

# PyTorch 講座 5

安藤 龍太郎

# 前回のおさらい

- レイヤーについて
  - Convolution
  - Linear
  - BatchNormalization
  - ReLU
- モデルの作成
  - nn.Module を必ず継承
  - forward: 入力から出力までの処理を記述

5~6回でやること

# PyTorch - データセット・データローダー -

- 読み込み・前処理など、学習に用いるデータ関連全ての処理を担当
- データセット: データを読み込み、前処理を施して返すまでの流れを記述
- データローダー: データセットからデータを1つ受け取り、バッチ化する

```
from PIL import Image
from pathlib import Path
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):

    def __init__(self, dataset_dir):
        dir_path_resolved = Path(dataset_dir).resolve()
        self.paths = list(dir_path_resolved.glob("*"))

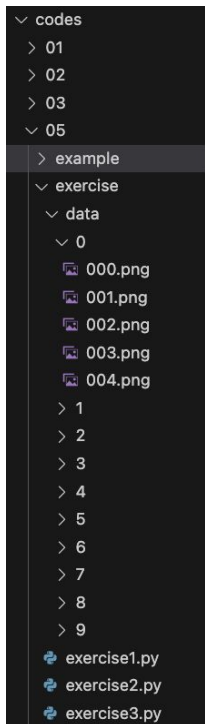
    def __len__(self):
        return len(self.paths)

    def __getitem__(self, idx):
        path = self.paths[idx]
        return path

if __name__ == "__main__":
    dataloader = DataLoader([dataset=MyDataset("."), batch_size=2])
```

# データセットのダウンロード

[Github](#)のpytorch-training/codes/05/exercise/dataから取得してください。



cifar10

物体カラー写真の画像データセット

乗り物や動物

```
git clone https://github.com/shun74/pytorch-training
```

gitから取得できない人は下記から※[Google Drive](#)からダウンロード

# Pathクラスについて

# Path

- なぜ学ぶか

データセットを構築する際、データが格納されているフォルダ構造やファイル名にアクセスする必要がある。

- Pathクラスとは

ファイルパスの操作をPathクラス一つで完結することができる。  
簡潔で直感的なコード

pathlibライブラリのPathクラス

```
from pathlib import Path
```

# Path

Pathクラスの以下のメソッドについて説明

- resolve
- glob
- **【/】**演算子を使ってパスを追加
- exists
- mkdir



# Path

- resolve  
絶対パスの取得

```
data_directory = "./data"
data_directory_path = Path(data_directory).resolve()
print("---Displaying the absolute path---")
print(data_directory_path)
```

```
---Displaying the absolute path---
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/dir05/data
Display all files underneath
```

- glob  
配下にあるファイルの取得

```
# dataディレクトリ以下のすべてのファイルを取得
file_list = list(data_directory.glob("*"))
```

```
---Display all files underneath---
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/dir05/data/002.png
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/dir05/data/003.png
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/dir05/data/001.png
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/dir05/data/000.png
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/dir05/data/004.png
```

recursive ...

# Path

- 【/】演算子を使ってパスを追加  
パス連結

```
base_directory = Path("/base_dir")
subdirectory_name = "data_dir"

new_path = base_directory / subdirectory_name

---Display the concatenated path---
/base_dir/data_dir
```

- exists  
存在するか

```
path = Path(file_path)

# ファイルが存在するか判定
if path.exists():
    print(f"The file '{file_path}' exists.")
else:
    print(f"The file '{file_path}' does not exist.")
```

# Path

- mkdir  
ディレクトリ作成

```
# ディレクトリを作成したいパス
directory_path = Path("./target_dir")

# ディレクトリを作成
directory_path.mkdir(parents=True, exist_ok=True)
```


# 演習

1. dataディレクトリの絶対パスを出力してください。
2. dataディレクトリ配下のディレクトリを出力してください。  
(1のインスタンスを使用)
3. dataディレクトリ配下の画像ファイルの数を出力してください。  
(1のインスタンスを使用)

▽ exercise

▽ data

▽ 0

 000.png

 001.png

 002.png

 003.png

 004.png

> 1

> 2

> 3

> 4

> 5


> 6


> 7

> 8

> 9

 exercise1.py

 exercise2.py

 exercise3.py

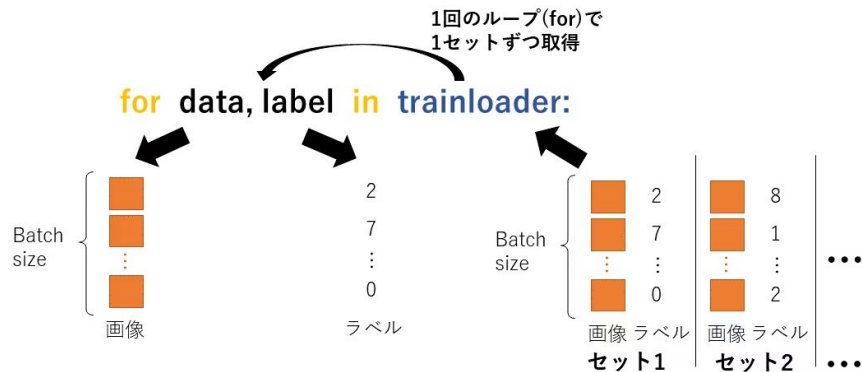
## 解答例

```
===== problem1 =====  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data  
===== problem2 =====  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/0  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/1  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/2  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/3  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/4  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/5  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/6  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/7  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/8  
/Users/andoryutaro/Documents/東京理科大学/M1/pytorch_learning/pytorch-training/codes/05/exercise/data/9  
===== problem3 =====  
50
```

# 学習準備の事前知識

# どのように学習するか

モデルにデータを入力し  
予測結果と実際のラベルを比較する



```
# データローダーからミニバッチを取り出すループ
for inputs, labels in dataloaders_dict[phase]:

    # GPUが使えるならGPUにデータを送る
    inputs = inputs.to(device)
    labels = labels.to(device)

    # optimizerを初期化
    optimizer.zero_grad()

    # 順伝搬 (forward) 計算
    with torch.set_grad_enabled(phase == 'train'):
        outputs = net(inputs)
        loss = criterion(outputs, labels) # 損失を計算
        _, preds = torch.max(outputs, 1) # ラベルを予測

    # 訓練時はバックプロパゲーション
    if phase == 'train':
        loss.backward()
        optimizer.step()

    # 結果の計算
    epoch_loss += loss.item() * inputs.size(0) # lossの合計を更新
    # 正解数の合計を更新
    epoch_corrects += torch.sum(preds == labels.data)
```

# One-Hotとは

機械学習のカテゴリカルデータ(質的データ)を処理するための手法の一つ

カテゴリカルデータ(質的データ)を数値データに変換

カテゴリカルデータとは

数量的な大小関係がないデータ  
(色、血液型、地域...)

- 名義尺度  
ラベルに順序がない(赤、青、黄)
- 順序尺度  
カテゴリに順序がある(低、中、大)

血液型				
A型	A型	B型	O型	AB型
A型	1	0	0	0
B型	0	1	0	0
O型	0	0	1	0
AB型	0	0	0	1

One-Hot  
エンコーディング





データセットについて

# データセットとは

データセット: データを読み込み、前処理を施して返すまでの流れを記述

torch.utils.data.Datasetクラスを継承する必要がある

実装する必要があるメソッド

- `__getitem__()`
  - Datasetから一つのデータを取り出すメソッド
- `__len__()`
  - Datasetのファイル数を返すメソッド

```
from PIL import Image
from pathlib import Path
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):

    def __init__(self, dataset_dir):
        dir_path_resolved = Path(dataset_dir).resolve()
        self.paths = list(dir_path_resolved.glob("*"))

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

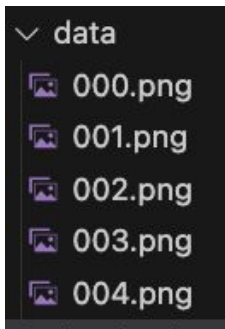
    def __getitem__(self, idx):
        path = self.paths[idx]
        return path

if __name__ == "__main__":
    dataloader = DataLoader([MyDataset("."), batch_size=2])
```

# 例

- `__getitem__()`  
インデックスを内部で加算することでデータを参照する

コード実行時のファイル構造



```
---Number of files in the dataset---  
5  
---Output the size of the image file---  
(32, 32)
```

```
import matplotlib.pyplot as plt  
from PIL import Image  
from pathlib import Path  
from torch.utils.data import Dataset  
  
class MyDataset(Dataset):  
  
    def __init__(self, dataset_dir):  
        dir_path_resolved = Path(dataset_dir).resolve()  
        self.img_list = list(dir_path_resolved.glob("*.png"))  
  
    def __len__(self):  
        return len(self.img_list)  
  
    def __getitem__(self, idx):  
        img_path = self.img_list[idx]  
        img = Image.open(img_path)  
        return img  
  
if __name__ == "__main__":  
    my_dataset = MyDataset("./data")  
    print("---Number of files in the dataset---")  
    print(len(my_dataset))  
    print("---Output the size of the image file---")  
    print(my_dataset[0].size)  
    plt.imshow(my_dataset[0])  
    plt.show()
```

# 演習

1. ダウンロードした画像ファイルのデータセットクラスを作成してください。

1.1. データセットの枚数を出力してください

1.2. 画像サイズを出力してください

## ヒント

- `Image.open`でやっていること  
画像ファイルを開いて`Image`オブジェクトを返す。
- サイズ出力するメソッド  
`size`を使用することで大きさを取得する  
`img`: `Image`クラスのインスタンス  
`img.size`

## 解答例

```
===== problem1.1 =====  
50  
===== problem1.2 =====  
(32, 32)
```

# 実際に使われる例

`__getitem__()`:

データとラベルをペアで返す

tensor型に変更する

`transforms.Compose`関連は次回説明

`img_path`: `Path`クラスのインスタンス  
`img_path.parts`

```
/dir1/dir2/file1
```

```
['/', 'dir1', 'dir2', 'file1']
```

```
---Number of files in the dataset---
```

```
5
```

```
img.size : torch.Size([3, 32, 32])
```

```
label    : 002
```

```
class MyDataset(Dataset):
```

```
    def __init__(self, dataset_dir):
```

```
        dir_path_resolved = Path(dataset_dir).resolve()
```

```
        self.img_list = list(dir_path_resolved.glob("*.png"))
```

```
        self.transform = transforms.Compose([
```

```
            transforms.ToTensor(),
```

```
        ])
```

```
    def __len__(self):
```

```
        return len(self.img_list)
```

```
    def __getitem__(self, idx):
```

```
        img_path = self.img_list[idx]
```

```
        img = Image.open(img_path)
```

```
        img_tensor = self.transform(img)
```

```
        # ファイル名からラベルの取得
```

```
        img_path = Path(img_path)
```

```
        parts = img_path.parts
```

```
        label = int(parts[-1][:3])
```

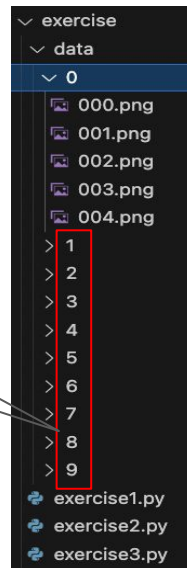
```
        return img_tensor, label
```

# 演習

1. データセットで画像ファイルの大きさとラベルを一緒に出力してください。  
ラベルは今回は画像ファイルの親ディレクトリの名前としてください。

- 1.1. 画像ファイルの大きさを出力してください。
- 1.2. ラベルを出力してください。

ラベル  
親dir



## 解答例

```
===== problem1.1 =====  
torch.Size([3, 32, 32])  
===== problem1.2 =====  
9
```



# 次回

- Dataloaderの説明
- transformsの説明

## 演習

5,6を組み合わせた出力