十个残差块代码如下：

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 ResBlock(nn.Module):  *# 定义残差块*
27. def \_\_init\_\_(self, n\_channels):
28. super().\_\_init\_\_()
29. self.conv2 = nn.Conv2d(n\_channels, n\_channels, kernel\_size=3, stride=1, padding=1)
30. self.bn = nn.BatchNorm2d(n\_channels)
31. def forward(self, x):  *# 残差块实现过程*
32. x1 = self.conv2(x)
33. r1 = torch.relu(self.bn(x1))
34. x = x + r1
35. return x
36. class Myconv(nn.Module):
37. def \_\_init\_\_(self):
38. super().\_\_init\_\_()  *# 初始化*
39. self.conv1 = nn.Conv2d(1, 16, kernel\_size=3, stride=1, padding=1)
40. *# self.conv2 = nn.Conv2d(16, 16, kernel\_size=3, stride=1, padding=1)*
41. *# self.bn = nn.BatchNorm2d(16)*
42. self.resBlocks = nn.Sequential(  *# 序列形式执行*
43. \*([ResBlock(16)] \* 10)  *# 变成列表乘以残差块个数*
44. )
45. self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)
46. self.pool2 = nn.AvgPool2d(kernel\_size=2, stride=2)
47. self.out = nn.Linear(16 \* 7 \* 7, 10)  *# 输入，输出*
48. 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*
49. self.optmiser = torch.optim.SGD(self.parameters(), lr=0.05)  *# SGD优化器，学习率选择为0.05*
50. self.count = 0  *# 训练计数*
51. self.progress = []  *# 进度表示需要存储的内容，用列表来表示*
52. def forward(self, x):  *# 将输入矩阵转换为输出*
53. x = self.conv1(x)  *#*
54. x = self.pool(x)
55. x = self.resBlocks(x)
56. x = self.pool2(x)
57. x = x.view((16 \* 7 \* 7))  *# 变成一纬然后送到fc层*
58. x = torch.sigmoid(self.out(x))
59. *# x = torch.relu(self.out(x))*
60. *# x = torch.sigmoid(self.out(x))*
61. return x
62. def train(self, input, target):  *# 定义训练方法*
63. output = self.forward(input)
64. myloss = self.loss(output, target)  *# output,target为矩阵可能包含很多数据*
65. self.optmiser.zero\_grad()  *# 梯度清零*
66. myloss.backward()  *# 用误差做反向传播计算*
67. self.optmiser.step()  *# 调用优化器step方法进行梯度更新*
68. if self.count % 1000 == 0:
69. self.progress.append(myloss.item())  *# 每隔1000次记录一次误差，并添加到progress列表里*
70. self.count = self.count + 1
71. net = Myconv()  *# 定义对象net*
72. *#net.load\_state\_dict("d:/dataset/deeplearningclass/model.2pt")  # 加载以前的模型*
73. traindataset = MyDataset("d:/dataset/deeplearningclass/mnist\_train.csv")  *# 包含很多行的训练数据*
74. testdataset = MyDataset("d:/dataset/deeplearningclass/mnist\_test.csv")
75. count = 0
76. item = 0
77. *# 训练并查看结果*
78. for epoch in range(0, 3):
79. for lables, valtensorf, target in traindataset:  *# 每次都执行一次getitem*
80. valtensorf = valtensorf.view(1, 1, 28, 28)
81. net.train(valtensorf, target)
82. if net.count % 1000 == 0:
83. print(net.progress[-1])
84. if net.count % 10000 == 0:
85. torch.save(net.state\_dict(), "d:/dataset/deeplearningclass/model" + str(epoch) + ".pt")  *# 每一万次保存一轮次*
86. *# torch.save(net.state\_dict(),"d:/dataset/deeplearningclass/model.pt")*
87. *# 测试集测试准确率*
88. for lables, data, target in testdataset:
89. output = net.forward(data.view((1, 1, 28, 28))).detach().numpy()
90. if output.argmax() == lables:  *# argmax数字最大的位置*
91. count = count + 1
92. item = item + 1
93. print(count / item)  *# 正确率*

运行结果如下：文本

中度可信度描述已自动生成

疑问：精确度下降很多，而且损失下降很慢，且有时候损失都是0.10000那个，不知道为啥