09. Custom Dataset

AILab
Hanyang Univ.

오늘 실습 내용

1. Pytorch Custom Dataset

1. Pytorch Custom Dataset

Pytorch Custom Dataset

- Dataset and Dataloader

```
import torchvision
import torchvision.transforms as transforms

train_dataset = torchvision.datasets.MNIST(root="MNIST_data/",
                    train=True,
                    transform=transforms.ToTensor(),
                    download=True)
test_dataset = torchvision.datasets.MNIST(root="MNIST_data/",
                    train=False,
                    transform=transforms.ToTensor(),
                    download=True)
```

```
batch_size = 128

train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)
```

- Pytorch가 내가 원하는 데이터 셋을 제공하지 않는다면,,,?

Pytorch Custom Dataset

- Custom Dataset
- torch.nn.utils.Dataset의 하위 클래스

- 반드시 필요한 세가지 function
  - __init__
    - run once when instantiating the Dataset object
  - __len__
    - 데이터 셋의 길이 정의
  - __getitem__
    - Dataloader가 학습되어
    - loads and returns a sample from the dataset at the given index idx

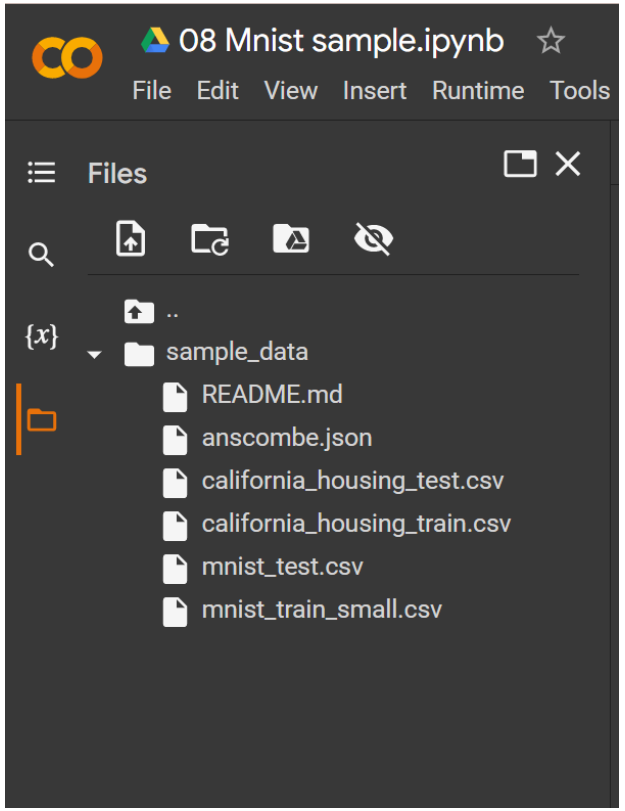
- 이미지 데이터에서 특히 유용함
  - 이미지에서 PIL image를 일일이 읽어와야하므로
  - __getitem__이 매우 유용

```python
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                                self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:]
        landmarks = np.array([landmarks])
        landmarks = landmarks.astype('float').reshape(-1, 2)
        sample = {'image': image, 'landmarks': landmarks}

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

        return sample
```

https://pytorch.org/tutorials/beginner/data_loading_tutorial.html#dataset-class

Pytorch Custom Dataset

- Example 1) MNIST



A	B	C	D	E	F	G	H	I
6	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

라벨 이미지 28*28 으로 나타냄

Pytorch Custom Dataset

- Load data

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from torch.utils.data import Dataset, DataLoader

import pandas as pd
df_train = pd.read_csv('./sample_data/mnist_train_small.csv', header=None)
df_test = pd.read_csv('./sample_data/mnist_test.csv', header=None)

train_labels = df_train.iloc[:, 0]
train_images = df_train.iloc[:, 1:]
```

Pytorch Custom Dataset

- Dataset 정의

```
class MNISTDataset(Dataset):
    def __init__(self, images, labels=None, transforms=None):
        self.images = images
        self.labels = labels
        self.transforms = transforms

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

    def __getitem__(self, idx):

        image = self.images.iloc[idx, :]
        image = np.asarray(image).astype(np.uint8).reshape(28, 28)
        label = self.labels[idx]

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

        return image, torch.tensor(label)
```

Pytorch Custom Dataset

- Dataset, Dataloader

```
import torchvision.transforms as transforms
transform = transforms.Compose([
                                transforms.ToTensor(),
                                ])

train_dataset = MNISTDataset(train_images, train_labels, transform)

train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)
```

Pytorch Custom Dataset

- Custom Dataset
- torch.nn.utils.Dataset의 하위 클래스

- 반드시 필요한 세가지 function
  - __init__
    - run once when instantiating the Dataset object
  - __len__
    - 데이터 셋의 길이 정의
  - __getitem__
    - Dataset
    - loads and returns a sample from the dataset at the given index idx

- 이미지 데이터에서 특히 유용함
  - 이미지에서 PIL image를 일일이 읽어와야 하므로
  - __getitem__이 매우 유용

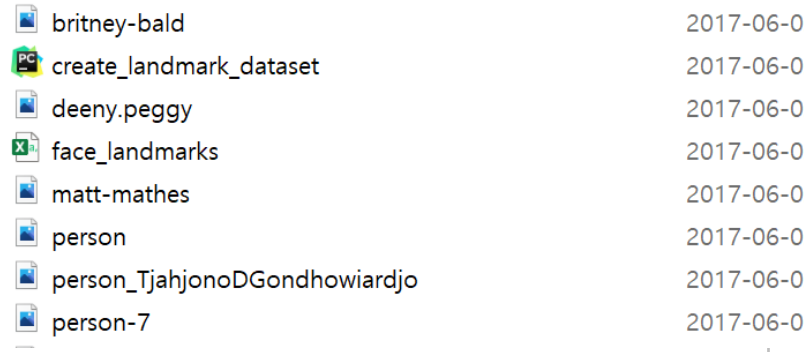
```python
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                                self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:]
        landmarks = np.array([landmarks])
        landmarks = landmarks.astype('float').reshape(-1, 2)
        sample = {'image': image, 'landmarks': landmarks}

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

        return sample
```

https://pytorch.org/tutorials/beginner/data_loading_tutorial.html#dataset-class

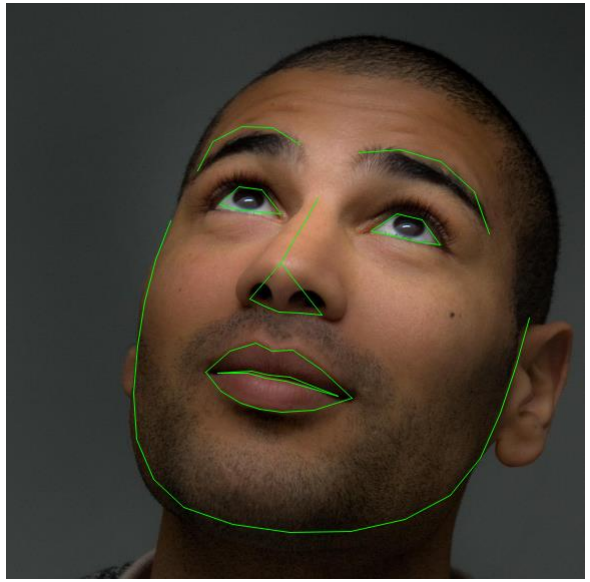
Pytorch Custom Dataset

- Example 2) Face Data
- 앞의 웹페이지에서 해당 부분에서 파일 다운로드 후 구글 드라이브에 업로드!

> NOTE
>
> Download the dataset from here so that the images are in a directory named 'data/faces/'. This dataset was actually generated by applying excellent dlib's pose estimation on a few images from imagenet tagged as 'face'.



- 얼굴의 표정을 표현한 데이터셋, 68개의 다른 점으로 나타내어 있다.



점을 표현한 파일 : face_landmarks.csv (x,y)로 이루어짐

	A	B	C	D	E	F	G	H	I	J	K	L	M
	image_name	part_0_x	part_0_y	part_1_x	part_1_y	part_2_x	part_2_y	part_3_x	part_3_y	part_4_x	part_4_y	part_5_x	part_5_y
	0805personali01.jpg	27	83	27	98	29	113	33	127	39	139	49	150
	1084239450_e76e00b7e7.jpg	70	236	71	257	75	278	82	299	90	320	100	340
	10comm-decarlo.jpg	66	114	65	128	67	142	68	156	72	169	80	180
	110276240_bec305da91.jpg	42	140	45	161	51	180	61	200	73	220	89	238
	1198_0_861.jpg	138	392	141	427	145	464	152	501	166	536	186	567
	137341995_e7c48e9a75.jpg	-4	129	1	178	10	231	30	276	59	314	98	349
	1383023626_8a49e4879a.jpg	0	48	-6	94	-10	141	-8	186	6	228	26	265

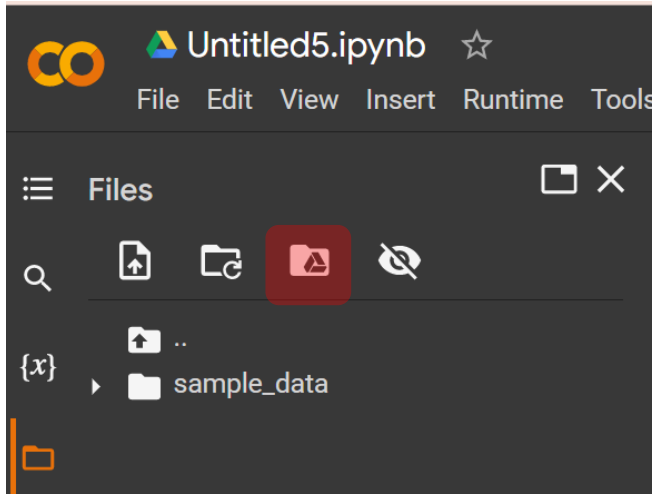
이미지 파일이름

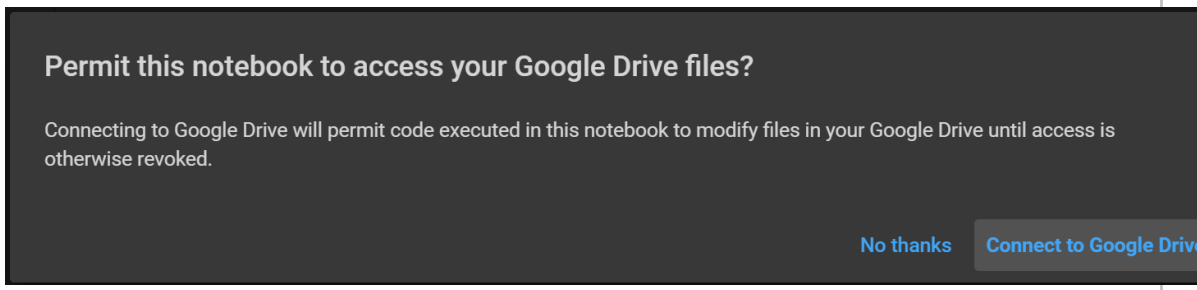
68개의 점 위치

Pytorch Custom Dataset

- Colab에 파일 올리기
  - Mount Google Drive

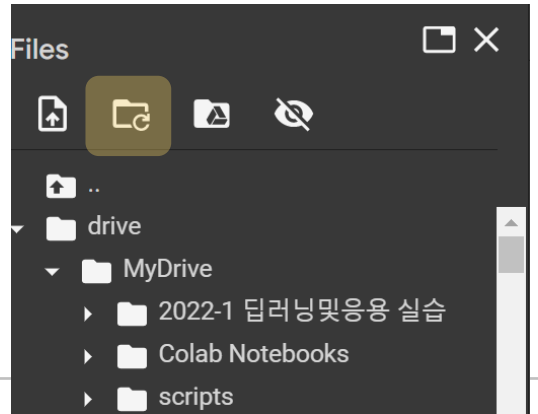
1. 구글 드라이브와 연동하기






```
from google.colab import drive
drive.mount('/content/drive')
```

2. Refresh를 눌러 path확인



3. 추가한 폴더 path 작성해 놓기

```
path='drive/MyDrive/Colab Notebooks/data/faces/'
```

Pytorch Custom Dataset

- Dataset

```
landmarks_frame = pd.read_csv(path+'/face_landmarks.csv')

n = 65
img_name = landmarks_frame.iloc[n, 0]
landmarks = landmarks_frame.iloc[n, 1:]
landmarks = np.asarray(landmarks)
landmarks = landmarks.astype('float').reshape(-1, 2)

print('Image name: {}'.format(img_name))
print('Landmarks shape: {}'.format(landmarks.shape))
print('First 4 Landmarks: {}'.format(landmarks[:4]))
```

```
(136,)
Image name: person-7.jpg
Landmarks shape: (68, 2)
First 4 Landmarks: [[32. 65.]
 [33. 76.]
 [34. 86.]
 [34. 97.]]
```

점을 표현한 파일 읽기

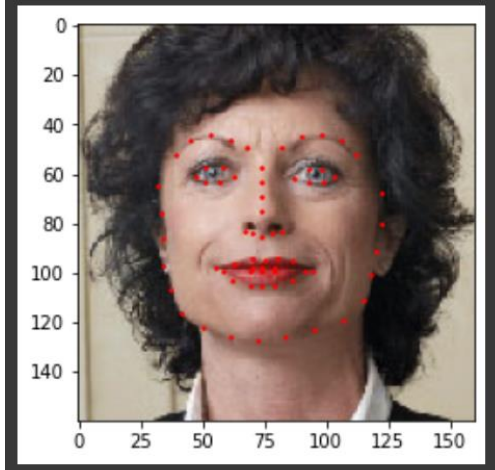
66번째 파일 샘플

일렬로 된 (x,y) 점을 (68,2)로 바꿔준다.

Pytorch Custom Dataset

- Dataset

```
def show_landmarks(image, landmarks):
    """Show image with landmarks"""
    plt.imshow(image)
    plt.scatter(landmarks[:, 0], landmarks[:, 1], s=10, marker='.', c='r')
    plt.pause(0.001)  # pause a bit so that plots are updated

plt.figure()
show_landmarks(io.imread(os.path.join(path, img_name)),
               landmarks)
plt.show()
```



(x, y)로 이미지 위에 점을 보이게 함

Pytorch Custom Dataset

- Dataset Class 정의

```
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                                self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:]
        landmarks = np.array([landmarks])
        landmarks = landmarks.astype('float').reshape(-1, 2)
        sample = {'image': image, 'landmarks': landmarks}

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

        return sample
```

annoatation.csv의 첫번째 값인 파일 이름과
루트 path 합쳐서 image 읽음

Dictionary 형태로 data return하게 정의
다른 방법으로 `return image, landmarks`
image, label 따로 받게함

Pytorch Custom Dataset

- Dataset 생성

```
]  face_dataset = FaceLandmarksDataset(csv_file=path+'face_landmarks.csv',
                                        root_dir=path)

   fig = plt.figure()

   for i in range(len(face_dataset)):
       sample = face_dataset[i]

       print(i, sample['image'].shape, sample['landmarks'].shape)

       ax = plt.subplot(1, 4, i + 1)
       plt.tight_layout()
       ax.set_title('Sample #{}'.format(i))
       ax.axis('off')
       show_landmarks(**sample)

       if i == 3:
           plt.show()
           break
```

Dictionary 형태로 data return하게 정의

```
return image, landmarks
→ image, landamarks = face_dataset[i]
```

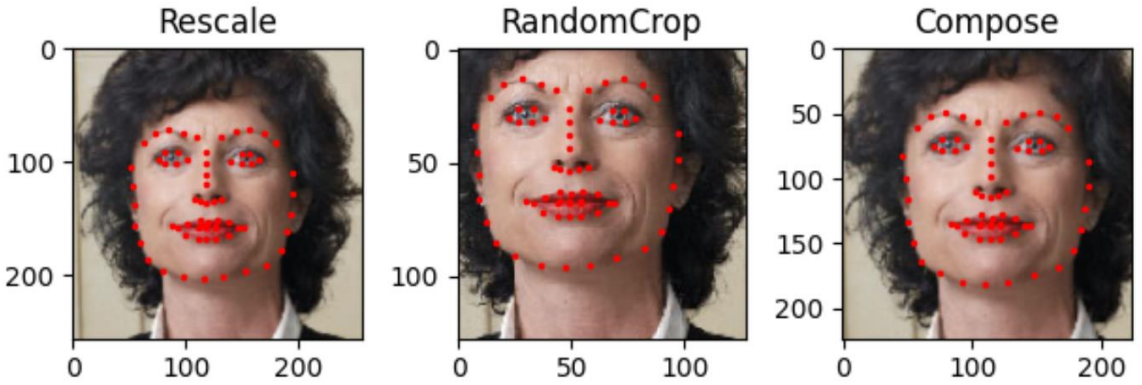
Pytorch Custom Dataset

- Transform

```python
class Rescale(object):
    """Rescale the image in a sample to a given size.

    Args:
        output_size (tuple or int): Desired output size. If tuple, output is
            matched to output_size. If int, smaller of image edges is matched
            to output_size keeping aspect ratio the same.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, sample):
        image, landmarks = sample['image'], sample['landmarks']

        h, w = image.shape[:2]
        if isinstance(self.output_size, int):
            if h > w:
                new_h, new_w = self.output_size * h / w, self.output_size
            else:
                new_h, new_w = self.output_size, self.output_size * w / h
        else:
            new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        img = transform.resize(image, (new_h, new_w))

        # h and w are swapped for landmarks because for images,
        # x and y axes are axis 1 and 0 respectively
        landmarks = landmarks * [new_w / w, new_h / h]

        return {'image': img, 'landmarks': landmarks}


class RandomCrop(object):
    """Crop randomly the image in a sample.

    Args:
        output_size (tuple or int): Desired output size. If int, square crop
            is made.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        if isinstance(output_size, int):
            self.output_size = (output_size, output_size)
        else:
            assert len(output_size) == 2
            self.output_size = output_size

    def __call__(self, sample):
        image, landmarks = sample['image'], sample['landmarks']

        h, w = image.shape[:2]
        new_h, new_w = self.output_size

        top = np.random.randint(0, h - new_h)
        left = np.random.randint(0, w - new_w)

        image = image[top: top + new_h,
                      left: left + new_w]

        landmarks = landmarks - [left, top]

        return {'image': image, 'landmarks': landmarks}


class ToTensor(object):
    """Convert ndarrays in sample to Tensors."""

    def __call__(self, sample):
        image, landmarks = sample['image'], sample['landmarks']

        # swap color axis because
        # numpy image: H x W x C
        # torch image: C x H x W
        image = image.transpose((2, 0, 1))
        return {'image': torch.from_numpy(image),
                'landmarks': torch.from_numpy(landmarks)}
```

Rescale : 크기 변환 (dataloader는 크기가 일정한 데이터 셋만 처리함)
RandomCrop : 사진을 random하게 자름 (데이터 수를 늘리는 작업)
ToTensor : dataloader는 tensor type만 처리

각 transform 정의가능 object class inherit
__call__ 수정



Pytorch Custom Dataset

- Transform Dataset

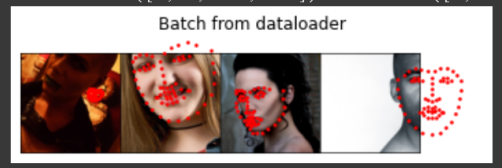
```
transformed_dataset = FaceLandmarksDataset(csv_file=path+'face_landmarks.csv',
                                           root_dir=path,
                                           transform=transforms.Compose([
                                               Rescale(256),
                                               RandomCrop(224),
                                               ToTensor()
                                           ]))

for i in range(len(transformed_dataset)):
    sample = transformed_dataset[i]

    print(i, sample['image'].size(), sample['landmarks'].size())

    if i == 3:
        break
```

Pytorch Custom Dataset

- Dataloader

```
[31] dataloader = DataLoader(transformed_dataset, batch_size=4,
                             shuffle=True, num_workers=0)


    # Helper function to show a batch
    def show_landmarks_batch(sample_batched):
        """Show image with landmarks for a batch of samples."""
        images_batch, landmarks_batch = ₩
                sample_batched['image'], sample_batched['landmarks']
        batch_size = len(images_batch)
        im_size = images_batch.size(2)
        grid_border_size = 2

        grid = utils.make_grid(images_batch)
        plt.imshow(grid.numpy().transpose((1, 2, 0)))

        for i in range(batch_size):
            plt.scatter(landmarks_batch[i, :, 0].numpy() + i * im_size + (i + 1) * grid_border_size,
                        landmarks_batch[i, :, 1].numpy() + grid_border_size,
                        s=10, marker='.', c='r')

            plt.title('Batch from dataloader')

    # if you are using Windows, uncomment the next line and indent the for loop.
    # you might need to go back and change "num_workers" to 0.

    # if __name__ == '__main__':
    for i_batch, sample_batched in enumerate(dataloader):
        print(i_batch, sample_batched['image'].size(),
              sample_batched['landmarks'].size())

        # observe 4th batch and stop.
        if i_batch == 3:
            plt.figure()
            show_landmarks_batch(sample_batched)
            plt.axis('off')
            plt.ioff()
            plt.show()
            break
```

```
0 torch.Size([4, 3, 224, 224]) torch.Size([4, 68, 2])
1 torch.Size([4, 3, 224, 224]) torch.Size([4, 68, 2])
2 torch.Size([4, 3, 224, 224]) torch.Size([4, 68, 2])
3 torch.Size([4, 3, 224, 224]) torch.Size([4, 68, 2])
```



Batch size 만큼의 데이터를 Dataset에서 가져온다.

오늘 실습 내용

1. Pytorch Custom Dataset

   Custom Dataset을 구축한 오픈 소스 해석할 수 있음

   - https://openai.com/dall-e-2/
   - unofficial implementation of dall-e-1 : https://github.com/lucidrains/DALLEpytorch/blob/fcd35de4571b50e2d051826514dafc0bd0c69d98/dalle_pytorch/loader.py#L10

   직접 구현해서 사용할 때도 정형화되게 작성할 수 있어서 편리함