DATASETS & DATALOADERS

Code for processing data samples can get messy and hard to maintain; we ideally want our dataset code to be decoupled from our model training code for better readability and modularity. PyTorch provides two data primitives: torch.utils.data.DataLoader and torch.utils.data.Dataset that allow you to use pre-loaded datasets as well as your own data. Dataset stores the samples and their corresponding labels, and DataLoader wraps an iterable around the Dataset to enable easy access to the samples.

PyTorch domain libraries provide a number of pre-loaded datasets (such as FashionMNIST) that subclass torch.utils.data.Dataset and implement functions specific to the particular data. They can be used to prototype and benchmark your model. You can find them here: [Image Datasets](https://pytorch.org/vision/stable/datasets.html), [Text Datasets](https://pytorch.org/text/stable/datasets.html), and [Audio Datasets](https://pytorch.org/audio/stable/datasets.html)

Loading a Dataset

Here is an example of how to load the [Fashion-MNIST](https://research.zalando.com/project/fashion_mnist/fashion_mnist/) dataset from TorchVision. Fashion-MNIST is a dataset of Zalando’s article images consisting of 60,000 training examples and 10,000 test examples. Each example comprises a 28×28 grayscale image and an associated label from one of 10 classes.

We load the [FashionMNIST Dataset](https://pytorch.org/vision/stable/datasets.html#fashion-mnist) with the following parameters:

* root is the path where the train/test data is stored,
* train specifies training or test dataset,
* download=True downloads the data from the internet if it’s not available at root.
* transform and target\_transform specify the feature and label transformations

**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()**

**)**

Out:

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-idx3-ubyte.gz

Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-idx1-ubyte.gz

Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz

Extracting data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz

Extracting data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to data/FashionMNIST/raw

## Iterating and Visualizing the Dataset

We can index Datasets manually like a list: training\_data[index]. We use matplotlib to visualize some samples in our training data.

**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]**

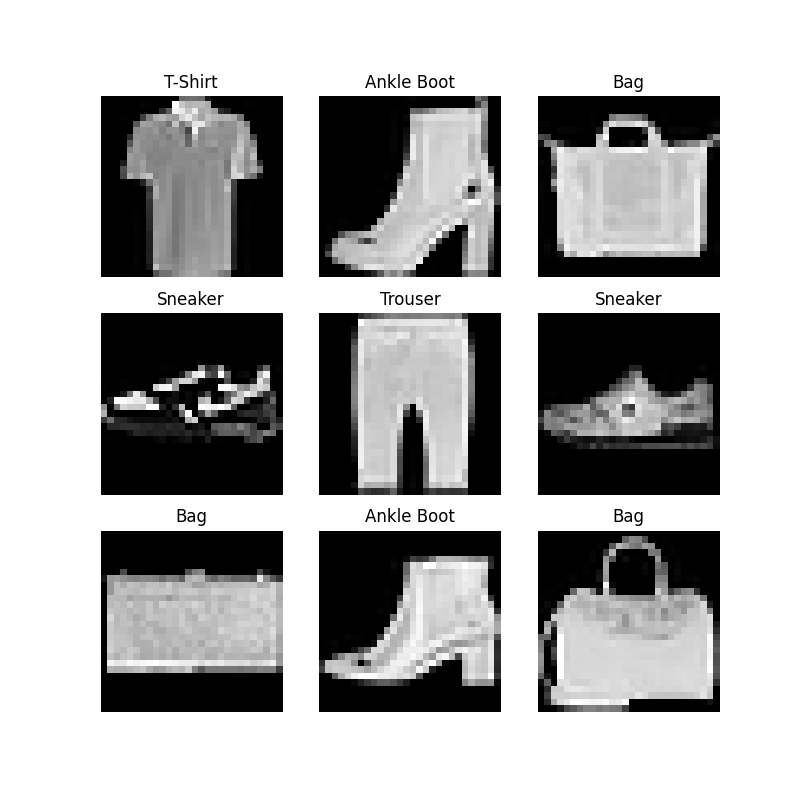
**figure.add\_subplot(rows,** **cols,** **i)**

**plt.title(labels\_map[label])**

**plt.axis(**"off"**)**

**plt.imshow(img.squeeze(),** **cmap=**"gray"**)**

**plt.show()**

[](https://pytorch.org/tutorials/_images/sphx_glr_data_tutorial_001.png)

## Creating a Custom Dataset for your files

A custom Dataset class must implement three functions: \_\_init\_\_, \_\_len\_\_, and \_\_getitem\_\_. Take a look at this implementation; the FashionMNIST images are stored in a directory img\_dir, and their labels are stored separately in a CSV file annotations\_file.

In the next sections, we’ll break down what’s happening in each of these functions.

**import** os

**import** pandas **as** pd

**from** torchvision.io **import** **read\_image**

**class** **CustomImageDataset(Dataset):**

**def** \_\_init\_\_**(**self**,** **annotations\_file,** **img\_dir,** **transform=None,** **target\_transform=None):**

self**.img\_labels** **=** **pd.read\_csv(annotations\_file)**

self**.img\_dir** **=** **img\_dir**

self**.transform** **=** **transform**

self**.target\_transform** **=** **target\_transform**

**def** \_\_len\_\_**(**self**):**

**return** len**(**self**.img\_labels)**

**def** \_\_getitem\_\_**(**self**,** **idx):**

**img\_path** **=** **os.path.join(**self**.img\_dir,** self**.img\_labels.iloc[idx,** 0**])**

**image** **=** **read\_image(img\_path)**

**label** **=** self**.img\_labels.iloc[idx,** 1**]**

**if** self**.transform:**

**image** **=** self**.transform(image)**

**if** self**.target\_transform:**

**label** **=** self**.target\_transform(label)**

**return** **image,** **label**

### \_\_init\_\_

The \_\_init\_\_ function is run once when instantiating the Dataset object. We initialize the directory containing the images, the annotations file, and both transforms (covered in more detail in the next section).

The labels.csv file looks like:

**tshirt1.jpg,** 0

**tshirt2.jpg,** 0

**......**

**ankleboot999.jpg,** 9

**def** \_\_init\_\_**(**self**,** **annotations\_file,** **img\_dir,** **transform=None,** **target\_transform=None):**

self**.img\_labels** **=** **pd.read\_csv(annotations\_file,** **names=[**'file\_name'**,** 'label'**])**

self**.img\_dir** **=** **img\_dir**

self**.transform** **=** **transform**

self**.target\_transform** **=** **target\_transform**

### \_\_len\_\_

The \_\_len\_\_ function returns the number of samples in our dataset.

Example:

**def** \_\_len\_\_**(**self**):**

**return** len**(**self**.img\_labels)**

### \_\_getitem\_\_

The \_\_getitem\_\_ function loads and returns a sample from the dataset at the given index idx. Based on the index, it identifies the image’s location on disk, converts that to a tensor using read\_image, retrieves the corresponding label from the csv data in self.img\_labels, calls the transform functions on them (if applicable), and returns the tensor image and corresponding label in a tuple.

**def** \_\_getitem\_\_**(**self**,** **idx):**

**img\_path** **=** **os.path.join(**self**.img\_dir,** self**.img\_labels.iloc[idx,** 0**])**

**image** **=** **read\_image(img\_path)**

**label** **=** self**.img\_labels.iloc[idx,** 1**]**

**if** self**.transform:**

**image** **=** self**.transform(image)**

**if** self**.target\_transform:**

**label** **=** self**.target\_transform(label)**

**return** **image,** **label**

Preparing your data for training with DataLoaders

The Dataset retrieves our dataset’s features and labels one sample at a time. While training a model, we typically want to pass samples in “minibatches”, reshuffle the data at every epoch to reduce model overfitting, and use Python’s multiprocessing to speed up data retrieval.

DataLoader is an iterable that abstracts this complexity for us in an easy API.

**from** torch.utils.data **import** **DataLoader**

**train\_dataloader** **=** **DataLoader(training\_data,** **batch\_size=**64**,** **shuffle=True)**

**test\_dataloader** **=** **DataLoader(test\_data,** **batch\_size=**64**,** **shuffle=True)**

Iterate through the DataLoader

We have loaded that dataset into the DataLoader and can iterate through the dataset as needed. Each iteration below returns a batch of train\_features and train\_labels (containing batch\_size=64 features and labels respectively). Because we specified shuffle=True, after we iterate over all batches the data is shuffled (for finer-grained control over the data loading order, take a look at [Samplers](https://pytorch.org/docs/stable/data.html#data-loading-order-and-sampler)).

*# Display image and label.*

**train\_features,** **train\_labels** **=** next**(**iter**(train\_dataloader))**

print**(**f"Feature batch shape: {**train\_features.size()**}"**)**

print**(**f"Labels batch shape: {**train\_labels.size()**}"**)**

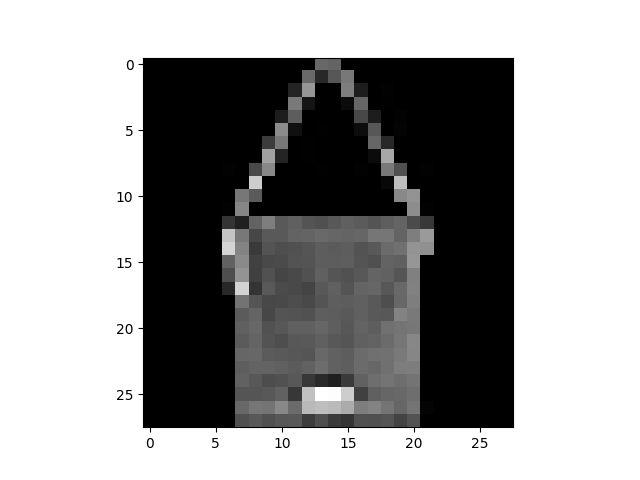
**img** **=** **train\_features[**0**].squeeze()**

**label** **=** **train\_labels[**0**]**

**plt.imshow(img,** **cmap=**"gray"**)**

**plt.show()**

print**(**f"Label: {**label**}"**)**



Out:

Feature batch shape: torch.Size([64, 1, 28, 28])

Labels batch shape: torch.Size([64])

Label: 8

Further Reading

* [torch.utils.data API](https://pytorch.org/docs/stable/data.html)

**Total running time of the script:** ( 0 minutes 5.523 seconds)