2025-2 데이터문제해결및실습1

12주차-2
[경진대회4] 항공 사진 내 선인장 식별_1

세종대학교
인공지능데이터사이언스학과
박동현 교수

- 본 강의는 골든래빗 출판사의 머신러닝/딥러닝 문제해결전략이 제공하는 강의 교안에 기반함.

Chapter

11

[경진대회4] 항공 사진 내 선인장 식별_1

□ 학습 목표

□ 학습 순서

□ 학습 키워드

딥러닝 모델을 활용한 쉬운 경진대회에 참가한다. 항공 사진에서 선인장을 식별하는 대회다. 이번 장을 통해 이미지 데이터 처리 방법, 신경망 모델 설계 방법, 그리고 파이토치 기본 활용법을 배운다. 점수를 향상 하는 방법보다는 딥러닝 모델을 다루는 방법 중심으로 학습한다.

- **유형 및 평가지표** : 이진분류, ROC AUC
- **데이터 준비** : 데이터셋, 데이터 로더, torchvision 모듈, 이미지 변환, 데이터 증강
- **딥러닝 모델** : CNN, 합성곱, 풀링, 평탄화, 전결합, 배치 정규화, ReLU, Leaky ReLU
- **훈련** : 교차 엔트로피 오차, 확률적 경사 하강법(SGD), 학습률, 에폭, Adamax

□ 핵심 요약

- 딥러닝 모델은 곳곳에서 난수를 활용하므로 매번 같은 결과를 얻으려면 **시드 값**을 잘 **고정**해야 한다.
- **GPU를 활용**하면 딥러닝의 방대한 연산 시간을 크게 단축할 수 있다.
- 파이토치에서는 **Dataset 클래스**를 상속하여 데이터셋을 준비한다.
- 파이토치의 **DataLoader 클래스**는 Dataset으로부터 배치 크기만큼씩 데이터를 불러와주는 역할을 한다.
- **이미지 변환기** : 원본 이미지를 특정한 형태로 변환한다.
- **데이터 증강** : 원본 이미지에 다양한 변환을 가하여 데이터 수를 늘리는 기법. 훈련 데이터 수가 부족할 때 유용하다.
- Torchvsion의 **transforms 모듈**은 다양한 이미지 변환기를 제공한다.
- **CNN** : 합성곱 계층을 포함한 딥러닝 모델로, 이미지(영상) 인식 분야에서 많이 활용된다.
- **손실 함수** : 모델 훈련 과정에서 예측값과 실젯값의 차이를 구하는 함수
- **활성화 함수** : 신경망 계층에서 입력값을 어떤 값으로 변환해 출력할지를 결정하는 함수, 이번 장에서는 ReLU와 Leaky ReLU를 이용한다.
- **옵티마이저** : 최적 가중치를 찾아주는 함수. 확률적 경사 하강법(SGD)을 기반 이론으로 하여 다양하게 보완하여 사용한다.
- **에폭** : '훈련 데이터 전체'를 '한 번' 훑었음을 뜻함.
- **배치 크기** : 매 훈련 이터레이션에서 한 번에 훈련할 데이터 개수
- **반복 횟수** : 1에폭의 훈련을 완료하는 데 필요한 훈련 이터레이션
- **배치 정규화** : 계층 간 데이터 분포의 편차를 줄이는 작업. 신경망 계층마다 입력 데이터 분포가 다르면 훈련 속도가 느려지고 과대적합될 수 있다.

□ 경진대회 정보

- **난이도** : ★☆☆
- **경진대회명** : 항공 사진 내 선인장 식별 경진대회
- **미션** : 항공 사진 내 선인장이 있을 확률 예측
- **문제 유형** : 이진분류
- **평가지표** : ROC AUC
- **데이터 크기** : 24.2MB
- **참가팀 수** : 1,221팀
- **제출 시 사용한 모델** : 기본 CNN
- **파이썬 버전** : 3.7.10
- **사용 라이브러리 및 버전**
    - numpy (numpy==1.19.5) / pandas (pandas==1.3.2)
    - torch (torch==1.7.1) / torchvision (torchvision==0.8.2)
    - sklearn (scikit-learn==0.23.2)
    - matplotlib (matplotlib==3.4.3)
    - cv2 (opencv-python==4.5.3.56)
    - zipfile, random, math, shutil, os

11.1 경진대회 이해

이번 경진대회는 데이터 크기가 작고 난이도도 낮은 플레이그라운드 대회로, 항공 사진에서 선인장(cactus)을 찾아내는 게 목표다.

멕시코는 자연을 보호하고자 멕시코는 VIGIA라는 프로젝트를 진행했다. VIGIA는 '자연보호 구역 자율 감시'를 위한 자동 시스템 개발 프로젝트이다. 자율 감시를 위한 첫 번째 단계가 보호 구역 내에 초목이 잘 자라는지 확인하는 작업이다. 드론과 이미지 인식 기술을 결합해 자율 감시 시스템을 만들었다.

이번 경진대회에서는 csv 파일에 더해 '이미지 파일'도 제공한다. 주어진 데이터는 다음과 같다.

- train.zip : 훈련 이미지 데이터(jpg 형식) 압축 파일
- test.zip : 테스트 이미지 데이터(jpg 형식) 압축 파일
- train.csv : 훈련 이미지 데이터 파일명 및 타깃값(타깃값은 0 또는 1)
- sample_submission.csv : 샘플 제출 파일

딥러닝 첫 대회인 만큼 리더보드 등수를 높이는 방법보다는 딥러닝 모델을 구축하는 방법을 중심으로 설명한다. 파이토치에 익숙해지는 기회도 될 것이다.

11.2 탐색적 데이터 분석

11.2.1 데이터 둘러보기

- csv 데이터를 불러와 데이터를 살펴본다.
- id 피처는 훈련 데이터의 이미지 파일명이다(확장자 jpg 포함).
- has_cactus는 타깃값이다.
- 해당 파일명을 가진 이미지가 선인장을 포함하는지 여부를 나타낸다. 0이면 선인장이 없고 1이면 있다.
- 훈련 데이터 14,500개, 테스트 데이터 3,000개다.

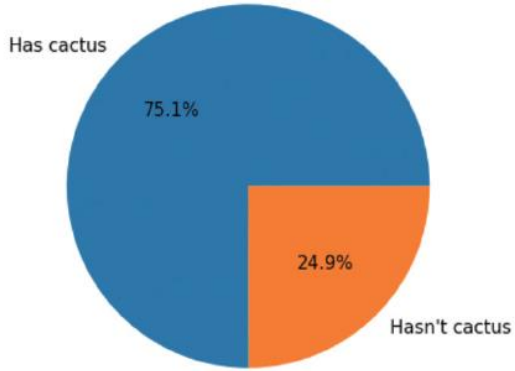
11.2 탐색적 데이터 분석

11.2.1 데이터 시각화

타깃값 분포

- 파이 그래프로 타깃값 분포를 그려본다.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

mpl.rc('font', size=15)
plt.figure(figsize=(7, 7))

label = ['Has cactus', 'Hasn\'t cactus'] # 타깃값 레이블
# 타깃값 분포 파이 그래프
plt.pie(labels['has_cactus'].value_counts(), labels=label, autopct='%.1f%%');
```



- 타깃값 0(Hasn't cactus)과 1(Has cactus)의 비율이 약 1:3이다.

11.2 탐색적 데이터 분석

11.2.1 데이터 시각화

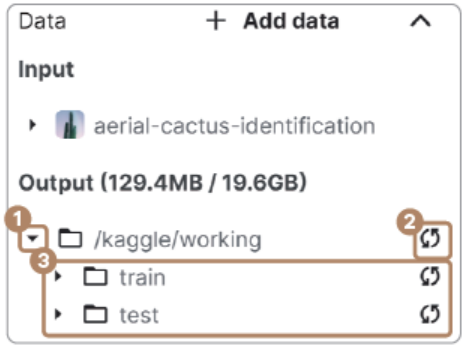
이미지 출력

- 어떤 이미지가 사용되는지 압축 파일을 풀어 확인한다.

```
from zipfile import ZipFile

# 훈련 이미지 데이터 압축 풀기
with ZipFile(data_path + 'train.zip') as zipper:
zipper.extractall()

# 테스트 이미지 데이터 압축 풀기
with ZipFile(data_path + 'test.zip') as zipper:
zipper.extractall()
```

- 타깃값 0(Hasn't cactus)과 1(Has cactus)의 비율이 약 1:3이다.
- 압축이 풀린 파일들은 작업 디렉터리에서 확인한다.
  코드를 실행한 다음, 오른쪽 상단 [Data] 탭을 확인한다.
  zipper.extractall( )을 호출해 푼 이미지가 (3)에 저장된다.

▼ 압축이 풀린 이미지 파일 디렉터리



11.2 탐색적 데이터 분석

11.2.1 데이터 시각화

이미지 출력

- OpenCV 라이브러리를 사용해 훈련 이미지 데이터 몇 개 출력해보기

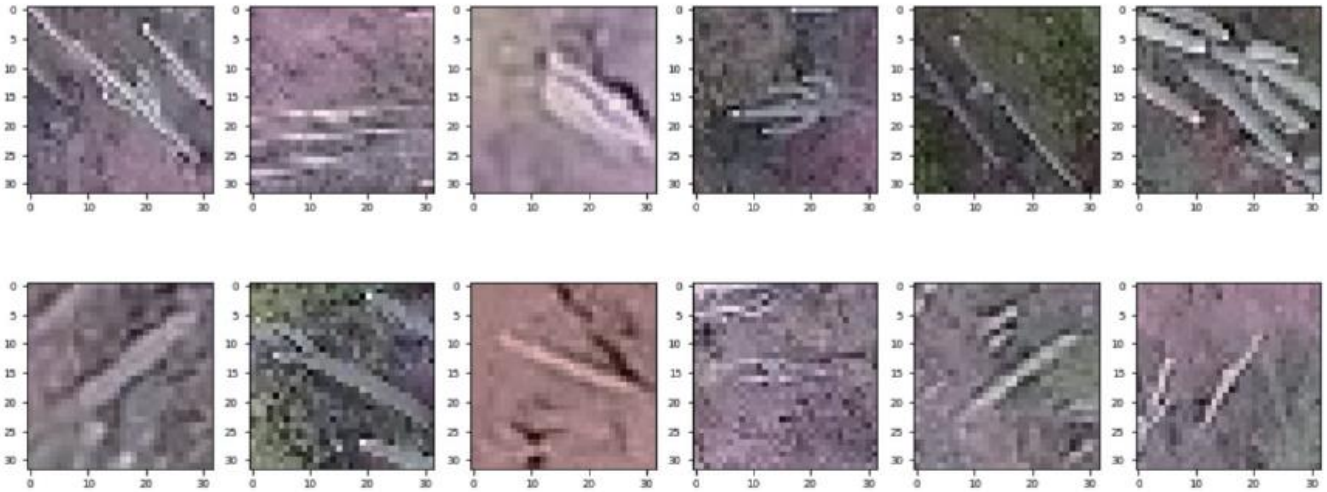
```
import matplotlib.gridspec as gridspec
import cv2 # OpenCV 라이브러리 임포트

mpl.rc('font', size=7)
plt.figure(figsize=(15, 6)) # 전체 Figure 크기 설정
grid = gridspec.GridSpec(2, 6) # 서브플롯 배치(2행 6열로 출력)

# 선인장을 포함하는 이미지 파일명(마지막 12개) 1
last_has_cactus_img_name = labels[labels['has_cactus']==1]['id'][-12:]

# 이미지 출력 2
for idx, img_name in enumerate(last_has_cactus_img_name):
    img_path = 'train/' + img_name # 이미지 파일 경로 3
    image = cv2.imread(img_path) # 이미지 파일 읽기 4
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # 이미지 색상 보정 5
    ax = plt.subplot(grid[idx])
    ax.imshow(image) # 이미지 출력 6
```

11.2 탐색적 데이터 분석

11.2.1 데이터 시각화

이미지 출력

- 실행 결과(선인장을 포함하는 이미지 마지막 12장)



11.2 탐색적 데이터 분석

11.2.1 데이터 시각화

이미지 출력

- 선인장을 포함하지 않는 이미지 출력하기. has_cactus 피처가 0인 이미지 추출

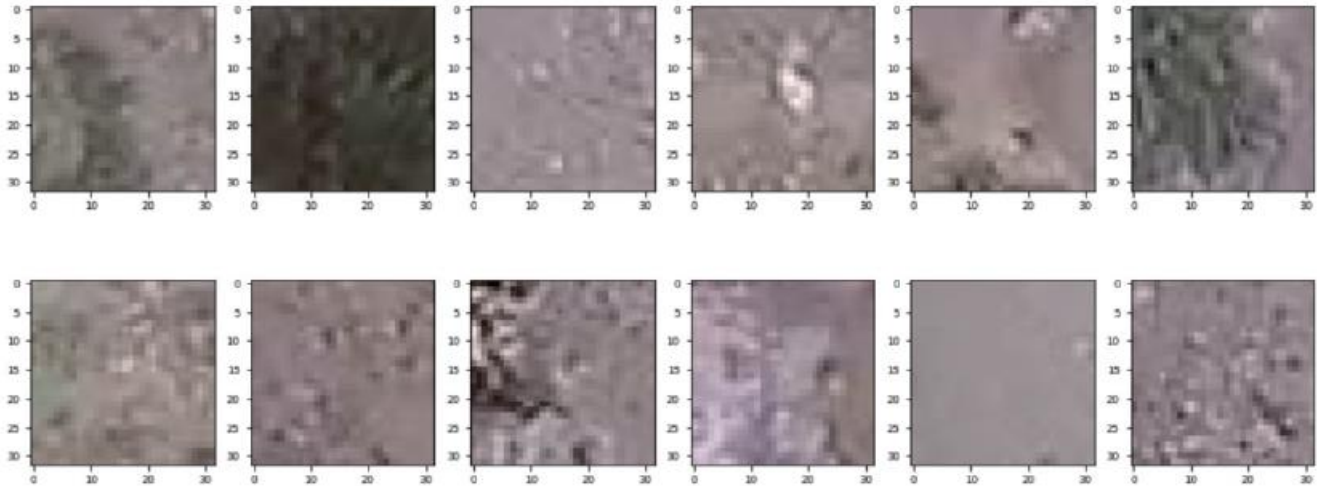
```
plt.figure(figsize=(15, 6)) # 전체 Figure 크기 설정
grid = gridspec.GridSpec(2, 6) # 서브플롯 배치

# 선인장을 포함하지 않는 이미지 파일명(마지막 12개)
last_hasnt_cactus_img_name = labels[labels['has_cactus']==0]['id'][-12:]

# 이미지 출력
for idx, img_name in enumerate(last_hasnt_cactus_img_name):
    img_path = 'train/' + img_name # 이미지 파일 경로
    image = cv2.imread(img_path) # 이미지 파일 읽기
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # 이미지 색상 보정
    ax = plt.subplot(grid[idx])
    ax.imshow(image) # 이미지 출력
```

11.2 탐색적 데이터 분석

11.2.1 데이터 시각화

이미지 출력

- 실행 결과(선인장을 포함하지 않는 이미지 마지막 12장)



11.2 탐색적 데이터 분석

11.2.1 데이터 시각화

이미지 출력

- 이미지 형상 출력하기

```
image.shape
```

- 이미지의 가로 픽셀 수, 세로 픽셀 수, 채널 수가 출력된다.
- 가로, 세로 크기는 32 x 32이며, 채널은 3개(데이터가 빨강(R), 초록(G), 파랑(B)으로 이루어진 컬러 이미지이기 때문)

분석 정리 및 모델링 전략

분석 정리

1. csv 파일의 id 피처는 이미지 파일명이다. 파일의 경로명만 추가하면 파일의 위치를 바로 얻어올 수 있다.
2. 제공된 이미지 파일들은 낮은 해상도의 컬러 이미지(32 x 32 x 3)이다.

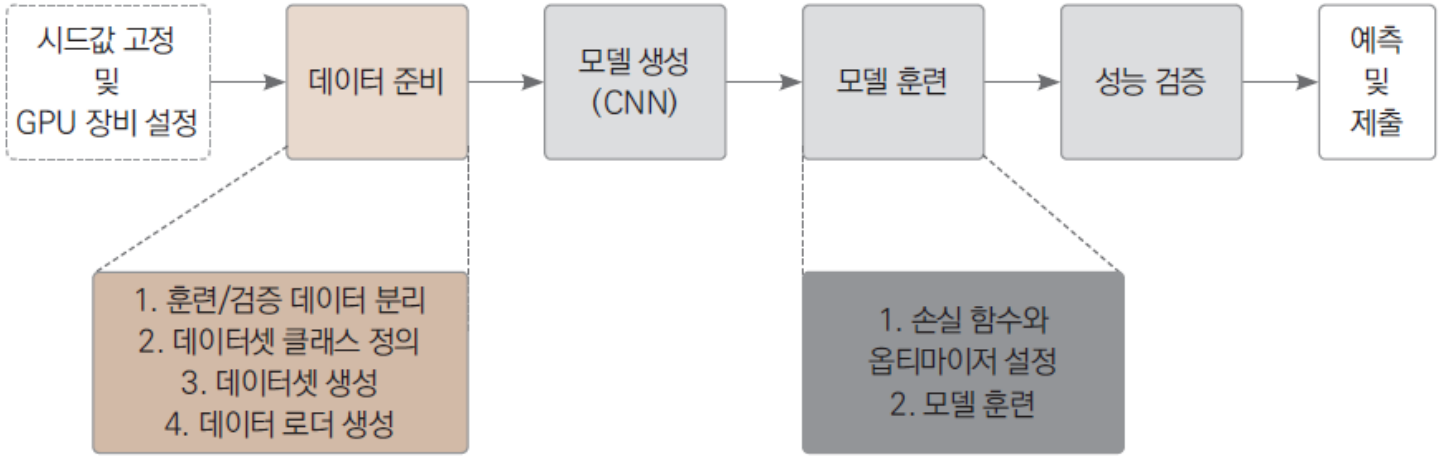
모델링 전략

- **베이스라인 모델** : 얕은 CNN
    - **신경망 구조** : 합성곱 x 2, 풀링, 평탄화, 전결합
    - **옵티마이저** : SGD
- **성능 개선** : 살짝 깊은 CNN
    - **데이터 증강** : 다양한 변환기 사용
    - **신경망 구조** : 합성곱 x 5, 배치 정규화, 풀링, 평탄화, 전결합 x 2
    - **옵티마이저** : Adamax
    - **기타** : 훈련 에폭 수 증가

*베이스라인 모델과 성능 개선은 본 경진대회에서 추천수가 12번째로 많은 다음 노트북을 리팩터링하여 작성함.
https://www.kaggle.com/bonhart/simple-cnn-on-pytorch-for-beginers

11.3 베이스라인 모델

파이토치를 활용해 딥러닝 모델을 만든다. 베이스라인은 간단한 CNN 모델을 활용한다. 파이토치를 활용한 딥러닝 모델 절차는 다음과 같다.



11.3 베이스라인 모델

11.3.1 시드값 고정 및 GPU 장비 설정

시드값 고정

- 파이토치를 임포트하고 시드값을 고정한다. 다시 실행해도 같은 결과를 얻기 위해서 시드값을 고정한다. (https://www.kaggle.com/werooring/ch11-baseline)

```
import torch # 파이토치
import random
import numpy as np
import os

# 시드값 고정
seed = 50
os.environ['PYTHONHASHSEED'] = str(seed)
random.seed(seed) # 파이썬 난수 생성기 시드 고정
np.random.seed(seed) # 넘파이 난수 생성기 시드 고정
torch.manual_seed(seed) # 파이토치 난수 생성기 시드 고정 (CPU 사용 시)
torch.cuda.manual_seed(seed) # 파이토치 난수 생성기 시드 고정 (GPU 사용 시)
torch.cuda.manual_seed_all(seed) # 파이토치 난수 생성기 시드 고정 (멀티GPU 사용 시)
torch.backends.cudnn.deterministic = True # 확정적 연산 사용
torch.backends.cudnn.benchmark = False # 벤치마크 기능 해제
torch.backends.cudnn.enabled = False # cudnn 사용 해제
```

11.3 베이스라인 모델

11.3.1 시드값 고정 및 GPU 장비 설정

GPU 장비 설정

- 비정형 데이터(이미지, 음성, 텍스트 등)를 주로 다루는 딥러닝 경진대회에서는 연산량이 많아진다. CPU로는 감당하기 벅찰 정도라서 훈련 시간이 너무 길어지므로, GPU를 사용한다. 다행히도 캐글에서 GPU 환경까지 제공한다. 다음 코드로 연산에 이용할 장비를 할당한다.

```
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')
```

```
device
```

- GPU를 사용하기 위해 설정을 변경한다. [Settings] > Accelerator > GPU로 변경
- Accelerator를 변경하면 코드 환경 전체가 초기화되므로 코드를 다시 실행해 device 변수에 CUDA가 할당되었는지 확인한다.

11.3 베이스라인 모델

11.3.2 데이터 준비

훈련 데이터, 검증 데이터 분리

- train_test_split( ) 함수를 사용해 이 데이터를 훈련 데이터와 검증 데이터로 나눈다.

```
from sklearn.model_selection import train_test_split

# 훈련 데이터, 검증 데이터 분리
train, valid = train_test_split(labels,
                                test_size=0.1, # 1
                                stratify=labels['has_cactus'], # 2
                                random_state=50)
    device = torch.device('cpu')
```

- 결과 확인

```
print('훈련 데이터 개수:', len(train))
print('검증 데이터 개수:', len(valid))
```

11.3 베이스라인 모델

11.3.2 데이터 준비

데이터셋 클래스 정의

- 먼저 데이터셋 생성에 필요한 라이브러리 임포트

```
import cv2 # OpenCV 라이브러리
from torch.utils.data import Dataset # 데이터 생성을 위한 클래스
```

- Dataset을 상속받은 다음 특수 메서드인 __len__( )과 __getitem__( )을 재정의(오버라이딩)
  - **__len__()** : 데이터셋 크기를 반환
  - **__getitem__()** : 인덱스를 전달받아 인덱스에 해당하는 데이터를 반환
  - **df** : DataFrame 객체. train 혹은 valid를 df 파라미터에 전달.
  - **img_dir** : 이미지 데이터를 포함하는 경로
  - **transform** : 이미지 변환기. 이미지 데이터셋을 만들 때 기본적인 전처리를 할 수 있는데, 전처리를 하려면 이미지 변환기를 넘겨주면 된다.

11.3 베이스라인 모델

```
class ImageDataset(Dataset):
    # 초기화 메서드(생성자) 1
    def __init__(self, df, img_dir='./', transform=None):
        super().__init__() # 상속받은 Dataset의 생성자 호출 2
        # 전달받은 인수들 저장 3
        self.df = df
        self.img_dir = img_dir
        self.transform = transform

    # 데이터셋 크기 반환 메서드 4
    def __len__(self):
        return len(self.df)

    # 인덱스(idx)에 해당하는 데이터 반환 메서드 5
    def __getitem__(self, idx):
        img_id = self.df.iloc[idx, 0] # 이미지 ID
        img_path = self.img_dir + img_id # 이미지 파일 경로 6
        image = cv2.imread(img_path) # 이미지 파일 읽기
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # 이미지 색상 보정
        label = self.df.iloc[idx, 1] # 이미지 레이블(타깃값)

        if self.transform is not None:
            image = self.transform(image) # 변환기가 있다면 이미지 변환 7
        return image, label # 8
```

11.3 베이스라인 모델

11.3.2 데이터 준비

데이터셋 생성

- 앞서 정의한 ImageDataset 클래스를 이용해서 데이터셋을 만든다. 파이토치 모델로 이미지를 다루려면 이미지 데이터를 텐서 타입으로 바꿔야 한다.

```
from torchvision import transforms # 이미지 변환을 위한 모듈

transform = transforms.ToTensor()
```

- 앞에서 정의한 ImageDataset( ) 클래스를 사용해 훈련 데이터셋과 검증 데이터셋 만들기

```
dataset_train = ImageDataset(df=train, img_dir='train/', transform=transform)
dataset_valid = ImageDataset(df=valid, img_dir='train/', transform=transform)
```

11.3 베이스라인 모델

11.3.2 데이터 준비

데이터 로더 생성

- 데이터 로더 : 지정한 배치 크기만큼씩 데이터를 불러오는 객체
- 딥러닝 모델을 훈련할 때는 주로 배치 단위로 데이터를 가져와 훈련한다. torch.utils.data의 DataLoader 클래스로 데이터 로더를 만들 수 있다.

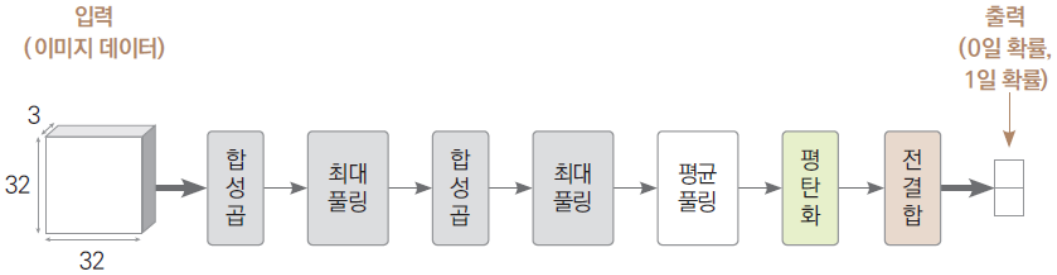
```
from torch.utils.data import DataLoader # 데이터 로더 클래스

loader_train = DataLoader(dataset=dataset_train, batch_size=32, shuffle=True)
loader_valid = DataLoader(dataset=dataset_valid, batch_size=32, shuffle=False)
```

- **dataset** : 앞서 만든 이미지 데이터셋을 전달
- **batch_size** : 배치 크기
- **shuffle** : 데이터를 섞을지 여부를 결정

11.3 베이스라인 모델

11.3.3 모델 생성

- 기본적인 합성곱 신경망(CNN) 모델 만들기



- CNN 모델을 만드는 데 필요한 모듈 불러오기

```
import torch.nn as nn # 신경망 모듈
import torch.nn.functional as F # 신경망 모듈에서 자주 사용되는 함수
```