《计算机视觉》实验报告

姓名：王子棠 学号：21122897

实验 10

一. 任务 1

a) 核心代码：

```
# 超参数设置
batch_size = 64
learning_rate = 0.01
momentum = 0.5
EPOCH = 10
# 数据集为 MNIST，从 torchvision.datasets 中下载获取
train_dataset = datasets.MNIST(root='./data/mnist', train=True, download=False,
transform=transform)
test_dataset = datasets.MNIST(root='./data/mnist', train=False, download=False,
transform=transform)  # train=True 训练集，=False 测试集
# CNN 搭建
class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 10, kernel_size=5), # 卷积
            torch.nn.ReLU(), # 激活
            torch.nn.MaxPool2d(kernel_size=2), # 池化
        )
        self.conv2 = torch.nn.Sequential(
```

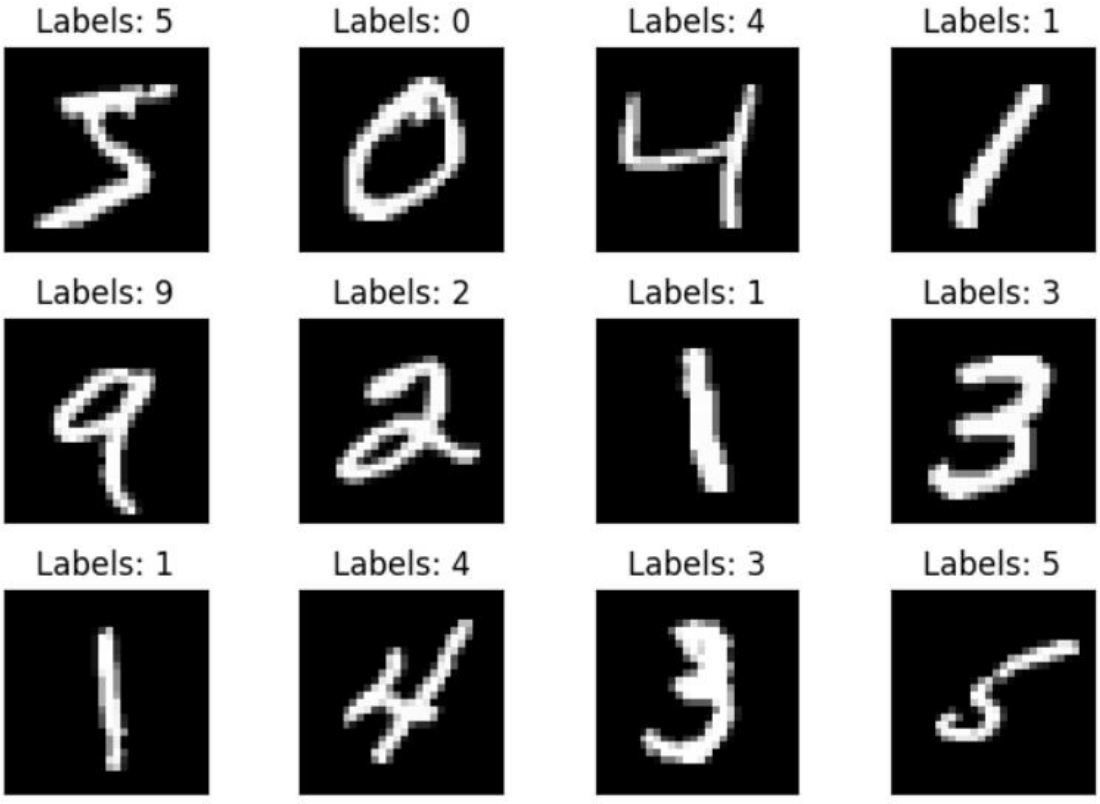
```
            torch.nn.Conv2d(10, 20, kernel_size=5),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2),
        )
        self.fc = torch.nn.Sequential(
            torch.nn.Linear(320, 50),
            torch.nn.Linear(50, 10),
        )
    def forward(self, x):
        batch_size = x.size(0)
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(batch_size, -1)
        x = self.fc(x)
        return x
# 使用交叉熵损失函数
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate,
momentum=momentum)
# 训练
def train(epoch):
    running_loss = 0.0
    running_total = 0
    running_correct = 0
    for batch_idx, data in enumerate(train_loader, 0):
        inputs, target = data
        optimizer.zero_grad()

        # forward + backward + update
        outputs = model(inputs)
```

```
            loss = criterion(outputs, target)


            loss.backward()
            optimizer.step()


            # 把 loss 累加起来
            running_loss += loss.item()
            # 计算运行中的 acc
            _, predicted = torch.max(outputs.data, dim=1)
            running_total += inputs.shape[0]
            running_correct += (predicted == target).sum().item()
            # 每 300 次计算一个平均损失和准确率
            if batch_idx % 300 == 299:
                print('[%d, %5d]: loss: %.3f , acc: %.2f %%'
                              % (epoch + 1, batch_idx + 1, running_loss / 300, 100 *
running_correct / running_total))
                running_loss = 0.0
                running_total = 0
                running_correct = 0
# 测试
def test():
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs.data, dim=1)
            total += labels.size(0)  # 张量之间的比较运算
            correct += (predicted == labels).sum().item()
```

```
        acc = correct / total
        print('[%d / %d]: Accuracy on test set: %.1f %% ' % (epoch+1, EPOCH,
100 * acc))  # 计算测试的准确率
        return acc
# 训练 10 次，每训练一轮就进行测试
acc_list_test = []
for epoch in range(EPOCH):
    train(epoch)
    acc_test = test()
    acc_list_test.append(acc_test)
plt.plot(acc_list_test)
plt.xlabel('Epoch')
plt.ylabel('Accuracy On TestSet')
plt.show()
```

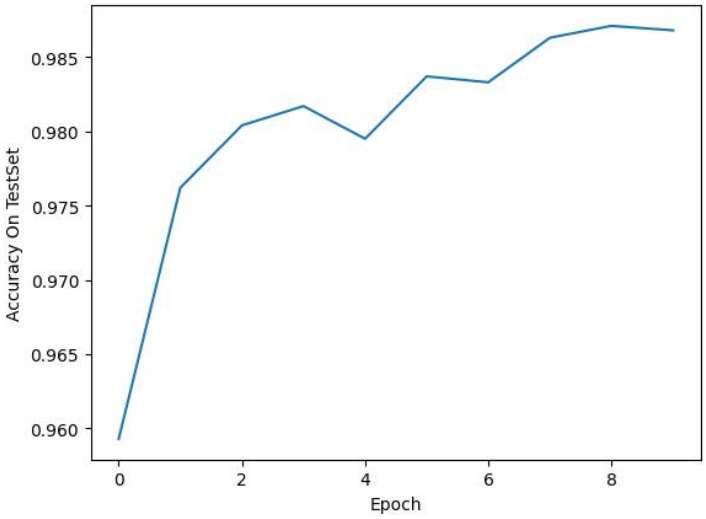
b) 实验结果截图

• 数据集结构图

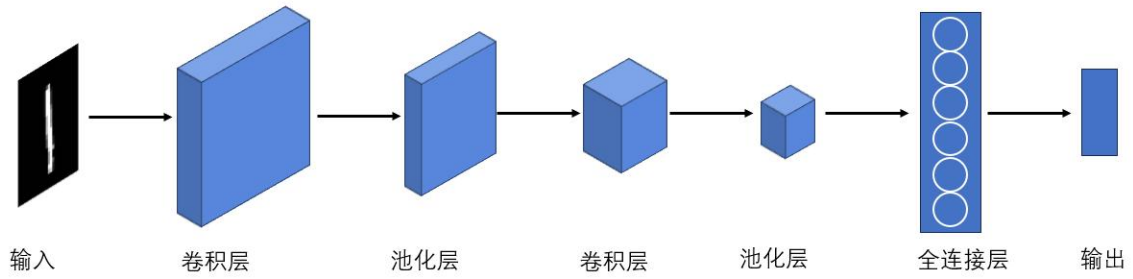


• **训练和测试结果，可以看到每迭代 300 次就计算一次损失和准确率，随着训练轮数增加，测试集上的准确率也随之增加，训练集上的 loss 随之降低**

```
[1,   300]: loss: 0.722 , acc: 79.67 %
[1,   600]: loss: 0.207 , acc: 93.69 %
[1,   900]: loss: 0.157 , acc: 95.36 %
[1 / 10]: Accuracy on test set: 95.9 %
[2,   300]: loss: 0.125 , acc: 96.34 %
[2,   600]: loss: 0.100 , acc: 96.93 %
[2,   900]: loss: 0.099 , acc: 96.99 %
[2 / 10]: Accuracy on test set: 97.6 %
[3,   300]: loss: 0.084 , acc: 97.39 %
[3,   600]: loss: 0.082 , acc: 97.53 %
[3,   900]: loss: 0.074 , acc: 97.76 %
[3 / 10]: Accuracy on test set: 98.0 %
[4,   300]: loss: 0.068 , acc: 97.93 %
[4,   600]: loss: 0.067 , acc: 97.90 %
[4,   900]: loss: 0.065 , acc: 98.03 %
[4 / 10]: Accuracy on test set: 98.2 %
[5,   300]: loss: 0.059 , acc: 98.12 %
[5,   600]: loss: 0.060 , acc: 98.20 %
[5,   900]: loss: 0.054 , acc: 98.31 %
[5 / 10]: Accuracy on test set: 98.0 %
[6,   300]: loss: 0.049 , acc: 98.56 %
[6,   600]: loss: 0.056 , acc: 98.27 %
[6,   900]: loss: 0.052 , acc: 98.35 %
[6 / 10]: Accuracy on test set: 98.4 %
[7,   300]: loss: 0.047 , acc: 98.43 %
...
[10,   300]: loss: 0.036 , acc: 98.88 %
[10,   600]: loss: 0.039 , acc: 98.81 %
[10,   900]: loss: 0.038 , acc: 98.79 %
[10 / 10]: Accuracy on test set: 98.7 %
```



• **CNN 网络结构示意图**



c) 实验小结

本次实验我学会自己搭建 pytorch 环境，搭建 CNN 卷积神经网络，并进行手写数字识别。卷积神经网络的基本结构是卷积层、激活函数、池化层、全连接层、输出层等，通过自己搭建 CNN，我了解了这些组成部分的作用和工作原理，对于后续深度学习相关内容的学习有很大帮助。