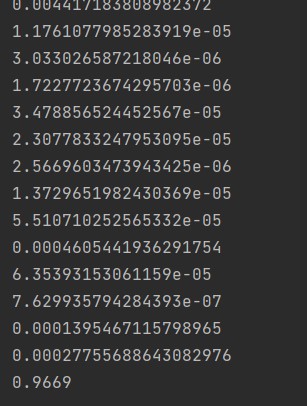
原版网络模型：

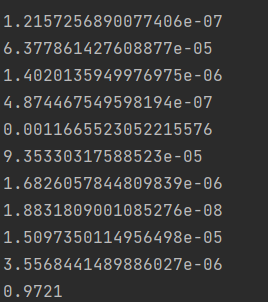
代码：

1. import torch
2. import torch.nn as nn
3. import torchvision
4. import PIL.Image as Image
5. import math  *# 主要做最后输出楼层向上取整*
6. import random
7. import pandas
8. from torch.utils.data import Dataset
9. import numpy
10. class MyDataset(Dataset):
11. def \_\_init\_\_(self, file):
12. *# 读取csv文件，头设置为空，df像一个二维数组,*
13. self.df = pandas.read\_csv(file, header=None)
14. *# 因为在getitem和len都要用，需要做指定成类级别的变量*
15. pass
16. def \_\_getitem\_\_(self, index):  *# 取i行的数据*
17. data = self.df.iloc[index]  *# 指定行数对于内容*
18. lables = data[0]  *# 第0列为标签*
19. values = data[1:]  *# 从第1列开始都是数值*
20. valtensorf = torch.tensor(values.values, dtype=torch.float32) / 255.0  *# 数据的数值改为tensorfloat32并归一化*
21. target = torch.zeros((10))  *# 生成一维的10个0*
22. target[lables] = 1.0  *# lables位置为1.0*
23. return lables, valtensorf, target
24. def \_\_len\_\_(self):  *# 数据集的长度，*
25. return len(self.df)
26. class Myconv(nn.Module):
27. def \_\_init\_\_(self):
28. super().\_\_init\_\_()  *# 初始化*
29. self.conv1 = nn.Conv2d(1, 16, kernel\_size=3, stride=1, padding=1)
30. self.conv2 = nn.Conv2d(16, 16, kernel\_size=3, stride=1, padding=1)
31. self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)
32. self.pool2 = nn.AvgPool2d(kernel\_size=2, stride=2)
33. self.out = nn.Linear(16 \* 7 \* 7, 10)  *# 输入，输出*
34. self.loss = nn.MSELoss()  *# MSEloss就是前面的误差(误差平方），self.loss现在是函数形式*
35. self.optmiser = torch.optim.SGD(self.parameters(), lr=0.05)  *# SGD优化器，学习率选择为0.05*
36. self.count = 0  *# 训练计数*
37. self.progress = []  *# 进度表示需要存储的内容，用列表来表示*
38. def forward(self, x):  *# 将输入矩阵转换为输出*
39. x = self.conv1(x)
40. x = self.pool(x)
41. x = self.conv2(x)
42. x = self.pool2(x)
43. x = x.view((16 \* 7 \* 7))  *# 变成一纬然后送到fc层*
44. x = torch.sigmoid(self.out(x))
45. return x
46. def train(self, input, target):  *# 定义训练方法*
47. output = self.forward(input)
48. myloss = self.loss(output, target)  *# output,target为矩阵可能包含很多数据*
49. self.optmiser.zero\_grad()  *# 梯度清零*
50. myloss.backward()  *# 用误差做反向传播计算*
51. self.optmiser.step()  *# 调用优化器step方法进行梯度更新*
52. if self.count % 1000 == 0:
53. self.progress.append(myloss.item())  *# 每隔1000次记录一次误差，并添加到progress列表里*
54. self.count = self.count + 1
55. net = Myconv()  *# 定义对象net*
56. traindataset = MyDataset("d:/dataset/deeplearningclass/mnist\_train.csv")  *# 包含很多行的训练数据*
57. testdataset = MyDataset("d:/dataset/deeplearningclass/mnist\_test.csv")
58. count = 0
59. item = 0
60. *# 训练并查看结果*
61. for epoch in range(0, 3):
62. for lables, valtensorf, target in traindataset:  *# 每次都执行一次getitem*
63. valtensorf = valtensorf.view(1, 1, 28, 28)
64. net.train(valtensorf, target)
65. if net.count % 1000 == 0:
66. print(net.progress[-1])
67. *# 测试集测试准确率*
68. for lables, data, target in testdataset:
69. output = net.forward(data.view((1, 1, 28, 28))).detach().numpy()
70. if output.argmax()==lables:   *# argmax数字最大的位置*
71. count = count+1
72. item = item+1
73. print(count/item)   *# 正确率*

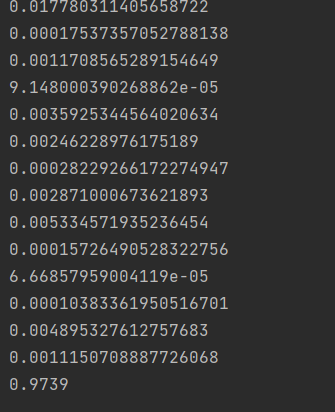
运行结果，准确率96.69％



单纯提高学习率到0.085，准确率提升至97.21％



在原网络模型基础上更换激活函数为relu，准确率打到97.39：

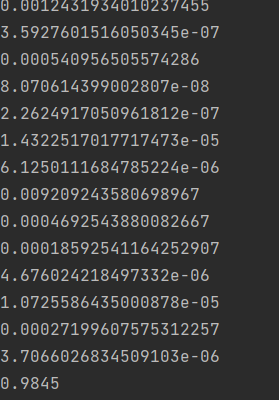


学习率0.05激活函数为sigmoid的情况下，将第二个卷积改为残差块自定义训练代码如下：

1. def forward(self, x):  *# 将输入矩阵转换为输出*
2. x1 = self.conv1(x)  *#*
3. *# r1 = torch.relu(self.bn(x1))*
4. y=self.pool(x1)
5. *# y = self.pool(x+r1)*
6. x2 = self.conv2(y)
7. r2 = torch.relu(self.bn(x2))
8. x = y+r2
9. y2 = self.pool2(x)
10. fc = y2.view((16 \* 7 \* 7))  *# 变成一纬然后送到fc层*
11. fo = torch.sigmoid(self.out(fc))
12. *# x = torch.relu(self.out(x))*
13. *# x = torch.sigmoid(self.out(x))*
14. return fo

且在原始代码32行处添加批归一化代码：self.bn = nn.BatchNorm2d(16)

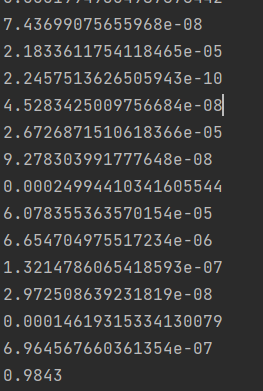
准确率提升至98.45%如图：



学习率0.05激活函数为sigmoid的情况下，将第一个卷积与第二个卷积各自改为残差块（总共2个）自定义训练处代码如下：

1. def forward(self, x):  *# 将输入矩阵转换为输出*
2. x1 = self.conv1(x)  *#*
3. r1 = torch.relu(self.bn(x1))
4. y=self.pool(x+r1)
5. x2 = self.conv2(y)
6. r2 = torch.relu(self.bn(x2))
7. x = y+r2
8. y2 = self.pool2(x)
9. fc = y2.view((16 \* 7 \* 7))  *# 变成一纬然后送到fc层*
10. fo = torch.sigmoid(self.out(fc))
11. *# x = torch.relu(self.out(x))*
12. *# x = torch.sigmoid(self.out(x))*
13. return fo

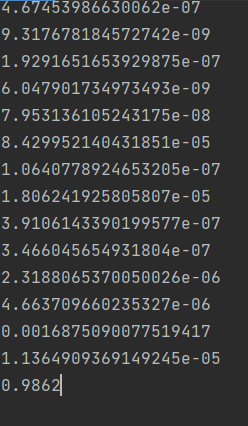
准确率达到98.43%结果如下图：



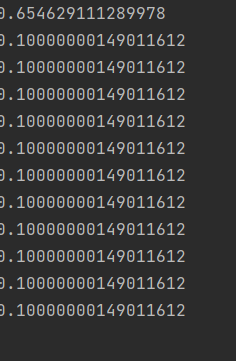
学习率0.05激活函数为sigmoid的情况下，将第一个卷积改为残差块，第二个卷积改为5个连续的残差块（总共6个），自定义训练处代码如下：

1. def forward(self, x):  *# 将输入矩阵转换为输出*
2. x1 = self.conv1(x)  *#*
3. r1 = torch.relu(self.bn(x1))
4. y=self.pool(x+r1)
5. x2 = self.conv2(y)
6. r2 = torch.relu(self.bn(x2))
7. x = r2+y
8. x3 = self.conv2(x)
9. r3 = torch.relu(self.bn(x3))
10. x = r3 + y
11. x4 = self.conv2(x)
12. r4 = torch.relu(self.bn(x4))
13. x = r4 + y
14. x5 = self.conv2(x)
15. r5 = torch.relu(self.bn(x5))
16. x = r5 + y
17. x6 = self.conv2(x)
18. r6 = torch.relu(self.bn(x6))
19. x = r6 + y
20. y2 = self.pool2(x)
21. fc = y2.view((16 \* 7 \* 7))  *# 变成一纬然后送到fc层*
22. fo = torch.sigmoid(self.out(fc))
23. *# x = torch.relu(self.out(x))*
24. *# x = torch.sigmoid(self.out(x))*
25. return fo

准确率达到98.62%如图所示：



一个疑问，如果使用残差块，激活函数使用relu的话会得到如下误差：

最终结果非常差劲，不知道为什么