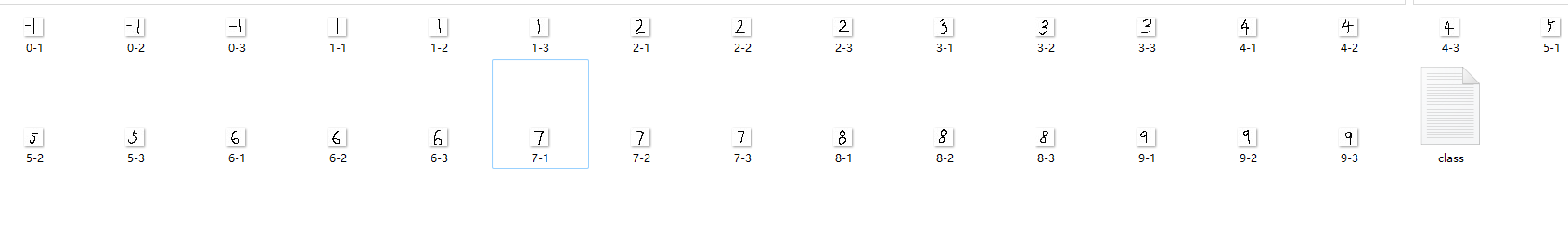
自建数据集：



Class.txt文本文档存储图片名称和标签如下：

0-1.png,1

0-2.png,1

0-3.png,1

1-1.png,2

1-2.png,2

1-3.png,2

2-1.png,3

2-2.png,3

2-3.png,3

3-1.png,4

3-2.png,4

3-3.png,4

4-1.png,5

4-2.png,5

4-3.png,5

5-1.png,6

5-2.png,6

5-3.png,6

6-1.png,7

6-2.png,7

6-3.png,7

7-1.png,8

7-2.png,8

7-3.png,8

8-1.png,9

8-2.png,9

8-3.png,9

9-1.png,10

9-2.png,10

9-3.png,10

代码如下：

1. import random
2. import torch
3. import torch.nn as nn
4. import PIL.Image as Image  *# Image 可以从图像读取数据*
5. import math    *# 主要做最后输出楼层向上取整*
6. data = []
7. flag = []  *# 期望值标签列表组成的矩阵*
8. with open("d:/dataset/deeplearningclass/class.txt") as f:
9. for line in f:  *# 循环line次*
10. words = line.split(",")  *# words为列表*
11. img = Image.open("d:/dataset/deeplearningclass/" + words[0])  *# 打开图片装载进内存，这里第0项是文件名*
12. img = img.convert("L")  *# convert 把原来的图像做转换，L为二值图像只有黑色白色 tips：原图片像素为三原色*
13. iml = list(img.getdata())  *# 转换成列表  getdata从图里面把每一个像素的值取出*
14. data.append([iml])
15. i = int(words[1])  *# 取标签*
16. d = [0.0]\*10      *# 初始化标签列表*
17. d[i - 1] = 1.0
18. flag.append([d])
19. *# print(flag)*
20. X = torch.tensor(data) / 255.0  *# 把data数据转换成tensor，并转换成浮点数做归一化*
21. *# print(X)*
22. S = torch.tensor(flag)  *# 标签列表转换成tensor*
23. class MyClass(nn.Module):  *# 继承nn.moudle*
24. def \_\_init\_\_(self):
25. super().\_\_init\_\_()  *# super指代父类，这指示nn.Moudle   这里主要做初始化*
26. self.modle = nn.Sequential(  *# Sequential可以按序执行*
27. nn.Linear(400, 50),
28. nn.Sigmoid(),
29. nn.Linear(50, 10),
30. nn.Sigmoid()
31. )
32. 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*
33. self.optmiser = torch.optim.SGD(self.parameters(), lr=0.05)  *# SGD优化器，学习率选择为0.05*
34. self.count = 0  *# 训练计数*
35. self.progress = []  *# 进度表示需要存储的内容，用列表来表示*
36. def forward(self, input):  *# 定义网络计算方法*
37. return self.modle(input)
38. def train(self, input, target):  *# 定义训练方法*
39. output = self.forward(input)
40. myloss = self.loss(output, target)  *# output,target为矩阵可能包含很多数据*
41. self.optmiser.zero\_grad()  *# 梯度清零*
42. myloss.backward()  *# 用误差做反向传播计算*
43. self.optmiser.step()  *# 调用优化器step方法进行梯度更新*
44. if self.count % 1000 == 0:
45. self.progress.append(myloss.item())  *# 每隔1000次记录一次误差，并添加到progress列表里*
46. self.count = self.count + 1
47. net = MyClass()  *# 定义对象net*
48. epoch = 100000
49. for j in range(epoch):  *# 循环epoch次，这里为100000次*
50. i = random.randint(0, 29)
51. net.train(X[i], S[i])
52. ot = net.forward(X[i])
53. loss = net.progress[-1]
54. tag = math.ceil((i+1)/3)  *# 输出的时候对应的标签值，为了方便观察输出结果*
55. if j % 1000 == 0:
56. *# print(net.progress[-1])  # 输出损失列表的最后一项*
57. *# print(net.forward(X[i]))*
58. print('i=', i, 'tag：', tag, 'loss=', loss, 'final output is:', ot)

结果显示如下：

