



본 영상 교재는
2025년도 과학기술정보통신부 및 정보통신기획평가원의
SW중심대학 사업의 지원을 받아 제작되었습니다.



한국항공대학교
Korea Aerospace University

—KAU—
AI융합대학

SW중심대학



PyTorch 코딩

Dataset

PyTorch Dataset

01

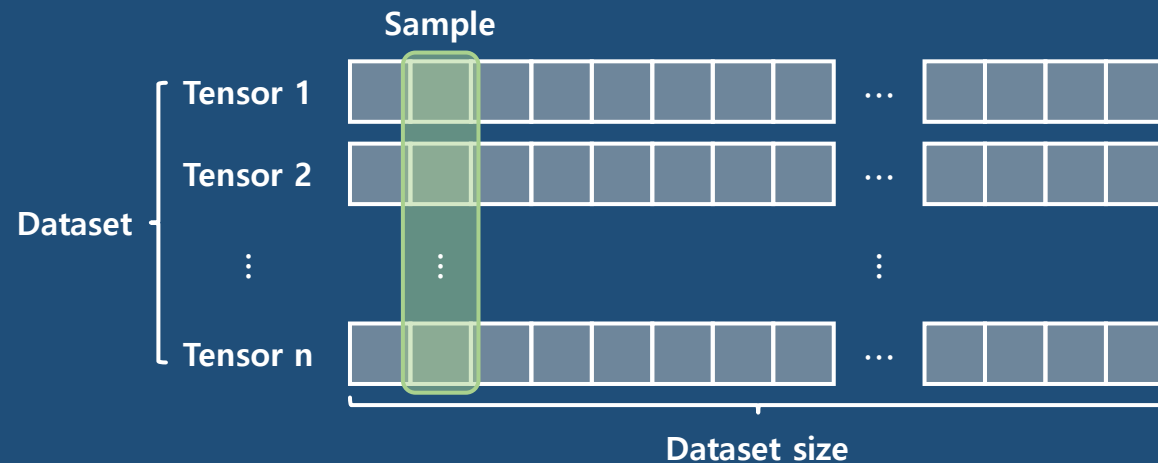
PyTorch Dataset

- PyTorch Dataset
 - PyTorch의 ANN training 및 test를 위한 데이터 관리하는 클래스
 - DataLoader 클래스와 함께 활용하여 mini-batch를 쉽게 생성할 수 있음
 - Mini-batch 관련 기능은 DataLoader 클래스가 전담
 - Dataset 클래스는 mini-batch 생성과 상관 없이 쉽게 개발할 수 있음
 - Data 처리 작업을 mini-batch 생성 및 ANN 코드와 분리하여 프로그램의 개발 및 유지 / 보수를 쉽게 해줌
- TensorDataset, ImageFolder 등, 가지고 있는 데이터를 하나의 dataset으로 쉽게 만들어주는 클래스 지원
- Torchvision, Torchaudio 등, 다양한 유명 dataset을 쉽게 활용하기 위한 클래스 제공

PyTorch Dataset

- TensorDataset

- 임의의 tensor들을 모아 하나의 dataset으로 만들어주는 클래스



- 모든 tensor들의 첫 번째 차원이 **dataset 크기**가 됨
 - 따라서 모든 tensor들의 첫 번째 차원이 **같아야** 함

PyTorch Dataset

- TensorDataset

- 2개의 tensor로 TensorDataset 인스턴스 생성하기

```

> import torch
> from torch.utils.data import TensorDataset, DataLoader

> dataset_size = 1024
> tensor1 = torch.rand([dataset_size])
> tensor2 = tensor1 * 2

> dataset = TensorDataset(tensor1, tensor2)
> print(f'Dataset size: {len(dataset)}')

> x, t = dataset[0]
> print(x, t)

> mini_batch_size = 4
> dataloader = DataLoader(dataset, mini_batch_size, shuffle = True)

> for x, t in dataloader:
>     print(x, t)
>     break
    
```

TensorDataset 클래스

임의의 tensor 생성

Tensor들을 모아 하나의 dataset으로 생성, 모든 tensor들의 첫 번째 차원이 같아야 함
Dataset 크기 확인

2개의 tensor로 이루어져 있으므로, 각 sample는 2개의 원소로 이루어짐

위에서 생성한 dataset을 data loader와 연결

2개의 tensor로 이루어져 있으므로, 각 sample는 2개의 원소로 이루어짐

PyTorch Dataset

- TensorDataset

- 2개 이상의 tensor로 TensorDataset 인스턴스 생성하기

```
> import torch
> from torch.utils.data import TensorDataset, DataLoader

> dataset_size = 1024
> tensor1 = torch.rand([dataset_size, 3])
> tensor2 = torch.rand([dataset_size, 3])
> tensor3 = torch.sum(tensor1 + tensor2, dim = 1)
> tensor4 = torch.sum(tensor1 * tensor2, dim = 1)

> dataset = TensorDataset(tensor1, tensor2, tensor3, tensor4)
> print(f'Dataset size: {len(dataset)}')

> x1, x2, t1, t2 = dataset[0]
> print(x1, x2, t1, t2)

> mini_batch_size = 4
> dataloader = DataLoader(dataset, mini_batch_size, shuffle = True)

> for x1, x2, t1, t2 in dataloader:
>     print(x1, x2, t1, t2)
>     break
```

다차원 tensor도 활용 가능

Tensor들을 모아 하나의 dataset으로 생성, 모든 tensor들의 첫 번째 차원이 같아야 함

4개의 tensor로 이루어져 있으므로, 각 sample는 4개의 원소로 이루어짐

4개의 tensor로 이루어져 있으므로, 각 sample는 4개의 원소로 이루어짐

Torchvision Dataset

02

Torchvision Dataset

- Torchvision Datasets

- PyTorch가 제공하는 이미지 처리 dataset 라이브러리

- Torchvision가 지원하는 다양한 이미지 처리 dataset

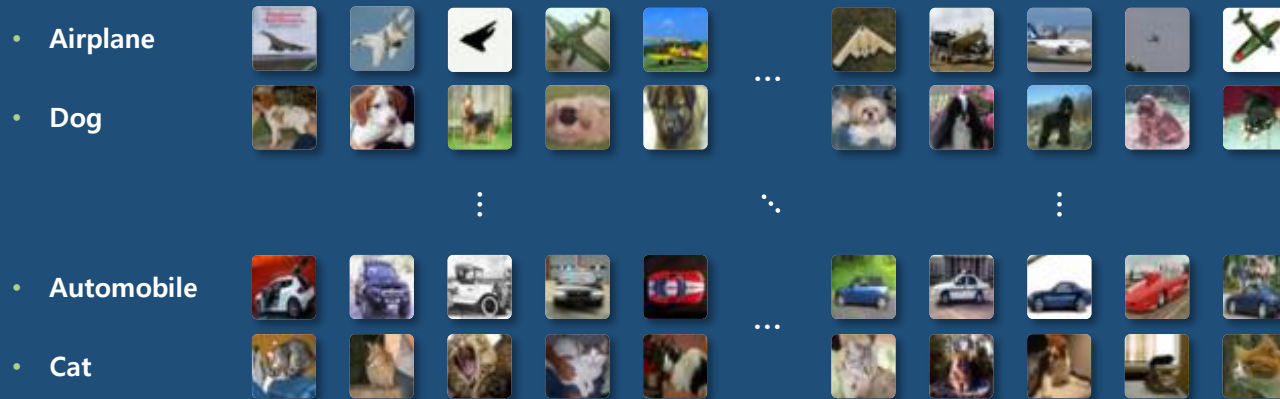
- CIFAR, ImageNet, MNIST 등 : 이미지 속 주요 물체의 종류를 맞추는 **classification** dataset
- PASCAL VOC, MS coco 등 : 이미지 속 여러 물체들의 위치, 크기, 종류를 맞추는 **object detection** dataset
- PASCAL VOC, Cityscapes 등 : 이미지 속 물체가 무엇인지 픽셀별로 맞추는 **semantic segmentation** dataset
- KITTI, CarlaStereo 등 : **스테레오** 이미지 속 동일 물체가 나타난 위치를 맞추는 **stereo matching** dataset
- KITTI, FlyingChairs 등 : **연속** 이미지 속 물체의 이동 방향과 속도를 픽셀별로 맞추는 **optical flow** dataset
- MS coco 등 : 이미지의 내용을 **자연어**로 묘사한 **image caption** dataset
- 기타 등등

- 일부 dataset의 경우 데이터 파일을 **직접** 다운 받은 뒤 활용해야 함
- 각 dataset 클래스의 활용법은 Torchvision documentation에서 확인 가능
 - Torchvision dataset : <https://pytorch.org/vision/stable/datasets.html>

Torchvision Dataset

- Torchvision Datasets

- CIFAR 100



- 32 × 32 RGB 이미지를 **100가지**의 class로 나눈 dataset
- 총 60,000개의 sample 제공(50,000개의 training sample과 10,000개의 test sample 제공)

Torchvision Dataset

- Torchvision Datasets

- CIFAR 100

```
> import matplotlib.pyplot as plt

> import torch
> from torchvision.datasets import CIFAR100 # Torchvision의 CIFAR 100 dataset 클래스

> training_dataset = CIFAR100(             # CIFAR 100 dataset 인스턴스 생성
>     root = 'data',                       # Dataset 파일 위치
>     train = True,                         # Training을 위한 dataset
>     transform = None,                    # Input image에 적용할 이미지 변환 작업
>     download = True                      # Dataset 파일이 없는 경우, 인터넷을 통해 다운로드
> )

> x, t = training_dataset[49999]           # 50,000 번째 sample

> print(type(x))                           # Pillow image 클래스, ToTensor 클래스를 적용해야 PyTorch의 ANN이 활용할 수 있음
> plt.imshow(x)
> plt.show()                               # Matplotlib를 통해 input 이미지 출력

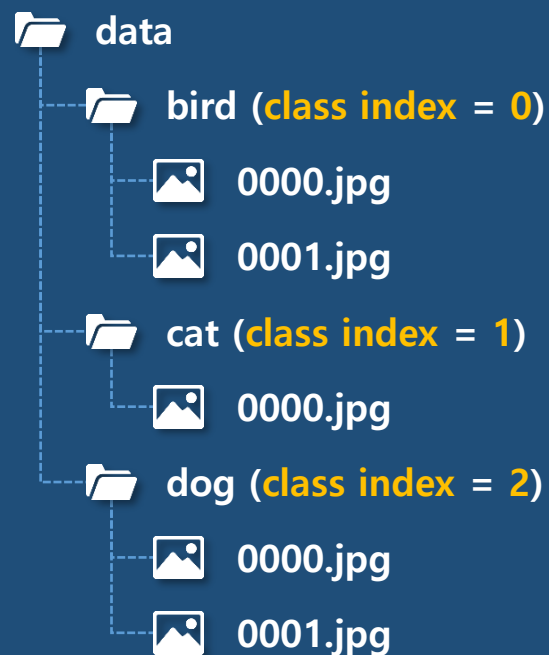
> print(type(t))
> print(t)
```

Torchvision Dataset

- Torchvision Datasets

- ImageFolder dataset

- 이미지 파일을 포함한 폴더를, **디렉토리** 구조를 통해 classification dataset을 **자동**으로 생성



Torchvision Dataset

- Torchvision Datasets

- ImageFolder dataset

```
> import matplotlib.pyplot as plt

> import torch
> from torch.utils.data import Dataset
> from torchvision.datasets import ImageFolder
> from torchvision import transforms

> input_image_transform = transforms.Compose(
>     [transforms.Resize((64, 128)),
>      transforms.CenterCrop((32, 64)),
>      transforms.ToTensor()]
> )

> image_dataset = ImageFolder(
>     root = 'data/cifar10_example',
>     transform = input_image_transform
> )

> x, t = image_dataset[0]
> tensor_to_image = transforms.ToPILImage()
> plt.imshow(tensor_to_image(x))
> plt.show()

> classes = ['airplane', 'car', 'cat', 'dog']
> print(classes[t])
```

Torchvision의 ImageFolder dataset 클래스
이미지를 PyTorch ANN이 처리하기 쉬운 형태로 변환해주는 라이브러리

여러가지 이미지 처리 작업들을 하나의 인스턴스로 통합하여, 차례대로 수행하도록 함
이미지 크기 변경 작업, 아규먼트로 이미지의 세로 및 가로 픽셀 크기 지정
이미지 주변을 잘라내는 작업, 아규먼트로 이미지에서 남길 세로 및 가로 픽셀 크기 지정

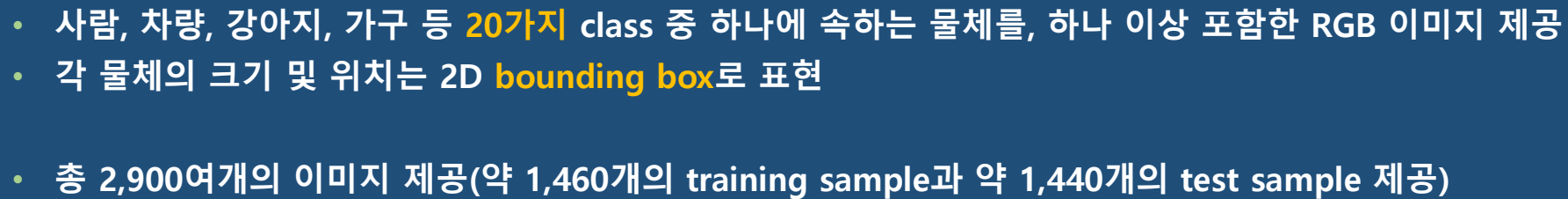
ImageFolder dataset 인스턴스 생성
읽으려는 디렉토리의 최상위 위치
각 이미지에 자동으로 적용할 transforms 인스턴스 지정

ImageFolder dataset의 sample은 2개의 원소(input image, target class index)를 가짐
PyTorch tensor를 Pillow 이미지로 변환

Input 이미지 출력

Target class label 출력

- **Torchvision Datasets**
 - PASCAL VOC detection



Torchvision Dataset

- Torchvision Datasets
 - Cityscapes



- 2,048 × 1,024 RGB 이미지의 각 픽셀이 어떤 물체인지 구분한 dataset
- 도로, 인도, 승용차, 트럭, 건물, 나무 등 약 30가지 class 제공
- 총 5,000개의 RGB 이미지 제공(약 3,000 training sample과 약 2,000개의 test sample 제공)

Torchvision Dataset

- Torchvision Datasets
 - KITTI 2015 stereo

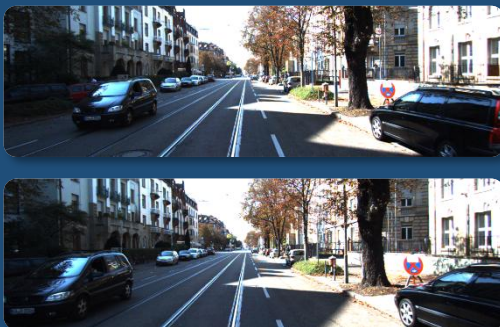


- 1,242 × 375 RGB **스테레오 이미지**로 이루어진 dataset
- 스테레오로 정렬된 두 카메라 이미지의 픽셀 별 **disparity**를 제공
 - LIDAR 등의 센서로 데이터 수집이 어려운 부분의 값은 0으로 제공됨
 - 값이 0이 아닌 픽셀의 disparity는 수식을 통해 쉽게 **depth**로 변환할 수 있음
- 총 200개의 sample 제공(100개의 training sample과 100개의 test sample 제공)

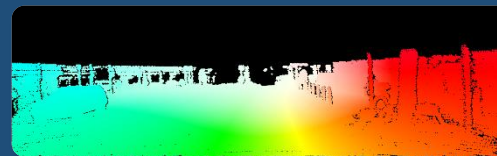
Torchvision Dataset

- Torchvision Datasets
 - KITTI flow

연속된 RGB 이미지



Optical flow



- 연속된 1,242 × 375 RGB 이미지로 이루어진 dataset
- 연속 프레임에서, 각 픽셀들이 가로 및 세로로 몇 픽셀 씩 움직였는지(optical flow)를 제공
 - Target은 가로 및 세로로 이동한 양을 나타내는 channel 2개로 이루어짐
- 총 400개의 RGB sample 제공(200개의 training sample과 200개의 test sample 제공)

Torchvision Dataset

- Torchvision Datasets
 - MS coco caption



The man at bat readies to swing at the pitch while the umpire looks on



A large bus sitting next to a very tall building

- RGB 이미지의 내용을 설명하는 영어 **caption**으로 이루어진 dataset
 - 하나의 이미지를 설명하는 **여러 개**의 caption이 있을 수 있음
- 약 16만개의 RGB 이미지와 약 100만개의 caption을 제공
 - 16만개의 RGB 이미지 중, 12만개는 training 및 validation, 나머지는 test용
 - 100만개의 영어 caption 중 61만개는 training 및 validation, 나머지는 test용

MS Excel 파일을 위한 맞춤형 Dataset

03

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습
 - Cell 1 : 맞춤형 dataset 및 MS Excel 라이브러리 불러오기

```
> import pandas as pd                                # MS Excel 파일을 읽어오기 위한 라이브러리
> import torch
> print(f'PyTorch version: {torch.__version__}')

> from torch import nn
> from torch.utils.data import Dataset, DataLoader    # 맞춤형 dataset 클래스
```

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습
 - Cell 3-1 : MS Excel 파일을 읽어와 training dataset으로 만들기

```
> class IrisTrainDataset(Dataset):
>     def __init__(self):
>         data_frame = pd.read_excel('data/iris_example/iris_train.xlsx')
>         self.input_data_frame = data_frame[[
>             'sepalLength',
>             'sepalWidth',
>             'petalLength',
>             'petalWidth'
>         ]]
>         self.target_data_frame = data_frame['irisClass']

>     def __len__(self):
>         return len(self.input_data_frame)

>     def __getitem__(self, index):
>         x = torch.tensor(
>             self.input_data_frame.iloc[(index)],
>             dtype = torch.float32
>         )
>         t = torch.tensor(
>             self.target_data_frame.iloc[index],
>             dtype = torch.int64
>         )

>         return x, t
```

PyTorch dataset 클래스를 상속받아 맞춤형 dataset 클래스 생성
Dataset을 초기화하는 작업 수행, 반드시 구현해야 함
Pandas 라이브러리를 통해 training data 파일을 읽음

Dataset의 크기를 return, 반드시 구현해야 함

Index를 통해 특정 sample을 추출해 return, 반드시 구현해야 함

PyTorch ANN은 input으로 32bit float를 사용하므로 이에 맞춰줌

PyTorch는 classification target으로 64bit integer를 사용하므로 이에 맞춰줌

Input과 target을 동시에 return

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습
 - Cell 3-2 : MS Excel 파일을 읽어와 test dataset으로 만들기

```
> class IrisTestDataset(Dataset):
>     def __init__(self):
>         data_frame = pd.read_excel('data/iris_example/iris_test.xlsx')    # Pandas 라이브러리를 통해 test data 파일을 읽음
>         self.input_data_frame = data_frame[[
>             'sepalLength',
>             'sepalWidth',
>             'petalLength',
>             'petalWidth'
>         ]]
>         self.target_data_frame = data_frame['irisClass']
>
>     def __len__(self):
>         return len(self.input_data_frame)
>
>     def __getitem__(self, index):
>         x = torch.tensor(
>             self.input_data_frame.iloc[(index)],
>             dtype = torch.float32
>         )
>         t = torch.tensor(
>             self.target_data_frame.iloc[index],
>             dtype = torch.int64
>         )
>
>         return x, t
```

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습
 - Cell 3-3 : Dataset 및 data loader 인스턴스 생성하기

```
> training_dataset = IrisTrainDataset()    # 3-1에서 만든 training dataset 인스턴스 생성
> test_dataset     = IrisTestDataset()     # 3-2에서 만든 test dataset 인스턴스 생성

> mini_batch_size  = 64
> training_data_loader = DataLoader(training_dataset, batch_size = mini_batch_size , shuffle = True)
> test_data_loader   = DataLoader(test_dataset, batch_size = mini_batch_size) # Test dataset의 경우 크기가 64보다 작지만 문제 없음

> print(f'Training dataset size: {len(training_dataset)}')
> ...
```

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습
 - Cell 4-1 : ANN architecture 정의하기

```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def __init__(self):
>         super().__init__()
>
>         self.linear_stack = nn.Sequential(
>             nn.Linear(4, 64),           # Dataset의 input dimension에 맞게 첫 layer dimension 수정
>             nn.Tanh(),
>             nn.Linear(64, 128),
>             nn.Tanh(),
>             nn.Linear(128, 128),
>             nn.Tanh(),
>             nn.Linear(128, 64),
>             nn.Tanh(),
>             nn.Linear(64, 8),
>             nn.Tanh(),
>             nn.Linear(8, 3),           # Dataset의 target dimension에 맞게 마지막 layer dimension 수정
>             nn.Softmax(dim = 1)
>         )
>
>         self.linear_stack.apply(self.weight_initializer)
>
>     ...
```


MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습
 - Cell 4-2 : ANN feed-forward 작업 정의하기

```
> class NeuralNetwork(nn.Module):  
>     ...  
  
>     def forward(self, x):  
>         y = self.linear_stack(x)  
  
>         return y
```

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습

- Cell 4-3 : Learning rate 수정하기

```
> model = NeuralNetwork().to(device)

> optimizer = torch.optim.Adam(
>     model.parameters(),
>     lr = 2.5e-4,                # Dataset 및 ANN architecture가 달라졌으므로 learning rate를 변경해야 할 수 있음
>     betas = (0.9, 0.999),
>     weight_decay = 1e-4
> )
```

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습
 - Cell 5 : 개별 input에 대한 inference 작업 정의하기

```
> def inference(device, data, model):  
>     x = data.view(1, 4) # 변경된 ANN architecture에 맞게 input dimension 변경  
  
>     model.eval()  
>     with torch.no_grad():  
>         x = x.to(device)  
>         y = model(x)  
  
>     return y
```

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습

- Cell 6 : 최대 epoch 횟수 수정하기

```
> max_epoch = 5 # Dataset 및 ANN architecture가 달라졌으므로 최대 epoch 횟수를 수정해야 할 수 있음
> for t in range(max_epoch):
>     print(f'Epoch {t + 1 :>3d} / {max_epoch}')
>     ...
```

MS Excel 파일을 위한 맞춤형 Dataset

- MS Excel 파일을 위한 맞춤형 dataset 실습

- Cell 7 : 임의의 sample에 대한 inference 성능 확인하기

```
> import random

> model = NeuralNetwork().to(device)
> model.load_state_dict(torch.load('model.pth'))

> test_sample_index = random.randint(0, len(test_dataset) - 1)
> x, t = test_dataset[test_sample_index]

> y = inference(device, x, model)

> classes = ['Setosa', 'Versicolor', 'Virginica'] # Iris dataset 에 맞게 class 이름 변경
> predicted, actual = classes[y.argmax(dim = 1)], classes[t]
> print('Random sample inference')
> print(f' Predicted: "{predicted}", Actual: "{actual}"')
```

이미지 파일을 위한 맞춤형 Dataset

04

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 1 : 맞춤형 dataset 및 이미지 라이브러리 불러오기

```
> import os                                # 파일 리스트를 얻기 위한 라이브러리
> import numpy as np                       # Disparity 파일을 읽기 위한 라이브러리
> from PIL import Image                   # 이미지 파일을 읽기 위한 클래스

> import torch
> print(f'PyTorch version: {torch.__version__}')

> from torch import nn
> from torch.utils.data import Dataset, DataLoader    # 맞춤형 dataset 클래스
> from torchvision import transforms
> from torchvision.transforms import CenterCrop, ToTensor    # Input 이미지 및 target disparity size를 통일하기 위한 이미지 처리 클래스
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 3-1 : Data 파일 리스트 불러오기

```
> class Kitti2015StereoDataset(Dataset):
>     def __init__(self, training):
>         if training == True:
>             data_path = 'data/kitti_2015_stereo_example/training'
>             print('Loading training dataset on RAM')
>         else:
>             data_path = 'data/kitti_2015_stereo_example/validation'
>             print('Loading test dataset on RAM')
>
>         left_image_file_list = os.listdir(f'{data_path}/left_image')
>         right_image_file_list = os.listdir(f'{data_path}/right_image')
>         disparity_file_list = os.listdir(f'{data_path}/disparity')
>         ...
```

```
# PyTorch dataset 클래스를 상속받아 맞춤형 dataset 클래스 생성
# 아규먼트를 통해 training 및 test dataset을 구분하여 생성
# Training dataset을 불러올 때
```

```
# Validation dataset을 불러올 때
```

```
# 특정 폴더에 있는 파일 이름 리스트
```


이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 3-2 : 스테레오 이미지 파일을 읽어와 dataset으로 만들기

```
> class Kitti2015StereoDataset(Dataset):
>     def __init__(self, training):
>         ...
>
>         image_to_frame_transform = transforms.Compose(
>             [CenterCrop((256, 512)),
>              ToTensor()]
>         )
>
>         self.left_image_memory = []
>         for file_name in left_image_file_list:
>             image = Image.open(f'{data_path}/left_image/{file_name}')
>             tensor = image_to_frame_transform(image)
>             self.left_image_memory.append(tensor)
>
>         self.right_image_memory = []
>         for file_name in right_image_file_list:
>             image = Image.open(f'{data_path}/right_image/{file_name}')
>             tensor = image_to_frame_transform(image)
>             self.right_image_memory.append(tensor)
>         ...
```

모든 input 이미지의 크기가 세로 256px, 가로 512px이 되도록 이미지 주변을 자름

Pillow Image 클래스를 통해 이미지 파일 열기
이미지 전처리
이미지 tensor들을 RAM에 저장해, ANN training 속도를 높임

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 3-3 : Disparity 파일을 읽어와 dataset으로 만들기

```
> class Kitti2015StereoDataset(Dataset):
>     def __init__(self, training):
>         ...
>
>         self.disparity_memory = []
>         self.valid_mask_memory = []
>         for file_name in disparity_file_list:
>             array = np.asarray(Image.open(f'{data_path}/disparity/{file_name}')) / 256.0
>             # 일반 이미지가 아니므로, Pillow Image로 활용하지 않고 Numpy array로 변환
>             # Numpy array로 변환하는 과정에서, dimension이 [W, H]에서 [H, W]로 바뀜
>             # 실제 disparity 값에 256을 곱한 값이 저장되어 있으므로 256.0으로 나눔
>
>             center_crop = CenterCrop((256, 512))
>             tensor = center_crop(
>                 torch.Tensor(array).
>                 unsqueeze(0)
>             )
>             self.disparity_memory.append(tensor)
>
>             valid_mask = tensor.bool().float()
>             self.valid_mask_memory.append(valid_mask)
>         ...
```

Torchvision의 이미지 처리 작업을 적용하기 위해 Numpy array를 tensor로 변환
Disparity 파일은 channel 차원이 없으므로, PyTorch ANN을 위해 [H, W]를 [C, H, W]로 변환
Disparity tensor들을 RAM에 저장해, ANN training 속도를 높임
Target에서 측정 값이 있는 픽셀만 1.0으로, 나머지는 0.0으로 변환하여 mask 생성

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 3-4 : Dataset의 sample 추출 기능 추가하기

```
> class Kitti2015StereoDataset(Dataset):
>     ...
>
>     def __len__(self):
>         return len(self.left_image_memory)
>
>     def __getitem__(self, index):
>         left_image_memory    = self.left_image_memory[index]
>         right_image_memory   = self.right_image_memory[index]
>         disparity_memory     = self.disparity_memory[index]
>         valid_mask_memory    = self.valid_mask_memory[index]
>
>         return left_image_memory, right_image_memory, disparity_memory, valid_mask_memory # 각 sample이 4개의 원소를 가짐
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 3-5 : Dataset 및 dataloader 인스턴스 만들기

```

> ...

> print('-----')
> training_dataset      = Kitti2015StereoDataset(training = True)      # 아규먼트를 통해 training dataset 인스턴스 생성
> test_dataset          = Kitti2015StereoDataset(training = False)    # 아규먼트를 통해 test dataset 인스턴스 생성

> mini_batch_size       = 16                                          # 이미지 크기가 커진 만큼 VRAM 공간을 고려하여 mini-batch 크기를 줄여야 할 수 있음
> training_data_loader  = DataLoader(training_dataset, batch_size = mini_batch_size, shuffle = True)
> test_data_loader      = DataLoader(test_dataset, batch_size = mini_batch_size)

> ...

> for x_left, x_right, t, _ in training_data_loader:                  # 각 sample이 4개의 원소를 가짐
>     print('-----')
>     print(f'Shape of left input:      {x_left.shape} {x_left.dtype}')
>     print(f'Shape of right input:     {x_right.shape} {x_right.dtype}')
>     print(f'Shape of target:          {t.shape} {t.dtype}')
>     print('-----')
>     break
    
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 4-1 : ANN layer 선언 및 초기화하기

```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def __init__(self):
>         super().__init__()
>
>         self.cnn_stack1 = nn.Sequential(
>             nn.Conv2d(in_channels = 6, out_channels = 16, kernel_size = 5, stride = 2, padding = 2, padding_mode = 'zeros'),
>             nn.GELU(), # ReLU를 개선시켜 모든 구간에서 연속적인 activation function
>             nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 5, stride = 2, padding = 2, padding_mode = 'zeros'),
>             nn.GELU()
>         )
>
>         self.cnn_stack2 = nn.Sequential(
>             nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size = 5, stride = 2, padding = 2, padding_mode = 'zeros'),
>             nn.GELU(),
>             nn.Conv2d(in_channels = 64, out_channels = 128, kernel_size = 5, stride = 2, padding = 2, padding_mode = 'zeros'),
>             nn.GELU(),
>             nn.Conv2d(in_channels = 128, out_channels = 256, kernel_size = 5, stride = 2, padding = 2, padding_mode = 'zeros'),
>             nn.GELU()
>         )
>         ...
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 4-2 : ANN layer 선언 및 초기화하기

```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def __init__(self):
>         ...
>
>         self.decnnc_stack1 = nn.Sequential(
>             nn.ConvTranspose2d(in_channels = 256, out_channels = 128, kernel_size = 2, stride = 2), # Convolutional layer와 반대로, input의 size를 늘려줌
>             nn.GELU(),
>             nn.ConvTranspose2d(in_channels = 128, out_channels = 64, kernel_size = 2, stride = 2),
>             nn.GELU(),
>             nn.ConvTranspose2d(in_channels = 64, out_channels = 32, kernel_size = 2, stride = 2),
>             nn.GELU(),
>         )
>
>         self.decnnc_stack2 = nn.Sequential(
>             nn.ConvTranspose2d(in_channels = 32, out_channels = 8, kernel_size = 2, stride = 2),
>             nn.GELU(),
>             nn.ConvTranspose2d(in_channels = 8, out_channels = 1, kernel_size = 2, stride = 2) # 별도의 activation function이 없는 regression output
>         )
>
>         self.cnn_stack1.apply(self.weight_initializer)
>         self.cnn_stack2.apply(self.weight_initializer)
>         self.decnnc_stack1.apply(self.weight_initializer)
>         self.decnnc_stack2.apply(self.weight_initializer)
>         ...
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 4-3 : ANN feed-forward 작업 정의하기

```

> class NeuralNetwork(nn.Module):
>     ...
>
>     def forward(self, x_left, x_right):      # 이 ANN은 2개의 input을 받음
>         x = torch.concat(                  # 여러 tensor들을 하나로 concatenate
>             [x_left, x_right],              # Concatenate할 tensor 리스트
>             dim = 1                          # 합칠 차원 index, 양쪽 카메라의 이미지를 [B, C, H, W] 중 C 차원으로 합침
>         )
>
>         cnn1 = self.cnn_stack1(x)
>         cnn2 = self.cnn_stack2(cnn1)
>
>         decnn1 = self.decnn_stack1(cnn2)
>         decnn2 = self.decnn_stack2(decnn1 + cnn1)
>
>         if not self.training:              # ANN이 training 중이 아닐 때
>             decnn2 = torch.clamp(decnn2, 0.0, 256.0) # Tensor 각 원소의 최소 및 최대값을 지정
>
>         return decnn2
    
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습

- Cell 4-4 : Learning rate 수정하기

```
> model = NeuralNetwork().to(device)

> optimizer = torch.optim.Adam(
>     model.parameters(),
>     lr = 1e-3,
>     betas = (0.9, 0.999),
>     weight_decay = 1e-4
> )
```

Dataset 및 ANN architecture가 달라졌으므로 learning rate를 변경해야 할 수 있음

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 5-1 : Training epoch 정의하기

```

> def train(device, data_loader, model, optimizer):
>     total_loss = 0

>     model.train()
>     for x_left, x_right, t, mask in data_loader:           # 각 sample이 4개의 원소를 가짐
>         x_left      = x_left.to(device)
>         x_right     = x_right.to(device)

>         y           = model(x_left, x_right)               # 이 ANN은 2개의 input을 받음
>         t           = t.to(y.device)

>         mask        = mask.to(device)
>         y           = y * mask                             # Output에 mask를 곱해, 비교할 target이 있는 픽셀만 남김
>         loss        = torch.nn.functional.mse_loss(y, t)   # 이번 ANN은 Regression output이므로 mean squared error 활용

>         ...

>         mini_batch_size = x_left.shape[0]                 # Input mini-batch를 통해 mini-batch 크기 확인
>         total_loss += loss.item() * mini_batch_size

>         ...

>     return average_loss
    
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 5-2 : Test 작업 정의하기

```
> def test(device, data_loader, model):
>     total_loss = 0
>     # total_correct = 0                                # 정답률 관련 코드 삭제

>     model.eval()
>     with torch.no_grad():
>         for x_left, x_right, t, mask in data_loader :    # Sample의 원소 개수가 달라졌으므로 수정
>             x_left = x_left.to(device)
>             x_right = x_right.to(device)

>             y      = model(x_left, x_right)              # 이 ANN은 2개의 input을 받음
>             t      = t.to(y.device)

>             mask   = mask.to(device)
>             y      = y * mask                            # Output에 mask를 곱해, 비교할 target이 있는 픽셀만 남김
>             loss   = torch.nn.functional.mse_loss(y, t)  # 이번 ANN은 Regression output이므로 mean squared error 활용

>             mini_batch_size = x_left.shape[0]
>             total_loss += loss.item() * mini_batch_size
>             # total_correct += (y.argmax(dim = 1) == t) ... # 정답률 관련 코드 삭제

>     dataset_size = len(data_loader.dataset)
>     average_loss = total_loss / dataset_size
>     # accuracy = total_correct / dataset_size            # 정답률 관련 코드 삭제

>     return average_loss #, accuracy                     # 정답률 관련 코드 삭제
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 5-3 : 개별 input에 대한 inference 작업 정의하기

```
> def inference(device, x_left, x_right, model):
>     # 이미지 처리를 위한 코드          # 일반 이미지인 x_left와 x_right에 CenterCrop 및 ToTensor 클래스 적용 작업 필요
>
>     x_left  = x_left.view(1, 3, 256, 512)
>     x_right = x_right.view(1, 3, 256, 512)
>
>     model.eval()
>     with torch.no_grad():
>         x_left  = x_left.to(device)
>         x_right = x_right.to(device)
>
>         y      = model(x_left, x_right)
>         y      = y.view(256, 512)      # Disparity 파일 형식에 맞게 channel 차원 제거
>
>     return y
```

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 6 : Training loop 정의하기

```
> max_epoch = 500
> for t in range(max_epoch):
>     if t % 50 == 0 or t == max_epoch - 1:
>         print(f'Epoch {t + 1 :>3d} / {max_epoch}')

>     training_loss = train(device, training_data_loader, model, optimizer)
>     if t % 50 == 0 or t == max_epoch - 1:
>         print(' Training progress')
>         print(f' Average loss: {training_loss :>8f}')

>     if t % 50 == 0 or t == max_epoch - 1:
>         test_loss = test(device, test_data_loader, model)
>         print(' Validation performance')
>         print(f' Average loss: {test_loss :>8f}')
>         # print(f' Accuracy: {(100 * test_accuracy) :>0.2f}%', end='')

>     if t + 1 < max_epoch:
>         print()

> torch.save(model.state_dict(), 'model.pth')
> ...
```

쉬운 training이 아니기 때문에 최대 epoch 횟수를 크게 늘림

최대 epoch 횟수가 늘어난 만큼, 간격을 두고 training 진행 상황을 출력

정답률 관련 코드 삭제

이미지 파일을 위한 맞춤형 Dataset

- 이미지 파일을 위한 맞춤형 dataset 실습
 - Cell 7 : 임의의 sample에 대한 inference 성능 확인하기

```
> import random
> import matplotlib.pyplot as plt

> model = NeuralNetwork().to(device)
> model.load_state_dict(torch.load('model.pth', weights_only = True))

> test_sample_index = random.randint(0, len(test_dataset) - 1)
> x_left, x_right, t, _ = test_dataset[test_sample_index]
> y = inference(device, x_left, x_right, model)

> tensor_to_image = transforms.ToPILImage()
> left_image = tensor_to_image(x_left)
> right_image = tensor_to_image(x_right)
> target_image = t.view(256, 512)
> output_image = y.to('cpu')

> figure, axes = plt.subplots(4, 1, figsize = (20, 30))
> axes[0].imshow(left_image)
> axes[1].imshow(right_image)
> axes[2].imshow(target_image, cmap = 'gray')
> axes[3].imshow(output_image, cmap = 'gray')
> plt.savefig('result.png')
> plt.show()
```

여러 이미지를 하나의 그래프로 합치기 위한 라이브러리

Sample의 원소 개수가 달라졌으므로 수정

PyTorch tensor를 Pillow 이미지로 변환

Grayscale 이미지를 출력하려면 channel 차원을 제거해야 함
VRAM에 저장된 output을 Numpy로 처리하려면 RAM으로 복사해와야 함

실행 결과를 확인하기 위해 disparity를 이미지로 출력

Input 스테레오 이미지, target 및 output disparity를 이미지 파일로 저장
Input 스테레오 이미지, target 및 output disparity를 화면에 출력

Have a nice day!

