PyTorch로 딥러닝 제대로 배우기

- 기초편 -

Part4. 데이터

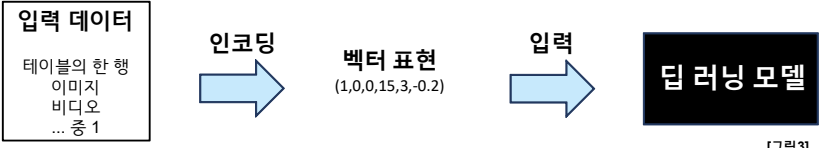
강사: 김 동 희

목차

II. 인코딩

1. 인코딩

- 컴퓨터는 결국 수를 다루는 계산기이다. 테이블, 이미지, 비디오 등의 입력 데이터를 수치로 변환하는 과정을 **인코딩(encoding)** 작업이라고 한다.
  - 전처리 작업 중 하나
- 인코딩을 하고 나면, 입력 데이터는 정해진 개수의 차원으로 이루어진 **벡터(vector)**로 변환된다.



[그림3]

- 실제로는 각 유형의 데이터를 인코딩하고, 전처리할 때 다양한 기법을 적용한다.
  - 노이즈를 줄이고 정확도를 높이기 위함
- 어떤 입력 데이터든 결국 수치로 변환되어 딥 러닝 모델에 입력된다는 점을 기억한다.

2. Table Data 인코딩

입력 데이터

상장유무	상장
규모	중소기업
직원 수	153
매출	278억

[표3]

- 남은 차원들은 연속형 차원이므로 수치를 그대로 사용하여 인코딩
- 단, 이 경우 값의 단위가 다르므로 각 차원의 최솟값을 0, 최댓값을 1로 두는 식의 정규화를 적용 가능
  - min-max normalizatoin

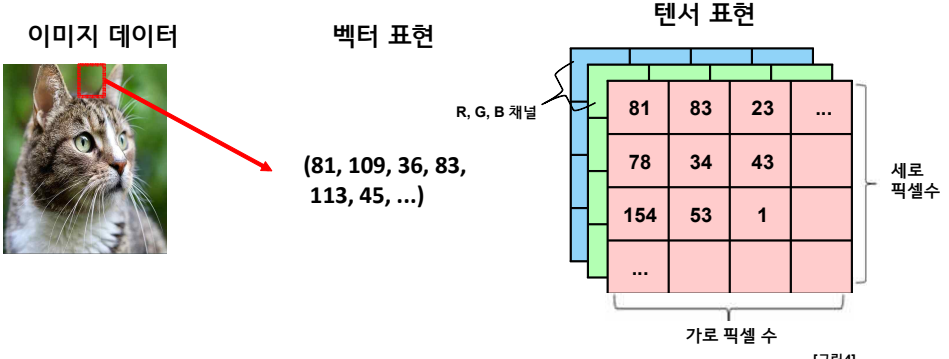
벡터 표현

(0, 0, 0, 1, 153, 278)

3. 이미지 데이터 인코딩

❑ 이미지 인코딩

- 일정한 구조를 갖지 않는 데이터(unstructured data)

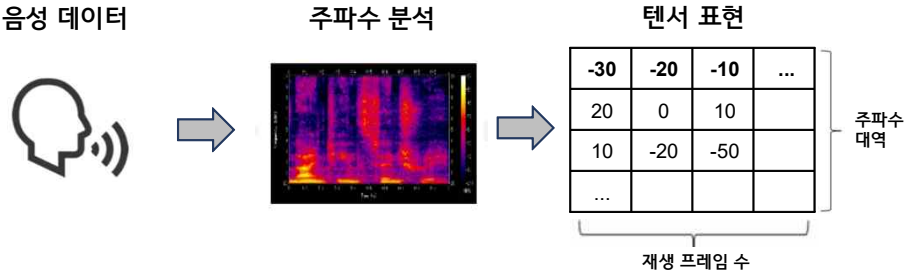


[그림4]

4. 음성 데이터 인코딩

❑ 음성 데이터 인코딩

- 일정한 구조를 갖지 않는 데이터(unstructured data)



[그림5]

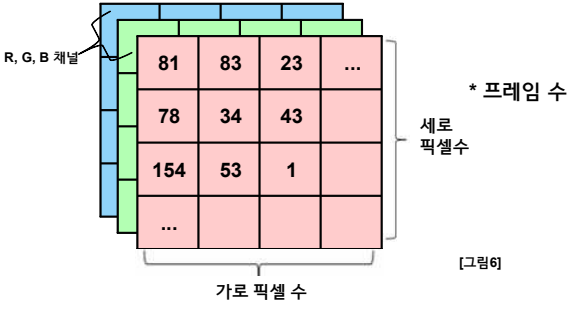
5. 비디오 데이터

❑ 비디오 데이터 인코딩

- 일정한 구조를 갖지 않는 데이터(unstructured data)

비디오 데이터

텐서 표현



[그림6]

III. Dataset & Data Loader

1. Loading Dataset

❑ 개요

- 더 나은 가독성과 모듈성을 위해 데이터 처리 코드가 별도로 존재해야 함
- 데이터를 다운 받거나, 조작을 하거나, 배치를 만드는 등 데이터 조작에 있어서 다양한 기능들이 필요함
- 다양한 기능들을 매번 개발하는 것은 번거롭기 때문에 PyTorch는 두 가지 기능을 제공
  - torch.utils.data.DataLoader: DataLoader는 샘플에 쉽게 접근할 수 있도록 데이터셋에 패키징
  - torch.utils.data.Dataset: 데이터 세트는 샘플과 해당 라벨을 저장

1. Loading Dataset

❑ **Dataset Loading example**

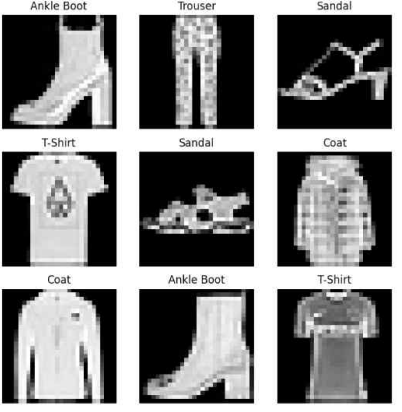
- root: train/test 데이터의 저장 장소
- train: train과 test 데이터 여부
- download: root 디렉토리에 데이터가 없을 때, 인터넷으로부터 다운로드
- transform and target_transform: specify the feature and label transformation

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt


training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

1. Loading Dataset

❑ Iterating and Visualizing the Dataset

```
labels_map = {
    0: "T-Shirt",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle Boot",
}
figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(training_data), size=(1,)).item()
    img, label = training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```



2. Creating a Custom Dataset for your files

❑ Overview

```
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None,
target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx,
0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

2. Creating a Custom Dataset for your files

❑ **__init__**

- __Init__ 함수는 데이터 세트 객체를 인스턴스화할 때 한 번 실행
- 이미지, 주석 파일 및 두 변환이 포함된 디렉토리를 초기화

```
def __init__(self, annotations_file, img_dir, transform=None,
target_transform=None):
    self.img_labels = pd.read_csv(annotations_file)
    self.img_dir = img_dir
    self.transform = transform
    self.target_transform = target_transform
```

❑ **__len__**

- __Len__ 함수는 데이터 세트의 샘플 수를 반환

```
def __len__(self):
    return len(self.img_labels)
```

2. Creating a Custom Dataset for your files

❑ __getitem__

- __Getitem__ 함수는 주어진 인덱스 idx의 데이터 세트에서 샘플을 로드하고 반환

1) 인덱스를 기반으로 디스크에서 이미지의 위치를 식별
2) read_image를 사용하여 텐서로 변환
3) self.img_labels의 csv 데이터에서 해당 라벨을 검색
4) 변환 함수를 호출
5) 튜플에서 텐서 이미지와 해당 라벨을 반환

```
def __getitem__(self, idx):
    img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
    image = read_image(img_path)
    label = self.img_labels.iloc[idx, 1]
    if self.transform:
        image = self.transform(image)
    if self.target_transform:
        label = self.target_transform(label)
    return image, label
```

3. DataLoader

❑ Data Loader

- 데이터셋은 한 번에 하나의 샘플(feature and label)을 반환
- 모델을 훈련시킬 때, 일반적으로 "미니배치 " 단위로 모델에 샘플을 전달
- 모델 과적합을 줄이기 위해 매 에폭마다 데이터를 reshuffle
- 파이썬의 멀티프로세싱을 사용하여 데이터 검색 속도 항상

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64,
shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

3. DataLoader

❑ Iterate through the DataLoader

- 데이터 셋을 DataLoader에 로드 한 후, 필요에 따라 데이터 셋을 반복 호출 가능
- 각 반복은 train_features와 train_labels의 배치를 반환 (각각 batch_size=64 뭉치와, 라벨 포함)
- shuffle=True를 지정했기 때문에 모든 배치를 반복한 후 데이터를 셔플링

```
# Display image and label.
train_features, train_labels = next(iter(train_dataloader))
print(f"Feature batch shape: {train_features.size()}")
print(f"Labels batch shape: {train_labels.size()}")
img = train_features[0].squeeze()
label = train_labels[0]
plt.imshow(img, cmap="gray")
plt.show()
print(f"Label: {label}")
```

감사합니다.