PyTorch
- building, manipulating, and fetching datasets from torchvision.datasets libray

Dr. Huiping Cao

Work with data - PyTorch

- Manipulating tensors
- Organizing data into formats that we can iterate over during training
- PyTorch has two primitives to work with data:
  - torch.utils.data.DataLoader and torch.utils.data.Dataset.
  - Dataset stores the samples and their corresponding labels,
  - DataLoader wraps an iterable around the Dataset.

Creating a dataset from files on your local storage disk

- We will build a dataset from image files stored on disk. There is an image folder associated with the **online content of this chapter**.

Creating a dataset - Step 1

- Step 1: Use the **pathlib library** to generate a list of image files:

```
from google.colab import drive
import pathlib

drive.mount('/content/drive’)

imgdir_path =
pathlib.Path('/content/drive/MyDrive/ColabNotebooks/data
/mlbook_images')
file_list = sorted([str(path) for path in
             imgdir_path.glob('*.jpg')])
print(file_list)
```

Output:

```
['/content/drive/MyDrive/ColabNotebooks/data/mlbo
ok_images/cat-01.jpg',
'/content/drive/MyDrive/ColabNotebooks/data/mlbo
ok_images/cat-02.jpg',
'/content/drive/MyDrive/ColabNotebooks/data/mlbo
ok_images/cat-03.jpg',
'/content/drive/MyDrive/ColabNotebooks/data/mlbo
ok_images/dog-01.jpg',
'/content/drive/MyDrive/ColabNotebooks/data/mlbo
ok_images/dog-02.jpg',
'/content/drive/MyDrive/ColabNotebooks/data/mlbo
ok_images/dog-03.jpg']
```

Creating a dataset

- Step 1.2 (optional)s: Visualize these image examples using Matplotlib:

```
import matplotlib.pyplot as plt
import os
from PIL import Image

fig = plt.figure(figsize=(10, 5))
for i, file in enumerate(file_list):
        img = Image.open(file)
        print('Image shape:', np.array(img).shape)
        #commands to plot images
plt.show()
```

Output:

```
Image shape: (900, 1200, 3)
Image shape: (900, 1200, 3)
Image shape: (900, 742, 3)
Image shape: (800, 1200, 3)
Image shape: (800, 1200, 3)
Image shape: (900, 1200, 3)
```

Create a dataset – step 2

- Step 2: set up labels of these files based on their file names (assigning label 1 to dogs and label 0 to cats.

```
labels = [1 if 'dog' in os.path.basename(file)
          else 0
          for file in file_list]
print(labesls)
```

Output:

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

Create a dataset – step 3

- Step 3: create a Dataset object using the list of filenames and the list of their labels

```
class ImageDataset(Dataset):
    def __init__(self, file_list, labels):
        self.file_list = file_list
        self.labels = labels

    def __getitem__(self, index):
        file = self.file_list[index]
        label = self.labels[index]
        return file, label

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

image_dataset = ImageDataset(file_list, labels)
for file, label in image_dataset:
        print(file, label)
```

Output:

```
/content/drive/MyDrive/ColabNotebooks/data/mlboo
k_images/cat-01.jpg 0
/content/drive/MyDrive/ColabNotebooks/data/mlboo
k_images/cat-02.jpg 0
/content/drive/MyDrive/ColabNotebooks/data/mlboo
k_images/cat-03.jpg 0
/content/drive/MyDrive/ColabNotebooks/data/mlboo
k_images/dog-01.jpg 1
/content/drive/MyDrive/ColabNotebooks/data/mlboo
k_images/dog-02.jpg 1
/content/drive/MyDrive/ColabNotebooks/data/mlboo
k_images/dog-03.jpg 1
```

Create a dataset – step 4

- Step 4: apply transformation to this dataset
  - Load the image content from its file path
  - Decode the raw content
  - Resize it to a desired size (e/g., 80 *120)
- Step 4.1: Define the transform using **torchvision.transforms** module to resize the images and convert the loaded pixels into tensors

```
img_height, img_width = 80, 120
transform = transforms.Compose([
                transforms.ToTensor(),
                transforms.Resize((img_height, img_width)),
])
```

Create a dataset – step 4.2: update the ImageDataset class with the transform

```
class ImageDataset(Dataset):
    def __init__(self, file_list, labels):
        self.file_list = file_list
        self.labels = labels

    def __getitem__(self, index):
        file = self.file_list[index]
        label = self.labels[index]
        return file, label

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

image_dataset = ImageDataset(file_list, labels)
for file, label in image_dataset:
        print(file, label)
```

```
class ImageDataset(Dataset):
    def __init__(self, file_list, labels, transform=None):
        self.file_list = file_list
        self.labels = labels
        self.transform = transform

     def __getitem__(self, index):
        img = Image.open(self.file_list[index])
        if self.transform is not None:
                img = self.transform(img)
                label = self.labels[index]
        return img, label

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

image_dataset = ImageDataset(file_list, labels, transform)
```

Create a dataset – step 4

- If visualize these transformed image examples
- The new images are resized.

```
for i, example in enumerate(image_dataset):
        img=example[0]
        labeli=example[1]
print('i:', i,', Image shape:', np.array(img).shape,', label=',labeli)
```

```
Output:
i: 0 , Image shape: (3, 80, 120) , label= 0
i: 1 , Image shape: (3, 80, 120) , label= 0
i: 2 , Image shape: (3, 80, 120) , label= 0
i: 3 , Image shape: (3, 80, 120) , label= 1
i: 4 , Image shape: (3, 80, 120) , label= 1
i: 5 , Image shape: (3, 80, 120) , label= 1
```

Fetching available datasets from the torchvision.datasets library

- The **torchvision.datasets** library provides a nice collection of freely available image datasets for training or evaluating deep learning models.
- The **torchtext.datasets** library provides datasets for natural language processing.
- The torchvision datasets (https://pytorch.org/vision/stable/datasets.html) are nicely formatted and come with informative descriptions, including
  - the format of features and labels
  - their type and dimensionality
  - the link to the original source of the dataset.
  - these datasets are all subclasses of **torch.utils.data.Dataset**

Make sure torchvision library is installed

- If you haven’t already installed torchvision together with PyTorch earlier, you need to install the torchvision library.

```
$ conda search torchvision

Or
$ conda install torchvision
```

Output:

```
Loading channels: done
# Name                    Version           Build  Channel
torchvision                 0.4.2 cpu_py37h1ea6fa9_0  pkgs/main
torchvision                 0.8.2 cpu_py37hde629fd_0  pkgs/main
torchvision                 0.8.2 cpu_py38hde629fd_0  pkgs/main
torchvision                 0.8.2 cpu_py39hde629fd_0  pkgs/main
```

Fetching available datasets example

- MNIST dataset
- CelebA dataset (http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html)
  - torchvision.datasets.CelebA
  - **CelebFaces** Attributes Dataset (CelebA) is a large-scale face attributes dataset with more than **200K** celebrity images, each with 40 attribute annotations.
  - The images in this dataset cover large pose variations and background clutter.
  - CelebA has large diversities, large quantities, and rich annotations, including
    - 10,177 number of **identities**,
    - 202,599 number of **face images**, and
    - 5 **landmark** locations,
    - 40 binary **attributes annotations** per image.

MNIST dataset

- torchvision.datasets.MNIST (https://pytorch.org/vision/stable/datasets.html#mnist)
- The database has **two partitions**, 'train' and 'test'. We need to select a specific subset to load.
- The images are stored in **PIL.Image** format.
- There are **10 classes** for the target, from 0 to 9.

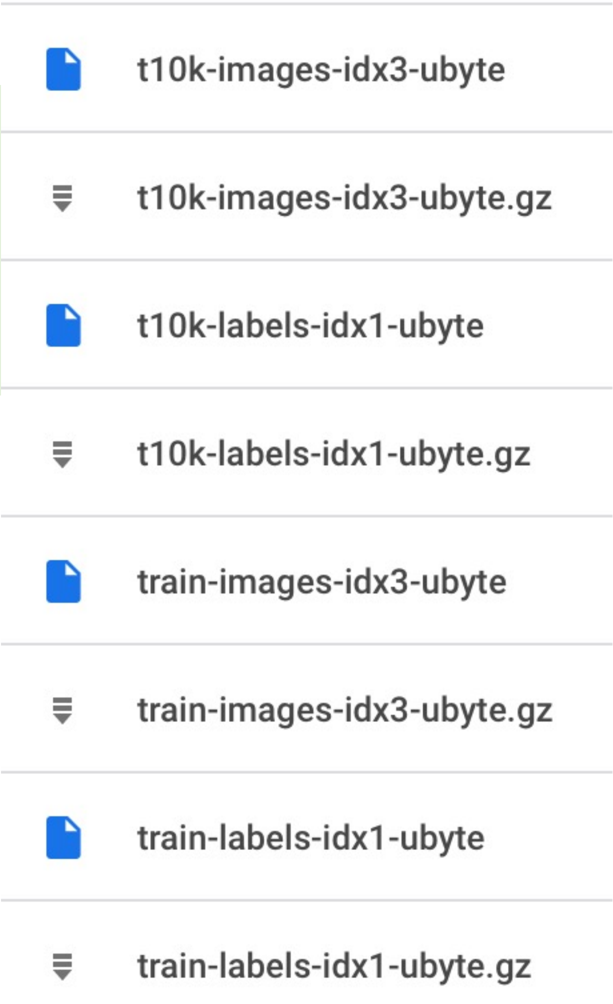
Download MNIST dataset

```
import torchvision

image_path = '/content/drive/MyDrive/ColabNotebooks/data/'

mnist_dataset = torchvision.datasets.MNIST(image_path, 'train', download=True)
```

- In the “image_path”, a folder “MNIST/raw” is created.
- In this folder, you will see 8 files



t10k-images-idx3-ubyte

t10k-images-idx3-ubyte.gz

t10k-labels-idx1-ubyte

t10k-labels-idx1-ubyte.gz

train-images-idx3-ubyte

train-images-idx3-ubyte.gz

train-labels-idx1-ubyte

train-labels-idx1-ubyte.gz

Operate on MNIST dataset

```
from itertools import islice

mnist_dataset = torchvision.datasets.MNIST(image_path,
'train', download=False)
assert isinstance(mnist_dataset, torch.utils.data.Dataset)
example = next(iter(mnist_dataset))

print(example)

for i, (image, label) in islice(enumerate(mnist_dataset), 10):
            print('i=',i, np.array(image).shape, 'label=',label)
```

Output:

```
(<PIL.Image.Image image mode=L
size=28x28 at 0x7F842B474FD0>, 5)

i= 0 (28, 28) label= 5
i= 1 (28, 28) label= 0
i= 2 (28, 28) label= 4
i= 3 (28, 28) label= 1
i= 4 (28, 28) label= 9
i= 5 (28, 28) label= 2
i= 6 (28, 28) label= 1
i= 7 (28, 28) label= 3
i= 8 (28, 28) label= 1
i= 9 (28, 28) label= 4
```

- **assert**() function checks if the object is of the torch.utils.data.Dataset class.
- **itertools** — Functions creating iterators for efficient looping
- **islice**(), arguments: seq, [start,] stop [, step]; meaning: get elements from seq[start:stop:step]

CelebA dataset

- It has three subsets, 'train', 'valid', and 'test’.
  - We can select a specific subset or load all of them with the split parameter.
- The images are stored in PIL.Image format.
  - We can **obtain a transformed version** using a custom transform function, such as transforms.ToTensor and transforms.Resize.
- There are **different types of targets** we can use, including 'attributes', 'identity', and 'landmarks’.
  - 'attributes' is 40 facial attributes for the person in the image, such as facial expression, makeup, hair properties, and so on;
  - 'identity' is the person ID for an image;
  - and 'landmarks' refers to the dictionary of extracted facial points, such as the position of the eyes, nose, and so on.

CelebA dataset

```
import torchvision

image_path = '/content/drive/MyDrive/ColabNotebooks/data/'
celeba_dataset = torchvision.datasets.CelebA(
            image_path, split='train', target_type='attr', download=True)
```

- You may run into a BadZipFile: File is not a zip file error, etc.
- Reason: Google Drive has a daily maximum quota that is exceeded by the CelebA files.
- Workaround:
  - manually download the files from the source: http://mmlab.ie.cuhk.edu.hk/projects/ CelebA.html.
  - In the downloaded folder, celeba/, you can unzip the img_align_celeba.zip file.
  - image_path is the root of the downloaded folder, celeba/.
- If you have already downloaded the files once, you can simply set download=False.

References

- Chapter 12: By Sebastian Raschka , Yuxi (Hayden) Liu , Vahid Mirjalili: Machine Learning with PyTorch and Scikit-Learn, Packt.
- https://pytorch.org/tutorials/
  - Most materials of this lecture are from https://pytorch.org/tutorials/beginner/basics/intro.html
- https://www.youtube.com/watch?v=c36lUUr864M&t=9613s (4.5 hours video)
- https://github.com/yunjey/pytorch-tutorial
- https://github.com/MorvanZhou/PyTorch-Tutorial