华中科技大学

2023

计算机视觉实验　课程设计报告

专　　业：	计算机科学与技术
班　　级：	计卓 2101 班
学　　号：	U202112071
姓　　名：	王彬
邮　　件：	wangbin2002@hust.edu.cn
完成日期：	2023.12.26

目 录

实验二：基于卷积神经网络的 MNIST 手写体数字识别

1.1 任务描述

设计一个卷积神经网络，并在其中使用 ResNet 模块，在 MNIST 数据集上实现 10 分类手写体数字识别。

注意事项：

1. 深度学习框架任选。
2. 不能直接导入现有的 ResNet 网络。
3. 可以尝试不同的网络架构、激活函数、训练超参数等，至少尝试两种，观察并比较网络性能。
4. 实验报告需包含神经网络架构、每一轮 mini-batch 训练后的模型在训练集和测试集上的损失、最终的训练集和测试集准确率，测试集十分类中每一类的准确率，不同设计变化导致的网络性能差异，以及对应的实验分析。

1.2 实验内容

(一) 数据集获取与重排序

我们首先加载 MNIST 数据集并进行预处理，将其导入模型中。这里我们直接使用 torch.utils.data.DataLoader()方法对 train_dataset 和 test_dataset 的训练集与测试集进行加载和重排序。我们的代码如下：

```
# 加载MNIST 数据集并进行预处理
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])
```

```
    train_dataset = datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
    test_dataset = datasets.MNIST(root='./data', train=False,
download=True, transform=transform)
    train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=60, shuffle=True, num_workers=2)
    test_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=1000, shuffle=False, num_workers=2)
```

(二) 构建 ResNet 网络模型

我们设计残差网络 ResNet 模型。我们分别运行了多种激活函数的（ReLU、Softplus）和多种残差网络架构（变化网络层数及每层的神经元个数的）网络。

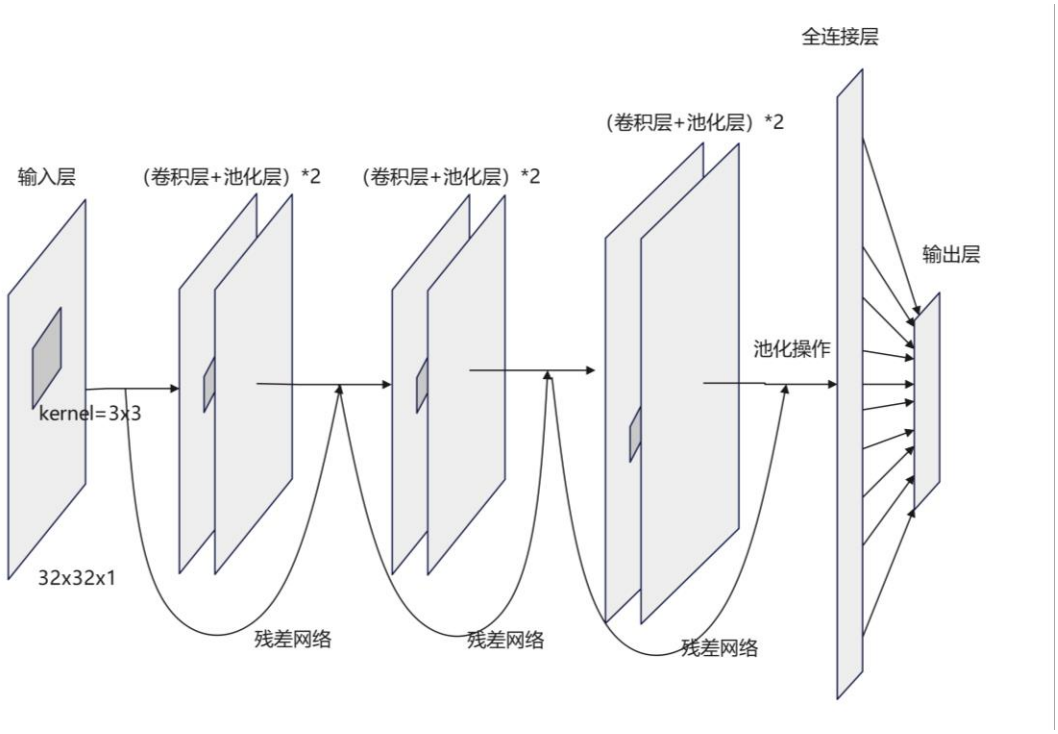


图 1 残差神经网络

我们的模型定义为：

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
```

```
        # 3x3 卷积层
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,
stride=stride, padding=1, bias=False)
        # 归一
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        #再卷一次
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,
stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.stride = stride

    def forward(self, x):
        residual = x
        # 卷积操作
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        if self.stride != 1 or residual.shape[1] != out.shape[1]:
            residual = nn.Conv2d(residual.shape[1], out.shape[1],
kernel_size=1, stride=self.stride, bias=False)(residual)
        out += residual
        out = self.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, in_channels=1, num_classes=10):
        super(ResNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, 64, kernel_size=7, stride=2,
padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
```

```
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer0 = self.make_layer(64, 64, 2)
        self.layer1 = self.make_layer(64, 64, 2)
        self.layer2 = self.make_layer(64, 128, 2, stride=2)
        self.layer3 = self.make_layer(128, 256, 2, stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(256, num_classes)

    def make_layer(self, in_channels, out_channels, num_blocks, stride=1):
        layers = []
        layers.append(ResidualBlock(in_channels, out_channels, stride))
        for _ in range(num_blocks - 1):
            layers.append(ResidualBlock(out_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.maxpool(out)
        # out = self.layer0(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.maxpool(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

这里我们将残差块封装至 ResidualBlock 类中，每个残差块先进行卷积后归一化和 ReLU 池化，而后再进行这样的操作，再与残差项加和后池化输出。残差网络定义再 ResNet 中，它使用四层 ResidualBlock 堆叠，

(三) 训练与结果绘制

由于篇幅限制，训练代码详见附录中的代码文件（./train_cuda.py）在测试集上可达到 99.47%准确率。

这里需要注意的是，我们在训练一定轮数（既一个 batch 后）就会对测试集的所有样本进行测试，并将测试集的结果写入输出列表。因此我们下文中所有准确率，均为实际测试样本的准确率。

对于每个类别的训练和测试结果，也就是每种数字的识别正确率，详见附录中的结果文件（./Results/acc_for_each_classification.txt）。该结果中有十个列表，分别是数字 0~9 的每个 batch 的正确率测试变化。

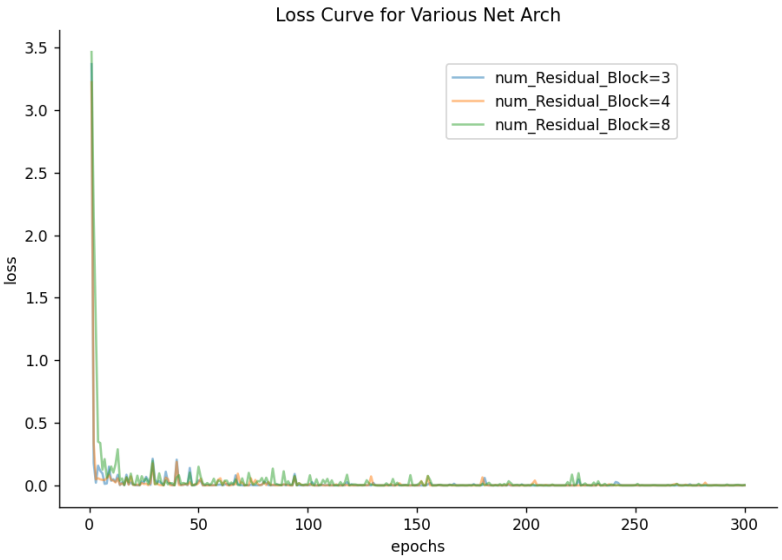
对于图形绘制，代码详见（./matplot.py）。我们调用 matplotlib.pyplot 画出 Loss 和 Acc 曲线。这里，绘图函数需要传入文件地址参数，以便对多个曲线统一绘制，并进行对比分析。我们还编写了一个直接对列表绘制曲线的方法，在本实验中我们主要使用该方法进行图形绘制。

1.3 实验结果及其分析

(一) 网络结构对于训练结果的影响

我们首先对比不同网络结构对于训练的影响。我们堆叠不同深度的残差块，每个残差块内部具有两层卷积神经网络和池化层，对于 3 层、4 层和 8 层 ResNet 的测试损失和准确率结果如图 2 所示。

我们注意到对于深层神经网络，由于其待训练参数较多，在训练初期，其损失值下降会略慢一些，但总体的 Acc 也没有 3 层或 4 层 ResNet 高。这不应当被归结于梯度消失，可能由于我们对于网络的设计不当，使得较深层的神经元尚未学习到较多特征。而对比 3 层和 4 层的 ResNet，我们注意到 4 层的效果略好一些，最高准确率从 99.36%上升至 99.47%。



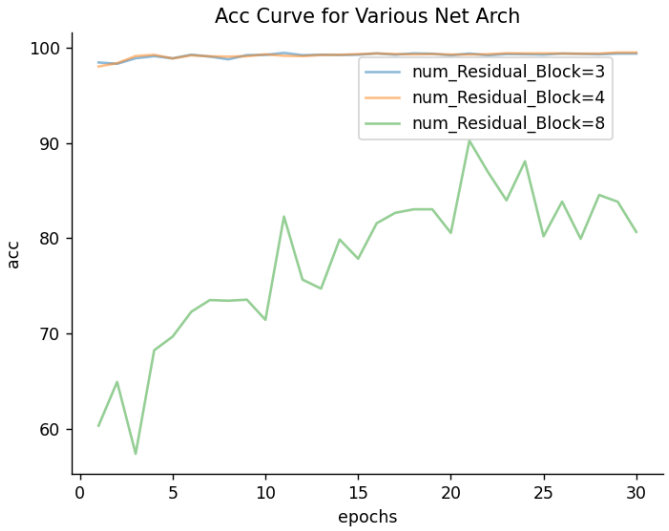
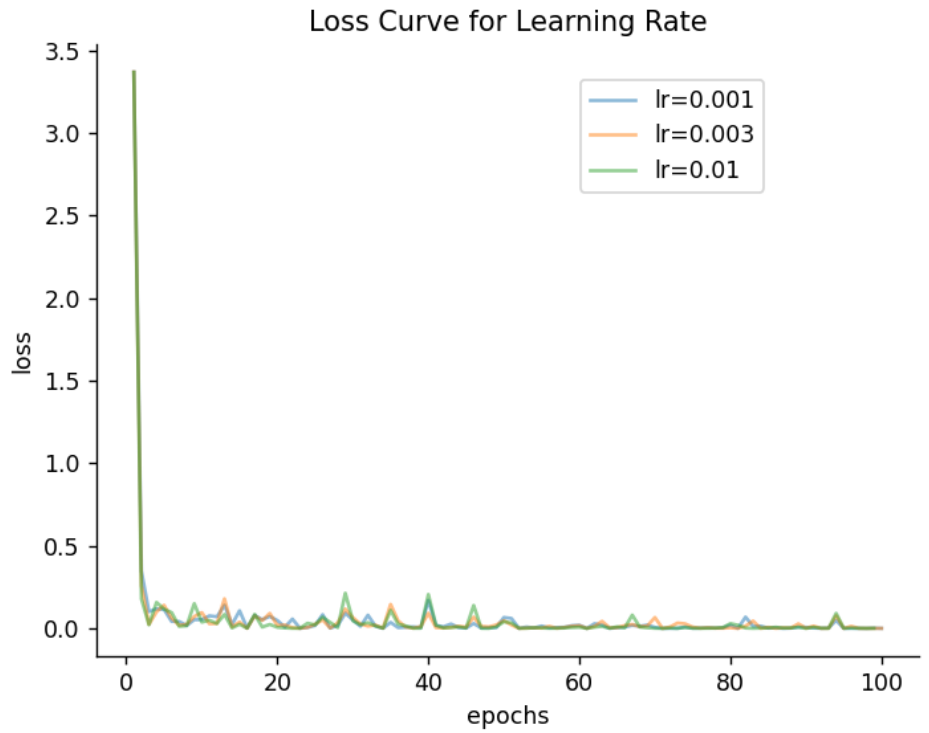


图 2 不同网络结构下的 Loss 及准确率变化

(二) 超参数对于训练结果的影响

我们测试不同学习率下的网络损失和正确率值，我们绘制的对于 lr=0.01、lr=0.003 和 lr=0.001 的测试结果数据如图 3 所示。我们注意到 lr 较大时收敛速度较快，但是其最终得到的模型准确率相对较低。这是因为它得到的分布相比小学习率者是不够精确的，容易发生统计分布振荡。这时应当调小学习率，一个较好的办法是使用自适应学习率。



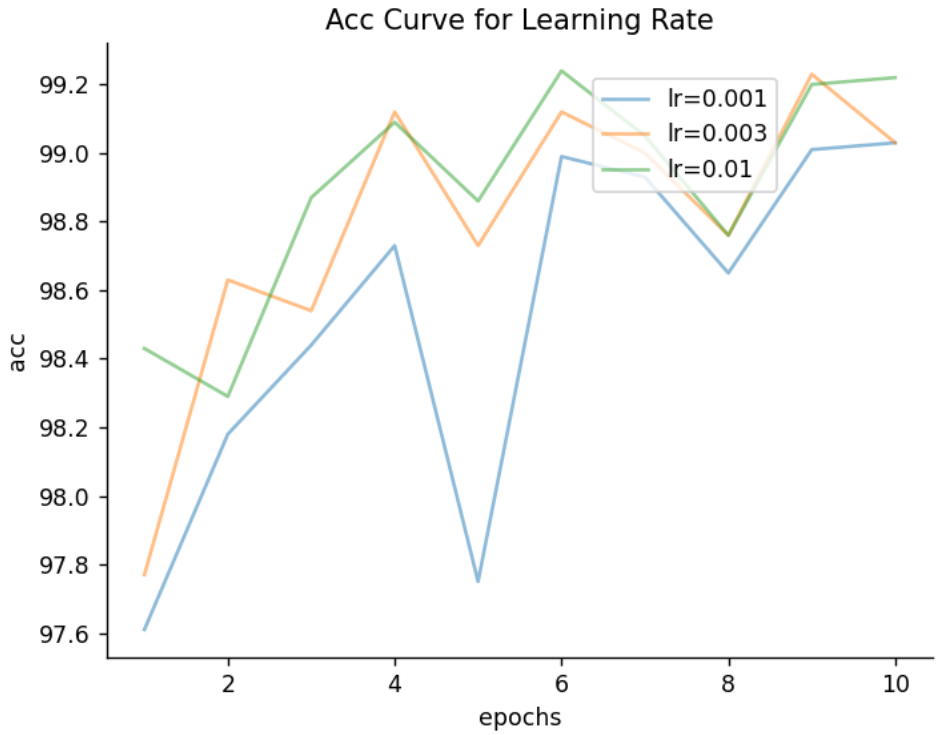


图 3 修改超参数的训练效果

(三) 激活函数对于训练结果的影响

我们使用不同的激活函数探讨其对卷积神经网络训练的影响。我们测试了使用 ReLU 函数以及 Softplus+ReLU 混合训练的方式，这里我们注意到 ReLU 的效果较好。

可能是因为使用混合激活函数的话，在反向传播时在 ReLU 层的导数为 0，而在 Softplus 层则具有一定误差。因此我们这样训练反而引入了误差，导致训练结果较差。我们注意到一般情况下 ResNet 都是采用 ReLU 层进行池化操作，这是因为 ReLU 在 x>0 区间求导为 1，不易产生梯度消失问题，而且计算速度较快，对于高层堆叠的神经网络也可以快速进行前馈和反向的传播。

随着训练的进行，其 Loss 将会趋同，Softplus 组将达到与 ReLU 组相近的 0.02 水平，但其波动仍然相对较大。这仍然是因为使用交替堆叠的结构会引入计算误差。

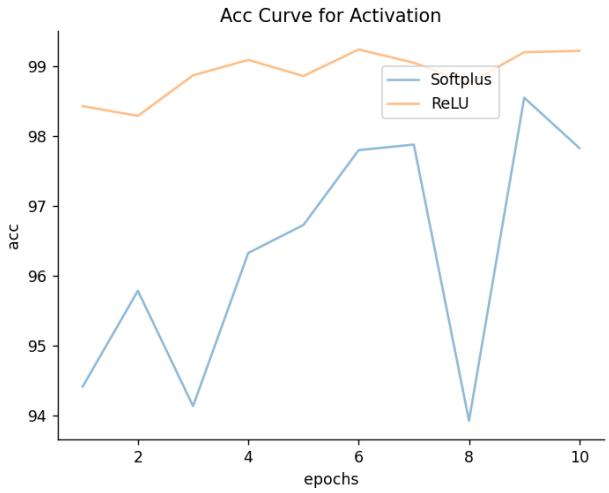


图 4 不同激活函数的对应准确率

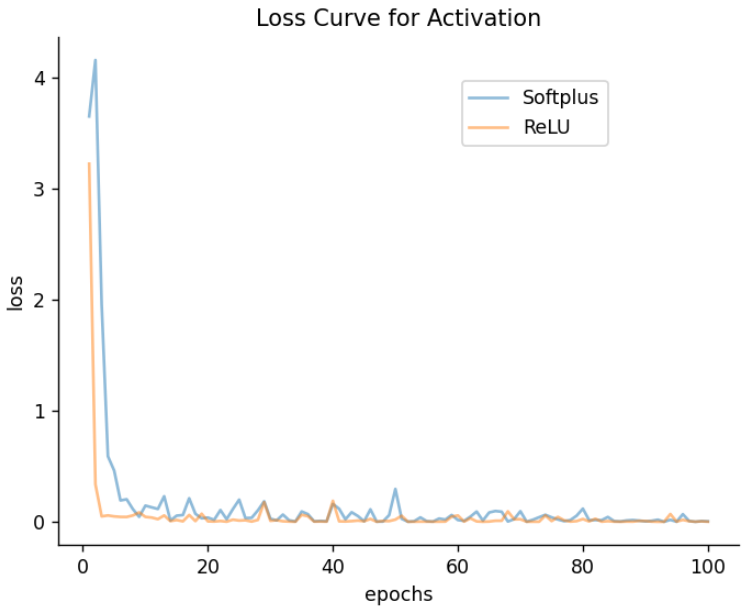


图 5 不同激活函数的训练 Loss 曲线

1.4 讨论

我们测试了不同架构、不同超参数和不同激活函数影响下对于残差神经网络的影响。其中，对于 3 层残差块，lr=0.01 的 ResNet 测试的对于不同类别数字的具体识别效果如图 6 所示。

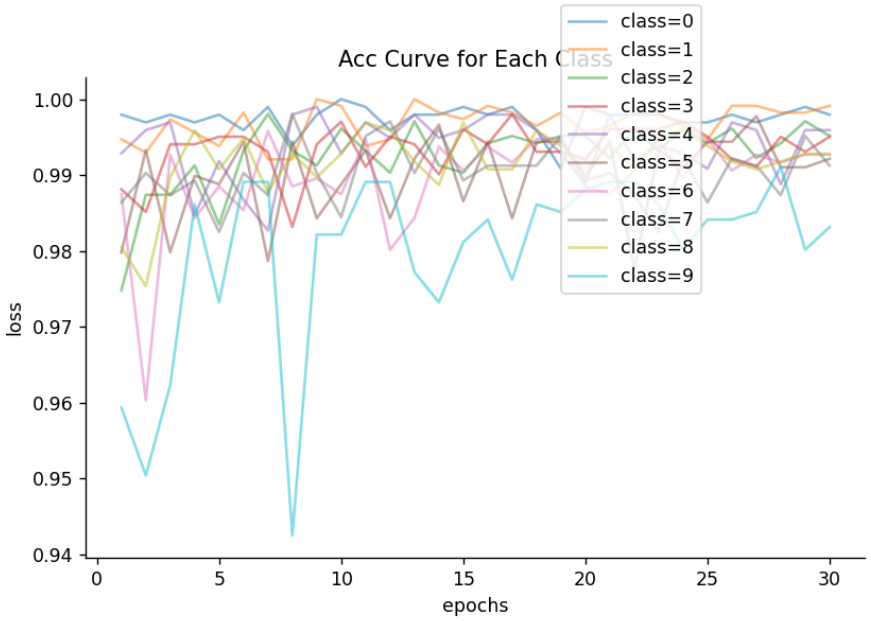


图 6 不同类别分类效果

我们注意到对于数字 0 和数字 1 的分类准确率几乎接近 1，但是对于数字 9 的分类准确率则徘徊在 0.98 附近。这是因为数据集的影响，不同的数字识别的效果不同。

我们最终得到的测试集正确率达到 99.47%，我们可以通过使用自适应学习率的方法进一步提高该准确率；同时，也可以更加优化网络架构，使用经典的 ResNet18 或 ResNet34 对于网络进行更换。