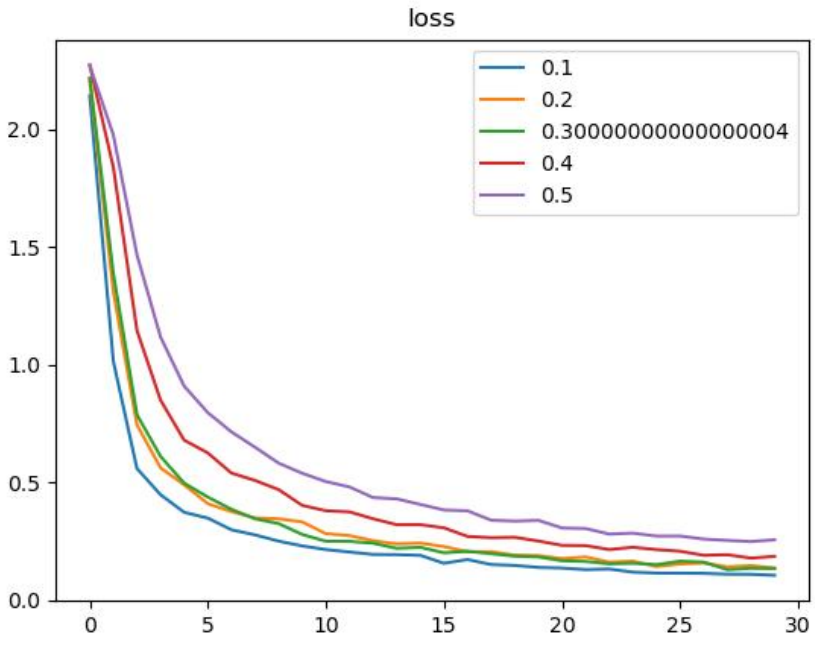
```
dropout参数p为0.1
使用relu函数作为激活函数
训练时间为：8.63s
accuracy: 0.943, precision: 0.943, recall: 0.942, f1: 0.942
dropout参数p为0.1
使用sigmoid函数作为激活函数
训练时间为：9.211s
accuracy: 0.569, precision: 0.555, recall: 0.635, f1: 0.493
dropout参数p为0.1
使用tanh函数作为激活函数
训练时间为：8.279s
accuracy: 0.943, precision: 0.943, recall: 0.943, f1: 0.943
dropout参数p为0.1
使用leaky_relu函数作为激活函数
训练时间为：9.063s
accuracy: 0.948, precision: 0.947, recall: 0.947, f1: 0.947
```

从上图看出，这几个函数相比而言 sigmoid 函数的训练效果最差，在所有指标上面都远差于其它几个函数。Relu 和 tanh 收敛速度较快，但是从测试结果来看，效果要比 leaky relu 函数差一点。由于一般我们采用 relu 函数作为激活函数，因此下面的测试均采用 relu 函数。

b) Dropout 测试：

本实验对 dropout 的取值进行测试。测试结果如下：

下图为 loss 下降的过程：



下图对模型进行评价：

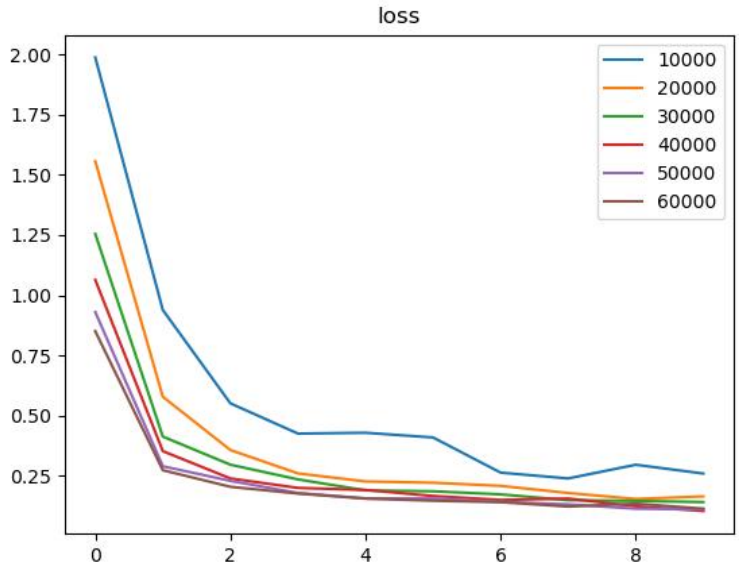
```
dropout参数p为0.1
使用relu函数作为激活函数
训练时间为：8.444s
accuracy: 0.946, precision: 0.946, recall: 0.946, f1: 0.946
dropout参数p为0.2
使用relu函数作为激活函数
训练时间为：8.465s
accuracy: 0.939, precision: 0.938, recall: 0.939, f1: 0.938
dropout参数p为0.30000000000000004
使用relu函数作为激活函数
训练时间为：8.557s
accuracy: 0.948, precision: 0.947, recall: 0.948, f1: 0.947
dropout参数p为0.4
使用relu函数作为激活函数
训练时间为：8.526s
accuracy: 0.932, precision: 0.931, recall: 0.932, f1: 0.931
dropout参数p为0.5
使用relu函数作为激活函数
训练时间为：8.537s
accuracy: 0.93, precision: 0.93, recall: 0.93, f1: 0.929
```

从上图看出，当 dropout 参数为 0.3 时，也就是说每次训练屏蔽 30%的节点时，测试效果最好，尽管 loss 不是最低，这说明当 loss 下降到一定程度的时候会出现过拟合的现象，而 dropout 刚好缓解了这种现象，并且我们经过实验可知，dropout 参数为 0.3 的时候效果最好。

c) 样本数量测试：

本实验对训练轮数的取值进行测试。测试结果如下：

下图为 loss 下降的过程：



下图对模型进行评价：

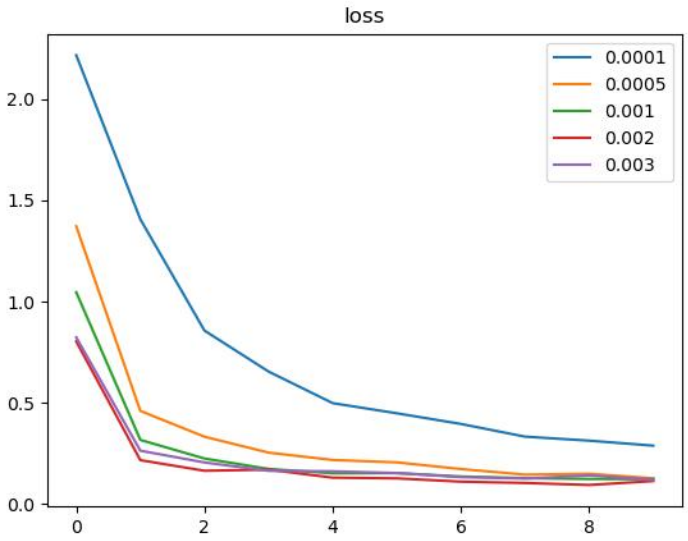
```
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：3.309s
accuracy: 0.953, precision: 0.952, recall: 0.954, f1: 0.952
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：4.413s
accuracy: 0.968, precision: 0.969, recall: 0.968, f1: 0.968
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：5.482s
accuracy: 0.979, precision: 0.979, recall: 0.979, f1: 0.979
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：6.353s
accuracy: 0.979, precision: 0.979, recall: 0.979, f1: 0.979
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：7.658s
accuracy: 0.977, precision: 0.977, recall: 0.978, f1: 0.977
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：9.065s
accuracy: 0.978, precision: 0.978, recall: 0.978, f1: 0.978
```

从上图看出，训练样本越多，训练时间越长，同时训练效果也越好，但是也需要防止数据过多时过拟合的发生，尽管在这里并没有出现。

d) 初始学习率测试：

本实验对初始学习率的取值进行测试。测试结果如下：

下图为 loss 下降的过程：



下图对模型进行评价：

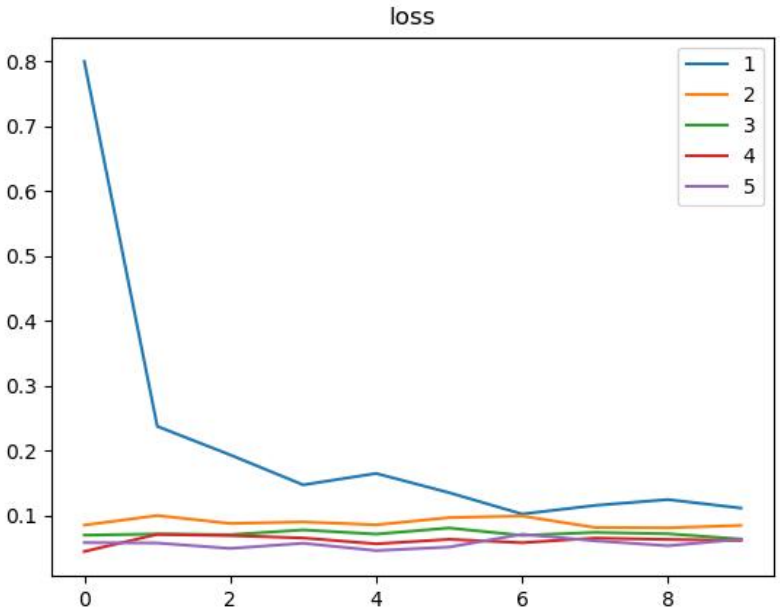
```
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：8.959s
accuracy: 0.951, precision: 0.951, recall: 0.95, f1: 0.95
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：8.828s
accuracy: 0.975, precision: 0.975, recall: 0.975, f1: 0.975
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：8.909s
accuracy: 0.977, precision: 0.977, recall: 0.977, f1: 0.977
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：8.731s
accuracy: 0.983, precision: 0.983, recall: 0.983, f1: 0.983
dropout参数p为0.3
使用relu函数作为激活函数
训练时间为：8.864s
accuracy: 0.982, precision: 0.982, recall: 0.982, f1: 0.982
```

从上图可以看出，并不是学习率越高收敛速度越快，因为这里用的是 adam 算法，它会根据每个参数的变化情况针对性的调整学习率。在这里当学习率为 0.002 时收敛速度最快，同时测试效果最好。

e) 训练轮数：

本实验对训练轮数的取值进行测试。测试结果如下：

下图为 loss 下降的过程：



下图对模型进行评价：

```
dropout参数p为0.3
使用relu函数作为激活函数
epoch: 0
训练时间为：8.867s
accuracy: 0.983, precision: 0.983, recall: 0.983, f1: 0.983
epoch: 1
训练时间为：8.836s
accuracy: 0.987, precision: 0.987, recall: 0.987, f1: 0.987
epoch: 2
训练时间为：8.847s
accuracy: 0.986, precision: 0.986, recall: 0.986, f1: 0.986
epoch: 3
训练时间为：8.834s
accuracy: 0.989, precision: 0.989, recall: 0.989, f1: 0.989
epoch: 4
训练时间为：8.855s
accuracy: 0.988, precision: 0.988, recall: 0.988, f1: 0.988
```

从上图可以看出，当训练轮数增加以后，loss 非但没有下降反而还上升了一点，但测试效果比先前好了许多，这说明当 loss 太低的时候发生了过拟合的现象，但是随着训练轮数的增加，损失函数逐渐趋于收敛，此时再进行训练也意义不大了。因此在这里训练轮数为 4 时最好。

5. 实验总体结论

a) 模型的激活函数不再使用 sigmoid 函数，而应该选取 relu，tanh 等函数。
b) 使用 dropout 防止过拟合：当 loss 下降到一定程度的时候会出现过拟合的现象，而 dropout 刚好缓解了这种现象。
c) 训练样本越多，训练时间越长，同时训练效果也越好，但是也需要防止数据过多时过拟合的发生，尽管在这里并没有出现。
d) 学习率越小，迭代次数越多，训练时间越长，训练效果越好；学习率偏大的时候容易出现震荡的现象，虽然用时短了，但是训练效果偏差。
e) 当训练轮数增加以后，loss 非但没有下降反而还上升了一点，但测试效果比先前好了许多，这说明当 loss 太低的时候发生了过拟合的现象，但是随着训练轮数的增加，损失函数逐渐趋于收敛，此时再进行训练也意义不大了。

6. 完整实验代码

main.py

```python
import time
import matplotlib.pyplot as plt
import numpy as np
import torch
```

```
import torchvision
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
import torch.optim as optim
from LeNet import LeNet
from torch import nn

batch_size = 30
lr = 0.002
epoch = 5
activate_list = ['relu', 'sigmoid', 'tanh', 'leaky_relu']
activate_test = False
dropout_list = np.arange(0.1, 0.6, 0.1)
dropout_test = False
lr_list = [0.0001, 0.0005, 0.001, 0.002, 0.003]
lr_test = False
bs_list = np.arange(10, 100, 10)
bs_test = False
num_list = np.arange(10000, 60001, 10000)
num_test = False


def test_net():
    net.eval()
    correct = 0
    confusion_matrix = torch.zeros(10, 10, dtype=torch.int32)
    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data
            outputs = net(inputs)
            _, predicts = torch.max(outputs.detach(), 1)
            correct += (predicts == labels).sum().item()
            for p, l in zip(predicts, labels):
                confusion_matrix[p, l] += 1
    return confusion_matrix


def train_net():
    global run_loss, loss_list
    net.train()
    print(f'epoch: {e}')
    for i, data in enumerate(train_loader):
        inputs, labels = data
```

```
48.         optimizer.zero_grad()
49.         outputs = net(inputs)
50.         loss = criterion(outputs, labels)
51.         loss.backward()
52.         optimizer.step()
53.         run_loss += loss.item()
54.         if i % print_num == print_num - 1:
55.             run_loss /= print_num
56.             loss_list.append(run_loss)
57.             # print(f'[{e + 1}, {(i + 1) * batch_size}:{tra
in_num}] \tloss: {round(run_loss, 5)}')
58.             run_loss = 0.0
59.
60.
61. def plot_loss(loss_lists, name):
62.     plt.title("loss")
63.     for loss_list, label in zip(loss_lists, name):
64.         num = len(loss_list)
65.         x = np.arange(num)
66.         plt.plot(x, loss_list, label=label)
67.     plt.legend()
68.     plt.show()
69.
70.
71. def cal_conmat(con_mat):
72.     p_num = torch.sum(con_mat, 0)
73.     l_num = torch.sum(con_mat, 1)
74.     tp = torch.diagonal(con_mat)
75.
76.     precision = tp / p_num
77.     recall = tp / l_num
78.     accuracy = (torch.sum(tp) / torch.sum(con_mat)).item()

79.     f1 = 2 / (1 / precision + 1 / recall)
80.
81.     esti_dic = {"accuracy": accuracy,
82.                 "precision": precision,
83.                 "recall": recall,
84.                 "f1": f1}
85.     return esti_dic
86.
87.
88. def plot_conmat(con_mat):
89.     fig = plt.figure()
```

```
90.     num = con_mat.size(0)
91.     plt.imshow(con_mat, interpolation='nearest', cmap=plt.c
m.Oranges)
92.     plt.title('confusion_matrix')
93.     plt.colorbar()
94.     tick_marks = np.arange(num)
95.     plt.xticks(tick_marks, tick_marks)
96.     plt.yticks(tick_marks, tick_marks)
97.     for i in range(num):
98.         for j in range(num):
99.             plt.text(i, j, format(con_mat[i, j]), va='cente
r', ha='center')
100.        fig.show()
101.
102.
103.   def print_esti(estimate: dict):
104.        accuracy = estimate['accuracy']
105.        precision = torch.mean(estimate['precision']).item()

106.        recall = torch.mean(estimate['recall']).item()
107.        f1 = torch.mean(estimate['f1']).item()
108.        print(f'accuracy: {round(accuracy, 3)}, precision: {
round(precision, 3)}, '
109.              f'recall: {round(recall, 3)}, f1: {round(f1, 3)
}')
110.
111.
112.   if __name__ == '__main__':
113.        transform = transforms.Compose([transforms.ToTensor()
,
114.                                        transforms.Normalize
(0.5, 0.5)])
115.        train_data = torchvision.datasets.MNIST('mnist-data',
 train=True, transform=transform)
116.        test_data = torchvision.datasets.MNIST('mnist-data',
 train=False, transform=transform)
117.        train_loader = DataLoader(dataset=train_data, batch_
size=batch_size, shuffle=True, num_workers=2)
118.        test_loader = DataLoader(dataset=test_data, batch_si
ze=batch_size, shuffle=True, num_workers=2)
119.        # print(type(train_data[0][1]))
120.        # print(train_data)
121.        # print(test_data.data.size())
122.        train_num = len(train_data)
```

```
123.        test_num = len(test_data)
124.        print_num = int(train_num / (batch_size * 10))
125.
126.        if activate_test:
127.            loss_lists = []
128.            for act in activate_list:
129.                net = LeNet(activate=act)
130.                optimizer = optim.Adam(net.parameters(), lr=
lr)
131.
132.                criterion = nn.CrossEntropyLoss()
133.                run_loss = 0.0
134.                loss_list = []
135.                for e in range(epoch):
136.                    start = time.time()
137.                    train_net()
138.                    end = time.time()
139.                    print(f'训练时间为:
{round(end - start, 3)}s')
140.                    con_matrix = test_net()
141.                loss_lists.append(loss_list)
142.                estimate = cal_conmat(con_matrix)
143.                print_esti(estimate)
144.                plot_conmat(con_matrix)
145.            plot_loss(loss_lists, activate_list)
146.
147.        elif dropout_test:
148.            loss_lists = []
149.            for dropout in dropout_list:
150.                net = LeNet(p=dropout)
151.                optimizer = optim.Adam(net.parameters(), lr=
lr)
152.
153.                criterion = nn.CrossEntropyLoss()
154.                run_loss = 0.0
155.                loss_list = []
156.                for e in range(epoch):
157.                    start = time.time()
158.                    train_net()
159.                    end = time.time()
160.                    print(f'训练时间为:
{round(end - start, 3)}s')
161.                    con_matrix = test_net()
162.                loss_lists.append(loss_list)
```

```
163.                 estimate = cal_conmat(con_matrix)
164.                 print_esti(estimate)
165.                 plot_conmat(con_matrix)
166.             plot_loss(loss_lists, dropout_list)
167.
168.         elif lr_test:
169.             loss_lists = []
170.             for lr in lr_list:
171.                 net = LeNet()
172.                 optimizer = optim.Adam(net.parameters(), lr=
lr)
173.
174.                 criterion = nn.CrossEntropyLoss()
175.                 run_loss = 0.0
176.                 loss_list = []
177.                 for e in range(epoch):
178.                     start = time.time()
179.                     train_net()
180.                     end = time.time()
181.                     print(f'训练时间为:
{round(end - start, 3)}s')
182.                     con_matrix = test_net()
183.                 loss_lists.append(loss_list)
184.                 estimate = cal_conmat(con_matrix)
185.                 print_esti(estimate)
186.                 plot_conmat(con_matrix)
187.             plot_loss(loss_lists, lr_list)
188.
189.         elif num_test:
190.             loss_lists = []
191.             for num in num_list:
192.                 train_data = torchvision.datasets.MNIST('mni
st-data', train=True, transform=transform)
193.                 train_data.data = train_data.data[:num]
194.                 train_data.targets = train_data.targets[:num]

195.                 train_loader = DataLoader(dataset=train_data,
 batch_size=batch_size, shuffle=True, num_workers=2)
196.                 train_num = len(train_data)
197.                 test_num = len(test_data)
198.                 print_num = int(train_num / (batch_size * 10)
)
199.                 net = LeNet()
```

```
200.                 optimizer = optim.Adam(net.parameters(), lr=
lr)
201.
202.                 criterion = nn.CrossEntropyLoss()
203.                 run_loss = 0.0
204.                 loss_list = []
205.                 for e in range(epoch):
206.                     start = time.time()
207.                     train_net()
208.                     end = time.time()
209.                     print(f'训练时间为:
{round(end - start, 3)}s')
210.                     con_matrix = test_net()
211.                 loss_lists.append(loss_list)
212.                 estimate = cal_conmat(con_matrix)
213.                 print_esti(estimate)
214.                 plot_conmat(con_matrix)
215.             plot_loss(loss_lists, num_list)
216.
217.         else:
218.             loss_lists = []
219.             net = LeNet()
220.             optimizer = optim.Adam(net.parameters(), lr=lr)

221.             criterion = nn.CrossEntropyLoss()
222.             run_loss = 0.0
223.             for e in range(epoch):
224.                 loss_list = []
225.                 start = time.time()
226.                 train_net()
227.                 end = time.time()
228.                 print(f'训练时间为:
{round(end - start, 3)}s')
229.                 con_matrix = test_net()
230.                 loss_lists.append(loss_list)
231.                 estimate = cal_conmat(con_matrix)
232.                 print_esti(estimate)
233.             plot_conmat(con_matrix)
234.             plot_loss(loss_lists, np.arange(1, epoch+1))
```

Lenet.py

```
1. # --*-- coding:utf-8 --*--
2. """
3. @Filename: LeNet.py
4. @Author: Keyan Xu
```

```
5. @Time: 2023-10-30
6. """
7. import torch
8. from torch import nn
9. import torch.nn.functional as F
10.
11.
12. class LeNet(nn.Module):
13.     def __init__(self, activate='relu', p=0.3):
14.         super().__init__()
15.         self.conv1 = nn.Conv2d(1, 8, 5)
16.         self.conv2 = nn.Conv2d(8, 16, 5)
17.         self.pool = nn.MaxPool2d(2)
18.         self.linear1 = nn.Linear(4 * 4 * 16, 120)
19.         self.linear2 = nn.Linear(120, 84)
20.         self.linear3 = nn.Linear(84, 10)
21.         print(f'dropout 参数 p 为{p}')
22.         self.dropout = nn.Dropout(p=p)
23.
24.         if activate == 'relu':
25.             print('使用 relu 函数作为激活函数')
26.             self.activate = nn.ReLU()
27.         elif activate == 'sigmoid':
28.             print('使用 sigmoid 函数作为激活函数')
29.             self.activate = nn.Sigmoid()
30.         elif activate == 'tanh':
31.             print('使用 tanh 函数作为激活函数')
32.             self.activate = nn.Tanh()
33.         elif activate == 'leaky_relu':
34.             print('使用 leaky_relu 函数作为激活函数')
35.             self.activate = nn.LeakyReLU()
36.         else:
37.             print('使用 relu 函数作为激活函数')
38.             self.activate = nn.ReLU()
39.
40.     def forward(self, x):
41.         x = self.activate(self.conv1(x))
42.         x = self.pool(x)
43.         x = self.activate(self.conv2(x))
44.         x = self.pool(x)
45.         x = torch.flatten(x, start_dim=1, end_dim=3)
46.         x = self.activate(self.linear1(x))
47.         x = self.dropout(x)
48.         x = self.activate(self.linear2(x))
```

```
        x = self.dropout(x)
        x = self.linear3(x)
        return x
```

7. 参考文献

无