人工智能Project1

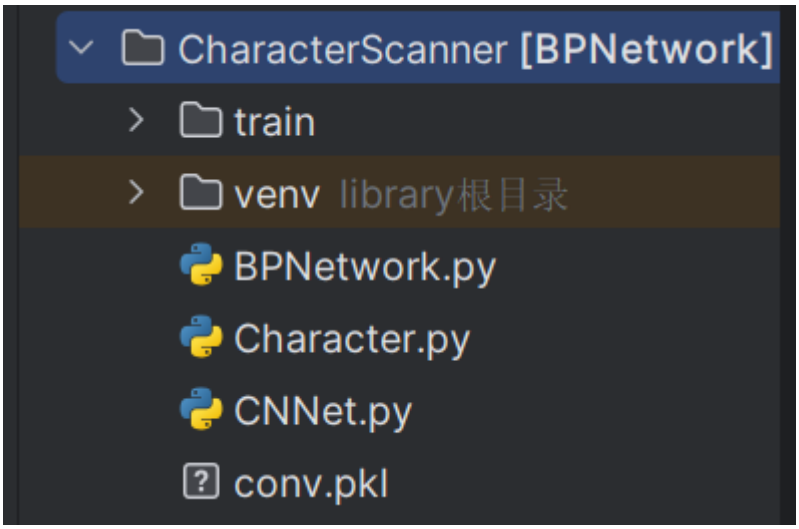
任务二：卷积神经网络

20302010075叶柯玲

项目简介

通过卷积神经网络实现12个手写体汉字识别，使用了pytorch

文件结构说明



train：保存手写字训练集

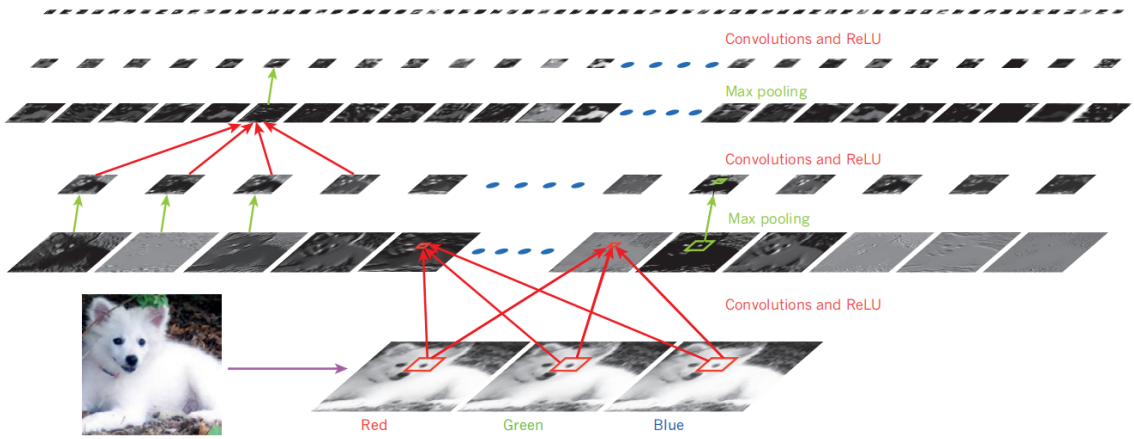
CNNet.py：卷积神经网络类，训练和测试

conv.pkl：已训练模型

卷积神经网络说明

卷积神经网络包括提取特征层和分类两大步

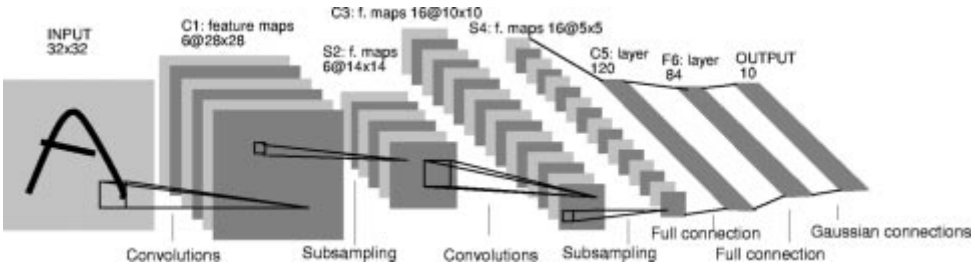
特征提取



上图展示了对图像进行卷积分类的过程

输入数据为三通道，再通过不同的卷积核对图形数据进行卷积，有多少个卷积核就得到多少个特征图输出，再对这些图进行最大池化（有的地方也用平均池化），目的是在不影响特征质量的情况下对图片进行压缩，再根据具体设计重复以上过程，完成特征提取的步骤

分类



在多次卷积和池化后讲数据展开至一维进行分类，进行这一步操作的模块叫全连接层，如果说特征提取是将原始数据映射到隐层特征空间，那么全连接就是将特征映射到样本标记空间，具体实现思路的核心同任务一反向传播相同，不加赘述

实现过程

卷积网络搭建

（1）卷积

卷积层的输入图像通道为 3，使用的训练图片有三个通道，输出通道为32（代表使用32个卷积核），一个卷积核产生一个单通道的特征图，卷积核kernel_size的尺寸为 3 * 3，stride 代表每次卷积核的移动像素个数为1，padding 填充，为1代表在图像长宽都多了两个像素

```
nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1)
```

（2）归一化

BatchNorm2d批量归一化，跟上一层的out_channels大小相等，否则不能连接好网络

```
nn.BatchNorm2d(num_features=32)
```

（3）激活函数

增加神经网络模型的非线性

```
nn.ReLU(inplace=True)
```

（4）最大池化

池化层的kernel_size 为2 * 2的滑动窗口

kernel_size 为2 * 2的滑动窗口

```
nn.MaxPool2d(kernel_size=2, stride=2)
```

在对以上四步重复后最终[28,28]的输入数据规模变为[7,7]

(5) 全连接

```
        self.classifier = nn.Sequential(
            # dropout层
            nn.Dropout(p=0.5),
            nn.Linear(64*7*7, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(512, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Linear(512, 12),
        )
```

(7) 综合输出

```
    def forward(self, x):
        x = self.features(x)
        # 展开为一维向量
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x
```

模型训练和测试

(1) 数据获取

利用datasets.ImageFolder获取本地中的图片作为数据集，地址需为分类图片坐在文件夹的上一级，utils.data.random_split在数据集中按照指定比例随机分配训练集和测试集，utils.data.DataLoader用于将文件转换为训练和测试过程使用的迭代器

```
# 训练集
TRAIN_DIR = "train"
# 样本规格化规则
train_transforms = transforms.Compose([transforms.ToTensor(),
                                       transforms.Normalize(mean=(0.5,), std=
(0.5,))])
# 获取图片样本
custom_datasets = datasets.ImageFolder(TRAIN_DIR, transform=train_transforms)
# 规定训练集测试集比例
train_size = int(len(custom_datasets) * 0.8)
test_size = len(custom_datasets) - train_size
train_datasets, test_datasets = torch.utils.data.random_split(custom_datasets,
[train_size, test_size])
# 生成训练集测试集迭代器
train_dataloader = torch.utils.data.DataLoader(train_datasets,
                                               batch_size=BATCH_SIZE,
                                               shuffle=True,
                                               num_workers=2)
test_dataloader = torch.utils.data.DataLoader(test_datasets,
                                               batch_size=BATCH_SIZE,
                                               shuffle=True,
```

```
                                                num_workers=2)
```

(2) 训练

epoch_num训练次数，model指定训练模型，device将模型和图片载入图片处理器，train_loader训练集迭代器，optimizer模型优化器，输入参数模型，定义初试学习率，lr_scheduler学习率调度器，使用它就不必像任务一中自己调整学习率了

```
def train(epoch_num, _model, _device, _train_loader, _optimizer, _lr_scheduler):
    _model.train()
    for epoch in range(epoch_num):
        for i, (images, labels) in enumerate(_train_loader):
            samples = images.to(_device)
            labels = labels.to(_device)
            output = _model(samples.reshape(-1, 3, 28, 28))
            loss = criterion(output, labels)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            if (i + 1) % 10 == 0:
                print("epoch:{}/{}, steps:{}, loss:{:.4f}".format(epoch + 1,
epoch_num, i + 1, loss.item()))
        _lr_scheduler.step()
```

(3) 测试

```
def test(_test_loader, _model, _device):
    _model.eval()
    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.reshape(-1, 3, 28, 28))
            loss += criterion(output, target).item()
            pred = output.data.max(1, keepdim=True)[1]
            correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    loss /= len(_test_loader.dataset)
    print('\nAverage loss: {:.4f}, Accuracy: {}/{} ({:.3f}%)\n'.format(
        loss, correct, len(_test_loader.dataset),
        100. * correct / len(_test_loader.dataset)))
```

训练结果

□□ - CNNet

```
D:\Desktop\CharacterScanner\venv\Scripts\python.exe D:\
Desktop\CharacterScanner\CNNet.py
epoch:1/1, steps:10, loss:0.0048
epoch:1/1, steps:20, loss:0.0629
epoch:1/1, steps:30, loss:0.0049
epoch:1/1, steps:40, loss:0.0086
epoch:1/1, steps:50, loss:0.0010

Average loss: 0.0000, Accuracy: 1487/1488 (99.933%)

epoch:1/2, steps:10, loss:0.0093
epoch:1/2, steps:20, loss:0.0034
epoch:1/2, steps:30, loss:0.0162
epoch:1/2, steps:40, loss:0.0085
epoch:1/2, steps:50, loss:0.0036
epoch:2/2, steps:10, loss:0.0007
epoch:2/2, steps:20, loss:0.0286
epoch:2/2, steps:30, loss:0.0337
epoch:2/2, steps:40, loss:0.0020
epoch:2/2, steps:50, loss:0.0151

Average loss: 0.0001, Accuracy: 1487/1488 (99.933%)

epoch:1/3, steps:10, loss:0.0168
epoch:1/3, steps:20, loss:0.0002
epoch:1/3, steps:30, loss:0.0028
epoch:1/3, steps:40, loss:0.0016
epoch:1/3, steps:50, loss:0.0001
epoch:2/3, steps:10, loss:0.0005
epoch:2/3, steps:20, loss:0.0011
epoch:2/3, steps:30, loss:0.0089
epoch:2/3, steps:40, loss:0.0061
epoch:2/3, steps:50, loss:0.0397
epoch:3/3, steps:10, loss:0.0055
epoch:3/3, steps:20, loss:0.0043
epoch:3/3, steps:30, loss:0.0025
epoch:3/3, steps:40, loss:0.0008
epoch:3/3, steps:50, loss:0.0010

Average loss: 0.0001, Accuracy: 1485/1488 (99.798%)
```

□□ 1 of 3

在训练了20次左右，最终准确率可达99.933%

困难点

在使用了pytorch框架后，实验的难度大大降低，唯一比较困难的点是连接好上一层输出和下一层输入，大小需保持一致，我的解决方法是在各层之间一点点调，调好上一层再去接下一层

对卷积神经网络的理解

我认为卷积神经网络再传统方向传播的基础上增加的特征值概念大大地提高了网络的速率和准确率，上上一个任务中，网络稍微复杂点，如中间超过三层，即需很长的运行时间，而且即使是复杂的层数也很难是准确率变得很高，而卷积网络一下子解决了这两个痛点