

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **卷积神经网络实验** |
| **姓 名：** | **束天成** |
| **学 号：** | **22121501** |
| **日 期：** | **2022/8/13** |

## 一、实验内容

本次实验为卷积神经网络的基本实验，本次实验主要包含了二维卷积实验，空洞卷积实验和残差网络实验，编程的语言为Python，其目的是了解和掌握卷积神经网络的基本原理，并将其运用到实际操作中。

#### 1.1 PyTorch基本操作实验

二维卷积实验包含了手写二维卷积的实现、使用torch.nn实现卷积两个实验，对比不同的实现方式对实验结果产生的不同影响，此外还需要设置不同的参数，对比不同的超参数和收敛度以及训练损失之间的联系。

#### 1.2 空洞卷积实验

空洞卷积实验在本质上也是二维卷积实验的一种，但是为了扩大感受视野和捕获多尺度的上下文信息，在代码格式上表现为dilation参数的不同。在实验结果上看，可以减少训练损失，提高训练精度。本次实验主要使用torch.nn实现，设置不同的参数，对比实验结果，进行简单分析。

#### 1.3 残差网络实验

## 残差网络使用多个不同的卷积层进行运算，在理论上有着更佳的训练向。本次实验的要求是给定结构的残差网络，在至少一个数据集上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验结果

## 二、实验设计

本次实验主要内容都为指定内容，均指定了实验需要的数据矩阵维度，矩阵构造方式，运算函数，使用的方法以及需要分析的内容，不包含过多需要自主设计的模型和方法等，因此不再赘述。

## 三、实验环境及实验数据集

Torch版本：1.5.1

操作系统：Windows10

硬件：

CPU：i5 9300h GPU：GTX1650

数据集：torch.nn，Fashion-MNIST数据集，nn.BCEWithLogitsLoss函数等

## 四、实验过程

实验过程包括在编写代码时一些需要注意的事项，可附代码片段进行说明；还应该包括在训练模型时进行的一些步骤、参数设置等内容。

注：为了代码片段尽量的美观、统一，建议附代码片段时只附加关键的片段，不要全部粘贴，并尽量使用下面提供的网站进行代码高亮等格式转换后再粘贴。

<http://www.planetb.ca/syntax-highlight-word>

#### 4.1 二维卷积实验

**4.1.1**手写二维卷积的实现，并在至少一个数据集上进行实验，从训练时间、预测精度、Loss变化等角度分析实验结果。

1. import os
2. import torch.nn as nn
3. import torch.nn.functional as F
4. from PIL import Image
5. import matplotlib.pyplot as plt
6. import numpy as np
7. import torch
8. from torch.utils.data import Dataset
9. from torchvision import transforms
10. import random
11. *#导入做实验必要的数据包等相关内容*
12. num\_classes = 3  *# 分类数量*
13. batch\_size = 64 *#读取数量大小*
14. num\_epochs = 10  *# 训练轮次*
15. lr = 0.001  *#学习率*
16. device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')
17. file\_root = "D:\实验3数据包\数据包\车辆分类数据集"
18. classes = ['bus', 'car', 'truck']  *# 分别对应三个标签*
19. nums = [218, 779, 360]  *# 每种类别的个数*
20. def read\_data(path):
21. file\_name = os.listdir(path)  *# 打开对应的文件夹*
22. train\_data = []
23. train\_labels = []
24. test\_data = []
25. test\_labels = []
26. train\_num = [int(num \* 4 / 5) for num in nums]*#定义训练集数量*
27. test\_num = [nums[i] - train\_num[i] for i in range(len(nums))]*#定义测试集数量*
29. for idx, f\_name in enumerate(file\_name):  *# 定义数据标签等相关内容*
30. im\_dirs = path + '/' + f\_name
31. im\_path = os.listdir(im\_dirs)  *# 每个不同类别图像文件夹下所有图像的名称*
32. index = list(range(len(im\_path)))
33. random.shuffle(index)  *# 打乱顺序*
34. im\_path\_ = list(np.array(im\_path)[index])
35. test\_path = im\_path\_[:test\_num[idx]]  *# 测试数据的路径*
36. train\_path = im\_path\_[test\_num[idx]:]  *# 训练数据的路径*
38. for img\_name in train\_path:
39. if img\_name == 'desktop.ini':
40. continue
41. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
42. img = img.resize((32, 32), Image.ANTIALIAS)  *# 对图片进行变形*
43. train\_data.append(img)
44. train\_labels.append(idx)
45. for img\_name in test\_path:
46. if img\_name == 'desktop.ini':
47. continue
48. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
49. img = img.resize((32, 32), Image.ANTIALIAS)  *# 对图片进行变形*
50. test\_data.append(img)
51. test\_labels.append(idx)
52. print('训练集大小：', len(train\_data), ' 测试集大小：', len(test\_data))
53. return train\_data, train\_labels, test\_data, test\_labels
54. train\_data, train\_labels, test\_data, test\_labels = read\_data(file\_root)
55. *# 定义一个Transform操作*
56. transform = transforms.Compose(
57. [transforms.ToTensor(),  *# 变为tensor*
58. *# 对数据按通道进行标准化，即减去均值，再除以方差, [0-1]->[-1,1]*
59. transforms.Normalize(mean=[0.4686, 0.4853, 0.5193], std=[0.1720, 0.1863, 0.2175])
60. ]
61. )*#将数据类型进行转换，最终要转换成Tensor*
62. *# 自定义Dataset类实现每次取出图片，将PIL转换为Tensor*
63. class MyDataset(Dataset):
64. def \_\_init\_\_(self, data, label, trans):
65. self.len = len(data)
66. self.data = data
67. self.label = label
68. self.trans = trans
69. def \_\_getitem\_\_(self, index):  *# 根据索引返回数据和对应的标签*
70. return self.trans(self.data[index]), self.label[index]
71. def \_\_len\_\_(self):
72. return self.len
73. *# 调用自己创建的Dataset*
74. train\_dataset = MyDataset(train\_data, train\_labels, transform)
75. test\_dataset = MyDataset(test\_data, test\_labels, transform)
76. *# 生成data loader*
77. train\_iter = torch.utils.data.DataLoader(
78. dataset=train\_dataset,
79. batch\_size=batch\_size,
80. shuffle=True,
81. num\_workers=0
82. )*#得到数据训练集*
83. test\_iter = torch.utils.data.DataLoader(
84. dataset=test\_dataset,
85. batch\_size=batch\_size,
86. shuffle=False,
87. num\_workers=0
88. )*#得到数据测试集*
89. def conv2d(X, K):*#定义卷积运算，详情见PPT*
90. '''
91. :param X: 样本输入，shape(batch\_size,H,W)
92. :param K: 卷积核，shape(k\_h,k\_w)
93. :return: Y 卷积结果，shape(batch\_size, H-k\_h+1, W-k\_w+1)
94. '''
95. batch\_size, H, W = X.shape
96. k\_h, k\_w = K.shape
97. *# 初始化 Y*
98. Y = torch.zeros((batch\_size, H - k\_h + 1, W - k\_w + 1)).to(device)
99. for i in range(Y.shape[1]):
100. for j in range(Y.shape[2]):
101. Y[:, i, j] = (X[:, i: i + k\_h, j:j + k\_w] \* K).sum(dim=2).sum(dim=1)
102. return Y
103. def conv2d\_multi\_in(X, K):*#多通道卷积输入，详情见CSDN*
104. '''
105. :param X: (batch\_size, C\_in，H，W)代表有C个输入通道
106. :param K: (C\_in, k\_h, k\_w)
107. :return: (batch\_size, H\_out, W\_out)
108. '''
109. res = conv2d(X[:, 0, :, :], K[0, :, :])
110. for i in range(1, X.shape[1]):  *# 多个通道的结果相加*
111. res += conv2d(X[:, i, :, :], K[i, :, :])
112. return res
113. *# 实现多输出通道*
114. *# 输出通道数 = 卷积核个数*
115. def conv2d\_multi\_in\_out(X, K):*#实现卷积网络的多通道输出*
116. '''
117. :param X: (batch\_size, C\_in，H，W)代表有C个输入通道
118. :param K: (K\_num, C\_in, k\_h, k\_w) k\_num表示卷积核的个数
119. :return: (batch\_size, K\_num, H\_out, W\_out)
120. '''
121. return torch.stack([conv2d\_multi\_in(X, k) for k in K], dim=1)
122. class MyConv2D(torch.nn.Module):*#对卷积进行封装*
123. def \_\_init\_\_(self, in\_channels, out\_channels, kernel\_size):
124. super(MyConv2D, self).\_\_init\_\_()
125. *# 初始化卷积层2个参数：卷积核、偏差*
126. if isinstance(kernel\_size, int):  *# 如果kernel size是一个数*
127. kernel\_size = (kernel\_size, kernel\_size)
128. *# weight的shape:(卷积核个数/输出通道数，输入通道数，卷积核高，卷积核宽)*
129. *# torch.randn:返回一个符合均值为0，方差为1的正态分布（标准正态分布）中填充随机数的张量*
130. self.weight = torch.nn.Parameter(torch.randn((out\_channels, in\_channels) + kernel\_size))
131. self.bias = torch.nn.Parameter(torch.randn(out\_channels, 1, 1))
132. def forward(self, x):
133. '''
134. :param x:
135. :return:
136. '''
137. return conv2d\_multi\_in\_out(x, self.weight) + self.bias
138. class MyConvModule(torch.nn.Module):*#卷积网络的构建*
139. def \_\_init\_\_(self):
140. super(MyConvModule, self).\_\_init\_\_()
141. *# 定义一层卷积*
142. self.conv = torch.nn.Sequential(
143. MyConv2D(in\_channels=3, out\_channels=32, kernel\_size=3),
144. torch.nn.BatchNorm2d(32),
145. torch.nn.ReLU(inplace=True)  *# inplace=True表示计算出来的结果会替换掉原来的Tensor*
146. )
147. *# 输出层，将输出通道数变为分类数量*
148. self.fc = torch.nn.Linear(32, num\_classes)
149. def forward(self, X):
150. *# 图片经过一层卷积，输出（batch\_size，C\_out, H, W)*
151. out = self.conv(X)
152. *# 使用平均池化层将图片大小变为1\*1(图片原大小32\*32,卷积后为30\*30*
153. out = F.avg\_pool2d(out, 30)
154. *# 将out从shape batch\_size\*32\*1\*1变为batch\_size\*32*
155. out = out.squeeze()  *# squeeze的用法：*
156. *# 输入到全连接层*
157. out = self.fc(out)
158. return out
159. net = MyConvModule()
160. net.to(device)
161. *# 损失函数和优化器*
162. loss = torch.nn.CrossEntropyLoss()  *# 交叉熵损失函数*
163. optimizer = torch.optim.SGD(net.parameters(), lr=lr)
164. def train(net, data\_loader, device):*#模型训练*
165. net.train()  *# 指定为训练模式*
166. train\_batch\_num = len(data\_loader)
167. total\_loss = 0.0
168. correct = 0  *# 记录共有多少个样本被正确分类*
169. sample\_num = 0
170. *# 遍历每个batch进行训练*
171. for data, target in data\_loader:
172. *# 将图片和标签放入指定的device中*
173. data = data.to(device)
174. target = target.to(device)
175. *# 将当前梯度清零*
176. optimizer.zero\_grad()
177. *# 使用模型计算出结果*
178. y\_hat = net(data)
179. *# 计算损失*
180. loss\_ = loss(y\_hat, target)
181. *# 进行反向传播*
182. loss\_.backward()
183. optimizer.step()
184. total\_loss += loss\_.item()
185. cor = (torch.argmax(y\_hat, 1) == target).sum().item()
186. correct += cor
187. *# 累加当前的样本总数*
188. sample\_num += target.shape[0]
189. print('loss: %.4f  acc: %.4f' % (loss\_.item(), cor/target.shape[0]))
190. *# 平均loss和准确率*
191. loss\_ = total\_loss / train\_batch\_num
192. acc = correct / sample\_num
193. return loss\_, acc
194. *# 对模型进行测试*
195. def test(net, data\_loader, device):
196. net.eval()  *# 指定当前模式为测试模式（针对BN层和dropout层）*
197. test\_batch\_num = len(data\_loader)
198. total\_loss = 0
199. correct = 0
200. sample\_num = 0
201. *# 指定不进行梯度计算（没有反向传播也会计算梯度，增大GPU开销*
202. with torch.no\_grad():
203. for data, target in data\_loader:
204. data = data.to(device)
205. target = target.to(device)
206. output = net(data)
207. loss\_ = loss(output, target)
208. total\_loss += loss\_.item()
209. correct += (torch.argmax(output, 1) == target).sum().item()
210. sample\_num += target.shape[0]
211. loss\_ = total\_loss / test\_batch\_num
212. acc = correct / sample\_num
213. return loss\_, acc
214. *# 模型训练与测试*
215. train\_loss\_list = []
216. train\_acc\_list = []
217. test\_loss\_list = []
218. test\_acc\_list = []
219. for epoch in range(num\_epochs):
220. *# 在训练集上训练*
221. train\_loss, train\_acc = train(net, data\_loader=train\_iter, device=device)
222. *# 测试集上验证*
223. test\_loss, test\_acc = test(net, data\_loader=test\_iter, device=device)
224. train\_loss\_list.append(train\_loss)
225. train\_acc\_list.append(train\_acc)
226. test\_loss\_list.append(test\_loss)
227. test\_acc\_list.append(test\_acc)
228. print('epoch %d, train loss: %.4f, train acc: %.3f' % (epoch+1, train\_loss, train\_acc))
229. print('test loss: %.4f, test acc: %.3f' % (test\_loss, test\_acc))
230. *# 绘制函数*
231. def draw\_(x, train\_Y, test\_Y, ylabel):
232. plt.plot(x, train\_Y, label='train\_' + ylabel, linewidth=1.5)
233. plt.plot(x, test\_Y, label='test\_' + ylabel, linewidth=1.5)
234. plt.xlabel('epoch')
235. plt.ylabel(ylabel)
236. plt.legend()  *# 加上图例*
237. plt.show()
238. *# 绘制loss曲线*
239. x = np.linspace(0, len(train\_loss\_list), len(train\_loss\_list))
240. draw\_(x, train\_loss\_list, test\_loss\_list, 'loss')
241. draw\_(x, train\_acc\_list, test\_acc\_list, 'accuracy')

**4.1.2**使用torch.nn实现二维卷积，并在至少一个数据集上进行实验，从训练时间、 预测精度、Loss变化等角度分析实验结果（最好使用图表展示）

1. import os
2. import torch.nn as nn
3. import torch.nn.functional as F
4. from PIL import Image
5. import matplotlib.pyplot as plt
6. import numpy as np
7. import torch
8. from torch.utils.data import Dataset
9. from torchvision import transforms
10. import random
11. import time
12. *#导入做实验必要的数据包等相关内容*
13. num\_classes = 3  *# 分类数量*
14. batch\_size = 128 *#读取数量大小*
15. device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')
16. file\_root = "D:\实验3数据包\数据包\车辆分类数据集"
17. classes = ['bus', 'car', 'truck']  *# 分别对应三个标签*
18. nums = [218, 779, 360]  *# 每种类别的个数*
19. def read\_data(path):
20. file\_name = os.listdir(path)  *# 打开对应的文件夹*
21. train\_data = []
22. train\_labels = []
23. test\_data = []
24. test\_labels = []
25. train\_num = [int(num \* 4 / 5) for num in nums]*#定义训练集数量*
26. test\_num = [nums[i] - train\_num[i] for i in range(len(nums))]*#定义测试集数量*
28. for idx, f\_name in enumerate(file\_name):  *# 定义数据标签等相关内容*
29. im\_dirs = path + '/' + f\_name
30. im\_path = os.listdir(im\_dirs)  *# 每个不同类别图像文件夹下所有图像的名称*
31. index = list(range(len(im\_path)))
32. random.shuffle(index)  *# 打乱顺序*
33. im\_path\_ = list(np.array(im\_path)[index])
34. test\_path = im\_path\_[:test\_num[idx]]  *# 测试数据的路径*
35. train\_path = im\_path\_[test\_num[idx]:]  *# 训练数据的路径*
37. for img\_name in train\_path:
38. if img\_name == 'desktop.ini':
39. continue
40. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
41. img = img.resize((32, 32), Image.ANTIALIAS)  *# 对图片进行变形*
42. train\_data.append(img)
43. train\_labels.append(idx)
44. for img\_name in test\_path:
45. if img\_name == 'desktop.ini':
46. continue
47. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
48. img = img.resize((32, 32), Image.ANTIALIAS)  *# 对图片进行变形*
49. test\_data.append(img)
50. test\_labels.append(idx)
51. print('训练集大小：', len(train\_data), ' 测试集大小：', len(test\_data))
52. return train\_data, train\_labels, test\_data, test\_labels
53. train\_data, train\_labels, test\_data, test\_labels = read\_data(file\_root)
54. *# 定义一个Transform操作*
55. transform = transforms.Compose(
56. [transforms.ToTensor(),  *# 变为tensor*
57. *# 对数据按通道进行标准化，即减去均值，再除以方差, [0-1]->[-1,1]*
58. transforms.Normalize(mean=[0.4686, 0.4853, 0.5193], std=[0.1720, 0.1863, 0.2175])
59. ]
60. )*#将数据类型进行转换，最终要转换成Tensor*
61. *# 自定义Dataset类实现每次取出图片，将PIL转换为Tensor*
62. class MyDataset(Dataset):
63. def \_\_init\_\_(self, data, label, trans):
64. self.len = len(data)
65. self.data = data
66. self.label = label
67. self.trans = trans
68. def \_\_getitem\_\_(self, index):  *# 根据索引返回数据和对应的标签*
69. return self.trans(self.data[index]), self.label[index]
70. def \_\_len\_\_(self):
71. return self.len
72. *# 调用自己创建的Dataset*
73. train\_dataset = MyDataset(train\_data, train\_labels, transform)
74. test\_dataset = MyDataset(test\_data, test\_labels, transform)
75. *# 生成data loader*
76. train\_iter = torch.utils.data.DataLoader(
77. dataset=train\_dataset,
78. batch\_size=batch\_size,
79. shuffle=True,
80. num\_workers=0
81. )*#得到数据训练集*
82. test\_iter = torch.utils.data.DataLoader(
83. dataset=test\_dataset,
84. batch\_size=batch\_size,
85. shuffle=False,
86. num\_workers=0
87. )*#得到数据测试集*
88. *# 使用torch.nn实现二维卷积*
89. class ConvModule(torch.nn.Module):
90. def \_\_init\_\_(self):
91. super(ConvModule, self).\_\_init\_\_()
92. *# 定义一个三层卷积（卷积层越少曲线越平滑，卷积层越多acc越高）*
93. self.conv = torch.nn.Sequential(
94. *# 第一层卷积*
95. *# stride步长,padding填充*
96. torch.nn.Conv2d(in\_channels=3, out\_channels=32, kernel\_size=3, stride=1, padding=0),
97. torch.nn.BatchNorm2d(32),
98. torch.nn.ReLU(inplace=True),
99. *# 第二层卷积*
100. torch.nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, stride=1, padding=0),
101. torch.nn.BatchNorm2d(64),
102. torch.nn.ReLU(inplace=True),
103. *# 第三层卷积*
104. torch.nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, stride=1, padding=0),
105. torch.nn.BatchNorm2d(128),
106. torch.nn.ReLU(inplace=True)
107. )
108. *# 输出层，将输出通道数变为分类数*
109. self.fc = torch.nn.Linear(128, num\_classes)
110. def forward(self, X):
111. out = self.conv(X)  *# 输出维度(batch\_size, C\_out, H, W)*
112. *# 平均池化*
113. out = F.avg\_pool2d(out, 26)  *# 池化后图片大小1\*1*
114. *# 将out从batch size\*128\*1\*1变为batch size\*128*
115. out = out.squeeze()
116. out = self.fc(out)
117. return out
118. net = ConvModule()
119. net.to(device)
120. *# 损失函数和优化器*
121. loss = torch.nn.CrossEntropyLoss()  *# 交叉熵损失函数*
122. optimizer = torch.optim.SGD(net.parameters(), lr=lr)
123. num\_epochs = 100  *# 训练轮次*
124. lr = 0.01
125. def train(net, data\_loader, device):*#模型训练*
126. net.train()  *# 指定为训练模式*
127. train\_batch\_num = len(data\_loader)
128. total\_loss = 0.0
129. correct = 0  *# 记录共有多少个样本被正确分类*
130. sample\_num = 0
131. *# 遍历每个batch进行训练*
132. for data, target in data\_loader:
133. *# 将图片和标签放入指定的device中*
134. data = data.to(device)
135. target = target.to(device)
136. *# 将当前梯度清零*
137. optimizer.zero\_grad()
138. *# 使用模型计算出结果*
139. y\_hat = net(data)
140. *# 计算损失*
141. loss\_ = loss(y\_hat, target)
142. *# 进行反向传播*
143. loss\_.backward()
144. optimizer.step()
145. total\_loss += loss\_.item()
146. cor = (torch.argmax(y\_hat, 1) == target).sum().item()
147. correct += cor
148. *# 累加当前的样本总数*
149. sample\_num += target.shape[0]
150. print('loss: %.4f  acc: %.4f' % (loss\_.item(), cor/target.shape[0]))
151. *# 平均loss和准确率*
152. loss\_ = total\_loss / train\_batch\_num
153. acc = correct / sample\_num
154. return loss\_, acc
155. *# 对模型进行测试*
156. def test(net, data\_loader, device):
157. net.eval()  *# 指定当前模式为测试模式（针对BN层和dropout层）*
158. test\_batch\_num = len(data\_loader)
159. total\_loss = 0
160. correct = 0
161. sample\_num = 0
162. *# 指定不进行梯度计算（没有反向传播也会计算梯度，增大GPU开销*
163. with torch.no\_grad():
164. for data, target in data\_loader:
165. data = data.to(device)
166. target = target.to(device)
167. output = net(data)
168. loss\_ = loss(output, target)
169. total\_loss += loss\_.item()
170. correct += (torch.argmax(output, 1) == target).sum().item()
171. sample\_num += target.shape[0]
172. loss\_ = total\_loss / test\_batch\_num
173. acc = correct / sample\_num
174. return loss\_, acc
175. *# 模型训练与测试*
176. train\_loss\_list = []
177. train\_acc\_list = []
178. test\_loss\_list = []
179. test\_acc\_list = []
180. time\_list = []
181. timestart = time.clock()
182. for epoch in range(num\_epochs):
183. epochstart = time.clock()
184. *# 在训练集上训练*
185. train\_loss, train\_acc = train(net, data\_loader=train\_iter, device=device)
186. *# 测试集上验证*
187. test\_loss, test\_acc = test(net, data\_loader=test\_iter, device=device)
188. elapsed = (time.clock() - epochstart)
190. train\_loss\_list.append(train\_loss)
191. train\_acc\_list.append(train\_acc)
192. test\_loss\_list.append(test\_loss)
193. test\_acc\_list.append(test\_acc)
194. time\_list.append(elapsed)
195. print('epoch %d, train loss: %.4f, train acc: %.3f' % (epoch+1, train\_loss, train\_acc))
196. print('test loss: %.4f, test acc: %.3f' % (test\_loss, test\_acc))
197. print('Time used %.4f' % (elapsed))
199. timesum1 = (time.clock() - timestart)
200. print('Time used %.4f' % (timesum1))

203. *# 绘制函数*
204. def draw\_(x, train\_Y, test\_Y, ylabel):
205. plt.plot(x, train\_Y, label='train\_' + ylabel, linewidth=1.5)
206. plt.plot(x, test\_Y, label='test\_' + ylabel, linewidth=1.5)
207. plt.xlabel('epoch')
208. plt.ylabel(ylabel)
209. plt.legend()  *# 加上图例*
210. plt.show()
211. *# 绘制loss曲线*
212. x = np.linspace(0, len(train\_loss\_list), len(train\_loss\_list))
213. draw\_(x, train\_loss\_list, test\_loss\_list, 'loss')
214. draw\_(x, train\_acc\_list, test\_acc\_list, 'accuracy')

**4.1.3** 不同超参数的对比分析（包括卷积层数、卷积核大小、batchsize、lr等）选其中至少1-2个进行分析

代码和前面部分相同，分析内容见后文第五章结果分析。

#### 4.2 空洞卷积实验

**4.2.1** 使用torch.nn实现空洞卷积，要求dilation满足HDC条件（如1,2,5）且要堆叠多层并在至少一个数据集上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验结果（最好使用图表展示）

1. import os
2. import torch.nn as nn
3. import torch.nn.functional as F
4. from PIL import Image
5. import matplotlib.pyplot as plt
6. import numpy as np
7. import torch
8. from torch.utils.data import Dataset
9. from torchvision import transforms
10. import random
11. import time
12. *#导入做实验必要的数据包等相关内容*
13. num\_classes = 3  *# 分类数量*
14. batch\_size = 128 *#读取数量大小*
15. device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')
16. file\_root = "D:\实验3数据包\数据包\车辆分类数据集"
17. classes = ['bus', 'car', 'truck']  *# 分别对应三个标签*
18. nums = [218, 779, 360]  *# 每种类别的个数*
19. def read\_data(path):
20. file\_name = os.listdir(path)  *# 打开对应的文件夹*
21. train\_data = []
22. train\_labels = []
23. test\_data = []
24. test\_labels = []
25. train\_num = [int(num \* 4 / 5) for num in nums]*#定义训练集数量*
26. test\_num = [nums[i] - train\_num[i] for i in range(len(nums))]*#定义测试集数量*
28. for idx, f\_name in enumerate(file\_name):  *# 定义数据标签等相关内容*
29. im\_dirs = path + '/' + f\_name
30. im\_path = os.listdir(im\_dirs)  *# 每个不同类别图像文件夹下所有图像的名称*
31. index = list(range(len(im\_path)))
32. random.shuffle(index)  *# 打乱顺序*
33. im\_path\_ = list(np.array(im\_path)[index])
34. test\_path = im\_path\_[:test\_num[idx]]  *# 测试数据的路径*
35. train\_path = im\_path\_[test\_num[idx]:]  *# 训练数据的路径*
37. for img\_name in train\_path:
38. if img\_name == 'desktop.ini':
39. continue
40. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
41. img = img.resize((32, 32), Image.ANTIALIAS)  *# 对图片进行变形*
42. train\_data.append(img)
43. train\_labels.append(idx)
44. for img\_name in test\_path:
45. if img\_name == 'desktop.ini':
46. continue
47. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
48. img = img.resize((32, 32), Image.ANTIALIAS)  *# 对图片进行变形*
49. test\_data.append(img)
50. test\_labels.append(idx)
51. print('训练集大小：', len(train\_data), ' 测试集大小：', len(test\_data))
52. return train\_data, train\_labels, test\_data, test\_labels
53. train\_data, train\_labels, test\_data, test\_labels = read\_data(file\_root)
54. *# 定义一个Transform操作*
55. transform = transforms.Compose(
56. [transforms.ToTensor(),  *# 变为tensor*
57. *# 对数据按通道进行标准化，即减去均值，再除以方差, [0-1]->[-1,1]*
58. transforms.Normalize(mean=[0.4686, 0.4853, 0.5193], std=[0.1720, 0.1863, 0.2175])
59. ]
60. )*#将数据类型进行转换，最终要转换成Tensor*
61. *# 自定义Dataset类实现每次取出图片，将PIL转换为Tensor*
62. class MyDataset(Dataset):
63. def \_\_init\_\_(self, data, label, trans):
64. self.len = len(data)
65. self.data = data
66. self.label = label
67. self.trans = trans
68. def \_\_getitem\_\_(self, index):  *# 根据索引返回数据和对应的标签*
69. return self.trans(self.data[index]), self.label[index]
70. def \_\_len\_\_(self):
71. return self.len
72. *# 调用自己创建的Dataset*
73. train\_dataset = MyDataset(train\_data, train\_labels, transform)
74. test\_dataset = MyDataset(test\_data, test\_labels, transform)
75. *# 生成data loader*
76. train\_iter = torch.utils.data.DataLoader(
77. dataset=train\_dataset,
78. batch\_size=batch\_size,
79. shuffle=True,
80. num\_workers=0
81. )*#得到数据训练集*
82. test\_iter = torch.utils.data.DataLoader(
83. dataset=test\_dataset,
84. batch\_size=batch\_size,
85. shuffle=False,
86. num\_workers=0
87. )*#得到数据测试集*
88. *# 使用torch.nn实现二维卷积*
89. class ConvModule(torch.nn.Module):
90. def \_\_init\_\_(self):
91. super(ConvModule, self).\_\_init\_\_()
92. *# 定义一个三层卷积（卷积层越少曲线越平滑，卷积层越多acc越高）*
93. self.conv = torch.nn.Sequential(
94. *# 第一层卷积*
95. *# stride步长,padding填充*
96. torch.nn.Conv2d(in\_channels=3, out\_channels=32, kernel\_size=3, stride=1, padding=1),
97. torch.nn.BatchNorm2d(32),
98. torch.nn.ReLU(inplace=True),
99. *# 第二层卷积*
100. torch.nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, stride=1, padding=2),
101. torch.nn.BatchNorm2d(64),
102. torch.nn.ReLU(inplace=True),
103. *# 第三层卷积*
104. torch.nn.Conv2d(in\_channels=64, out\_channels=128, kernel\_size=3, stride=1, padding=5),
105. torch.nn.BatchNorm2d(128),
106. torch.nn.ReLU(inplace=True)
107. )
108. *# 输出层，将输出通道数变为分类数*
109. self.fc = torch.nn.Linear(128, num\_classes)
110. def forward(self, X):
111. out = self.conv(X)  *# 输出维度(batch\_size, C\_out, H, W)*
112. *# 平均池化*
113. out = F.avg\_pool2d(out, 26)  *# 池化后图片大小1\*1*
114. *# 将out从batch size\*128\*1\*1变为batch size\*128*
115. out = out.squeeze()
116. out = self.fc(out)
117. return out
118. net = ConvModule()
119. net.to(device)
120. *# 损失函数和优化器*
121. loss = torch.nn.CrossEntropyLoss()  *# 交叉熵损失函数*
122. optimizer = torch.optim.SGD(net.parameters(), lr=lr)
123. num\_epochs = 100  *# 训练轮次*
124. lr = 0.01
125. def train(net, data\_loader, device):*#模型训练*
126. net.train()  *# 指定为训练模式*
127. train\_batch\_num = len(data\_loader)
128. total\_loss = 0.0
129. correct = 0  *# 记录共有多少个样本被正确分类*
130. sample\_num = 0
131. *# 遍历每个batch进行训练*
132. for data, target in data\_loader:
133. *# 将图片和标签放入指定的device中*
134. data = data.to(device)
135. target = target.to(device)
136. *# 将当前梯度清零*
137. optimizer.zero\_grad()
138. *# 使用模型计算出结果*
139. y\_hat = net(data)
140. *# 计算损失*
141. loss\_ = loss(y\_hat, target)
142. *# 进行反向传播*
143. loss\_.backward()
144. optimizer.step()
145. total\_loss += loss\_.item()
146. cor = (torch.argmax(y\_hat, 1) == target).sum().item()
147. correct += cor
148. *# 累加当前的样本总数*
149. sample\_num += target.shape[0]
150. print('loss: %.4f  acc: %.4f' % (loss\_.item(), cor/target.shape[0]))
151. *# 平均loss和准确率*
152. loss\_ = total\_loss / train\_batch\_num
153. acc = correct / sample\_num
154. return loss\_, acc
155. *# 对模型进行测试*
156. def test(net, data\_loader, device):
157. net.eval()  *# 指定当前模式为测试模式（针对BN层和dropout层）*
158. test\_batch\_num = len(data\_loader)
159. total\_loss = 0
160. correct = 0
161. sample\_num = 0
162. *# 指定不进行梯度计算（没有反向传播也会计算梯度，增大GPU开销*
163. with torch.no\_grad():
164. for data, target in data\_loader:
165. data = data.to(device)
166. target = target.to(device)
167. output = net(data)
168. loss\_ = loss(output, target)
169. total\_loss += loss\_.item()
170. correct += (torch.argmax(output, 1) == target).sum().item()
171. sample\_num += target.shape[0]
172. loss\_ = total\_loss / test\_batch\_num
173. acc = correct / sample\_num
174. return loss\_, acc
175. *# 模型训练与测试*
176. train\_loss\_list = []
177. train\_acc\_list = []
178. test\_loss\_list = []
179. test\_acc\_list = []
180. time\_list = []
181. timestart = time.clock()
182. for epoch in range(num\_epochs):
183. epochstart = time.clock()
184. *# 在训练集上训练*
185. train\_loss, train\_acc = train(net, data\_loader=train\_iter, device=device)
186. *# 测试集上验证*
187. test\_loss, test\_acc = test(net, data\_loader=test\_iter, device=device)
188. elapsed = (time.clock() - epochstart)
190. train\_loss\_list.append(train\_loss)
191. train\_acc\_list.append(train\_acc)
192. test\_loss\_list.append(test\_loss)
193. test\_acc\_list.append(test\_acc)
194. time\_list.append(elapsed)
195. print('epoch %d, train loss: %.4f, train acc: %.3f' % (epoch+1, train\_loss, train\_acc))
196. print('test loss: %.4f, test acc: %.3f' % (test\_loss, test\_acc))
197. print('Time used %.4f' % (elapsed))
199. timesum1 = (time.clock() - timestart)
200. print('Time used %.4f' % (timesum1))

203. *# 绘制函数*
204. def draw\_(x, train\_Y, test\_Y, ylabel):
205. plt.plot(x, train\_Y, label='train\_' + ylabel, linewidth=1.5)
206. plt.plot(x, test\_Y, label='test\_' + ylabel, linewidth=1.5)
207. plt.xlabel('epoch')
208. plt.ylabel(ylabel)
209. plt.legend()  *# 加上图例*
210. plt.show()
211. *# 绘制loss曲线*
212. x = np.linspace(0, len(train\_loss\_list), len(train\_loss\_list))
213. draw\_(x, train\_loss\_list, test\_loss\_list, 'loss')
214. draw\_(x, train\_acc\_list, test\_acc\_list, 'accuracy')

**4.2.2**将空洞卷积模型的实验结果与卷积模型的结果进行分析比对，训练时间、 预测精度、Loss变化等角度分析

代码和前面部分相同，分析内容见后文第五章结果分析。

**4.2.3**不同超参数的对比分析（包括卷积层数、卷积核大小、不同dilation的选择，batchsize、lr等）选其中至少1-2个进行分析（选做）

代码和前面部分相同，分析内容见后文第五章结果分析。

#### 4.3 残差网络实验

**4.3.1** 实现给定结构的残差网络，在 至少一个数据集上进行实验， 从训练时间、预测精度、Loss 变化等角度分析实验结果（最好使用图表展示）

1. import os
2. import torch.nn as nn
3. import torch.nn.functional as F
4. from PIL import Image
5. import matplotlib.pyplot as plt
6. import numpy as np
7. import torch
8. from torch.utils.data import Dataset
9. from torchvision import transforms
10. import random
11. import time
12. *#导入做实验必要的数据包等相关内容*
13. batch\_size = 128
14. num\_classes = 3
15. lr = 0.001
16. epochs = 10
17. device = torch.device("cpu")
18. file\_root = "D:\实验3数据包\数据包\车辆分类数据集"
19. classes = ['bus', 'car', 'truck']  *# 分别对应三个标签*
20. nums = [218, 779, 360]  *# 每种类别的个数*
21. def read\_data(path):
22. file\_name = os.listdir(path)  *# 打开对应的文件夹*
23. train\_data = []
24. train\_labels = []
25. test\_data = []
26. test\_labels = []
27. train\_num = [int(num \* 4 / 5) for num in nums]*#定义训练集数量*
28. test\_num = [nums[i] - train\_num[i] for i in range(len(nums))]*#定义测试集数量*
30. for idx, f\_name in enumerate(file\_name):  *# 定义数据标签等相关内容*
31. im\_dirs = path + '/' + f\_name
32. im\_path = os.listdir(im\_dirs)  *# 每个不同类别图像文件夹下所有图像的名称*
33. index = list(range(len(im\_path)))
34. random.shuffle(index)  *# 打乱顺序*
35. im\_path\_ = list(np.array(im\_path)[index])
36. test\_path = im\_path\_[:test\_num[idx]]  *# 测试数据的路径*
37. train\_path = im\_path\_[test\_num[idx]:]  *# 训练数据的路径*
39. for img\_name in train\_path:
40. if img\_name == 'desktop.ini':
41. continue
42. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
43. img = img.resize((64, 64), Image.ANTIALIAS)  *# 对图片进行变形*
44. train\_data.append(img)
45. train\_labels.append(idx)
46. for img\_name in test\_path:
47. if img\_name == 'desktop.ini':
48. continue
49. img = Image.open(im\_dirs + '/' + img\_name)  *# 打开图片*
50. img = img.resize((64, 64), Image.ANTIALIAS)  *# 对图片进行变形*
51. test\_data.append(img)
52. test\_labels.append(idx)
53. print('训练集大小：', len(train\_data), ' 测试集大小：', len(test\_data))
54. return train\_data, train\_labels, test\_data, test\_labels
55. train\_data, train\_labels, test\_data, test\_labels = read\_data(file\_root)
56. *# 定义一个Transform操作*
57. transform = transforms.Compose(
58. [transforms.ToTensor(),  *# 变为tensor*
59. *# 对数据按通道进行标准化，即减去均值，再除以方差, [0-1]->[-1,1]*
60. transforms.Normalize(mean=[0.4686, 0.4853, 0.5193], std=[0.1720, 0.1863, 0.2175])
61. ]
62. )*#将数据类型进行转换，最终要转换成Tensor*
63. *# 自定义Dataset类实现每次取出图片，将PIL转换为Tensor*
64. class MyDataset(Dataset):
65. def \_\_init\_\_(self, data, label, trans):
66. self.len = len(data)
67. self.data = data
68. self.label = label
69. self.trans = trans
70. def \_\_getitem\_\_(self, index):  *# 根据索引返回数据和对应的标签*
71. return self.trans(self.data[index]), self.label[index]
72. def \_\_len\_\_(self):
73. return self.len
74. *# 调用自己创建的Dataset*
75. train\_dataset = MyDataset(train\_data, train\_labels, transform)
76. test\_dataset = MyDataset(test\_data, test\_labels, transform)
77. *# 生成data loader*
78. train\_iter = torch.utils.data.DataLoader(
79. dataset=train\_dataset,
80. batch\_size=batch\_size,
81. shuffle=True,
82. num\_workers=0
83. )*#得到数据训练集*
84. test\_iter = torch.utils.data.DataLoader(
85. dataset=test\_dataset,
86. batch\_size=batch\_size,
87. shuffle=False,
88. num\_workers=0
89. )*#得到数据测试集*
90. *# 使用torch.nn实现二维卷积*
91. *#残差网络块*
92. *#每个残差块都是两层*
93. *#默认3\*3卷积下padding为1，则大小不会变化，如变化则是步长引起的。*
94. class ResidualBlock(nn.Module):
95. def \_\_init\_\_(self, nin, nout, size, stride=1, shortcut=True):
96. super(ResidualBlock, self).\_\_init\_\_()
97. *#两层卷积层*
98. *#不同步长只有第一层卷积层不同*
99. self.block1 = nn.Sequential(nn.Conv2d(nin, nout, size, stride, padding=1),
100. nn.BatchNorm2d(nout),
101. nn.ReLU(inplace=True),
102. nn.Conv2d(nout, nout, size, 1, padding=1),
103. nn.BatchNorm2d(nout))
104. self.shortcut = shortcut
105. *#解决通道数变化以及步长不为1引起的图片大小的变化*
106. self.block2 = nn.Sequential(nn.Conv2d(nin, nout, size, stride, 1),
107. nn.BatchNorm2d(nout))
108. self.relu = nn.ReLU(inplace=True)
109. def forward(self, input):
110. x = input
111. out = self.block1(x)
112. '''若输入输出维度相等直接相加，不相等改变输入的维度--包括大小和通道'''
113. if self.shortcut:
114. out = x + out
115. else:
116. out = out + self.block2(x)
117. out = self.relu(out)
118. return out
119. *# In[ ]:*
120. *#定义给定的残差结构*
121. class resnet(nn.Module):
122. def \_\_init\_\_(self):
123. super(resnet, self).\_\_init\_\_()
124. self.block = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding=1),
125. nn.BatchNorm2d(64),
126. nn.ReLU())
127. *#t表示2个相同的残差块,每个残差块两个卷积*
128. self.d1 = self.make\_layer(64, 64, 3, stride=1, t=2)
129. self.d2 = self.make\_layer(64, 128, 3, stride=2, t=2)
130. self.d3 = self.make\_layer(128, 256, 3, stride=2, t=2)
131. self.d4 = self.make\_layer(256, 512, 3, stride=2, t=2)
132. self.avgp = nn.AvgPool2d(8)
133. self.exit = nn.Linear(512, 3)
134. def make\_layer(self, in1, out1, ksize, stride, t):
135. layers = []
136. for i in range(0, t):
137. if i == 0 and in1 != out1:
138. layers.append(ResidualBlock(in1, out1, ksize, stride, None))
139. else:
140. layers.append(ResidualBlock(out1, out1, ksize, 1, True))
141. return nn.Sequential(\*layers)
142. def forward(self, input):
143. x = self.block(input)  *# 输出维度 64 \* 64 \* 64    C \* H \* W*
144. x = self.d1(x)  *# 输出维度 64 \* 54 \* 54*
145. x = self.d2(x)  *# i=0 步长为2，输出维度128 \* 32 \* 32*
146. x = self.d3(x)  *# i=0 步长为2，输出维度256 \* 16 \* 16*
147. x = self.d4(x)  *# i=0 步长为2，输出维度512 \* 8 \* 8*
148. x = self.avgp(x)  *# 512 \* 1 \* 1*
149. *#将张量out从shape batchx512x1x1 变为 batch x512*
150. x = x.squeeze()
151. output = self.exit(x)
152. return output
153. *#初始化模型*
154. net = resnet().to(device)
155. *#使用多元交叉熵损失函数*
156. criterion = nn.CrossEntropyLoss()
157. loss = torch.nn.CrossEntropyLoss()  *# 交叉熵损失函数*
158. *#使用Adam优化器*
159. optimizer = torch.optim.SGD(net.parameters(), lr=lr)
160. def train(net, data\_loader, device):*#模型训练*
161. net.train()  *# 指定为训练模式*
162. train\_batch\_num = len(data\_loader)
163. total\_loss = 0.0
164. correct = 0  *# 记录共有多少个样本被正确分类*
165. sample\_num = 0
166. *# 遍历每个batch进行训练*
167. for data, target in data\_loader:
168. *# 将图片和标签放入指定的device中*
169. data = data.to(device)
170. target = target.to(device)
171. *# 将当前梯度清零*
172. optimizer.zero\_grad()
173. *# 使用模型计算出结果*
174. y\_hat = net(data)
175. *# 计算损失*
176. loss\_ = loss(y\_hat, target)
177. *# 进行反向传播*
178. loss\_.backward()
179. optimizer.step()
180. total\_loss += loss\_.item()
181. cor = (torch.argmax(y\_hat, 1) == target).sum().item()
182. correct += cor
183. *# 累加当前的样本总数*
184. sample\_num += target.shape[0]
185. print('loss: %.4f  acc: %.4f' % (loss\_.item(), cor/target.shape[0]))
186. *# 平均loss和准确率*
187. loss\_ = total\_loss / train\_batch\_num
188. acc = correct / sample\_num
189. return loss\_, acc
190. *# 对模型进行测试*
191. def test(net, data\_loader, device):
192. net.eval()  *# 指定当前模式为测试模式（针对BN层和dropout层）*
193. test\_batch\_num = len(data\_loader)
194. total\_loss = 0
195. correct = 0
196. sample\_num = 0
197. *# 指定不进行梯度计算（没有反向传播也会计算梯度，增大GPU开销*
198. with torch.no\_grad():
199. for data, target in data\_loader:
200. data = data.to(device)
201. target = target.to(device)
202. output = net(data)
203. loss\_ = loss(output, target)
204. total\_loss += loss\_.item()
205. correct += (torch.argmax(output, 1) == target).sum().item()
206. sample\_num += target.shape[0]
207. loss\_ = total\_loss / test\_batch\_num
208. acc = correct / sample\_num
209. return loss\_, acc
210. *# 模型训练与测试*
211. train\_loss\_list = []
212. train\_acc\_list = []
213. test\_loss\_list = []
214. test\_acc\_list = []
215. time\_list = []
216. timestart = time.clock()
217. for epoch in range(num\_epochs):
218. epochstart = time.clock()
219. *# 在训练集上训练*
220. train\_loss, train\_acc = train(net, data\_loader=train\_iter, device=device)
221. *# 测试集上验证*
222. test\_loss, test\_acc = test(net, data\_loader=test\_iter, device=device)
223. elapsed = (time.clock() - epochstart)
225. train\_loss\_list.append(train\_loss)
226. train\_acc\_list.append(train\_acc)
227. test\_loss\_list.append(test\_loss)
228. test\_acc\_list.append(test\_acc)
229. time\_list.append(elapsed)
230. print('epoch %d, train loss: %.4f, train acc: %.3f' % (epoch+1, train\_loss, train\_acc))
231. print('test loss: %.4f, test acc: %.3f' % (test\_loss, test\_acc))
232. print('Time used %.4f' % (elapsed))
234. timesum1 = (time.clock() - timestart)
235. print('Time used %.4f' % (timesum1))

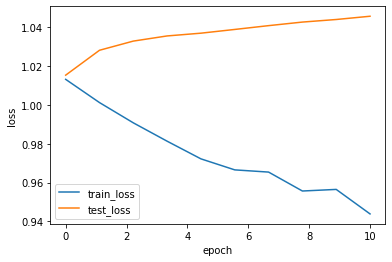
238. *# 绘制函数*
239. def draw\_(x, train\_Y, test\_Y, ylabel):
240. plt.plot(x, train\_Y, label='train\_' + ylabel, linewidth=1.5)
241. plt.plot(x, test\_Y, label='test\_' + ylabel, linewidth=1.5)
242. plt.xlabel('epoch')
243. plt.ylabel(ylabel)
244. plt.legend()  *# 加上图例*
245. plt.show()
246. *# 绘制loss曲线*
247. x = np.linspace(0, len(train\_loss\_list), len(train\_loss\_list))
248. draw\_(x, train\_loss\_list, test\_loss\_list, 'loss')
249. draw\_(x, train\_acc\_list, test\_acc\_list, 'accuracy')

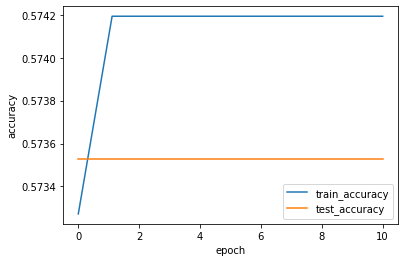
## 五、实验结果

实验结果包括程序运行结果以及对结果的分析，尽量用图表展示实验结果，并且通过结果进行相关的分析。

#### 5.1 二维卷积实验

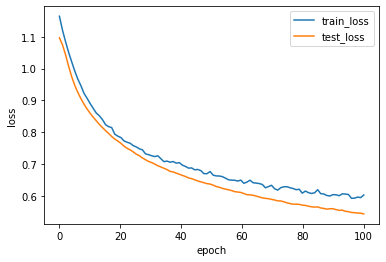
**5.1.1 实验4.1.1运行结果**

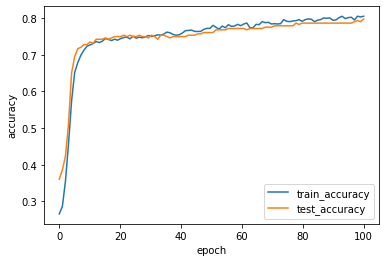




根据实验结果可以得知，如果手动实现二维卷积网络，在同样的测试条件下，首先该实验的准确度和使用torch.nn相比精度远远不够，最多也就是60%左右，而且读取的batch\_size数量也是非常小的，但是由于是手动构造模型的，训练出来的精度也在可接受范围内，因此实验结果仅供参考，不具备太多实际意义。

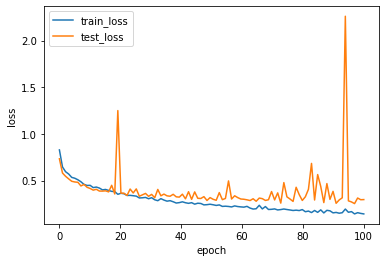
**5.1.2 实验4.1.2运行结果**

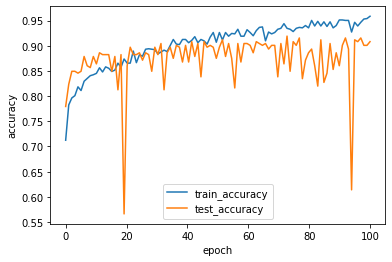




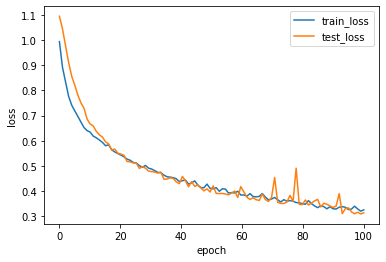
**分析：**由于这一部分使用了torch.nn，首先相比于手动实现在训练进度上有着较大的提高，经过为数不多的训练时间后就可以达到较高的训练精度，由于batch\_size较大，因此这里的训练精度最终结果超过80%，使用torch.nn得到的训练时间也有极大幅度的降低，且训练损失也维持在一个较为理想的水平。训练总共花费时长1400s左右，而且是CPU完全占用并用于完成其他测试的前提条件下，因此具备一定的参考价值，后续还会有对比实验**。**

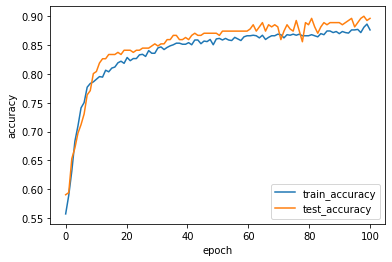
**5.1.3 实验4.1.3运行结果**





（对比实验1 batch\_size=32）



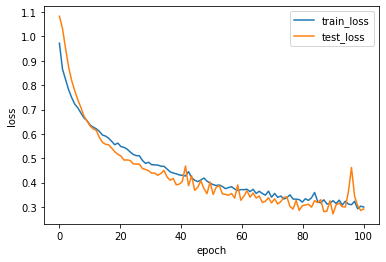


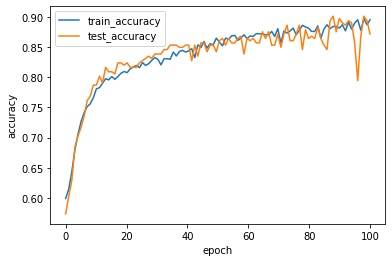
（对比实验1 Ir=0.008）

**分析：**这是一组对比实验，设置不同的读取量以及不同的学习率从而比较不同模型的好坏。根据数据分析可知，当batch\_size设置成一个较大的值时，虽然训练时间短，相对而言模型的训练精度就没有那么理想。在学习率设置更小的情况下，模型训练效果更好，但是也会出现过拟合现象。综合上述分析，是模型训练达到一个最佳的效果，必须要同时设置合理的学习率以及样本读取速率。

#### 5.2 空洞卷积实验

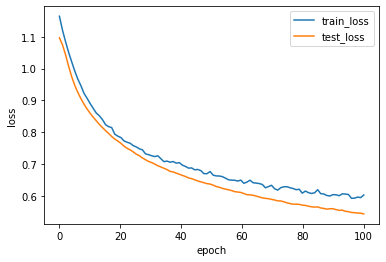
**5.2.1 实验4.2.1运行结果**

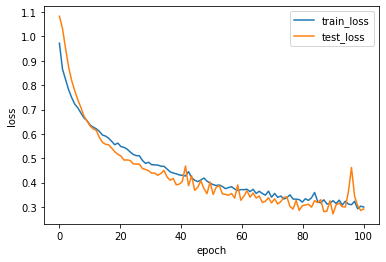




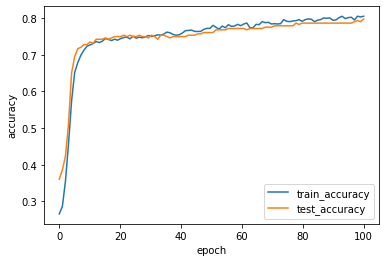
**分析：**这个实验是空洞卷积实验，其目的是设置一定的空洞数值，从而优化训练模型，达到一个更好的学习效果。根据实验要求，我们从训练时间、预测精度、Loss 变化等角度分析实验结果，训练时间为：7218 ，预测精度为： 0.87-0.9，Loss变化为：0.3左右 。

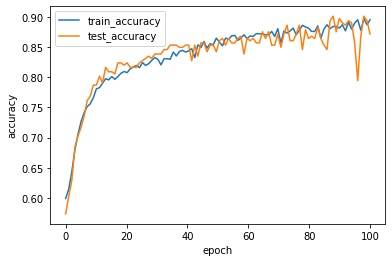
**5.2.2实验4.2.2运行结果**





（对比实验2-1 所以参数均为默认值）

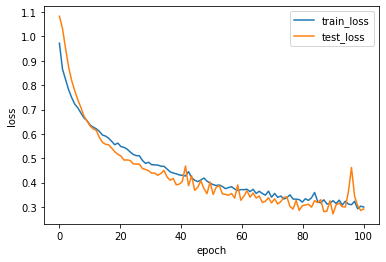


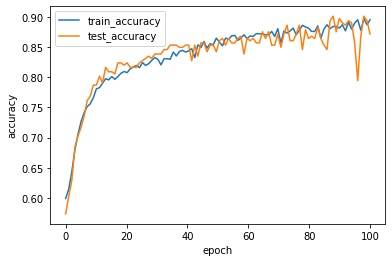


（对比实验2 lr=0.008）

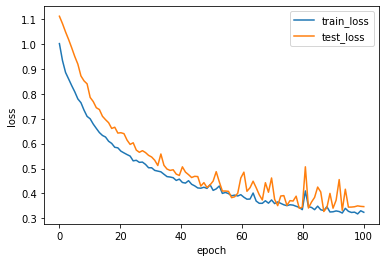
**分析：**这个实验是空洞卷积对比实验，其目的是设置一定的空洞数值，从而优化训练模型，达到一个更好的学习效果。在和实验1-2同样的参数条件下，设置空洞率的不同，结果如图所示。和预测分析的一样，新训练模型达到了一个较低的损失和更高的准确率，平均准确率更高，分类效果相对良好。但是也出现了过拟合，说明了后续参数还要进一步调整。其中二维卷积训练精度为0.8左右，空洞卷积为0.9，二维卷积训练损失为0.6-0.7左右，空洞卷积为0.3-0.4，二维卷积训练时间为1399秒左右，空洞卷积为7218秒左右。

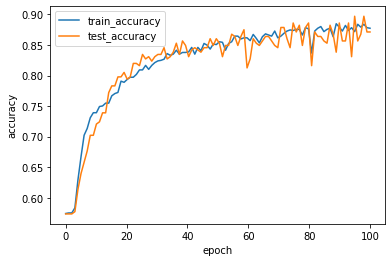
**5.2.3 实验4.2.3运行结果**



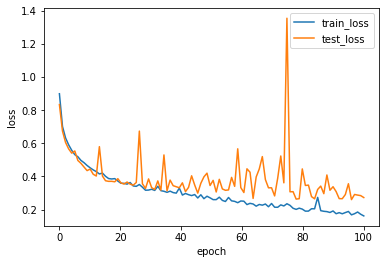


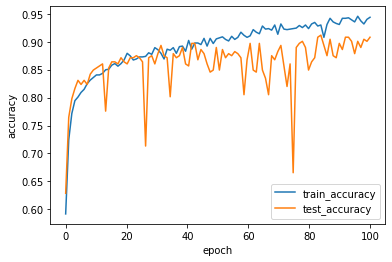
（正常实验 所有参数据选用默认值）



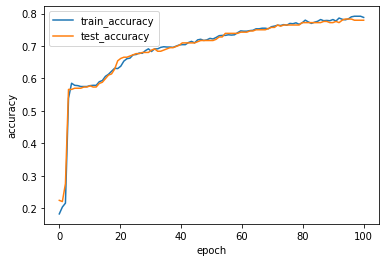
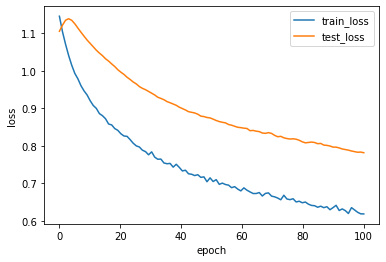


（对比实验1 改变空洞率）





（对比实验2 batch\_size=32）



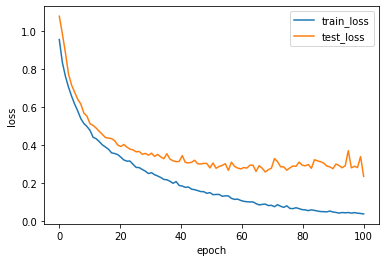
（对比实验3 lr=0.001）

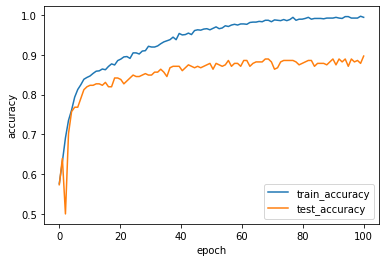
**分析：**这是一组对比实验，设置不同的读取量以及不同的学习率从而比较不同模型的好坏。根据数据分析可知，当batch\_size设置成一个较小的值时，虽然训练时间长，但是可以在较短的时间内完成较高精度的模型训练。在学习率设置更小的情况下，模型训练效果更好，但是也会出现过拟合现象。空洞率的不同对结果也会产生不一样的影响，结果如图所示，这也说明了模型的优化还需要进一步调整参数。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 参数 | 正常 | 改变空洞率 | 改变学习率 | 改变样本量 |
| Time | 7218 | 9158 | 7244 | 8417 |
| accuracy | 0.85 | 0.85+ | 0.8 | 0.9 |
| loss | 0.3-0.4 | 0.3-0.4 | 0.6-0.9 | 0.2-0.4 |

#### 5.3 残差网络实验

**5.3.1 实验4.3.1运行结果**





**分析：**根据结果分析可得，随着迭代次数的增加，函数的损失值不断减少，和理论上一样，说明了深度残差网络具有一定的学习能力。由于参数设置问题，这导致了深度残差网络的情况并不是特别理想的，还需要进一步优化。

## 六、实验心得体会

这次实验让我进一步步了解了PyTorch的基本原理和操作，学会了利用相关知识去解决实际问题，比如说汽车分类问题，虽然这个问题看似简单，却包含了许多深度学习的原理和知识。我学会了利用不同的方法去解决实际问题，在实际操作中提高了动手能力。此外我还通过设置不同的参数大小来改变模型的效果，这进一步巩固了我的知识，也提高了我的编程能力。然而，我也知道我的模型还有许多地方需要进一步优化，毕竟这次的实验还是有许多地方参考了CSDN上的东西，但是我接下来还会继续努力，争取做得更好。

## 七、参考文献

[1] https://blog.csdn.net/qq\_37534947/article/details/108179408

[2] 动⼿学深度学习-PyTorch

## 八、附录

需要补充说明的内容，如无可略。

**实验报告填写要求**

1. 正文要求小四号宋体，行间距1.5倍；

2. 英文要求小四号Times New Roman；

3. 在实验内容、实验过程、实验结果三部分需要针对当次实验不同的实验内容分别填写（模版以实验一为例），实验设计中如有必要也可以分开填写；

4. 实验报告配图的每幅图应有编号和标题，编号和标题应位于图下方处，居中，中文用五号宋体；

5. 表格应为三线表，每个表格应有编号和标题，编号和标题应写在表格上方正中，距正文段前0.5倍行距。表格中量与单位之间用“/”分隔，编号与标题中的中文用五号宋体；

6. 图、表、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。其标注形式应便于互相区别，可分别为：图1、表2、公式(5)等。