Table of Contents:

今日工作

今天主要对昨天写的数字识别模型进行了完善，包括以下内容：

获得模型训练数据

手写了0~9十个数字，拍照获得十张图片，编写对应的txt文件，标记每张图片对应的数字。虽然目前数量集包含的数据很少，只有十张图片，可以直接在代码中创建一个列表 `[0,1,2,3,4,5,6,7,8,9]` 与每张图片对应，但是使用文件标记图片对应数字的方法方便以后大数据集的使用。

编写数据集类HandwrittenDigitDataset

关键函数有两个 `__getitem__` 和 `__len__` 。 `__getitem__` 函数要返回图片矩阵的 `tensor` 对象 `image` 和预测结果的 `tensor` 对象 `target` 。

对于图片矩阵，注意我们将其转化为 `RGB` ，保证其为三通道。

对于预测结果，它是图片对应数字的'tensor'对象。

```
obj_id = self.ids[idx]  # ids是一个列表，ids[idx]表示第idx张图片对应的数字
target = torch.as_tensor(obj_id)
```

编写模型的训练函数

训练函数根据之前学习的PyTorch官网教程改写，简化了许多内容。没有预热过程，只有前向传播，计算损失，反向传播，打印信息。

遇到的困难

通过DataLoader获取数据

由于大部分代码直接copy了之前教程中的代码，在其代码中，自定义了 `collate_fn` ，这是由于其图像在 `__getitem__` 并没有进行统一 `size` 的操作，不同图像矩阵的 `size` 不同，因此不能进行 `stack` 操作，需要自定义方法，让一个batch中的图片不再被stack操作，可以存储在一个 `list` 中。

但是其自定义的 `collate_fn` 返回的是 `tuple` 类型的数据，而他的模型中应该有处理这个 `tuple` 对象的方法，将其转化为 `tensor` 对象，输入网络。

在我们的网络中，并没有上述条件，因此只能接受 `tensor` 对象。而由于我们的模型很简单，并且限制了输入图片的 `size` ，因此在 `__getitem__` 函数中，有一个 `resize` 操作，因此不需要自定义 `collate_fn` 。

定义Target

最开始，我想的是像模型输出一样定义 `target` 为一个 `list` ，例如图片中数字为 `0` 的图片的 `target` 为:

```
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

但是这样的定义在计算损失时有一点复杂。因此转换思路，让 `target` 表示图片对应的数字。

模型输入数据维度的问题

对于模型中的一层

```
self.conv1 = nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5, stride=1, padding=0)
```

这层接受的图像矩阵 `shape` 为 `3 * 32 * 32` 。需要注意的是，模型期望接收的是一组训练数据，因此，即使只有一个图片矩阵，我们需要给这个矩阵增加一维，让其 `shape` 为 `1 * 3 * 32 * 32` ，才能够输入。

同时，这层权重的 `shape` 为 `6 * 3 * 3 * 3` ，为4维，输入也为4维的化是否能够更方便地计算呢?

训练效果

模型使用的损失函数为**交叉熵函数**，这个损失函数适合用于多分类问题。在500次训练中，每次训练都会使用所有图片，损失函数的值一直在2.30、2.29之间波动。这说明这个模型的识别效果非常差，我想主要有以下原因：

1、简单来看，这是一个非常简陋的模型，为了适应模型输入条件，将原本大于 `3000 * 3000` 像素的图片 `resize` 成 `32 * 32` 。同时，图片效果也不好，背景有波纹。

2、对各种超参数的设置没有分析。

后续任务

1、在今天的编程过程中，意识对于损失函数的知识有很大的空缺，以后需要学习。

2、了解PyTorch记录向量操作的知识，打算自定义损失函数时，对于程序如何记录我自定义的操作存在疑惑。