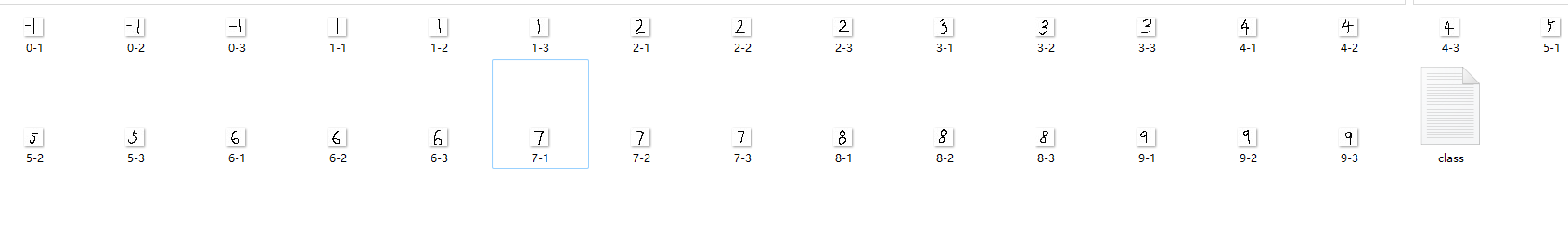
自建数据集：



Class.txt文本文档存储图片名称和标签如下：

0-1.png,1

0-2.png,2

0-3.png,3

1-1.png,4

1-2.png,5

1-3.png,6

2-1.png,7

2-2.png,8

2-3.png,9

3-1.png,10

3-2.png,11

3-3.png,12

4-1.png,13

4-2.png,14

4-3.png,15

5-1.png,16

5-2.png,17

5-3.png,18

6-1.png,19

6-2.png,20

6-3.png,21

7-1.png,22

7-2.png,23

7-3.png,24

8-1.png,25

8-2.png,26

8-3.png,27

9-1.png,28

9-2.png,29

9-3.png,30

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. data = []
6. flag = []  *# 期望值标签列表组成的矩阵*
7. with open("d:/dataset/deeplearningclass/class.txt") as f:
8. for line in f:  *# 循环line次*
9. words = line.split(",")  *# words为列表*
10. 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*
11. 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*
12. iml = list(img.getdata())  *# 转换成列表  getdata从图里面把每一个像素的值取出*
13. data.append([iml])
14. i = int(words[1])  *# 取标签*
15. d = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  *# 初始化标签列表*
16. 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
17. 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
18. d[i - 1] = 1.0
19. flag.append([d])
20. *# print(flag)*
21. 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*
22. *# print(X)*
23. S = torch.tensor(flag)  *# 标签列表转换成tensor*
24. '''X = torch.tensor([[[0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0]],  # 这个代表-1楼以此类推
25. [[0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0]],
26. [[1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0]],
27. [[1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0]],
28. [[0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0]],
29. [[1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0]],
30. [[1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0]],
31. [[1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0]],
32. [[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]],
33. [[1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0]]])
34. # 以下是期望输出值，用二进制数表示
35. S = torch.tensor([[[0.0, 0.0, 0.0, 0.0]],
36. [[0.0, 0.0, 0.0, 1.0]],
37. [[0.0, 0.0, 1.0, 0.0]],
38. [[0.0, 0.0, 1.0, 1.0]],
39. [[0.0, 1.0, 0.0, 0.0]],
40. [[0.0, 1.0, 0.0, 1.0]],
41. [[0.0, 1.0, 1.0, 0.0]],
42. [[0.0, 1.0, 1.0, 1.0]],
43. [[1.0, 0.0, 0.0, 0.0]],
44. [[1.0, 0.0, 0.0, 1.0]]])
45. '''
46. class MyClass(nn.Module):  *# 继承nn.moudle*
47. def \_\_init\_\_(self):
48. super().\_\_init\_\_()  *# super指代父类，这指示nn.Moudle   这里主要做初始化*
49. self.modle = nn.Sequential(  *# Sequential可以按序执行*
50. nn.Linear(400, 50),
51. nn.Sigmoid(),
52. nn.Linear(50, 30),
53. nn.Sigmoid()
54. )
55. 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*
56. self.optmiser = torch.optim.SGD(self.parameters(), lr=0.05)  *# SGD优化器，学习率选择为0.05*
57. self.count = 0  *# 训练计数*
58. self.progress = []  *# 进度表示需要存储的内容，用列表来表示*
59. def forward(self, input):  *# 定义网络计算方法*
60. return self.modle(input)
61. def train(self, input, target):  *# 定义训练方法*
62. output = self.forward(input)
63. myloss = self.loss(output, target)  *# output,target为矩阵可能包含很多数据*
64. self.optmiser.zero\_grad()  *# 梯度清零*
65. myloss.backward()  *# 用误差做反向传播计算*
66. self.optmiser.step()  *# 调用优化器step方法进行梯度更新*
67. if self.count % 1000 == 0:
68. self.progress.append(myloss.item())  *# 每隔1000次记录一次误差，并添加到progress列表里*
69. self.count = self.count + 1
70. net = MyClass()  *# 定义对象net*
71. epoch = 600000
72. for j in range(epoch):  *# 循环epoch次，这里为10000次*
73. i = random.randint(0, 29)
74. net.train(X[i], S[i])
75. ot = net.forward(X[i])
76. loss = net.progress[-1]
77. if j % 1000 == 0:
78. *# print(net.progress[-1])  # 输出损失列表的最后一项*
79. *# print(net.forward(X[i]))*
80. print('i=', i, 'loss=', loss, 'final output is:', ot)

结果显示如下：

