CNN手写汉字识别

CNN原理

CNN模型在处理高维数据时有较好的应用，其主要由下面几种元素组成：

1. 卷积层：卷积操作可以使用卷积核对输入数据进行滑动窗口处理，将每个窗口内的数据与卷积核进行点乘，得到卷积结果作核的大小和数量可以根据需求进行调整，不同的卷积核可以提取不同的特征，比如边缘、角点和纹理等。
2. 池化层：池化操作可以对特征图进行下采样处理，将每个池化窗口内的数据进行池化操作，比如取最大值或求平均值，得到降维后的特征图。池化操作可以减少特征图的尺寸和参数数量，从而加速计算，并提高模型的泛化性能。
3. 激活函数：卷积和池化操作之后，可以使用激活函数对特征图进行非线性变换，比如ReLU函数。
4. 全连接层：在最后一层特征图之后，可以添加全连接层来进行分类或回归等任务。全连接层将特征图展开为一维向量，再与权重矩阵进行矩阵乘法，得到输出类别或值。全连接层可以为不同的类别或值分配不同的权重，实现分类或回归的功能。

代码解释

```python
data_transforms = {
    'train': transforms.Compose([transforms.Grayscale(1), transforms.ToTensor(),]),
    'val': transforms.Compose([transforms.Grayscale(1), transforms.ToTensor(),])
}

dataset = torchvision.datasets.ImageFolder("train",transform=data_transforms["train"])

train_ratio = 0.9

train_dataset,val_dataset = data.random_split(dataset,[int(train_ratio*len(dataset)),len(dataset)-int(train_ratio*len(dataset))])

train_loader = data.DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = data.DataLoader(dataset=val_dataset,batch_size=BATCH_SIZE,shuffle=False)
```

从相对路径train读取数据，以9：1的比例划分为训练集和测试集，对训练集使用shuffle进行随机。

在加载数据的时候使用了pytorch框架下的torchvision.datasets.Imagefolder，它是PyTorch中用于处理图像数据集的一个类，可以用来读取指定目录下的图片数据集并进行预处理。它可以自动地将指定目录下的图像文件按照他们的目录名进行分类，即每个文件夹的名称对应着一个类别。这样，在加载图像数据集时，就会将每个图片的标签自动地设为其所在的类别。

class CNN

Python | 复制代码

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 16, 5, 1, 2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.5)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.5)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(32 * 7 * 7, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.5)
        )
        self.fc2 = nn.Linear(256, 12)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

cnn = CNN()
```

此部分利用pytorch框架建立自己的CNN模型，网络大体上由两个卷积层和两个全连接层组成，它们的实现都使用了nn.Sequential()方法将一系列不同的层组成一个以序列形式调用的块。

对于卷积层1，其主要实现为nn.Conv2d(1, 16, 5, 1, 2)：卷积层1包含一个二维卷积操作nn.Conv2d()，有1个输入通道，16个输出通道，窗口大小为 5x5，步长为1，padding为2；紧接着用nn.batchNorm2d实现一个归一化层，对输入的16个特征图进行归一化，使得不同特征图的均值和方差接近。这有助于加快训练收敛速度和提高模型泛化能力；在CNN中我们选用relu作为激励函数，它将负值截断为0，使得模型更稀疏且非线性，有助于提高模型的表达能力；然后是池化操作，nn.MaxPool2d(2)

将特征图的尺寸降低一半，有助于减小特征图的维数，提高模型的鲁棒性，并且能够识别比较显著的特征；最后使用nn.Dropout(0.5)进行Dropout 操作，随机将输入元素归零，提高模型的泛化能力，防止过拟合现象。

对于卷积层2，其实现的函数及顺序与卷积层1没有实质区别，只在输入输出的通道数上有所不同。

对于全连接层1，其主要实现为nn.Linear(32 * 7 * 7, 256)：全连接层1包含一个线性变换操作。它将卷积层2中的32个7x7（经过了两次池化操作，28*28的原图已经被压缩为7*7的特征图）的特征图展平成一个1维向量，并映射到一个256维的特征向量；接下来使用nn.BatchNorm1d(256)对全连接层1中的256维特征向量进行归一化操作；然后同样经过relu激活函数和随机丢弃。

对于全连接层2，它包含一个线性变换，将256维的特征向量映射到12个类别的输出。也就是最终我们要识别的分类数目。

forward函数展示了CNN网络向前传播的过程。先将输入的图像 x 通过卷积层1进行处理，得到卷积层1的输出；然后将卷积层1的输出 x 通过卷积层2进行处理，得到卷积层2的输出。x = x.view(x.size(0), -1)：将卷积层2的输出 x 展平，转换成一个一维向量，以便送入全连接层1中进行处理。x = self.fc1(x)：将展平后的特征向量 x 通过全连接层1进行处理；x = self.fc2(x)：将全连接层1的输出 x 通过全连接层2进行处理，得到最终的输出。

train Python 复制代码

```python
optimizer = torch.optim.Adam(cnn.parameters(), lr=LR)

loss_func = nn.CrossEntropyLoss()

for epoch in range(EPOCH):
    cnn.train()
    for i, x_y in tqdm(list(enumerate(train_loader))):
        batch_x = Variable(x_y[0])
        batch_y = Variable(x_y[1])
        output = cnn(batch_x)
        loss = loss_func(output, batch_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print('Train Epoch: {} Loss: {:.6f}'.format(epoch,loss.item()))
```

对于模型的自动优化仅需要一行函数即可实现，我们在这里定义使用Adam优化器，它是一种自适应优化算法，基本思想是根据每个参数的历史梯度值来自适应调整步长的大小，以便更快地收敛到最优值。这里我们将优化目标指定为卷积神经网络模型的参数，初始学习率为LR=0.001。

在计算loss时，由于是分类问题故使用CE作为损失函数。

开始训练模型，一共训练epoch=10轮，循环遍历训练集。将训练集数据的特征赋值给batch_x、训练集数据的标签赋值给batch_y；将batch_x作为输入，使用cnn模型进行前向传播，得到输出output；计算输出output与batch_y之间的交叉熵损失。利用optimizer.zero_grad(): 清除优化器中之前的梯度信息。loss.backward(): 反向传播，计算梯度并更新模型中的参数。optimizer.step(): 使用优化器对模型的参数进行更新。

test Python 复制代码

```python
cnn.eval()
val_loss = 0
correct = 0
with torch.no_grad():
    for data, target in val_loader:
        output = cnn(data)
        loss = loss_func(output, target)
        val_loss += loss.item() # sum up batch loss
        pred = output.argmax(dim=1, keepdim=True) # get the index of the max log-probability
        correct += pred.eq(target.view_as(pred)).sum().item()
        val_loss /= len(val_loader.dataset)

print('\nval set: Average loss: {:.4f}, Accuracy: {}/{} ({:.4f}%)\n'.format(
val_loss, correct, len(val_loader.dataset),
    100. * correct / len(val_loader.dataset)))
```

最后是性能测试，cnn.eval()使模型进入评估模式。在评估模式下，模型不会启用Batch Normalization和Dropout等操作。

with torch.no_grad()用于包围评估过程的代码块，它是一个上下文管理器，用于在计算图构建阶段禁用梯度计算的上下文环境中执行代码。在这个上下文环境中，所有的张量计算都不会被记录到计算图中，这样能够节省显存并加快计算速度。

防过拟合操作

batch_normalization

每层神经元的输入都可能因为激活函数和参数的非线性变换而产生高斯分布差异很大的数据，这样就会造成数据分布的偏移和不稳定，从而导致神经网络的收敛速度变缓甚至停滞，从而影响其泛化能力。

Batch normalization的主要作用是对每个mini-batch数据进行归一化处理，使得每个特征的数据均值为0，方差为1，从而提高深度神经网络的训练效果。

drop_out

Dropout的原理是在每次训练过程中，以一定的概率随机的将神经元的输出置为0，这样可以强制网络的每个神经元都不会过度依赖于其他神经元，使得网络不仅仅是学习到了所有可能的输入特征组合的信息，同时也学习到了这些特征组合的不同子集之间的关系。

实验结果

1. 在不进行normalization和dropout时，识别的精确度为：729/744

```
val set: Average loss: 0.0001, Accuracy: 729/744 (98%)
```

2. 在单独进行normalization或dropout时，识别的精确度均为：731/744

```
val set: Average loss: 0.0002, Accuracy: 731/744 (98%)
```

3. 两种防过拟合方法都使用，池化采用平均池化时，识别的精确度为：736/744

```
val set: Average loss: 0.0001, Accuracy: 736/744 (98.9247%)
```

4. 两种防过拟合方法都是用，池化采用最大池化时，识别的精确度为：737/744

```
val set: Average loss: 0.0001, Accuracy: 737/744 (99.0591%)
```

实验思考

- 最大池化是在池化区域内选取最大的数值作为该区域的输出，即将该区域内的所有数值中的最大值返回作为该区域的特征值，通常用来查找特征中的最突出部分，而忽略掉其他部分；平均池化是在池化区域内所有数值的平均值作为该区域的输出，通常用于减少特征图的大小，而且不像最大池化那样容易引入噪声，但是它无法突出特定区域的特征。通过本实验的结果，可以推测汉字识别任务对于突出局部特征的需求更大一些。
- 与卷积核大小为3相比，卷积核大小为5可以覆盖更大的感受野，从而对更大尺度的特征进行提取和处理，有助于更好地表示图像中的局部结构和全局信息。