1 任务要求

任务要求：设计一个卷积神经网络，输入为两张 MNIST 手写体数字图片，如果两张图片为同一个数字（注意，非同一张图片），输出为 1，否则为 0。

从 MNIST 数据集的训练集中选取 10%作为本实验的训练图片，从 MNIST 数据集的测试集中选取 10%作为本实验的测试图片。请将该部分图片经过适当处理形成一定数量的用于本次实验的训练集和测试集。

2 任务设计

2.1 数据集准备

使用随机下标+subSet 函数获取 10%训练样本和 10%测试样本。
首先获取全部样本，然后生成随机 10%大小的下标，利用下标取出数据。然后对于 10%训练数据和 10%测试数据，再各自分成一半，两两图片构成一组输入，用于训练或者测试。

```
# 加载完整的MNIST 数据集
train_dataset = datasets.MNIST('data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST('data', train=False, download=True, transform=transform)

# 计算10%的样本数量
train_subset_size = int(0.1 * len(train_dataset))
test_subset_size = int(0.1 * len(test_dataset))

# 从训练集中随机选择10%的样本
train_subset = Subset(train_dataset,
                      # torch.randperm 创建随机序列索引,然后取前trian_subset_size 个,
                      # 最后利用subset 函数取处随机的10%样本数
                      torch.randperm(len(train_dataset))[:train_subset_size]
                      )
# 从测试集中随机选择10%的样本
test_subset = Subset(test_dataset,
                     torch.randperm(len(test_dataset))[:test_subset_size]
                     )
# 将随机选取的10%数据分成两份,用于训练
train_subset_index = torch.randperm(len(train_subset))
# print(train_subset_index)
```

```
train_subset1 = Subset(train_subset,train_subset_index[:int(len(train_subset)/2)])
train_subset2 = Subset(train_subset,train_subset_index[int(len(train_subset)/2):])
# 将随机选取的10%数据分成两份，用于测试
test_subset_index = torch.randperm(len(test_subset))
test_subset1 = Subset(test_subset,test_subset_index[:int(len(test_subset)/2)])
test_subset2 = Subset(test_subset,test_subset_index[int(len(test_subset)/2):])
# 创建训练数据加载器，分别表示两张图片
train_loader1 = DataLoader(train_subset1, batch_size=train_batch_size, shuffle=True, drop_last=True)
train_loader2 = DataLoader(train_subset2, batch_size=train_batch_size, shuffle=True, drop_last=True)
# 创建测试数据加载器，分别表示两张图片
test_loader1 = DataLoader(test_subset1, batch_size=test_batch_size, shuffle=False, drop_last=True)
test_loader2 = DataLoader(test_subset2, batch_size=test_batch_size, shuffle=False, drop_last=True)
```

训练或者测试时，将两个数据加载器合并起来一起遍历。遍历运行时，先将两张图在通道维度上叠加起来，然后在输入模型。比如：

```
    for (data1, target1), (data2, target2) in zip(train_loader1, train_loader2):
        optimizer.zero_grad()
        # 将两张图叠在一起,形成输入数据
        data = torch.cat((data1,data2), dim=1)
        # print(data.shape)
        output = model(data)
        #  ....
```

2.2 网络结构

网络接收双通道的图数据，一个通道上放一张图。输出一个二维向量，通过 softmax 转化为对预测是否为同一张图片的概率。

本次实验采用卷积神经网络架构，将尝试 resnet 模型和简单的卷积神经神经网络模型。

resnet 的一个模块如下代码所示：

```
    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out += self.shortcut(residual)
        out = self.relu(out)
        return out
```

实验中将叠加三个模块构成 Resnet 结构。

简单的卷积神经网络如下代码所示：

```
    def forward(self, x1):
        x1 = self.relu(self.conv1(x1))
        x1 = self.maxpool(x1)
        x1 = self.relu(self.conv2(x1))
        x1 = self.maxpool(x1)
        x1 = x1.view(x1.size(0), -1)
        x1 = self.relu(self.fc1(x1))
        x1 = self.fc2(x1)
        return x1
```

3 实验

本实验尝试不同的卷积神经网络模型。通过调整各种参数以是每个模型达到性能最优。

实验之前通过统计各个数字出现的情况，对分布较为均匀的案例进行结果分析。统计如下：

训练集上

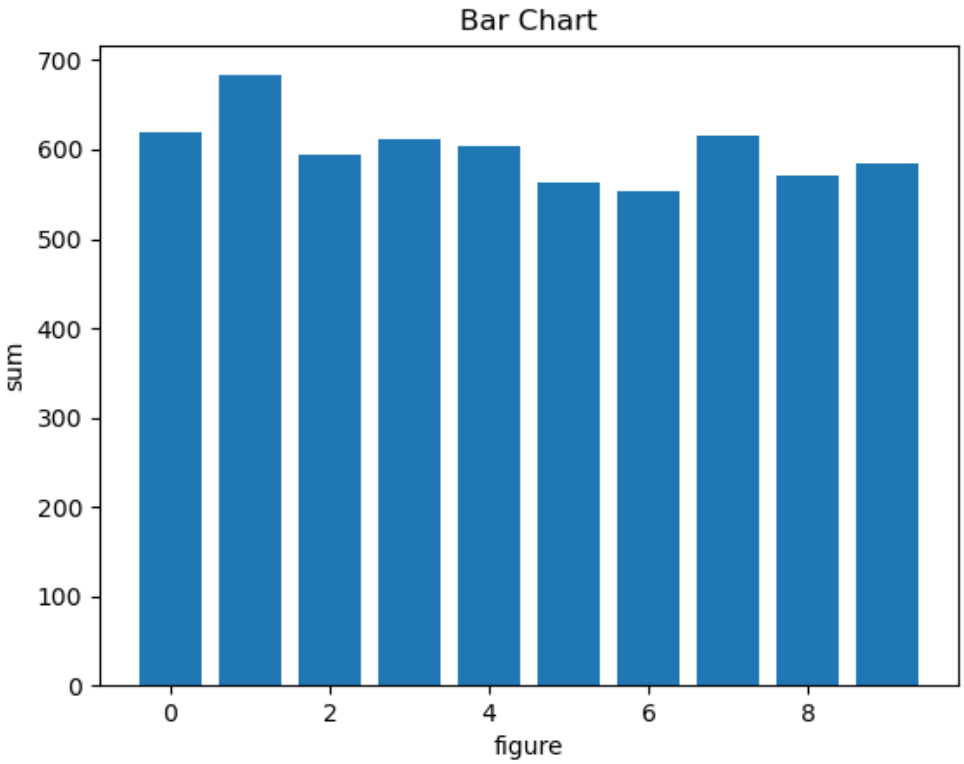

图 1 训练集上数字分布

测试集上：

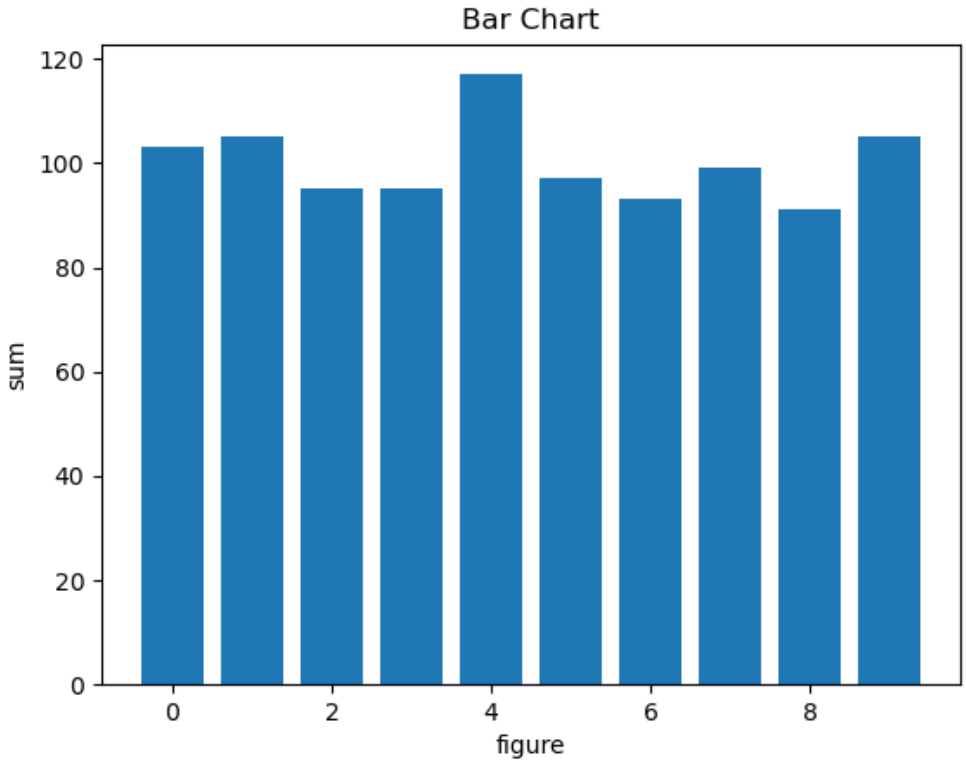

图 2 测试集上数字分布

可以看到，通过 2.1 所述的数据准备方法，基本可以做到均匀取值。满足实验要求。

3.1 Resnet 结构神经网络

经过尝试，激活函数采用 ReLu 函数，学习率设置为 0.001 时，该神经网络性能最佳。下面是 loss 和 accuracy 变化图。

训练集上：

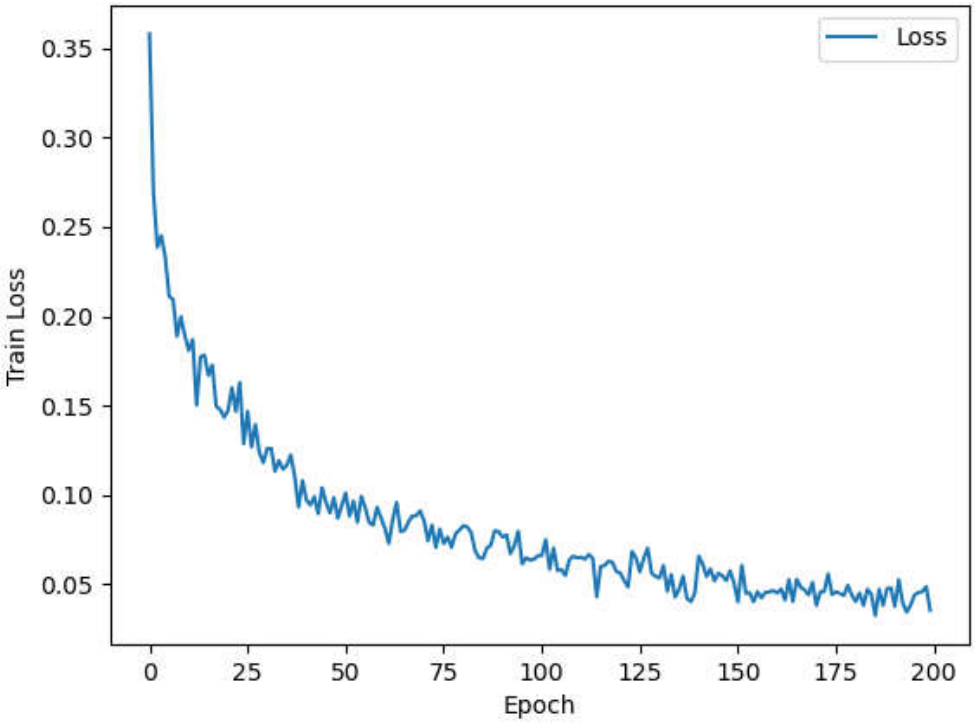

图 3 训练集 loss

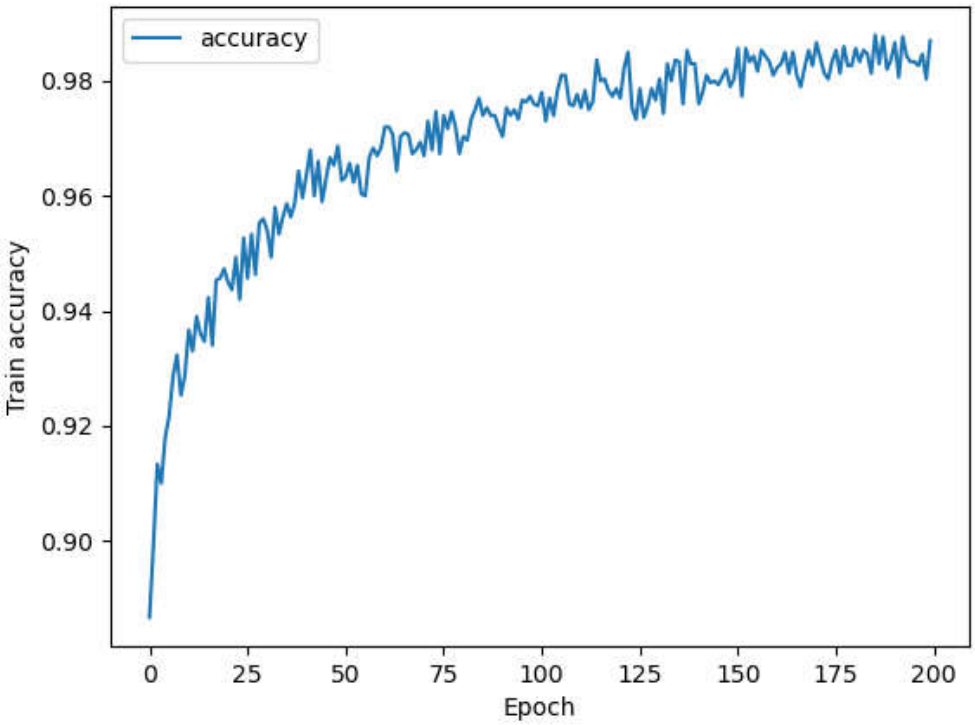

图 4 训练集 accuracy

测试集上：

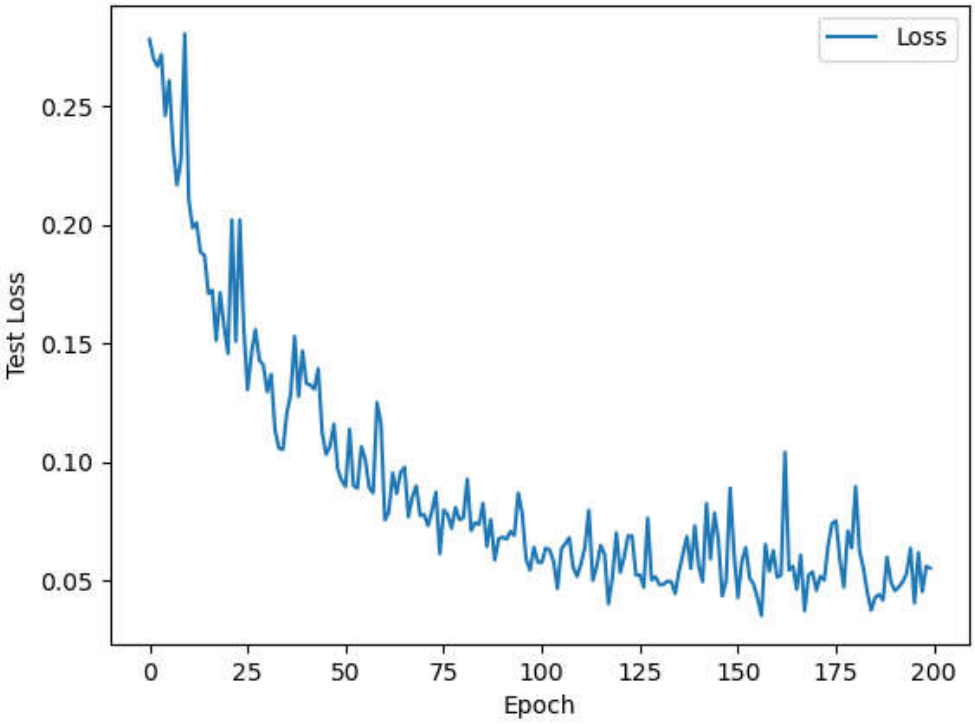

图 5 测试集 loss

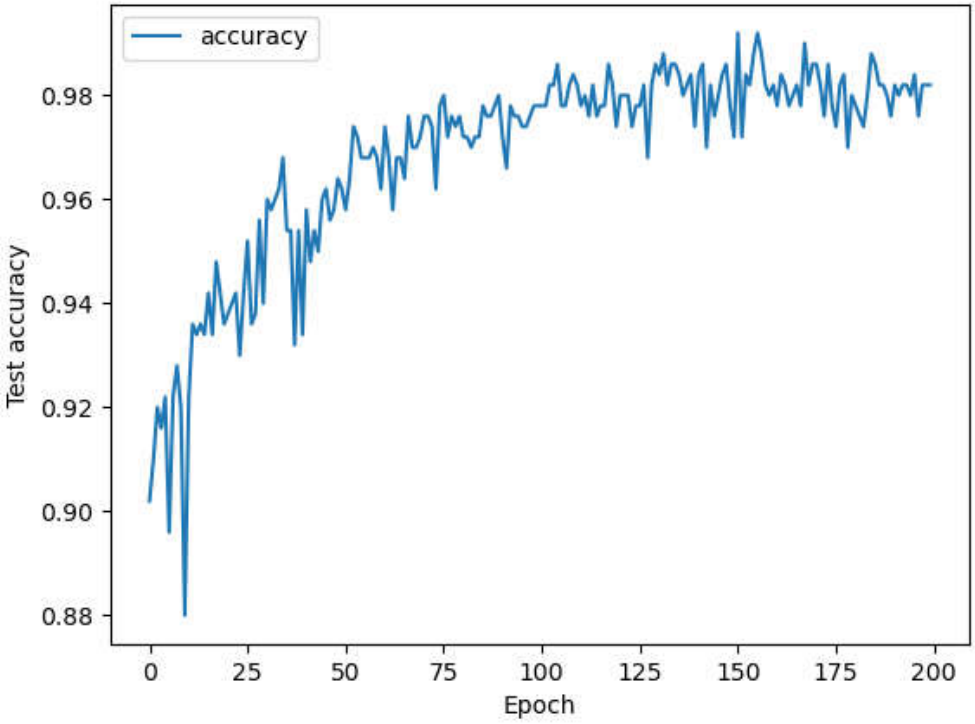

图 6 测试集 accuracy

最终训练集与测试集上的 loss 和 accuracy 如下：

	训练集		测试集	
	loss	accuracy	loss	accuracy
Resnet	0.035036	0.987000	0.055120	0.982000

每一轮 batch 训练后的在训练和测试集上的 loss 和 accuracy 数据见同目录下“Resnet 模型数据.txt”文件。

3.2 简单的卷积神经网络

采用 2.2 所描述的简单神经网络模型。

初始化如下：

```
    def __init__(self):
        super(MyJudgNet, self).__init__()
        self.conv1 = nn.Conv2d(2, 16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 2)
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu = nn.ReLU()
```

经过超参数尝试，选取激活函数为 ReLu，学习率 0.001，优化器为 Adam 网络性能最佳。

loss 和 accuracy 变化如下：

训练集上：

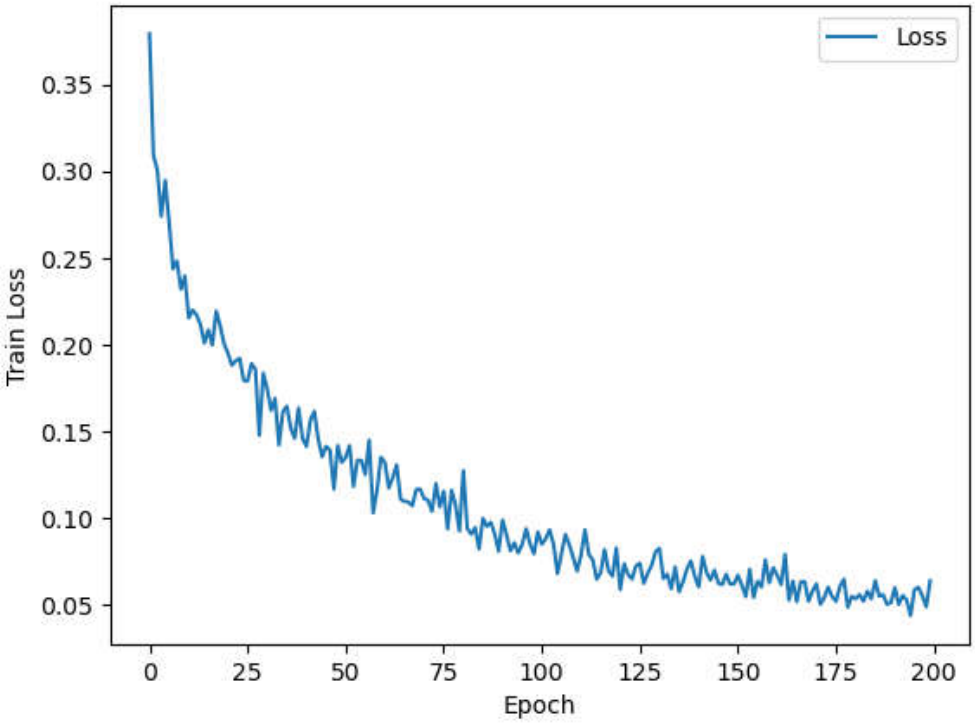

图 7 训练集 loss

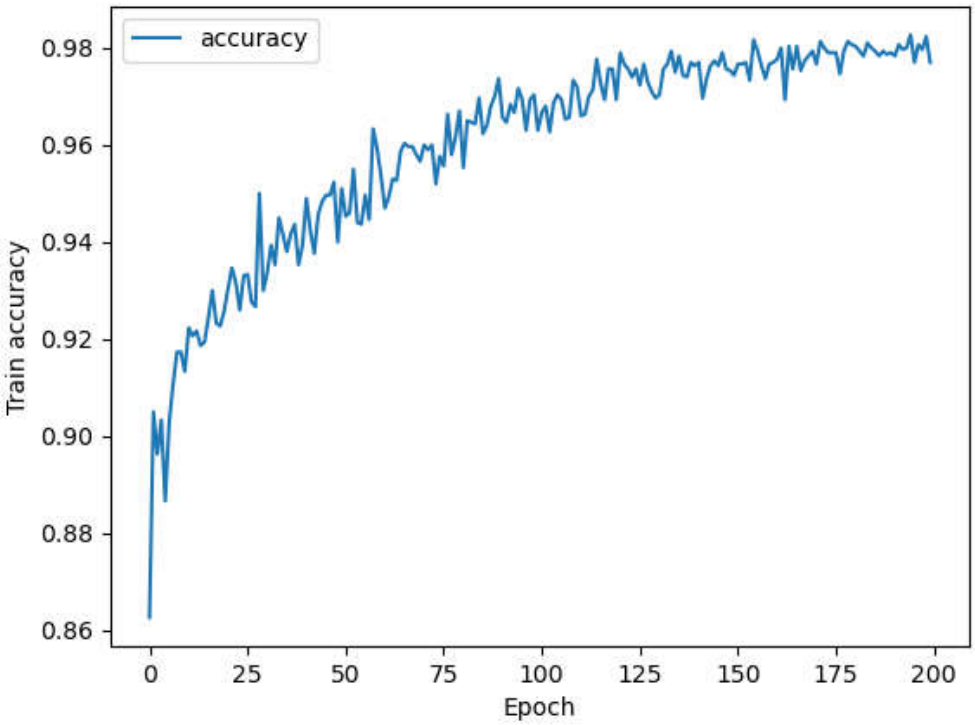

图 8 训练集 accuracy

测试集上：

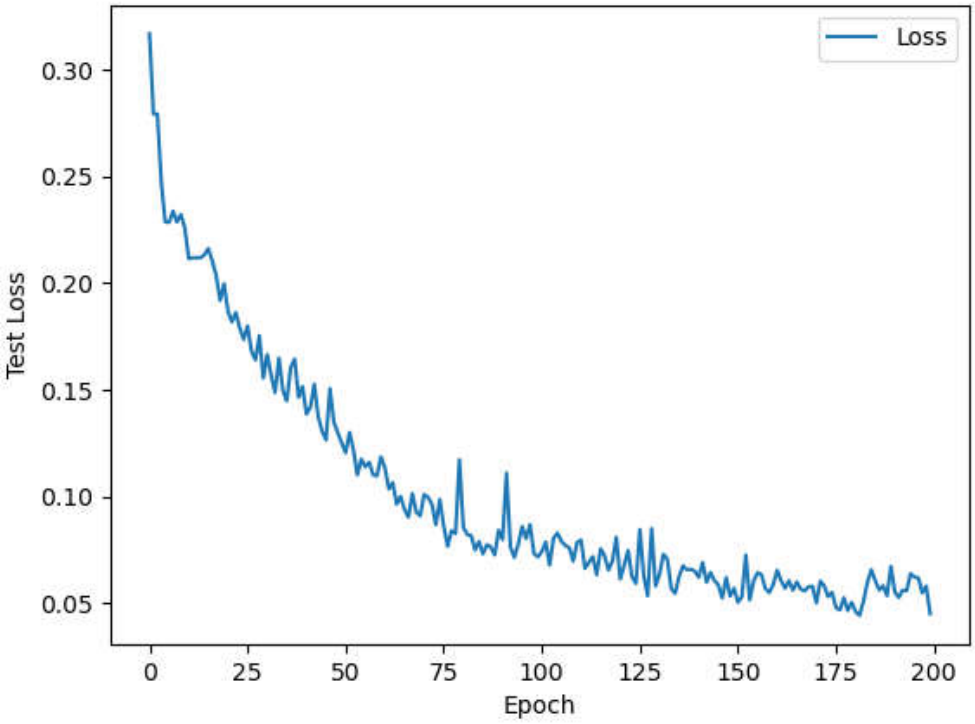

图 9 测试集 loss

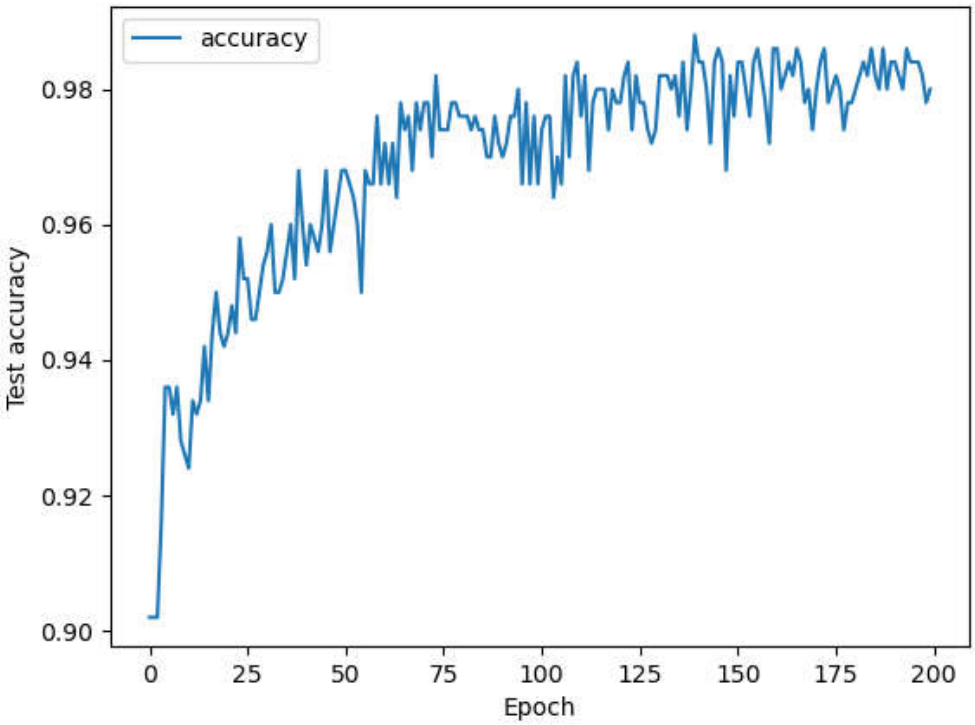

图 10 测试集 accuracy

最终训练集和测试集上的 loss 和 accuracy 如下：

	训练集		测试集	
	loss	accuracy	loss	accuracy
Resnet	0.063839	0.977000	0.045029	0.986000

每一轮训练过后训练集和测试集上的 loss、accuracy 见同目录下“简单卷积神经网络数据.txt”文件。

4 结论

本实验使用卷积神经网络架构完成。尝试了简单的卷积神经网络和简单的 Resnet 网络结构。实验过程中需要不断尝试超参数是网络模型达到最优。从上面的结果变化图和数据来看，两个网络均收敛较快，也比较平和。最终结果 Resnet 以微弱的优势获胜，但是 Resnet 叠加了 3 个 block，比另一个模型复杂了很多，而且耗时也较大，性价比不高。对于某种任务，不是网络越复杂越好，需综合实际考虑。