PyTorch 講座 6

安藤　龍太朗

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

```python
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)
```

前回のおさらい

- Pathクラス
  - resolve
  - glob
  - 【/】演算子を使ってパスを追加
  - exists

- データセット
  - __getitem__
  - __len__

今回やること

今回やること

- transformsの説明
  - 画像の前処理
- データローダーの説明
  - バッチ

演習

5,6を組み合わせた出力

前処理

モデルで学習する前に前処理を行う必要がある。

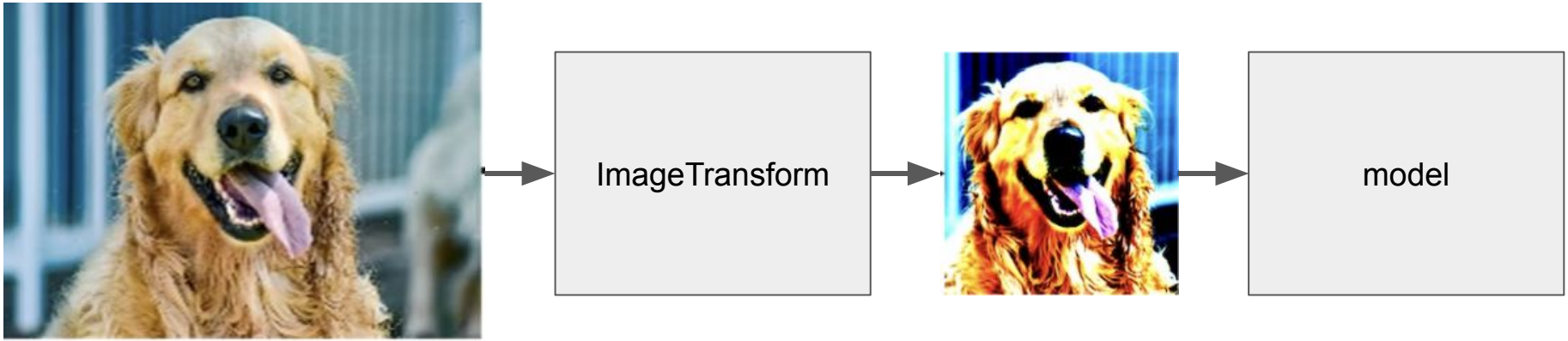
なんで前処理する必要があるか

- 汎用性向上
  - データの拡張
- データの正規化
  - 安定性の向上

前回のコード

```python
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
```

前処理



transforms

以下のメソッドについて説明

- `transforms.ToTensor()`
- `transforms.Resize((Height, Width))`
- `transforms.CenterCrop`
- `transforms.RandomResizedCrop`
- `transforms.RandomHorizontalFlip()`
- `transforms.Normalize(mean=[, , ], std=[, , ])`

pytorch公式サイト参照

ToTensor

PyTorchのテンソルに変換するための変換関数

- 画像の各ピクセルの値を
  [0, 255]から[0.0, 1.0]の範囲にスケーリング
- 軸が入れ替わる（H,W,C)→(C,H,W)
  - H:高さ
  - W:幅
  - C:チャネル

```
変換前の型表示：
 <class 'PIL.JpegImagePlugin.JpegImageFile'>
変換前のサイズ表示：
 (426, 640, 3)
変換後の型表：
 <class 'torch.Tensor'>
変換後のサイズ表示：
 torch.Size([3, 426, 640])
```

toTensor.py U

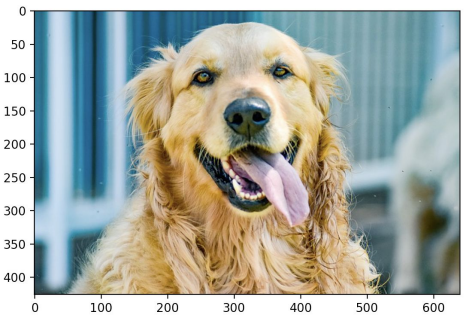
codes > 06 > example > toTensor.py > ...

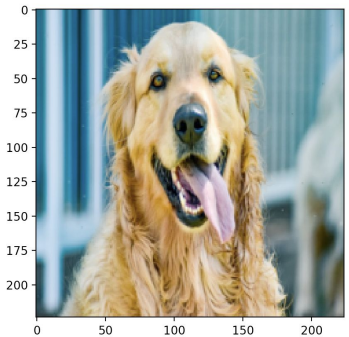
```python
from PIL import Image
from torchvision import transforms
import numpy as np

if __name__ == "__main__":
    image_path = "./example_data/dog_img.png"
    image = Image.open(image_path)

    transform = transforms.Compose([
        transforms.ToTensor()
    ])
    # 変換を適用
    transformed_image = transform(image)
    print("変換前の型表示:\n",type(image))
    # NumPy配列に変換
    image_array = np.array(image)
    print("変換前のサイズ表示:\n",image_array.shape)

    print("変換後の型表:\n",type(transformed_image))
    print("変換後のサイズ表示:\n",transformed_image.size())
```

Resize

指定されたサイズにリサイズする関数

```
変換前の画像サイズ
(640, 426)
変換後の画像サイズ
(224, 224)
```





resize.py U

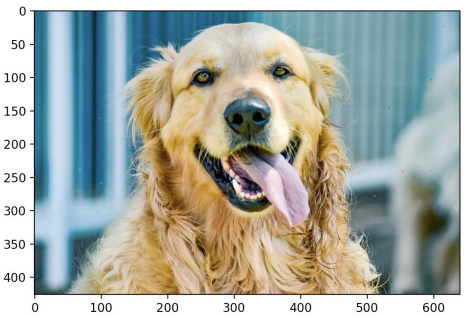
codes > 06 > example > resize.py > ...

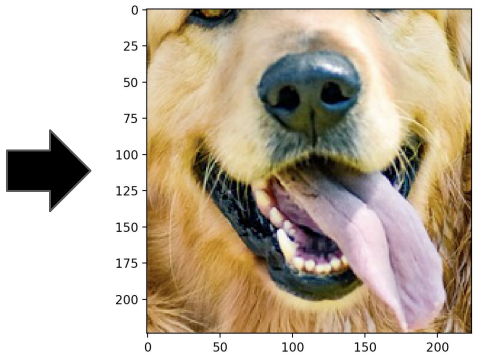
```python
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

if __name__ == "__main__":
    image_path = "./example_data/dog_img.png"
    image = Image.open(image_path)

    transform = transforms.Compose([
        transforms.Resize((224,224))
    ])
    # 変換を適用
    transformed_image = transform(image)
    print("変換前の画像サイズ")
    print(image.size)
    print("変換後の画像サイズ")
    print(transformed_image.size)
    plt.imshow(image)
    plt.show()
    plt.imshow(transformed_image)
    plt.show()
```

CenterClop

画像の中心を指定したサイズでクロップする関数

```
変換前の画像サイズ
(640, 426)
変換後の画像サイズ
(224, 224)
```





centerCrop.py U

codes > 06 > example > centerCrop.py > ...

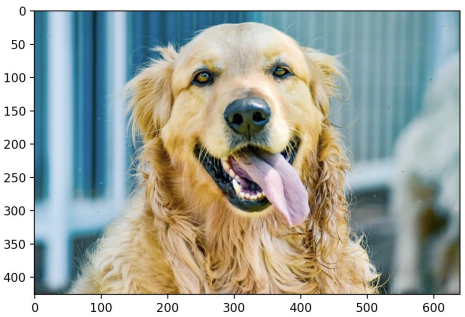
```python
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

if __name__ == "__main__":
    image_path = "./example_data/dog_img.png"
    image = Image.open(image_path)

    transform = transforms.Compose([
        transforms.CenterCrop(size=(224,224))
    ])
    # 変換を適用
    transformed_image = transform(image)
    print("変換前の画像サイズ")
    print(image.size)
    print("変換後の画像サイズ")
    print(transformed_image.size)
    plt.imshow(image)
    plt.show()
    plt.imshow(transformed_image)
    plt.show()
```

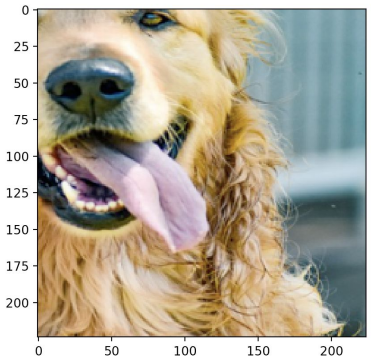
RandomResizedCrip

ランダムな位置から画像をクロップし、
指定されたサイズにリサイズする関数

scale:縮尺の度合い

```
変換前の画像サイズ
(640, 426)
変換後の画像サイズ
(224, 224)
```





randomResizedCrop.py U

codes > 06 > example > randomResizedCrop.py > ...

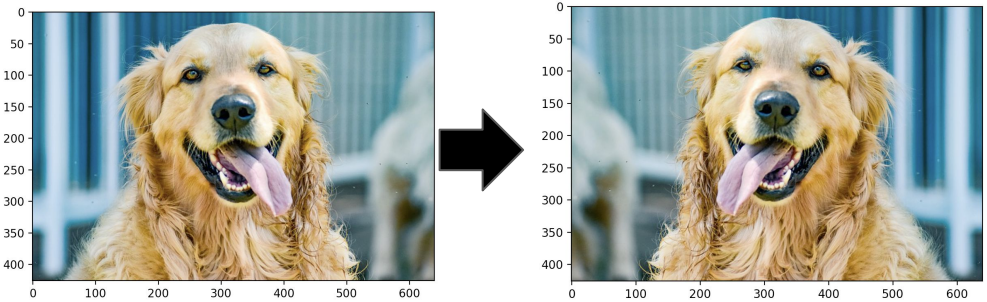
```python
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

if __name__ == "__main__":
    image_path = "./example_data/dog_img.png"
    image = Image.open(image_path)

    transform = transforms.Compose([
        transforms.RandomResizedCrop(size=(224, 224), scale=(0.08, 1.0)),
    ])
    # 変換を適用
    transformed_image = transform(image)
    print("変換前の画像サイズ")
    print(image.size)
    print("変換後の画像サイズ")
    print(transformed_image.size)
    plt.imshow(image)
    plt.show()
    plt.imshow(transformed_image)
    plt.show()
```

RandomHorizontalFlip

画像をランダムに水平方方向に反転する関数

transforms.RandomHorizontalFlip(p=0.5)

50%の確率で画像反転



randomHorizontalFlip.py U

codes > 06 > example > randomHorizontalFlip.py > ...

```python
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

if __name__ == "__main__":
    image_path = "./example_data/dog_img.png"
    image = Image.open(image_path)

    transform = transforms.Compose([
        transforms.RandomHorizontalFlip()
    ])
    # 変換を適用
    transformed_image = transform(image)
    # 画像出力
    plt.imshow(image)
    plt.show()
    plt.imshow(transformed_image)
    plt.show()
```

Normalize

画像データの正規化を行う関数

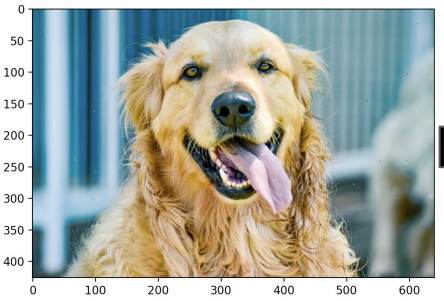
transforms.Normalize(mean,std)

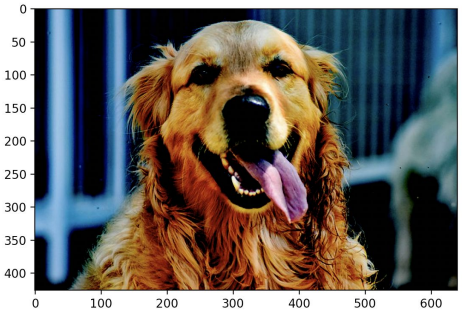
データを平均がmean、標準偏差がstdとなるように変換

※mean=[R,G,B],std=[R,G.B]　各チャネルに対して

ネットワークへの入力データのスケールを揃え、

学習の安定性や収束速度を安定させる





normalize.py U

codes > 06 > example > normalize.py > ...

```python
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

if __name__ == "__main__":
    image_path = "./example_data/dog_img.png"
    image = Image.open(image_path)

    # 画像をTensorに変換し、その後に正規化を行う変換を追加
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])

    # 変換を適用
    transformed_image = transform(image)

    # 順番入れ替え
    tensor_data_permuted = transformed_image.permute(1, 2, 0)
    # NumPy配列に変換
    numpy_array = tensor_data_permuted.numpy()

    # 画像出力
    plt.imshow(image)
    plt.show()
    plt.imshow(numpy_array)
    plt.show()
```

ColorJitter

明るさ・彩度などを変更することで画像の色情報を変更する

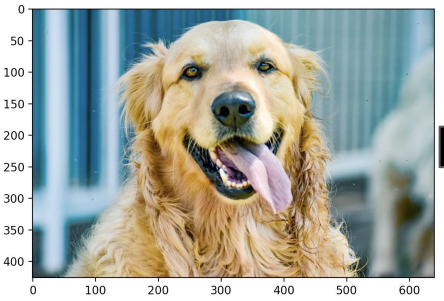
transforms.ColorJitter(brightness, contrast, saturation,hue)

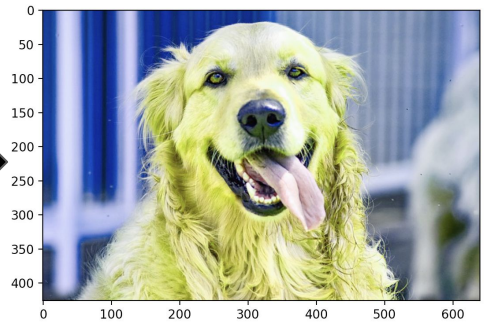
brightness:明るさをランダムに変更

contrast:コントラストをランダムに変更

saturation:彩度をランダムに変更

hue:色相をランダムに変更





colorJitter.py U X

codes > 06 > example > colorJitter.py > ...

```python
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

if __name__ == "__main__":
    image_path = "./example_data/dog_img.png"
    image = Image.open(image_path)

    transform = transforms.Compose([
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    ])
    # 変換を適用
    transformed_image = transform(image)
    plt.imshow(image)
    plt.show()
    plt.imshow(transformed_image)
    plt.show()
```

コントラストとは
明るい部分と暗い部分との相対的な差異や際立ちを示す尺度

学習時と推論時の使い分け

- 訓練データ
  - データ拡張が行われる
  - モデルは多くのバリエーションで学習することができ汎化能力を向上
- テストデータ
  - データ拡張を行わない
  - テストデータはモデルの性能を評価するもの

imageTransform.py U

codes > 06 > example > imageTransform.py > ...

```python
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms

class ImageTransform():
    def __init__(self, resize, mean, std):
        self.data_transform = {
            'train': transforms.Compose([
                transforms.RandomResizedCrop(
                    resize, scale=(0.5, 1.0)),
                transforms.RandomHorizontalFlip(),
                transforms.ToTensor(),
                transforms.Normalize(mean, std)
            ]),
            'val': transforms.Compose([
                transforms.Resize(resize),
                transforms.CenterCrop(resize),
                transforms.ToTensor(),
                transforms.Normalize(mean, std)
            ])
        }

    def __call__(self, img, phase='train'):
        """
        phase : 'train' or 'val'
        """
        return self.data_transform[phase](img)

if __name__ == "__main__":
    image_file_path = "./example_data/dog_img.png"
    img = Image.open(image_file_path)

    # 元の画像の出力
    plt.imshow(img)
    plt.show()

    # 画像の前処理と処理済み画像の表示
    size = 224
    mean = (0.485, 0.456, 0.406)
    std = (0.229, 0.224, 0.225)

    transform = ImageTransform(size, mean, std)
    img_transformed = transform(img, phase="train")

    # (色、高さ、幅)を (高さ、幅、色)に変換
    img_transformed = img_transformed.numpy().transpose((1, 2, 0))
    # 変換後の画像出力
    plt.imshow(img_transformed)
    plt.show()
```

訓練データとテストデータ

ファイル構造

5回のデータを使用

```
random_split(file_list, [train_size, test_size])
```

データセットをトレーニングデータと検証データにランダムに分割する

今回はトレーニングデータ80%
検証データ20%

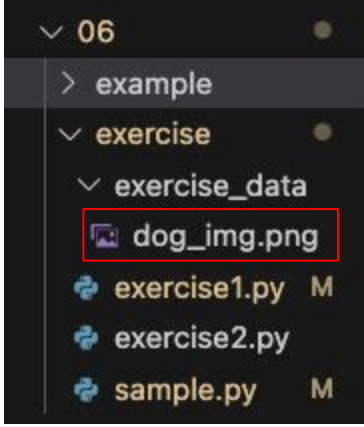
```
dataset size:  50
train dataset size: 40
val dataset size: 10
```

splitDataset.py U X

codes > 06 > example > splitDataset.py > ...

```python
from pathlib import Path
from torch.utils.data import random_split

if __name__ == "__main__":
    data_directory = "../../05/exercise/data"
    data_directory_path = Path(data_directory).resolve()
    dir_list = sorted(list(data_directory_path.glob("*")))
    file_list = []
    for dir in dir_list:
        file_path_list = list(dir.glob("*"))
        file_list += file_path_list

    # データセット全体のサイズ
    dataset_size = len(file_list)

    # データセットを訓練データとテストデータに分割する割合を設定
    train_ratio = 0.8
    train_size = int(train_ratio * dataset_size)
    val_size = dataset_size - train_size

    # ランダムに分割
    train_dataset, val_dataset = random_split(file_list, [train_size, val_size])

    # 分割されたデータセットのサイズを確認
    print(f"dataset size:  {len(file_list)}")
    print(f"train dataset size: {len(train_dataset)}")
    print(f"val dataset size: {len(val_dataset)}")

```

Githubに画像があるのでダウンロード後
右図の場所に配置してください



演習

以下の演習全てテンソル化も行ってください

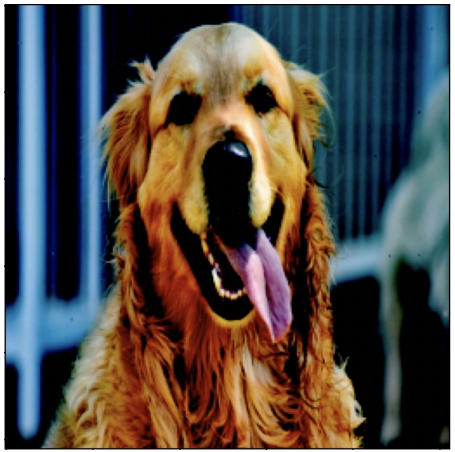
1. 画像のリサイズと正規化を行ってください
2. ランダムな水平反転と色調の変更を行ってください
3. ランダムな回転とクロップを行ってください

```
preprocess_1 = transforms.Compose([
    """
    記述
    """
])
# サンプル画像を読み込んで前処理を適用
image_path = "./exercise_data/dog_img.png"
image = Image.open(image_path)
processed_image = preprocess_1(image)
# テンソルに変換した画像を表示
plt.imshow(processed_image.permute(1, 2, 0))  # チャンネル次元を最後に移動
plt.show()
```

右図の”””記述”””の箇所を埋めて画像に適用して画像出力してください

解答例

1.



2.

3.

※2,3はランダムなため出力が異なることがあります

データローダー

データローダーとは

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

引数について（主に指定する引数）

- dataset
  - データセットオブジェクトを指定
    torch.utils.data.Dataset
- batch_size
  - ミニバッチのサイズを指定
    バッチについては後のスライドで説明

```python
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)
```

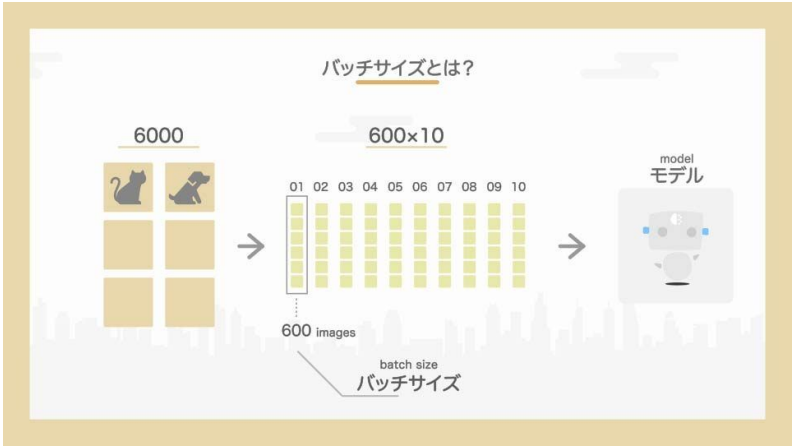
pytorch公式サイト説明

バッチとは

モデルの訓練や推論において複数の入力データを同時に処理するためのデータのまとまり

利点
- 計算量の削減
- 学習の安定化
  （backwardでバッチの誤差の平均？）

バッチサイズ：モデルが一度に処理するデータの数



引用

前回のコードに追記

次の演習では次のコードを参考にしてください。

演習1はコードの”””記述”””に書き加えてください

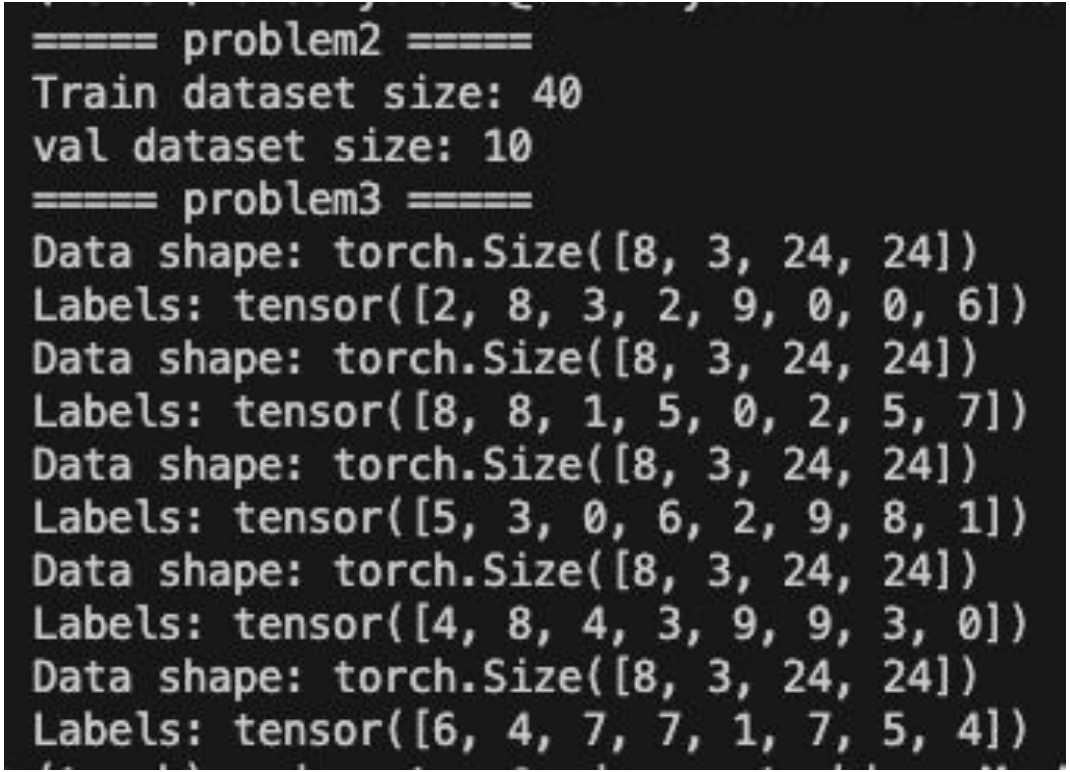
```python
class ImageTransform():
    def __init__(self, resize, mean, std):
        self.data_transform = {
            'train': transforms.Compose([
                """
                記述
                """
            ]),
            'val': transforms.Compose([
                """
                記述
                """
            ])
        }

    def __call__(self, img, phase='train'):
        return self.data_transform[phase](img)

class MyDataset(Dataset):
    def __init__(self, img_list, transform=None, phase='train'):
        self.transform = transform
        self.phase = phase
        self.img_list = img_list

    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, self.phase)
        # 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[-2])

        return img_tensor, label


if __name__ == "__main__":

    """
    記述
    """
    train_dataset =     "=" の右側に式が必要です
    val_dataset =     "=" の右側に式が必要です
    size = 24
    mean = (0.5,0.5,0.5)
    std = (0.5,0.5,0.5)
    train_dataset = MyDataset(train_dataset, transform=ImageTransform(size,mean,std), phase='train')
    val_dataset = MyDataset(val_dataset, transform=ImageTransform(size,mean,std), phase='val')
```

演習

1. スライドのコードのtransforms.Compose内を記述してください
   出力はなし
   - 学習
     - ランダムな水平反転
     - ランダムなクロップ
     - テンソル化
     - 正規化
   - 評価
     - リサイズ
     - テンソル化
     - 正規化
2. スライドを参考にして学習、評価に分割しそれぞれのDatasetオブジェクトを作成しデータセットの要素数を出力してください
3. データローダーを用いてバッチサイズ含めたサイズとラベルを出力してください
   （batch_size=8）

解答例

出力結果



コード

```
print("===== problem2 =====")
print(f"Train dataset size: {len(train_dataset)}")
print(f"val dataset size: {len(val_dataset)}")
```

```
print("===== problem3 =====")
for batch in train_dataloader:
    # batchはデータとラベルのタプル (data, labels)
    data, labels = batch
    print("Data shape:", data.shape)

    print("Labels:", labels)
```