**HW03-CNN神经网络的构建以及CIFAR-10数据集的训练报告**

2300017729 刘卓洋

**一、源代码文件**

源代码文件为“cifar10\_cnn.py”，已附加在压缩包中。

**二、训练日志**

训练日志为“训练日志1.txt”，已附加在压缩包中。

**三、实验报告**

**1、网络结构设计**

使用PyTorch的nn.Module基类创建一个自定义的卷积神经网络Net，其中，包含7个卷积层，1个池化层（此池化层使用了4次），2个线性层：

**1) 7个卷积层：**

**self.conv1:**

self.conv1 = nn.Sequential(

    # 输入通道数in\_channels=3，输出通道数out\_channels=64

        nn.Conv2d(3, 64, kernel\_size=3, stride=1, padding=1),

        # 对输入batch的每一个特征通道进行normalize,64表示输出的channel数

        nn.BatchNorm2d(64),

        nn.ReLU()

        )

①输入通道数：3

输入通道数为3是因为原始的训练集图像即为32x32x3(RGB)的图像，因此输入的channel数为3

②输出通道数：64

输出通道数为64，为自定义值，形成64个独立的特征映射，以提取更多精细化的图像信息

③卷积核大小(kernel\_size)：3x3

采用小尺寸的卷积核，以提高提取特征的能力

④步长(stride)：1

步长为1，同样是为了提取出更多的特征

⑤填充(padding)：1

假设输入图像的高度和宽度分别为 H 和 W，卷积核的尺寸为 h × w，步长为 s，填充为 p，则卷积之后输出特征图的高度和宽度分别计算如下：

输出高度 OH = (H + 2p - h) / s + 1

输出宽度 OW = (W + 2p - w) / s + 1

当padding = 1并且stride = 1时：

如果卷积核的尺寸是奇数，比如 h = w = 3，那么：

输出高度 OH = (H + 2 - 3) / 1 + 1 = H

输出宽度 OW = (W + 2 - 3) / 1 + 1 = W

保持了输出的图像尺寸与输入的图像尺寸完全一致

⑥BatchNorm2d：通过对对来自上一层（这里是 Conv2d）输出的所有通道（共 64 个）进行归一化处理，以加快训练速度并提高模型泛化能力。

⑦nn.ReLU() 是 ReLU（Rectified Linear Unit）激活函数层，对每个元素应用非线性变换，公式为 f(x) = max(0, x)，用于增加模型表达能力。

总结：对于self.conv1，其输入一个32x32x3的图像，输出一个32x32x64的特征图像。

**self.conv2:**

self.conv2=nn.Sequential(

            nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1),

            nn.BatchNorm2d(128),

            nn.ReLU()

        )

与self.conv1大致相同，区别在于输入通道数为64，输出通道数为128，通过对特征通道数的翻倍，实现了对高维细节和特征的提取。

下列self.conv3到self.conv7均类似，不展开说明。

**self.conv3:**

self.conv3=nn.Sequential(

            nn.Conv2d(128, 128, kernel\_size=3, stride=1, padding=1),

            nn.BatchNorm2d(128),

            nn.ReLU()

        )

**self.conv4:**

self.conv4=nn.Sequential(

            nn.Conv2d(128, 256, kernel\_size=3, stride=1, padding=1),

            nn.BatchNorm2d(256),

            nn.ReLU()

        )

**self.conv5:**

self.conv5=nn.Sequential(

            nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1),

            nn.BatchNorm2d(256),

            nn.ReLU()

        )

**self.conv6:**

self.conv6=nn.Sequential(

            nn.Conv2d(256, 512, kernel\_size=3, stride=1, padding=1),

            nn.BatchNorm2d(512),

            nn.ReLU()

        )

**self.conv7:**

self.conv7=nn.Sequential(

            nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1),

            nn.BatchNorm2d(512),

            nn.ReLU()

        )

**2) 1个池化层**

**self.pool:**

self.pool = nn.MaxPool2d(2, 2)

窗口大小（kernel size）：2

池化操作将在输入特征图上以 2x2 的窗口滑动。

步长（stride）：2

每次滑动窗口移动的距离为 2。这意味着池化窗口不会重叠，对于每个 2x2 的区域，仅保留该区域内最大的数值作为输出特征图对应位置的值。由于输入的图像为32x32，因此每次操作都会使图像的尺寸减半。

该池化层在前向传播中使用了4次，以达到使原始图像从32x32降维到2x2的目的，进而展平输入全连接层。

**3) 2个线性层（全连接层）**

**self.fc1,self.fc2:**

self.fc1 = nn.Linear(512\*2\*2, 240)

self.fc2=nn.Linear(240,10)

self.fc1创建了一个全连接层，输入维度是512 \* 2 \* 2，输出维度是240。前面的512 \* 2 \* 2通常表示经过一系列卷积层和池化层后，特征图展平（flatten）后的大小。例如，如果最后一个卷积层的输出特征图是512个通道，每个特征图的大小是2x2，那么将这些特征图沿着通道维度展平后，就会得到一个长度为512 \* 2 \* 2的一维向量。

self.fc2定义了第二个全连接层，它的输入维度是上一层（fc1）的输出维度240，输出维度是10。这个维度设置通常对应于最终的分类类别数目，例如在CIFAR-10数据集中，共有10种类别的图片，所以输出维度设为10。

**4) 前向传播**

def forward(self, x):

        x = x.view(-1, 3, 32, 32)

        #32x32

        x=self.conv1(x)

        x=self.pool(x)

        #16x16

        x=self.conv2(x)

        x=self.conv3(x)

        x=self.pool(x)

        #8x8

        x=self.conv4(x)

        x=self.conv5(x)

        x=self.pool(x)

        #4x4

        x=self.conv6(x)

        x=self.conv7(x)

        x=self.pool(x)

        #2x2

        x = x.view(-1, 512\*2\*2)

        x = F.tanh(self.fc1(x))

        x=F.softmax(self.fc2(x),dim=1)

        return x

如上代码所示，前向传播部分采用了四次池化层，以达到降维的目的，此外，最终的全连接层使用tanh函数进行梯度传播，tanh函数实测性能优于relu函数

**2、部分关键性能改进以及数据支持**

**1) 线性层的增加：**

在神经网络的结构设计中，原本只有一个全连接层：

self.fc1 = nn.Linear(512\*2\*2, 10)

但是在其他条件不变的条件下，只有一个全连接层的模型在50个epoch后测试集准确率大概在50%左右，远远低于当前模型的75%左右，最终100个epoch左右的准确率在80%上下浮动，无法稳定，而当前模型可以稳定在81%以上，因此新的全连接层的加入使得神经网络的表达能力增强，进而提高了模型的学习能力和准确性。

**2) 卷积层的增加以及填充参数的调整：**

原本设计了两个卷积层

self.conv1 = nn.Sequential(

    # 输入通道数in\_channels=3，输出通道数out\_channels=64

        nn.Conv2d(3, 64, kernel\_size=3, stride=1, padding=0),

        # 对输入batch的每一个特征通道进行normalize,64表示输出的channel数

        nn.BatchNorm2d(64),

        nn.ReLU()

        )

self.conv2=nn.Sequential(

        nn.Conv2d(64, 16, kernel\_size=3, stride=1, padding=0),

        nn.BatchNorm2d(16),

        nn.ReLU()

        )

且其padding均为0，但是padding=0不遵循还原规则，导致了图像信息的丢失和错误，因此在原本的设计中，准确率提升很慢，初期在20个epoch条件下只能达到30%左右（当前20个epoch准确率≈68%），卷积层数量的增加有助于提取细节特征，尤其是增加输出通道数量，以进行信息的提取的保留。

**3、数据增强技术的使用及其对模型性能的影响**

在当前模型中，使用了以下数据增强：

transform = transforms.Compose([

    # 随机裁剪为 32x32，填充为 4

    transforms.RandomCrop(32,padding=4),

    #随机水平翻转，发生的概率为0.5

    transforms.RandomHorizontalFlip(p=0.5),

    #随机垂直翻转，发生的概率为0.5

    #transforms.RandomVerticalFlip(p=0.5),

    # 随机旋转，随机角度为 15

    transforms.RandomRotation(15),

    # 转换为 Tensor

    transforms.ToTensor(),

    # 归一化，其中mean和std的数值是根据 CIFAR-10 数据集计算得到的，预先训练好的模型需要这些数值。

    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465), std=(0.2023, 0.1994, 0.2010)),

])

其中函数的功能已经注释在语句后端，数据增强功能通过对原有的图像（数据量有限）进行一系列操作，使得能够在原有数据的基础上得到新的数据，起到增加数据量、减少过拟合的作用，尤其是在CIFAR-10数据集中，训练数据数据量并不大，此时数据增强的功能至关重要，在当前模型中，通过上述数据增强过程，使得模型过拟合的情况得到改善：

当不添加数据增强时，训练集平均83%的准确率只能对应测试集平均77%的准确率，而添加数据增强时，训练集平均83%的准确率即可以对应测试集平均81%的准确率。

但是同样的，在训练过程中发现，CIFAR-10的训练精度主要取决于网络结构，而数据增强对准确率的提升较为不显著。

**4、训练过程中遇到的挑战以及解决方法**

**1) 测试集最终准确率长期收敛于或在79%左右。**

（详细数据举例参见训练日志1）

解决方法：通过增加全连接层self.fc2、调整激活函数从relu为为tanh，成功取得了90个epoch后几乎可以稳定在81%以上的训练成果。

**2) 免费的Google Colab使用时间有限**

解决方法：通过付费的Autodl租用服务器完成训练。

**5、最终性能评估**

（详细数据请参见训练日志3）

对于本模型，最终在100个epoch后，训练集准确率最高可达89%以上，测试集准确率稳定在80%以上，后10个epoch的平均测试集准确率达到81%左右，超过了要求的准确率标准，并且模型结构较为清晰，改进思路较为合理，性能较佳。