浙江大学

课程名称：计算机视觉

姓　　名：刘佳润

学　　院：计算机科学与技术学院

专　　业：数字媒体技术

学　　号：3180105640

指导教师：潘纲

2021 年 1 月 6 日

浙江大学实验报告

课程名称：计算机视觉 实验类型：综合

实验项目名称：CNN 卷积神经网络

学生姓名：刘佳润 专业：数字媒体技术 学号：3180105640

同组学生姓名： 指导老师：潘纲

实验地点： 实验日期：2021 年 1 月 6 日

一、 实验内容和要求

- 编写程序，实现 LeNet-5 卷积神经网络，对 MNIST 手写数字数据库进行训练与识别，展示准确率等。
- 自己选择神经网络，对 CIFAR-10 数据库进行图像物体训练与识别。

二、 实验器材

Python 3.7

开发平台：Windows10 Visual Studio Code

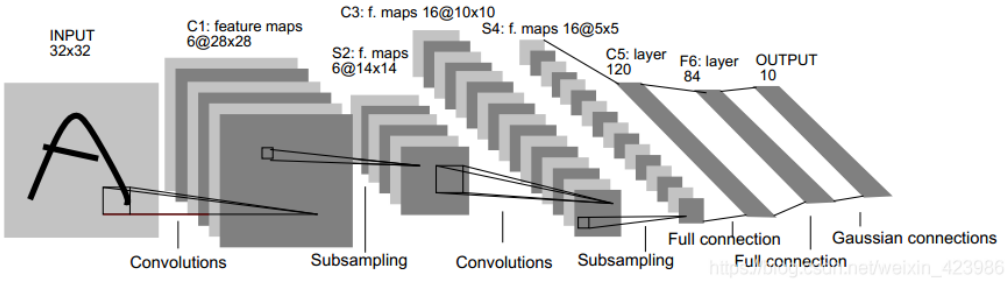
机器学习库：torch 1.6.0 torchvision 0.7.0

辅助：CUDA 10.2，用于进行 GPU 加速

三、 具体实现

1. LeNet-5 实现

使用 torch 的 nn.Module 类的派生，可以编写 LeNet5 的结构如下：其中调用 nn.Conv2d()函数进行卷积层设置，用 nn.Linear()函数进行全连接操作。在正向传导的过程中，规定了两次池化，使用 F.max_pool2d 函数。每一经过一层，对结果调用 F.relu()函数进行激活，形成新的输出。




```python
#  定义网络
class LeNet5(nn.Module):

    def __init__(self):
        super(LeNet5, self).__init__()      # 继承父类nn.Model的属性并初始化
        self.conv1 = nn.Conv2d(1, 6, 5)     # 卷积层1——LeNet5第一层
        self.conv2 = nn.Conv2d(6, 16, 5)    # 卷积层2——LeNet5第三层
        self.fc1 = nn.Linear(16 * 5 * 5, 120)   # 全连接层1——LeNet5第五层
        self.fc2 = nn.Linear(120, 84)       # 全连接层2——LeNet5第六层
        self.fc3 = nn.Linear(84, 10)        # 全连接层output——LeNet第七层

    def forward(self, x):
        """正向传导过程

        :param x: 输入样本
        """
        x = F.max_pool2d(F.relu(self.conv1(x)), (2,2)) # 池化层1——LeNet5第二层
        x = F.max_pool2d(F.relu(self.conv2(x)), (2,2)) # 池化层2——LeNet5第四层
        # x = x.view(-1, self.num_flat_features(x))   # 数据重组为256*(16*5*5)
        x = x.view(-1, reduce(lambda x,y:x*y, x.size()[1:]))   # 数据重组为256*(16*5*5)
        x = F.relu(self.fc1(x))             # 激活函数，产生新的输出
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

在实现卷积神经网络的过程中，调用 pytorch 的数据加载模块的部分是遇到的一个难点。调用 torch.utils.data.DataLoader()，设定批的大小，是否随机重组，以及 num_workers（进程数），由于使用的是 Windows 所以对多线程支持的并不好。

```python
train_loader = torch.utils.data.DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True, num_workers=4)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=BATCH_SIZE, shuffle=False, num_workers=4)
```

训练过程：使用优化函数 optimizer（选用 Adam 算法）和损失函数（交叉熵函数 CrossEntropyLoss），对 loss 调用 backard()函数进行反向传播过程。注意在训练前对网络进行 train()设置，启用 batchnormalization 和 dropout，防止网络过拟合。

```python
def train(model, device, train_loader, optimizer, criterion, epoch):
    """训练过程

    :param model: 输入网络模型
    :param device: 分配设备
    :param train_loader: 训练数据集加载
    :param optimizer: 优化函数
    :param criterion: 损失函数
    :param epoch: 训练总次数
    """
    model.train()   # 启用 BatchNormalization 和 Dropout，让model变成训练模式，起到防止网络过拟合的问题
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if (batch_idx + 1) % 30 == 0:
            print('Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

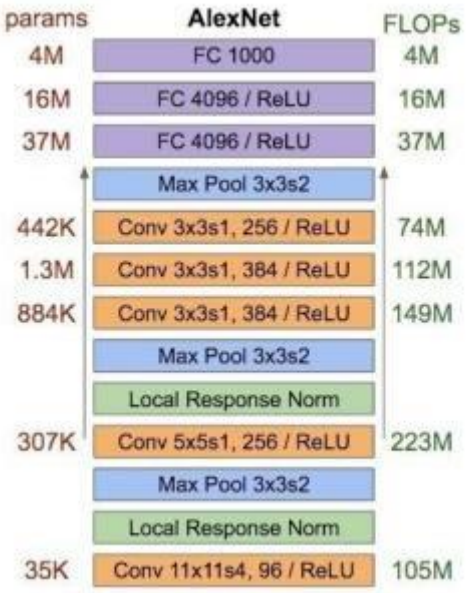
测试过程：启用 eval()模式，对输入数据进入网络进行传播，对输出的 output 取极大值作为预测结果 pred。

```python
def test(model, device, test_loader, criterion):
    """测试过程

    :param model: 输入网络模型
    :param device: 分配设备
    :param test_loader: 测试数据集加载
    :param criterion: 损失函数
    """
    model.eval()    # 不启用 BatchNormalization 和 Dropout，不会取平均，而是用训练好的值
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target)
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

2. AlexNet 实现

网络定义如下图：




```python
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.conv1 = nn.Conv2d(3,96,11,4)                       # 卷积层1，卷积核个数96，大小为11*11*3，步长为4
        self.conv2 = nn.Conv2d(96,256,5,padding=2,groups=2)     # 卷积层2，卷积核个数256，大小为5*5*48，步长为2，填充为2
        self.conv3 = nn.Conv2d(256,384,3,padding=1)             # 卷积层3，卷积核个数384，大小为3*3*256，填充为1
        self.conv4 = nn.Conv2d(384,384,3,padding=1, groups=2)   # 卷积层4，卷积核个数384，大小为3*3，填充为1
        self.conv5 = nn.Conv2d(384,256,3,padding=1, groups=2)   # 卷积层5，卷积核个数256，大小为3*3，填充为1
        self.fc1 = nn.Linear(256*6*6,4096)                      # 全连接层1，神经元个数4096
        self.fc2 = nn.Linear(4096,4096)                         # 全连接层2，神经元个数4096
        self.fc3 = nn.Linear(4096,num_classes)                  # 全连接层3，神经元个数4096

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)),(2,2))           # 对卷积层1的池化
        x = F.max_pool2d(F.relu(self.conv2(x)),(2,2))           # 对卷积层2的池化
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = F.max_pool2d(F.relu(self.conv5(x)),(2,2))           # 对卷积层5的池化
        x = x.view(x.size(0),256*6*6)
        x = F.dropout(F.relu(self.fc1(x)),p=0.5)                # 全连接层使用dropout和relu
        x = F.dropout(F.relu(self.fc2(x)),p=0.5)
        x = self.fc3(x)
        return x
```

注意在训练前也要对数据先做预处理，利用 torchvision 的处理函数进行 resize 和转换成张量（tensor）的处理。另外调用 Normalize 函数，将原来的 tensor 从(0,1)变换到(-1,1)区间。

```python
transform = transforms.Compose([
    transforms.Resize(227),
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
])
```

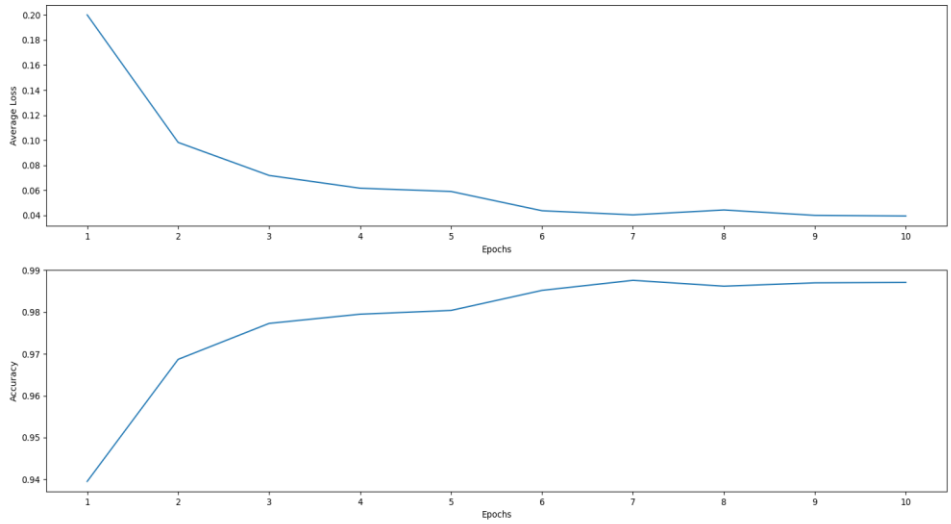
对 CIFAR-10 的训练和检测与 MNIST 的思想是类似的，不再赘述。

四、 实验结果与分析

1. LeNet-5 对 MNIST 的训练与识别

设置 BATCH_SIZE 为 512，总共训练 10 个 epoch。每次一个 epoch 在过完一遍训练数据之后再过一遍测试数据，得到一次准确度和损失函数的值。训练和测试的输出结果保存在 LeNet.log 里，模型保存为 LeNet.pth。

对训练结果进行可视化处理如下：



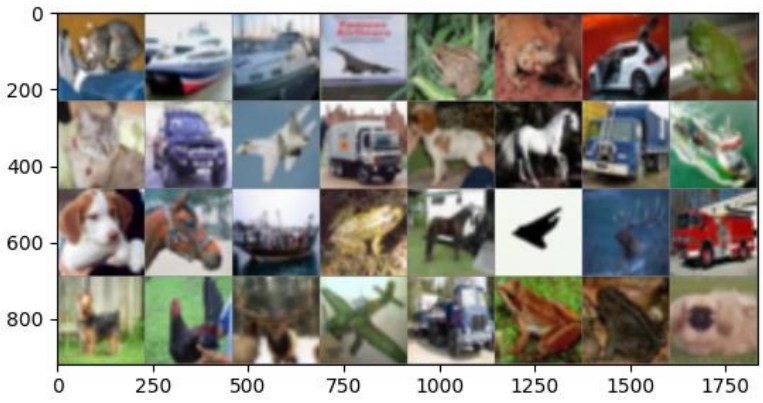
2. AlexNet 对 CIFAR-10 的训练与识别

设置 BATCH_SIZE 为 32，总共训练 20 个 epoch。每次一个 epoch 在过完一遍训练数据之后再过一遍测试数据，得到一次准确度和损失函数的值。训练和测试的输出结果保存 AlexNet.log 里，模型保存为 AlexNet.pth。

因为 AlexNet 网络比较复杂，而且 CIFAR-10 数据量也较大，现将训练的网络结构打印如下验证是否正确：

```
AlexNet(
  (conv1): Conv2d(3, 96, kernel_size=(11, 11), stride=(4, 4))
  (conv2): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), groups=2)
  (conv3): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=2)
  (conv5): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=2)
  (fc1): Linear(in_features=9216, out_features=4096, bias=True)
  (fc2): Linear(in_features=4096, out_features=4096, bias=True)
  (fc3): Linear(in_features=4096, out_features=10, bias=True)
)
```

我们对训练后的结果先随机选择一批数据进行测试：



对比实际标签和预测标签：在 32 张图中正确判断了 27 张，正确率约为 84%。

```
GroundTruth:  cat  ship  ship airplane  frog  frog  automobile  frog  cat  automobile
airplane truck  dog horse truck  ship  dog horse  ship  frog horse  airplane  deer  truck
 dog  bird  deer airplane truck  frog  frog  dog
Predicted:  cat  ship  ship airplane  frog  frog  truck  frog  cat  automobile airplane
 truck  dog horse truck  ship  dog horse  ship  frog horse  bird  airplane truck  deer
 frog  deer airplane truck  frog  frog  dog
```

另外，在五万张训练数据中测试的结果显示正确率为 92%，在一万张新的测试数据中的结果为 77%。在十个标签中，正确率最高的 ship 达到了 91%，最低的 cat 也有近六成的判断正确率。

```
Accuracy of the network on the 50000 train images: 92 %
Accuracy of the network on the 10000 test images: 77 %
Accuracy of   airplane : 77 %
Accuracy of automobile : 89 %
Accuracy of       bird : 65 %
Accuracy of        cat : 59 %
Accuracy of       deer : 80 %
Accuracy of        dog : 67 %
Accuracy of       frog : 88 %
Accuracy of      horse : 73 %
Accuracy of       ship : 91 %
Accuracy of      truck : 83 %
```

五、 心得与体会

我之前曾经用过 tensorflow 做过 MNIST 的识别，所以这次使用了 torch 这一工具，拓展一下视野和能力。torch 在对于小型网络方面的编写和运算都要比 tensorflow 快，适合轻量开发。

在跑 AlexNet 的时候，因为数据量大，网络层数多，计算量很大，用 CPU 跑的话要炸掉。所以花了很长时间配置 CUDA 和 pytorch 的 gpu 加速版本适配。但是最后很遗憾的发现 Windows 对于多线程支持很差，所以虽然有所提速，但还是跑的很慢很慢。最终用了大概五六个小时才得到网络模型，而且模型文件竟然有 200 多 M（就不附在上交的压缩文件里了）。以后要加速还是要用 Linux 啊。

编写的过程是简单的，对照着参考论文和示意图一步一步设定网络的层的功能和参数即可，训练和识别的过程大同小异，需要根据不同的数据类别加以调整即可。

通过本次实验，再次体验了 CNN 网络的训练和测试流程与原理，进一步理解的 CNN 的 pipeline，对机器学习有了初步了解。