

计算机视觉

课程实验报告

|  |  |
| --- | --- |
| 院系 | 计算机科学与技术 |
| 专业班级 | 本硕博2101 |
| 姓名 | 姚晨炫 |
| 学号 | U202115674 |
| 指导老师 | 杨卫 |

目录

[1 lab3 3](#_Toc152929906)

[2 网络架构 4](#_Toc152929907)

[3 数据集 6](#_Toc152929908)

[4 训练 8](#_Toc152929909)

[5 实验 9](#_Toc152929910)

# 网络架构

实验3的网络架构在卷积部分与lab2中的网络时相似的，目的是特征抽取，但是在MLP的部分，我将两张图片卷积后的结果展开，利用torch.cat 将他们拼接在一起，然后送入全连接层进行判断。网络结构如下。

# 定义卷积神经网络  
class CNN(nn.Module):  
 def \_\_init\_\_(self):  
 super(CNN, self).\_\_init\_\_()  
 self.conv1 = nn.Conv2d(1, 32, kernel\_size=5)  
 self.conv2 = nn.Conv2d(32, 64, kernel\_size=5)  
 self.fc1 = nn.Linear(2048, 512)  
 self.fc2 = nn.Linear(512, 128)  
 self.fc3 = nn.Linear(128, 1)  
 self.sigmoid = nn.Sigmoid()  
  
 def forward(self, x1, x2):  
 x1 = self.conv1(x1)  
 x1 = nn.functional.relu(x1)  
 x1 = nn.functional.max\_pool2d(x1, 2)  
 x1 = self.conv2(x1)  
 x1 = nn.functional.relu(x1)  
 x1 = nn.functional.max\_pool2d(x1, 2)  
 x1 = x1.view(-1, 1024)  
  
 x2 = self.conv1(x2)  
 x2 = nn.functional.relu(x2)  
 x2 = nn.functional.max\_pool2d(x2, 2)  
 x2 = self.conv2(x2)  
 x2 = nn.functional.relu(x2)  
 x2 = nn.functional.max\_pool2d(x2, 2)  
 x2 = x2.view(-1, 1024)  
  
 x = torch.cat((x1, x2), dim=1)  
 x = self.fc1(x)  
 x = nn.functional.relu(x)  
 x = self.fc2(x)  
 x = nn.functional.relu(x)  
 x = self.fc3(x)  
 x = self.sigmoid(x)  
 return x  
  
model = CNN()

# 数据集

这次的数据集有所不同，因为一次要抽取两张图片，所以我们使用from torch.utils.data import Dataset, DataLoader 来自定义我们的数据集。

# 定义用于加载两张图片的自定义数据集类  
class TwoMNISTDataset(Dataset):  
 def \_\_init\_\_(self, mnist\_dataset):  
 self.mnist\_dataset = mnist\_dataset  
  
 def \_\_len\_\_(self):  
 return len(self.mnist\_dataset)  
  
 def \_\_getitem\_\_(self, idx):  
 if(idx%2):  
 img1, label1 = self.mnist\_dataset[idx]  
 img2, label2 = self.mnist\_dataset[idx]  
 else:  
 img1, label1 = self.mnist\_dataset[abs(len(self.mnist\_dataset)-idx-1)]  
 img2, label2 = self.mnist\_dataset[idx]  
 return img1, label1, img2, label2, torch.tensor((label1==label2), dtype=torch.float32)  
  
mnist\_train = MNIST('./data', train=True, download=True, transform=torchvision.transforms.ToTensor())  
mnist\_test = MNIST('./data', train=False, download=True, transform=torchvision.transforms.ToTensor())  
  
train\_dataset = TwoMNISTDataset(mnist\_train)  
test\_dataset = TwoMNISTDataset(mnist\_test)

这里有一个很有意思的问题，就是假如我每一次都选取数据集中相邻的两张图片，然后将他们送入神经网络，这样的话，按照概率学上，label 为1的概率为10%，为0的概率为90%，那么对于神经网络来说，他只要重复的输出0，无论他的输入是什么，就能获得百分之90的准确率，这样会字节导致我们的训练出现大问题，所以我人为的改变了抽取样本的方式，在数据集中混入了百分之五十的两张相同的图片，也就是混入了百分之50的label为1的数据，这使得我们的训练得以成功的进行下去。

而且为什么使用两张相同的图片呢？我认为这个是有一定好处的，因为这里理论上根据我们网络的架构来说，根据对称性，两张图片经过的卷积层应该是相同的，如果我们在数据集中混入了两张相同的图片，会有利于网络的对称性。（只是臆测）

# 训练

采用标准的训练方式，代码如下，测试代码与训练相似

# 定义训练和测试函数  
def train(model, criterion, optimizer, train\_loader):  
 model.train()  
 train\_loss = 0.0  
 for data in train\_loader:  
 img1, \_, img2, \_, label = data  
 optimizer.zero\_grad()  
 output = model(img1, img2)  
 loss = criterion(output,label.unsqueeze(1))  
 loss.backward()  
 optimizer.step()  
 train\_loss += loss.item()  
 return train\_loss / len(train\_loader)  
  
def test(model, criterion, test\_loader):  
 model.eval()  
 correct = 0  
 total = 0  
 test\_loss = 0  
 with torch.no\_grad():  
 for data in test\_loader:  
 img1, \_, img2, \_, label = data  
 outputs = model(img1, img2)  
 loss = criterion(outputs,label.unsqueeze(1))  
 predicted = torch.round(outputs)  
 total += label.size(0)  
 correct += (predicted == label.unsqueeze(1)).sum().item()  
 test\_loss += loss.item()  
 return test\_loss/len(test\_loader), correct / total

# 实验

实验环境为MacOS Apple Silicon M1 Pro，Python版本3.9，Pytorch版本2.0.0(MacOS版)

训练了10个Epoch实验结果如下

