DATASETS & DATALOADERS

一、概要

数据集代码与模型训练代码分离，以提高可读性和模块化。PyTorch提供了两种数据原始类型：torch.utils.data.DataLoader和torch.utils.data.Dataset，它们允许使用预加载的数据集以及自己的数据。Dataset存储样本及其对应的标签，而DataLoader则在Dataset周围封装一个可迭代对象，以便于访问样本。

二、内容

一、 调用库中的数据

以下调用datasets包中的数据集来处理数据，使用以下参数加载FashionMNIST数据集： root是存储训练/测试数据的路径， train指定训练或测试数据集， download=True从互联网下载数据，如果root没有可用的数据。1 transform和target_transform指定特征和标签的变换。

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt


training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
```

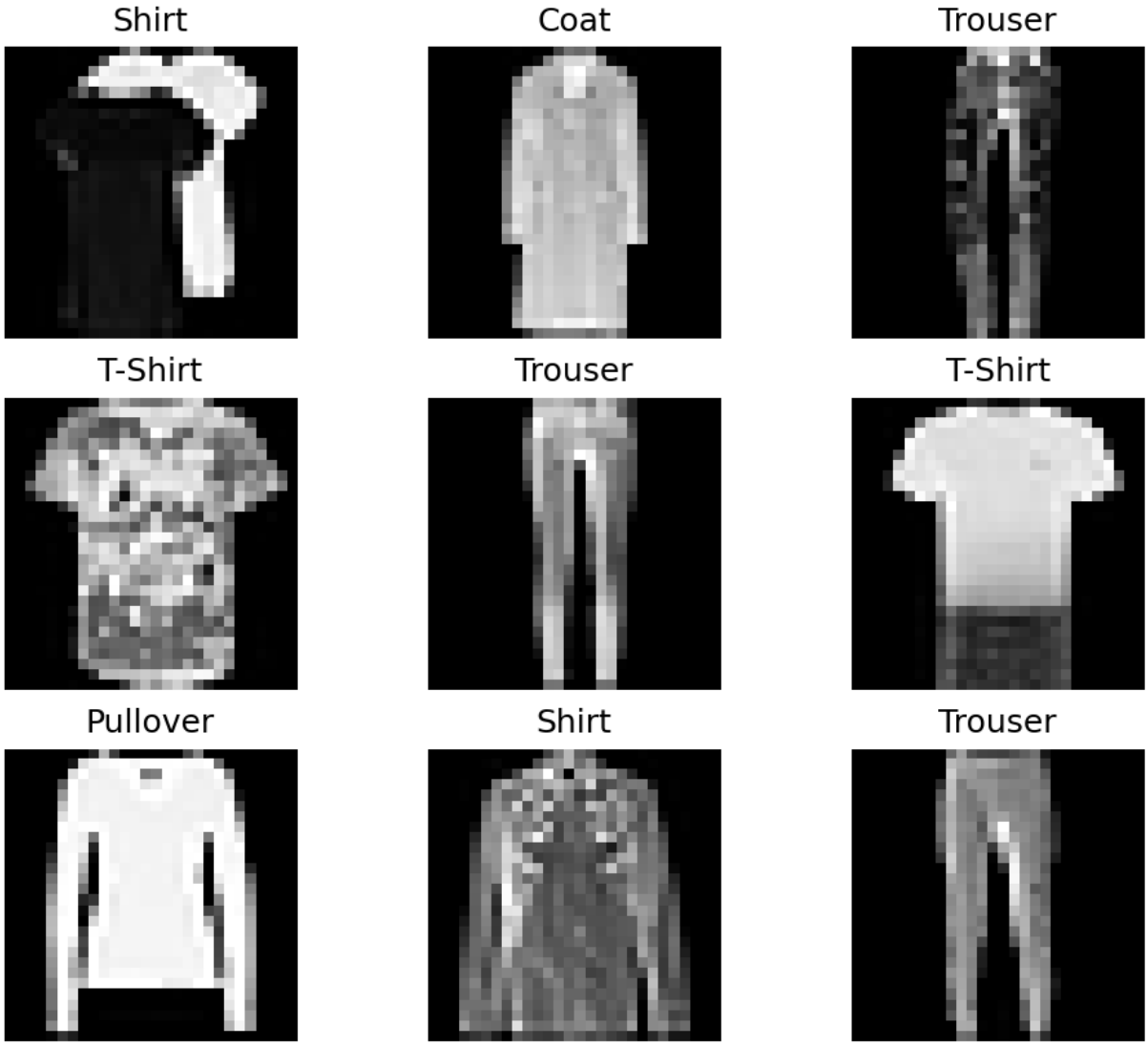
```
    download=True,
    transform=ToTensor()
)
```

显示数据

```
labels_map = {
    0: "T-Shirt",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle Boot",
}
figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(training_data), size=(1,)).item()
    img, label = training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```

这段代码使用了`matplotlib`库来绘制一个3x3的图像网格，显示来自训练数据集的9个随机样本。首先，它定义了一个`labels_map`字典，将类别标签（0-9）映射到对应的类别名称（如T-Shirt、Trouser等）。然后，它创建了一个8x8英寸的图像，并设置了列数和行数为3。

接下来，代码使用一个for循环来遍历每个网格单元。在每次迭代中，它首先使用`torch.randint`函数从训练数据集中随机选择一个样本索引。然后，它使用这个索引从训练数据集中获取对应的图像和标签。接着，它使用`figure.add_subplot`方法在图像上添加一个子图，并设置其标题为对应类别的名称（使用`labels_map`字典进行查找）。然后，它关闭坐标轴并使用`plt.imshow`方法显示图像（注意，由于图像是灰度图像，因此需要使用`squeeze`方法去除颜色通道，并使用`cmap="gray"`参数指定颜色映射）。最后，在循环结束后，代码使用`plt.show`方法显示整个图像。



二、创建自定义数据集

Dataset

自定义数据集类必须实现三个函数：init，len和getitem。

init函数初始化对象属性，并调用load_csv（自己写的函数）获得数据路径和标签。

len函数返回数据的数量

getitem函数从数据集中加载并返回给定索引inddx的样本。根据索引，它确定图像在磁盘上的位置。

```
class myDataset(Dataset):
    def __init__(self, *, root, train_data=True, transform=None, target_transform=None):
        self.root = root
        self.path = os.path.join(self.root, "train" if train_data is True else "test")
        # transform和target_transform是对数据和标签的预处理方法
        self.transform = transform
        self.target_transform = target_transform
```

```
            # 参过函数load_csv获得数据的路径和标签
            self.X, self.y = self.load_csv('save_path.csv')

      def __len__(self):
            return len(self.X)

      def __getitem__(self, index):
            x, label = self.X[index], self.y[index]

            x = Image.open(x)
            # 必须是tensor类型数据才能使用batch
            # x要先转成np才能转成tensor
            x = np.array(x)
            x = torch.tensor(x)
            label = torch.tensor(label)
            # 并在这里对数据预处理
            if self.transform is not None:
                  x = self.transform(x)
            if self.target_transform is not None:
                  label = self.target_transform(label)

            return x, label

      def load_csv(self, savepath):
            if not os.path.exists(os.path.join( self.path, savepath)):
                  data_X = []
                  data_y = []
                  # 如果该路径下的文件不存在就创建一个
                  # 因为每类数据都存在各自的文件夹下，所以要对每个数据生成唯一的路径方便索引
                  # 在数据集中，分为多个类别，每个类别的数据存在以类别命名的文件夹中,把路径集中起来
                  labels = os.listdir(self.path)
                  # 遍历每个类
                  for label in labels:
                        files = os.listdir(os.path.join(self.path, label))
                        # 遍历类中的每个文件
                        for file in files:
                              data_X.append(os.path.join(self.path, label, file))
                              data_y.append(label)

                  # 打乱顺序
                  key = np.array(range(len(data_X)))
                  np.random.shuffle(key)
                  data_X = np.array(data_X)
                  data_y = np.array(data_y)
                  # 变成np型式才不会报错
                  data_X = data_X[key]
                  data_y = data_y[key]
```

```
                # 保存成文件，方便直接从路径索引
                with open(os.path.join(self.path, savepath), mode='w', newline='') as f:
                        # mode='w'参数指定以写入模式打开文件，如果文件已经存在，则覆盖原有
内容。
                        # newline=''参数指定在写入文件时不添加额外的换行符。
                        writer = csv.writer(f)
                        for X, y in zip(data_X,data_y):
                                writer.writerow([X, y]) #
fashion_mnist_images\train\7\3151.png,7

        X = []
        y = []
        # 从保存文件中读数据
        with open(os.path.join(self.path, savepath)) as f:
                reader = csv.reader(f)
                for row in reader:
                        x, label = row
                        label = int(label)
                        X.append(x)
                        y.append(label)

        # 返回数据的路径和标签
        return X, y
```

其中的load_csv，在**第一次**调用时会生成一个'save_path.csv'，保存所有数据的路径和标签，**再次**调用次直接从'save_path.csv'查找数据的路径和标签。

上图是文件夹的结构，及'save_path.csv'存放的相对位置。

```
fashion_mnist_images\test\7\0472.png,7
fashion_mnist_images\test\9\0151.png,9
fashion_mnist_images\test\8\0635.png,8
fashion_mnist_images\test\4\0649.png,4
fashion_mnist_images\test\1\0934.png,1
fashion_mnist_images\test\3\0422.png,3
fashion_mnist_images\test\0\0629.png,0
fashion_mnist_images\test\8\0404.png,8
fashion_mnist_images\test\8\0778.png,8
fashion_mnist_images\test\5\0397.png,5
fashion_mnist_images\test\4\0420.png,4
fashion_mnist_images\test\2\0191.png,2
```

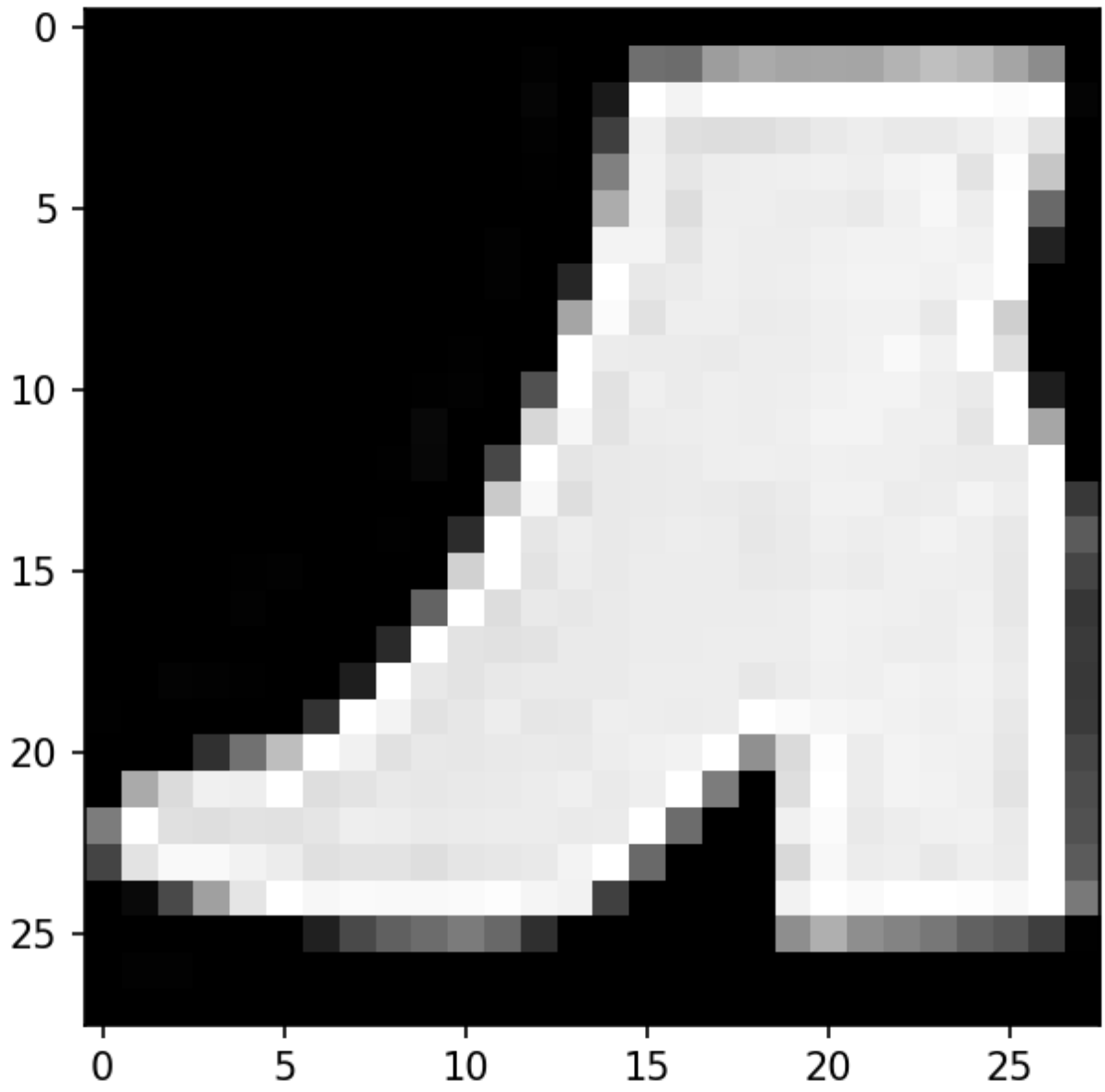
> 上图是'save_path.csv'内数据的存放结构。

DataLoader

数据集每次检索数据集的特征和标签一个样本。在训练模型时，通常希望以“小批量”传递样本，每个时期重新洗牌数据以减少模型过拟合，并使用Python的多进程来加速数据检索。DataLoader是一个可迭代的对象，它提供了一个简单的API来抽象这种复杂性。

```python
trainData = myDataset(root='fashion_mnist_images', train_data=False)
train_dataloader = DataLoader(trainData, batch_size=64, shuffle=True)
x, label = next(iter(train_dataloader))
print(x.shape)
print(label)
plt.imshow(x[0], cmap="gray")
plt.show()
```

```
torch.Size([64, 28, 28])
tensor([9, 8, 6, 1, 4, 9, 6, 8, 9, 7, 6, 3, 9, 3, 5, 6, 4, 4, 7, 7, 2, 9, 4, 8,
        2, 9, 3, 8, 5, 8, 7, 9, 3, 3, 8, 2, 2, 9, 1, 9, 9, 9, 7, 2, 8, 2, 5, 7,
        9, 9, 8, 1, 5, 7, 9, 4, 5, 4, 4, 7, 8, 3, 5, 3])
```


0
5
10
15
20
25
0
5
10
15
20
25