Program 9

PyTorch Custom Dataset and DataLoader: Learn to create a custom dataset class in PyTorch. Implement a DataLoader for efficient data handling and batching during training.

Explanation:

What is a DataLoader in PyTorch?

The **DataLoader** in PyTorch is a powerful utility that **loads data efficiently in batches**, handles **shuffling**, and supports **parallel processing** using multiple worker threads. It is used in conjunction with a Dataset to streamline the data feeding process during training or evaluation.

DataLoader is a smart loop that fetches data for your model in manageable chunks (batches), optionally shuffles it, and does this in parallel for speed.

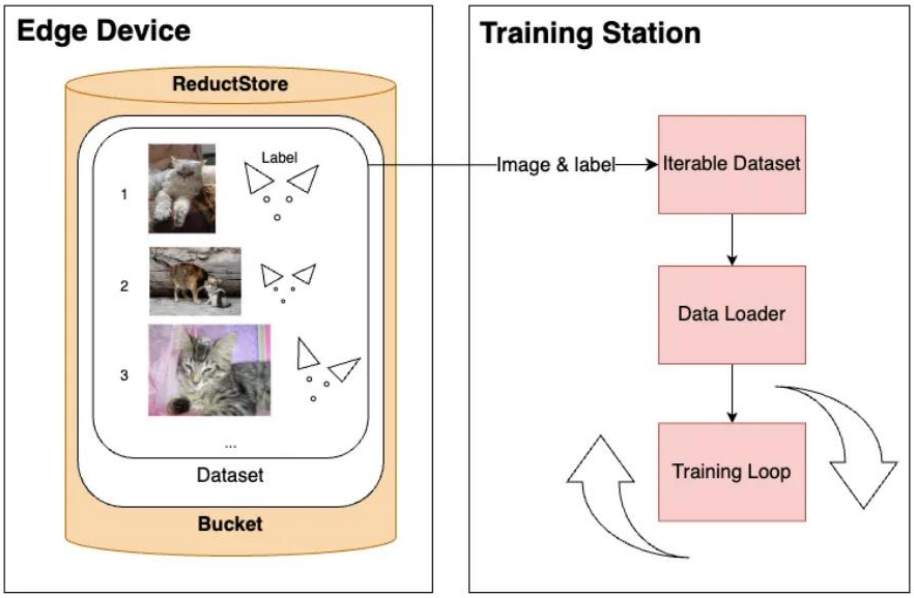

Image Source: https://www.reduct.store/blog/ai/datastreaming/pytorch/implement-database-data-streaming-pytorch

Key Features of DataLoader

Feature	Description
`batch_size`	How many samples per batch to load (e.g., `batch_size=32`)
`shuffle`	Whether to shuffle the data each epoch (`True` for training)
`num_workers`	Number of subprocesses to use for data loading (`>0` for faster loading)
`drop_last`	Drop the last incomplete batch if dataset size is not divisible by batch
`pin_memory`	Speeds up transfer of data to GPU

Example

```
from torch.utils.data import DataLoader

# Assuming `dataset` is an object of a class that inherits from
torch.utils.data.Dataset

dataloader = DataLoader(dataset, batch_size=4, shuffle=True, num_workers=2)

for batch in dataloader:
    images, labels = batch
    print(images.shape, labels)
```

Why use DataLoader?

- Efficient memory use (no need to load full dataset at once)
- Faster training (with `num_workers` > 0)
- Easy batching and shuffling
- Works seamlessly with any custom or built-in `Dataset`

Create a Custom Dataset
Assume we have a dataset in CSV format, where each row contains a file path to an image and its label.https://www.kaggle.com/datasets/samuelcortinhas/cats-and-dogs-image-classification?resource=download

Program code:

```
import torch
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from PIL import Image
import os
from torchvision import transforms

# Custom Dataset class
class CustomImageDataset(Dataset):
    def __init__(self, csv_file, root_dir, transform=None):
        self.data_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.root_dir, self.data_frame.iloc[idx, 0])
        image = Image.open(img_path).convert('RGB')
        label = int(self.data_frame.iloc[idx, 1])

        if self.transform:
            image = self.transform(image)

        return image, label

# Transformations
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
])

# Dataset and DataLoader
dataset = CustomImageDataset(csv_file='data.csv', root_dir='images', transform=transform)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True, num_workers=2)

# Iterate through DataLoader
for images, labels in dataloader:
    print(images.shape)
    print(labels)
```

Code explanation:

The goal here is to load a custom dataset (e.g., images and labels from a CSV file), preprocess it, and feed it in batches to a deep learning model using PyTorch's Dataset and DataLoader classes.

Step 1: Import Required Libraries

```
import torch
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from PIL import Image
import os
```

Explanation:

- `torch`: Core PyTorch library.
- `Dataset`: Base class to define a custom dataset.
- `DataLoader`: Wraps the dataset and provides batching, shuffling, and parallel loading.
- `pandas`: To read the CSV file where data (image paths and labels) is stored.
- `PIL.Image`: To load images.
- `os`: Helps in constructing the file path.

Step 2: Create a Custom Dataset Class

```
class CustomImageDataset(Dataset):
    def __init__(self, csv_file, root_dir, transform=None):
        self.data_frame = pd.read_csv(csv_file)      # Load CSV into a
DataFrame
        self.root_dir = root_dir                     # Directory where images
are stored
        self.transform = transform                   # Optional transforms to
apply

    def __len__(self):
        return len(self.data_frame)                  # Number of samples

    def __getitem__(self, idx):
        img_path = os.path.join(self.root_dir, self.data_frame.iloc[idx, 0])
        image = Image.open(img_path).convert('RGB')  # Load image
        label = int(self.data_frame.iloc[idx, 1])    # Get corresponding
label

        if self.transform:
            image = self.transform(image)            # Apply transformation

        return image, label
```

Explanation:

You are creating a **custom dataset** by extending `torch.utils.data.Dataset`.

- `__init__`: Reads the CSV file and stores the path to the image folder. `transform` lets you specify image preprocessing.
- `__len__`: Tells PyTorch how many samples are in the dataset.
- `__getitem__`: This method is used to fetch a single sample (`image`, `label`) by index:
    - `img_path` combines the folder path and the image file name.
    - `Image.open(...).convert('RGB')` ensures the image has 3 channels (RGB).
    - `self.transform(image)` applies any preprocessing you specify.

Step 3: Define Image Transformations (Optional but Recommended)

```
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize((128, 128)),  # Resize all images to 128x128
    transforms.ToTensor(),          # Convert PIL image to PyTorch tensor (0-
1 range)
])
```

Explanation:

- `Resize`: Ensures all images are the same size, which is required for training.
- `ToTensor`: Converts images to PyTorch tensors and normalizes pixel values from [0, 255] to [0.0, 1.0].

You can also include more transformations like `RandomCrop`, `Normalize`, `RandomFlip`, etc., for **data augmentation**.

Step 4: Instantiate the Dataset and DataLoader

```
dataset = CustomImageDataset(csv_file='data.csv', root_dir='images',
transform=transform)

dataloader = DataLoader(dataset, batch_size=4, shuffle=True, num_workers=2)
```

Explanation:

`CustomImageDataset(...)`

- Loads your dataset from the CSV and image folder with the specified transforms.

`DataLoader(...)`

- **`batch_size=4`**: Combines 4 samples into one batch.

- **`shuffle=True`**: Shuffles the data at every epoch, improving generalization.
- **`num_workers=2`**: Uses 2 subprocesses to load data in parallel, making it faster.

Step 5: Iterate Over the DataLoader During Training

```
for images, labels in dataloader:
    print(images.shape)  # Shape of the batch: [batch_size, channels, height,
width]
    print(labels)
    # Training code here
```

Explanation:

- When you loop over `dataloader`, it automatically:
  - Fetches batches of data.
  - Applies transformations.
  - Shuffles if specified.
- `images.shape` gives you something like `[4, 3, 128, 128]`, where:
  - 4 is the batch size
  - 3 is the number of channels (RGB)
  - 128x128 is the image resolution

This loop is typically used inside the training routine of your model.