Ứng dụng Trí tuệ nhân tạo trong Nuôi trồng thủy sản

NGUYỄN HẢI TRIỀU[1]

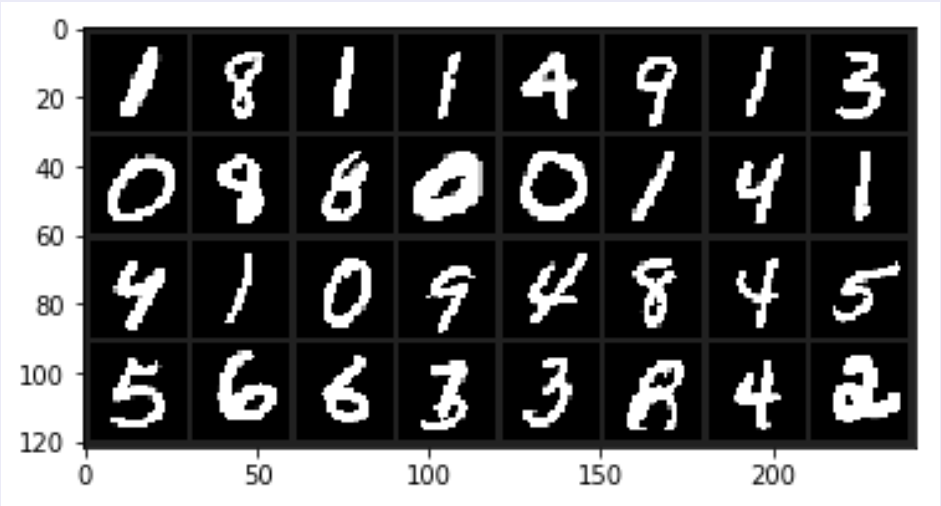
[1] Bộ môn Kỹ thuật phần mềm,

Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

Multilayer Perceptron cho MNIST Dataset I

Trong phần trước chúng ta đã huấn luyện Multilayer Perceptron trên tập dữ liệu XOR, trong phần này chúng ta sẽ huấn luyện trên tập dữ liệu MNIST. MNIST là tập dữ liệu chữ số viết tay từ 0 đến 9, mỗi hình là một ảnh đen trắng chứa một số được viết tay có kích thước là 28×28.



Chúng ta sẽ áp dụng MLPs cho bài toán phân lớp có 10 labels.

Multilayer Perceptron cho MNIST Dataset II

Tải MNIST dataset bằng thư viện PyTorch. Tập train có 60000 ảnh, tập test có 10000 ảnh.

```
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

train_dataset = datasets.MNIST(
  root="./mnist", train=True, transform=transforms.ToTensor(), download=True
)

test_dataset = datasets.MNIST(
  root="./mnist", train=False, transform=transforms.ToTensor()
)
```

Multilayer Perceptron cho MNIST Dataset III

Tạo tập validation từ tập training sử dụng *random_split* từ *torch.utils.data.dataset*

```
1  import torch
2  from torch.utils.data.dataset import random_split
3
4  torch.manual_seed(1)
5  train_dataset, val_dataset = random_split(train_dataset, lengths=[55000,
       5000])
6
7  train_loader = DataLoader(
8      dataset=train_dataset,
9      batch_size=64,
10     shuffle=True,
11 )
12
13 val_loader = DataLoader(
14     dataset=val_dataset,
15     batch_size=64,
16     shuffle=False,
```

Multilayer Perceptron cho MNIST Dataset IV

```
17  )
18
19  test_loader = DataLoader(
20      dataset=test_dataset,
21      batch_size=64,
22      shuffle=False,
23  )
```

Kiểm tra lại phân bố dữ liệu. Lưu ý: dữ liệu thường phải cân bằng giữa các lớp trong các bộ data.

```
1  from collections import Counter
2
3  train_counter = Counter()
4  for images, labels in train_loader:
5    train_counter.update(labels.tolist())
6
7  print("\nTraining label distribution:")
8  print(sorted(train_counter.items()))
```

Multilayer Perceptron cho MNIST Dataset V

```
9
10
11  val_counter = Counter()
12  for images, labels in val_loader:
13    val_counter.update(labels.tolist())
14
15  print("\nValidation label distribution:")
16  print(sorted(val_counter.items()))
17
18
19  test_counter = Counter()
20  for images, labels in test_loader:
21    test_counter.update(labels.tolist())
22
23  print("\nTest label distribution:")
24  print(sorted(test_counter.items()))
```

Multilayer Perceptron cho MNIST Dataset VI

Tính baseline cho bài toán phân loại, dùng để đánh giá hiệu quả thật sự của model trên tập testing. Nếu không dùng bất kì thuật toán phân loại nào thì chúng ta luôn dự đoán được 1 lớp có phân bố nhiều nhất trong tập test sẽ có tỉ lệ dự đoán đúng thấp nhất là kết quả của baseline.

```
majority_class = test_counter.most_common(1)[0]
print("Majority class:", majority_class[0])

baseline_acc = majority_class[1] / sum(test_counter.values())
print("Accuracy when always predicting the majority class:")
print(f"{baseline_acc:.2f} ({baseline_acc*100:.2f}%)")
```

Hiển thị dữ liệu từ training

Multilayer Perceptron cho MNIST Dataset VII

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import torchvision

for images, labels in train_loader:
  break

plt.figure(figsize=(8, 8))
plt.axis("off")
plt.title("Training images")
plt.imshow(np.transpose(torchvision.utils.make_grid(
  images[:64],
  padding=1,
  pad_value=1.0,
  normalize=True),
  (1, 2, 0)))
## np.transpose hoán vị các trục của tensor đầu vào. Trong trường hợp này,
    các trục được hoán vị từ (0, 1, 2) sang (1, 2, 0):
# Trục 0 (batch size hoặc số lượng hình ảnh) được chuyển thành trục 2.
```

Multilayer Perceptron cho MNIST Dataset VIII

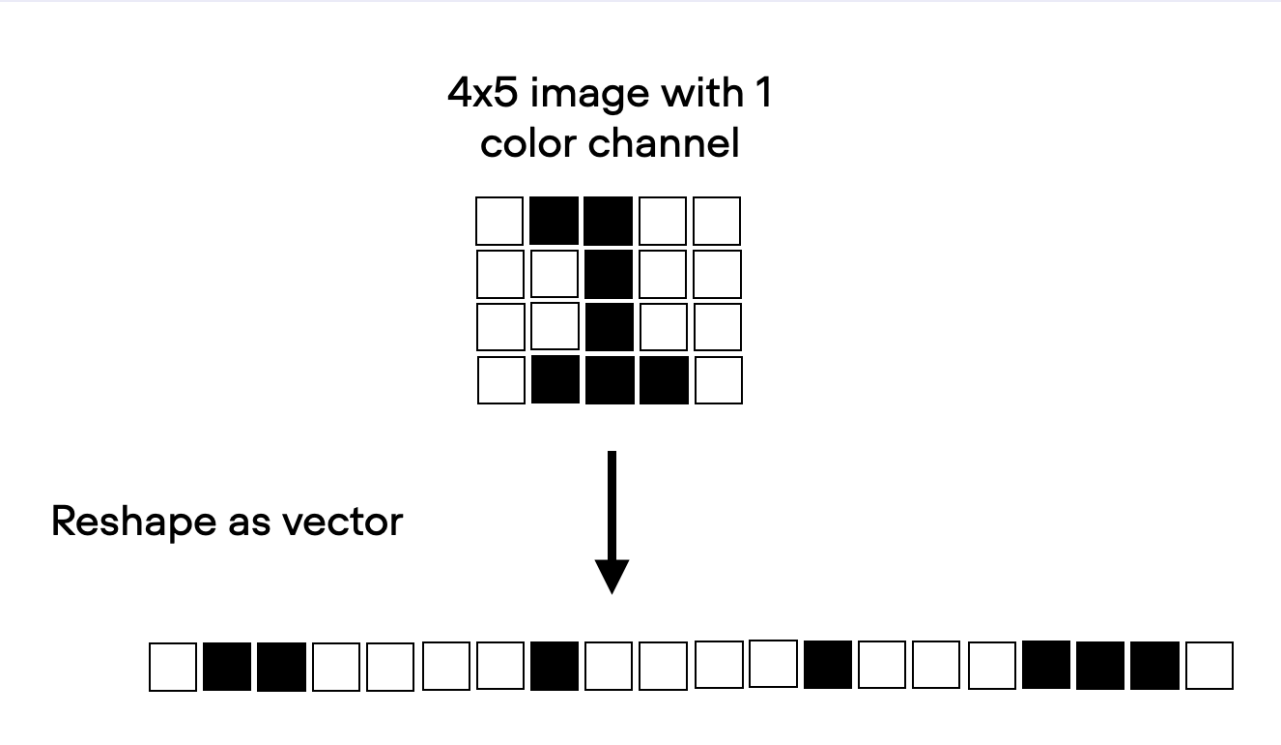
```
20  # Trục 1 (chiều cao của hình ảnh) giữ nguyên là trục 1.
21  # Trục 2 (chiều rộng của hình ảnh) giữ nguyên là trục 2.
22  # Trục 3 (số kênh màu, thường là 3 cho RGB) được chuyển thành trục 0.
23  # Mục đích của việc hoán vị này là đểlàm cho tensor phù hợp với định dạng mà
        plt.imshow yêu cầu. plt.imshow mong đợi dữ liệu đầu vào có dạng (height,
        width, channels).
24  plt.show()
```

Phân tích dữ liệu đầu vào để đưa vào mô hình

Khi sử dụng: *images.shape # batchsize, channel, height, width*, ta thu được: *torch.Size([64, 1, 28, 28])*. Trong đó, đầu vào là 64 bức ảnh 1 kênh màu (ảnh xám/trắng đen) có kích thước 28×28.

Multilayer Perceptron cho MNIST Dataset IX

Vậy để đưa vào model như dữ liệu của tập *XOR là từng vector*, chuyển đổi các ảnh đầu vào thành vector, rõ ràng với 64 ảnh ta sẽ thu được 64 vectors, mỗi vector có kích thước 784 features.



Multilayer Perceptron cho MNIST Dataset X

Sử dụng torch.flatten để chuyển ảnh thành vector

```
import torch
torch.flatten(images, start_dim=1).shape # batchsize, features
#torch.Size([64, 784])
```

Định nghĩa lớp MLPs trong PyTorch

```
import torch
class PyTorchMLP(torch.nn.Module):
    def __init__(self, num_features, num_classes):
        super().__init__()

        self.all_layers = torch.nn.Sequential(
            # 1st hidden layer
            torch.nn.Linear(num_features, 50),
            torch.nn.ReLU(),
            # 2nd hidden layer
            torch.nn.Linear(50, 25),
```

Multilayer Perceptron cho MNIST Dataset XI

```
12              torch.nn.ReLU(),
13              # output layer
14              torch.nn.Linear(25, num_classes),
15          )
16
17      def forward(self, x):
18          x = torch.flatten(x, start_dim=1)
19          logits = self.all_layers(x)
20          return logits
```

Định nghĩa quá trình huấn luyện

```
1  import torch.nn.functional as F
2
3  torch.manual_seed(1)
4  model = PyTorchMLP(num_features=784, num_classes=10)
5
6  optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
7
```

Multilayer Perceptron cho MNIST Dataset XII

```
8  num_epochs = 10
9
10 loss_list = []
11 train_acc_list, val_acc_list = [], []
12 for epoch in range(num_epochs):
13
14   model = model.train()
15   for batch_idx, (features, labels) in enumerate(train_loader):
16
17     logits = model(features)
18
19     loss = F.cross_entropy(logits, labels)
20
21     optimizer.zero_grad()
22     loss.backward()
23     optimizer.step()
24
25     if not batch_idx % 250:
26       ### LOGGING
27       print(
```

Multilayer Perceptron cho MNIST Dataset XIII

```
28          f"Epoch: {epoch+1:03d}/{num_epochs:03d}"
29          f" | Batch {batch_idx:03d}/{len(train_loader):03d}"
30          f" | Train Loss: {loss:.2f}"
31        )
32      loss_list.append(loss.item())
33
34    train_acc = compute_accuracy(model, train_loader)
35    val_acc = compute_accuracy(model, val_loader)
36    print(f"Train Acc {train_acc*100:.2f}% | Val Acc {val_acc*100:.2f}%")
37    train_acc_list.append(train_acc)
38    val_acc_list.append(val_acc)
```

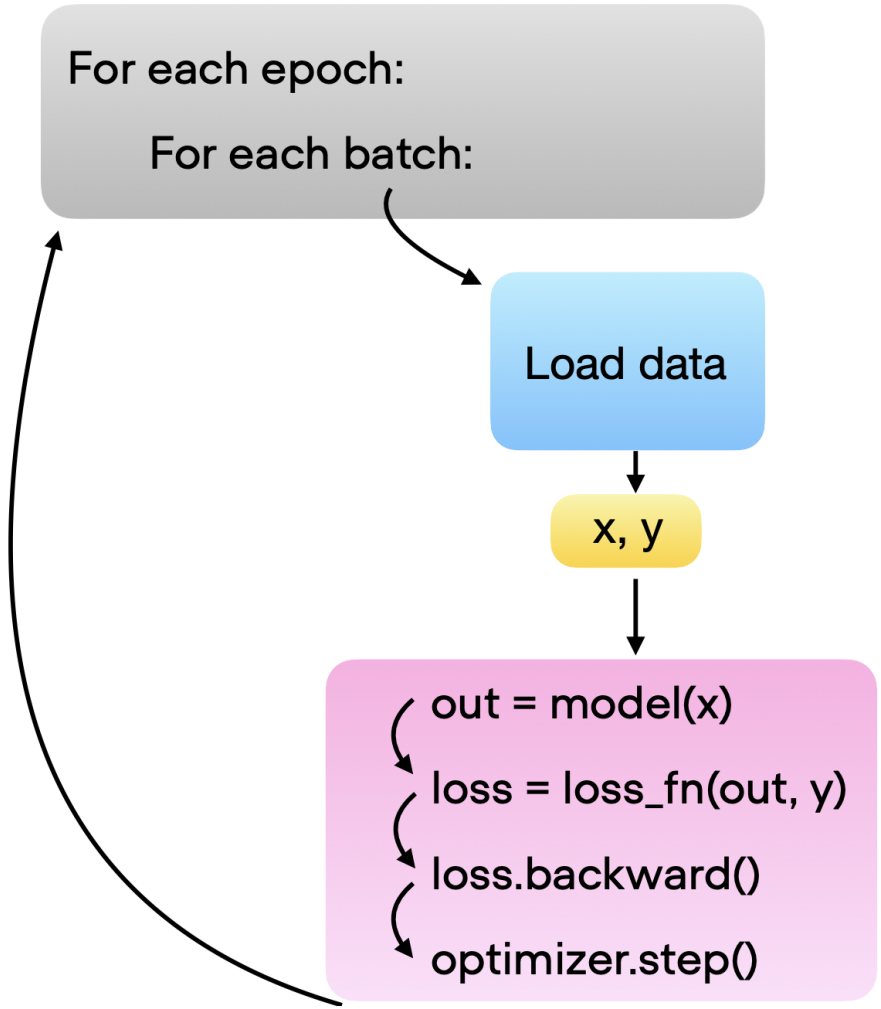
DataLoader I

Avoiding Data Loading Bottlenecks

Trong quá trình luyện, chúng ta đưa dữ liệu vào mô hình thông qua train_loader, tại sao cần phải sử dụng nó?

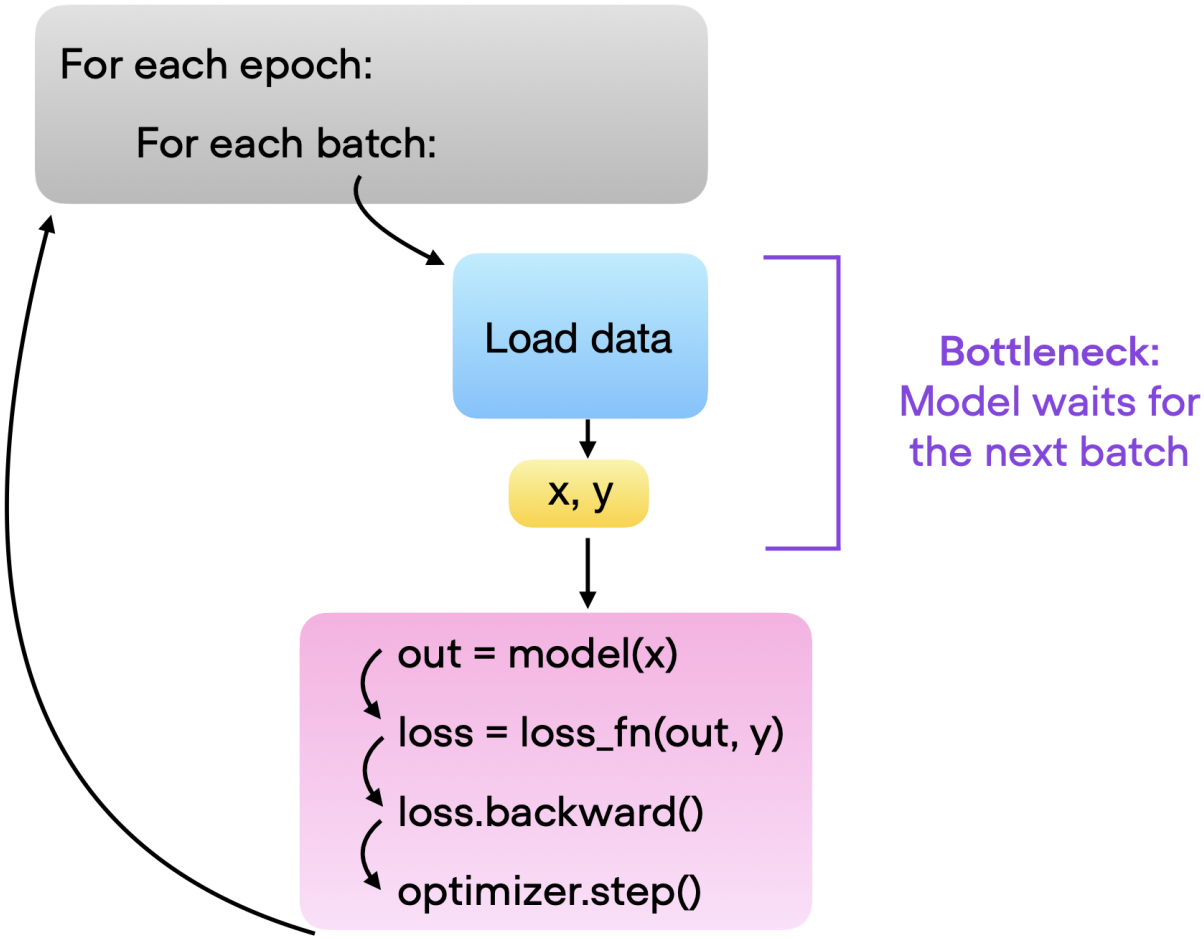
DataLoader II

Quy trình load dữ liệu thông thường



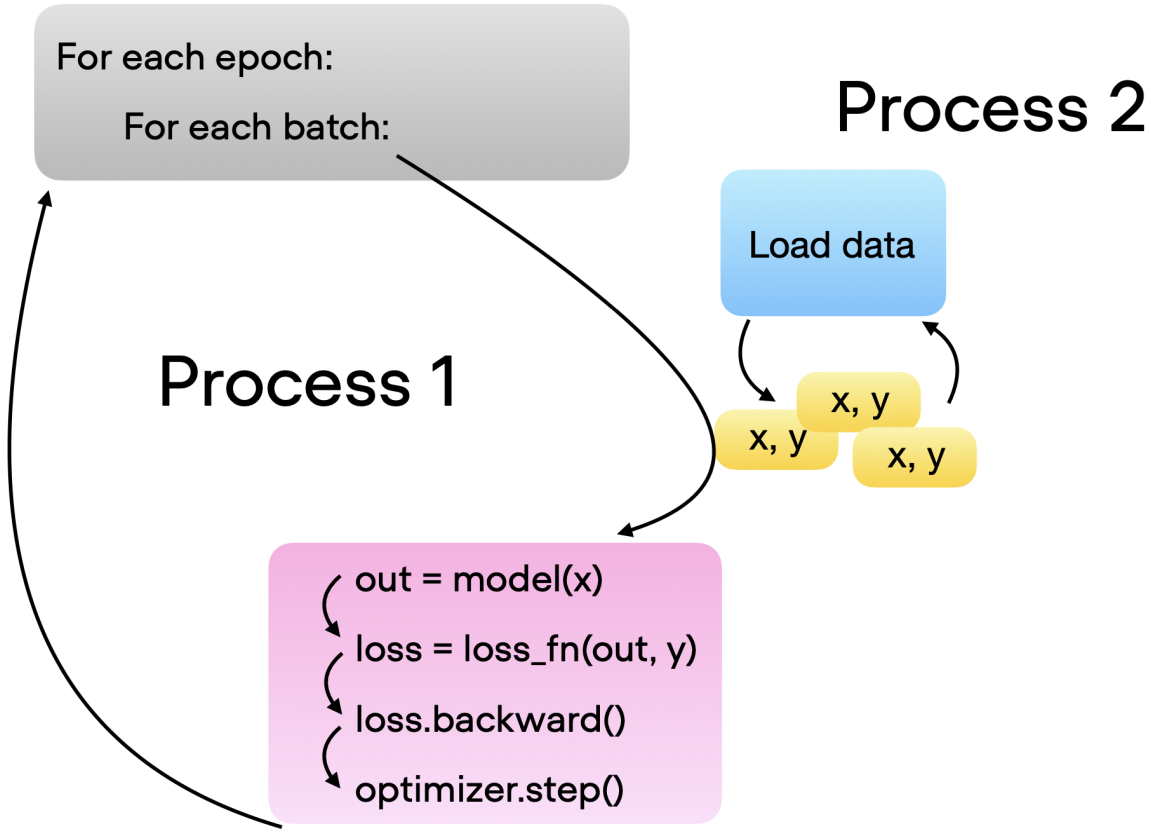
DataLoader III

Tuy nhiên cách làm này sẽ không thu được hiệu quả cao do tình trạng *nghẽn cổ chai*, model phải đợi *load dữ liệu từ batch tiếp theo*.



DataLoader IV

Sẽ hiệu quả hơn khi chúng ta load trước các mini-Batch để model không phải đợi nhận dữ liệu bằng cách chạy *data loading* và *model training* thành 2 quá trình riêng biệt.



DataLoader V

Để thực thi 2 quá trình riêng biệt trong PyTorch, chỉ cần sử dụng DataLoader từ PyTorch

```
from torch.utils.data import DataLoader

train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=32,
    shuffle=True,
    drop_last=True,
    num_workers=2,
)
```

Use multiple processes to load data in the background

Chú ý: nếu thiết lập *num_ workers=2* nghĩa là có 2 quá trình load trước các mini-batches tiếp theo.

DataLoader VI

```
from torch.utils.data import DataLoader

train_loader = DataLoader(
    dataset=train_dataset,
    batch_size=32,
    shuffle=True,
    drop_last=True,
    num_workers=2,
)
```

Just means we drop the last batch
if training set size is not
evenly divisible by batch size

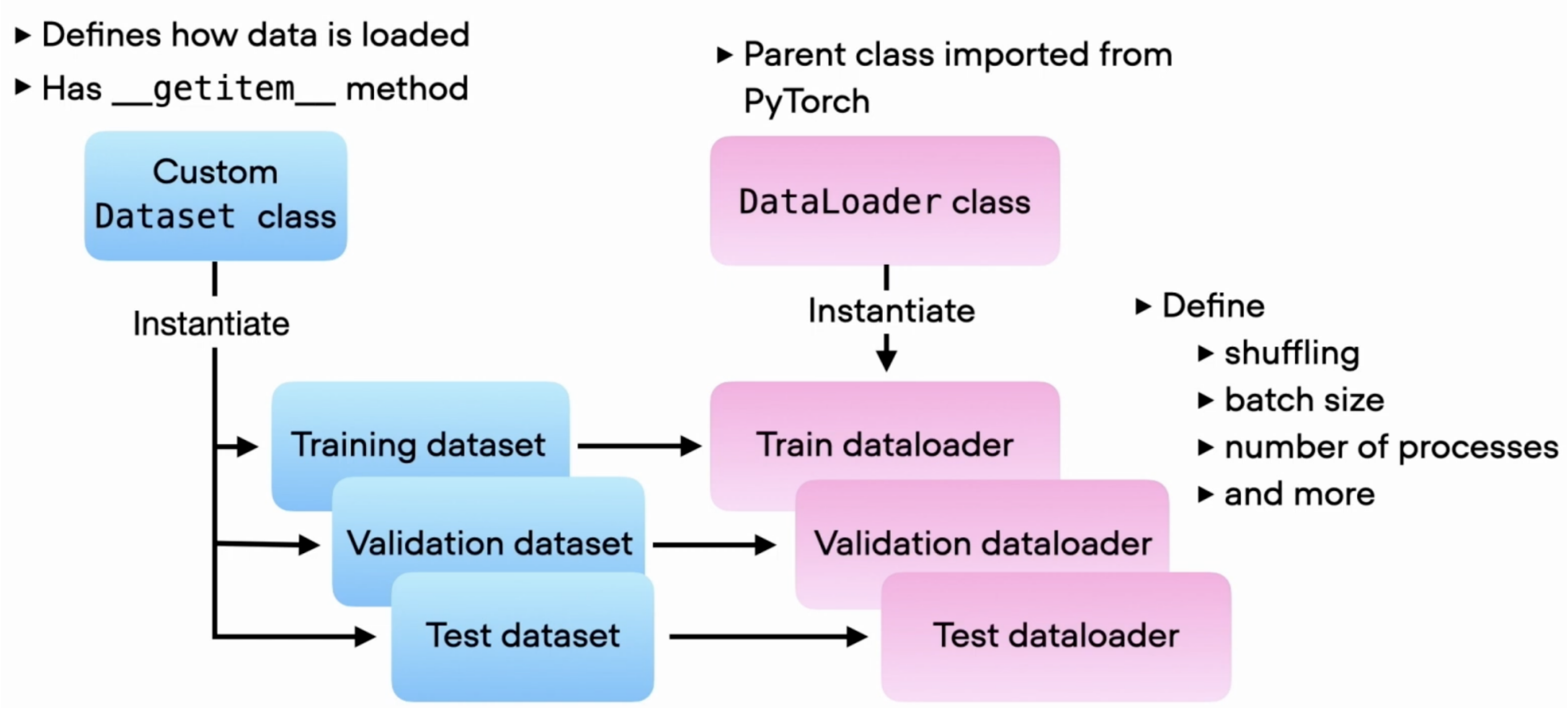
Chú ý: thiết lập *drop_last=True* chỉ dùng cho tập training nghĩa là chúng ta sẽ bỏ mini-batch cuối cùng nếu bộ dữ liệu huấn luyện của chúng ta không chia *hết* cho số batch size.

Loading Pipeline I

Về tổng quan, có 4 bước để triển khai **data loading pipeline**:

1. **Create dataset**: tạo một lớp cho tập dữ liệu tuỳ chỉnh.
2. **Instantiate datasets**: khởi tạo các đối tượng tập dữ liệu từ custom data set class *bước 1*.
3. **Instantiate Dataloaders**: khởi tạo các dataloaders dựa trên các datasets ở *bước 3*.
4. **Test/Use dataloaders**: đưa dataloaders đi chạy thử để đảm bảo rằng các bộ dữ liệu hoạt động trước khi đưa vào mô hình.

Loading Pipeline II



Loading Pipeline III

Bước 1: creating a custom data set class

The data set class needs to *define how the data is loaded* and it has to have a *get item method* here.

```
1  import os
2  import matplotlib.pyplot as plt
3  import pandas as pd
4  from PIL import Image
5  from torch.utils.data import Dataset
6
7  class MyDataset(Dataset):
8    def __init__(self, csv_path, img_dir, transform=None):
9
10     df = pd.read_csv(csv_path)
11     self.img_dir = img_dir
12     self.transform = transform
13
14     # based on DataFrame columns
```

Loading Pipeline IV

```
15      self.img_names = df["filepath"]
16      self.labels = df["label"]
17
18    def __getitem__(self, index):
19      img = Image.open(os.path.join(self.img_dir, self.img_names[index]))
20
21      if self.transform is not None:
22        img = self.transform(img)
23
24      label = self.labels[index]
25      return img, label
26
27    def __len__(self):
28      return self.labels.shape[0]
```

Loading Pipeline V

```
5  from torch.utils.data import Dataset
6
7  class MyDataset(Dataset):
8      def __init__(self, csv_path, img_dir, transform=None):
9
10         df = pd.read_csv(csv_path)
11         self.img_dir = img_dir
12         self.transform = transform
13
14         # based on DataFrame columns
15         self.img_names = df["filepath"]
16         self.labels = df["label"]
17
18     def __getitem__(self, index):
19         img = Image.open(os.path.join(self.img_dir, self.img_names[index]))
20
21         if self.transform is not None:
22             img = self.transform(img)
23
24         label = self.labels[index]
25         return img, label
26
27     def __len__(self):
28         return self.labels.shape[0]
```

set up attributes

Loading Pipeline VI

```
5  from torch.utils.data import Dataset
6
7  class MyDataset(Dataset):
8      def __init__(self, csv_path, img_dir, transform=None):
9
10         df = pd.read_csv(csv_path)
11         self.img_dir = img_dir
12         self.transform = transform
13
14         # based on DataFrame columns
15         self.img_names = df["filepath"]
16         self.labels = df["label"]
17
18     def __getitem__(self, index):
19         img = Image.open(os.path.join(self.img_dir, self.img_names[index]))
20
21         if self.transform is not None:
22             img = self.transform(img)
23
24         label = self.labels[index]
25         return img, label
26
27     def __len__(self):
28         return self.labels.shape[0]
```

Define how to load a single record (training example, test example)

Loading Pipeline VII

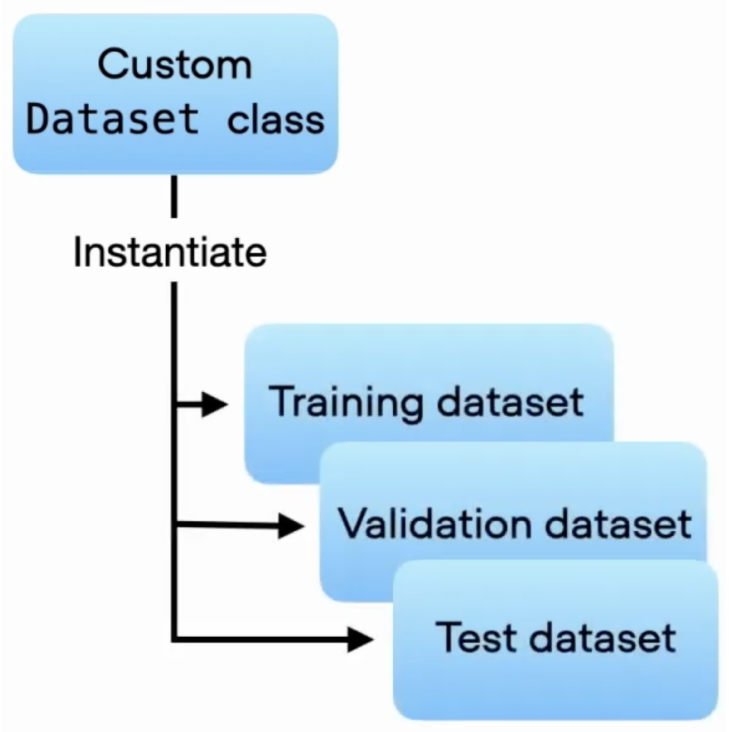
```
5  from torch.utils.data import Dataset
6
7  class MyDataset(Dataset):
8      def __init__(self, csv_path, img_dir, transform=None):
9
10         df = pd.read_csv(csv_path)
11         self.img_dir = img_dir
12         self.transform = transform
13
14         # based on DataFrame columns
15         self.img_names = df["filepath"]
16         self.labels = df["label"]
17
18     def __getitem__(self, index):
19         img = Image.open(os.path.join(self.img_dir, self.img_names[index]))
20
21         if self.transform is not None:
22             img = self.transform(img)
23
24         label = self.labels[index]
25         return img, label
26
27     def __len__(self):
28         return self.labels.shape[0]
```

the length of the dataset

Loading Pipeline VIII

Bước 2: Instantiating datasets

Khởi tạo training, validation và test datasets từ lớp dữ liệu tuỳ chỉnh ở *bước 1*.



Loading Pipeline IX

```
train_dataset = MyDataset(
  csv_path="mnist-pngs/new_train.csv",
  img_dir="mnist-pngs/",
  transform=data_transforms["train"],)
val_dataset = MyDataset(
csv_path="mnist-pngs/new_val.csv",
img_dir="mnist-pngs/",
transform=data_transforms["test"],)
test_dataset = MyDataset(
    csv_path="mnist-pngs/test.csv",
    img_dir="mnist-pngs/",
    transform=data_transforms["test"],)
```

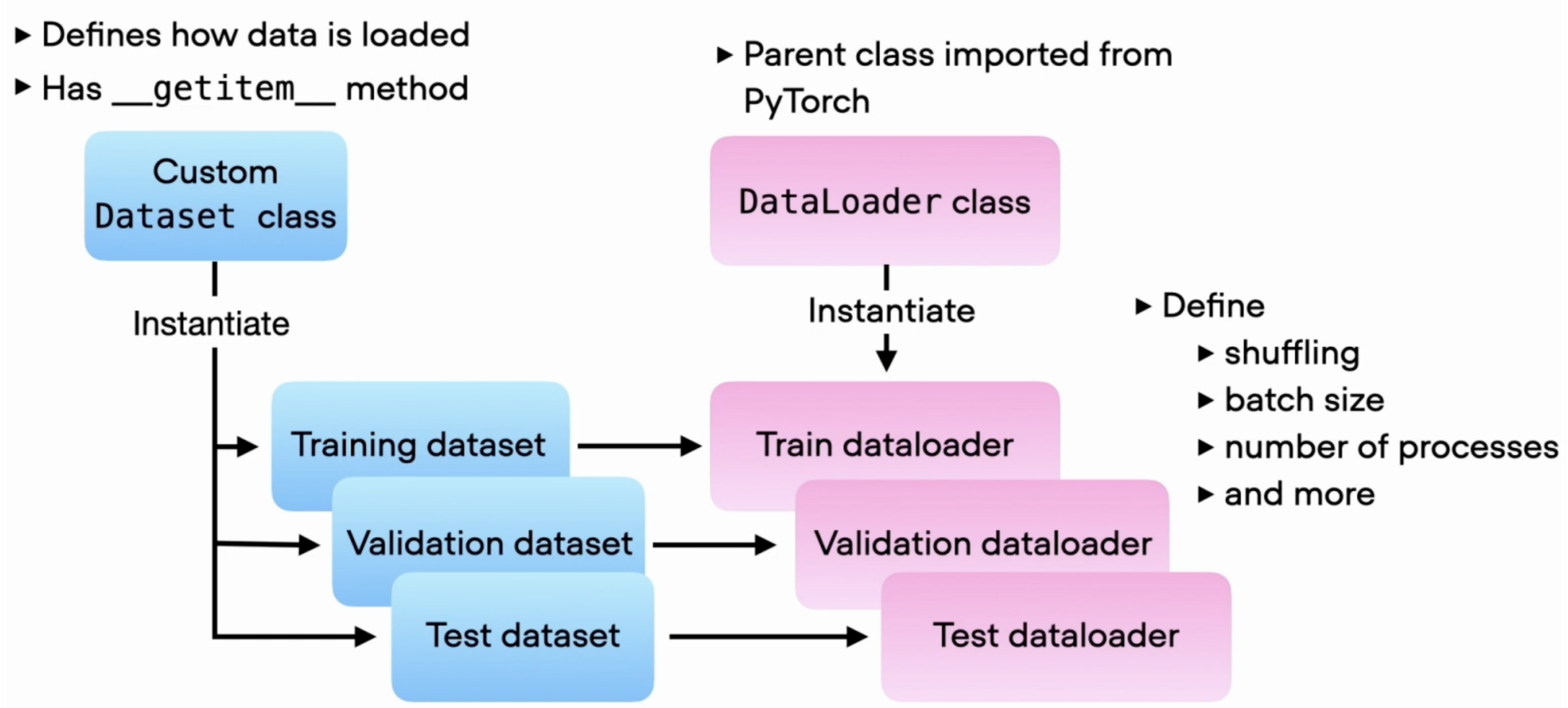
Loading Pipeline X

Lưu ý: *transform=data_transforms["train"]* dùng để biến đổi file ảnh sang PyTorch Tensor.

Bước 3: Instantiate Dataloaders

Khởi tạo các dataloaders cho các datasets đã tạo ở *bước 2*.

Loading Pipeline XI



Loading Pipeline XII

```
train_loader = DataLoader(
  dataset=train_dataset,
  batch_size=32,
  shuffle=True, # want to shuffle the dataset
  num_workers=0, # number processes/CPUs to use
  )
val_loader = DataLoader(
    dataset=val_dataset,
    batch_size=32,
    shuffle=False,
    num_workers=0,)
test_loader = DataLoader(
    dataset=test_dataset,
    batch_size=32,
    shuffle=False,
    num_workers=0)
```

Loading Pipeline XIII

Bước 4: Testing/Using dataloaders

Trước khi đưa các dataloader này vào model *chúng ta nên kiểm tra lại các dataloader có hoạt động đúng chưa vì khi đưa vào model dữ liệu không đúng sẽ rất khó debug*. Có thể sử dụng một vài bước model training loop (không dùng để huấn luyện) để in ra một vài mẫu dữ liệu được trả ra từ dataloaders.

```
import time
num_epochs = 1
for epoch in range(num_epochs):
  for batch_idx, (x, y) in enumerate(train_loader):
    time.sleep(1)
    if batch_idx >= 3:
      break
    print(" Batch index:", batch_idx, end="")
    print(" | Batch size:", y.shape[0], end="")
```

Loading Pipeline XIV

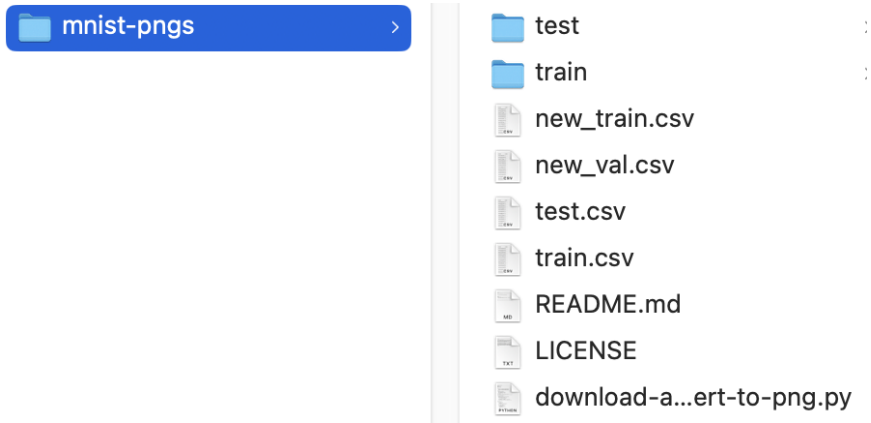
```
10      print(" | x shape:", x.shape, end="")
11      print(" | y shape:", y.shape)
12
13  print("Labels from current batch:", y)
14  #Batch index: 0 | Batch size: 32 | x shape: torch.Size([32, 1, 28, 28]) | y
        shape: torch.Size([32])
15   #Batch index: 1 | Batch size: 32 | x shape: torch.Size([32, 1, 28, 28]) | y
         shape: torch.Size([32])
16   #Batch index: 2 | Batch size: 32 | x shape: torch.Size([32, 1, 28, 28]) | y
         shape: torch.Size([32])
17  #Labels from current batch: tensor([9, 1, 3, 2, 8, 4, 8, 3, 1, 3, 6, 5, 4, 9,
        6, 2, 6, 0, 5, 8, 0, 5, 1, 0,
18          4, 2, 3, 8, 5, 5, 4, 6])
```

Chúng ta có thể thấy các dữ liệu đã được shuffled

Ứng dụng DataLoader cho bộ dữ tuỳ chỉnh MNIST

Ta có thể áp dụng trên bộ dữ liệu ảnh MNIST tuỳ chỉnh thay vì phải load từ bộ dữ liệu của PyTorch.

Bước 0. Tạo bộ dữ liệu ảnh tuỳ chỉnh MNIST



Lưu ý, để tải bộ dữ liệu ảnh từ github, cần cài thư viện: *pip install gitpython*

```
import os
from git import Repo

if not os.path.exists("mnist-pngs"):
  Repo.clone_from("https://github.com/rasbt/mnist-pngs", "mnist-pngs")
# Here, we check the CSV files listing the image names and labels
import pandas as pd

df_train = pd.read_csv('mnist-pngs/train.csv')
df_train.head()
df_test = pd.read_csv('mnist-pngs/test.csv')
df_test.head()
# Creating a validation split: MNIST doesn't come with a validation set
    partition, so we are creating it here from the training set, using 10%
    of the training data for validation.
df_train = pd.read_csv('mnist-pngs/train.csv')
df_train = df_train.sample(frac=1, random_state=123)

loc = round(df_train.shape[0]*0.9)
df_new_train = df_train.iloc[:loc]
df_new_val = df_train.iloc[loc:]
df_new_train.to_csv('mnist-pngs/new_train.csv', index=None)
df_new_val.to_csv('mnist-pngs/new_val.csv', index=None)
```

Bước 1. Defining the Dataset Class

```
1  import os
2  import matplotlib.pyplot as plt
3  import pandas as pd
4  from PIL import Image
5  from torch.utils.data import Dataset
6
7  class MyDataset(Dataset):
8    def __init__(self, csv_path, img_dir, transform=None):
9
10     df = pd.read_csv(csv_path)
11     self.img_dir = img_dir
12     self.transform = transform
13
14     # based on DataFrame columns
15     self.img_names = df["filepath"]
16     self.labels = df["label"]
17
18   def __getitem__(self, index):
19     img = Image.open(os.path.join(self.img_dir, self.img_names[index]))
20
21     if self.transform is not None:
22       img = self.transform(img)
```

```
23
24      label = self.labels[index]
25      return img, label
26
27  #Defining an optional batch visualization function
28  def __len__(self):
29    return self.labels.shape[0]
30
31    def viz_batch_images(batch):
32
33    plt.figure(figsize=(8, 8))
34    plt.axis("off")
35    plt.title("Training images")
36    plt.imshow(
37      np.transpose(
38        vutils.make_grid(batch[0][:64], padding=2, normalize=True), (1, 2, 0)
39      )
40    )
41    plt.show()
42
43  #Defining optional image transformations
44  from torchvision import transforms
45
46  data_transforms = {
```

```
47      "train": transforms.Compose(
48          [
49              transforms.Resize(32),
50              transforms.RandomCrop((28, 28)),
51              transforms.ToTensor(),
52              # normalize images to [-1, 1] range
53              transforms.Normalize((0.5,), (0.5,)),
54          ]
55      ),
56      "test": transforms.Compose(
57          [
58              transforms.Resize(32),
59              transforms.CenterCrop((28, 28)),
60              transforms.ToTensor(),
61              # normalize images to [-1, 1] range
62              transforms.Normalize((0.5,), (0.5,)),
63          ]
64      ),
65  }
```

Chú ý: image transformations

A *transformation* is essentially a data augmentation, a s*light change of the image*. So here we are *resizing* the images, *randomly cropping* them, and then *converting them to tensors*.

- *Random_ Crop* chỉ dùng cho tập train.
- Khi dùng *.ToTensor()*, ảnh sẽ được chuẩn hoá về vùng giá trị [-1,1]

Bước 2-3. Defining the data loaders. *Lưu ý:* Datasets đã tạo từ *bước 0*.

```
from torch.utils.data import DataLoader

train_dataset = MyDataset(
  csv_path="mnist-pngs/new_train.csv",
  img_dir="mnist-pngs/",
  transform=data_transforms["train"],
)

train_loader = DataLoader(
  dataset=train_dataset,
  batch_size=32,
  shuffle=True, # want to shuffle the dataset
  num_workers=0, # number processes/CPUs to use
)

val_dataset = MyDataset(
    csv_path="mnist-pngs/new_val.csv",
    img_dir="mnist-pngs/",
    transform=data_transforms["test"],
)

val_loader = DataLoader(
    dataset=val_dataset,
    batch_size=32,
```

```
    shuffle=False,
    num_workers=0,
)

test_dataset = MyDataset(
    csv_path="mnist-pngs/test.csv",
    img_dir="mnist-pngs/",
    transform=data_transforms["test"],
)

test_loader = DataLoader(
    dataset=test_dataset,
    batch_size=32,
    shuffle=False,
    num_workers=0
)
```

Testing the data loaders

```
import time

num_epochs = 1
for epoch in range(num_epochs):

  for batch_idx, (x, y) in enumerate(train_loader):
    time.sleep(1)
    if batch_idx >= 3:
      break
    print(" Batch index:", batch_idx, end="")
    print(" | Batch size:", y.shape[0], end="")
    print(" | x shape:", x.shape, end="")
    print(" | y shape:", y.shape)

print("Labels from current batch:", y)

# Uncomment to visualize a data batch:
# batch = next(iter(train_loader))
# viz_batch_images(batch[0])
```

Tài liệu tham khảo


Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.


Sebastian Raschka
MACHINE LEARNING Q AND AI: 30 Essential Questions and Answers on Machine Learning and AI (2024). ISBN-13: 978-1-7185-0377-9 (ebook).


LightningAI
LightningAI: PyTorch Lightning (2024) .