다섯째마당
딥러닝 활용하기

16장 이미지 인식의 꽃, 컨볼루션 신경망(CNN)

1 이미지를 인식하는 원리

2 딥러닝 기본 프레임 만들기

3 컨볼루션 신경망(CNN)

4 맥스 풀링, 드롭아웃, 플래튼

5 컨볼루션 신경망 실행하기

이미지 인식의 꽃, 컨볼루션 신경망(CNN)

- 이미지 인식의 꽃, 컨볼루션 신경망(CNN)

이미지 인식의 꽃, 컨볼루션 신경망(CNN)

- **이미지 인식의 꽃, 컨볼루션 신경망(CNN)**
  - 급히 전달받은 노트에 숫자가 적혀 있음
  - 뭐라고 썼는지 읽기에 그리 어렵지 않음
  - 일반적인 사람에게 이 사진에 나온 숫자를 읽어 보라고 하면 대부분 '504192'라고 읽음
  - 컴퓨터에 이 글씨를 읽게 하고 이 글씨가 어떤 의미인지 알게 하는 과정은 쉽지 않음
  - 사람이 볼 때는 쉽게 알 수 있는 글씨라 해도 숫자 5는 어떤 특징을 가졌고, 숫자 9는 6과 어떻게 다른지 기계가 스스로 파악해 정확하게 읽고 판단하게 만드는 것은 머신 러닝의 오랜 진입 과제

이미지 인식의 꽃, 컨볼루션 신경망(CNN)

- **이미지 인식의 꽃, 컨볼루션 신경망(CNN)**
  - MNIST 데이터셋은 미국 국립표준기술원(NIST)이 고등학생과 인구조사국 직원 등이 쓴 손글씨를 이용해 만든 데이터로 구성되어 있음
  - 7만 개의 글자 이미지에 각각 0부터 9까지 이름표를 붙인 데이터셋으로, 머신러닝을 배우는 사람이라면 자신의 알고리즘과 다른 알고리즘의 성과를 비교해 보고자 한 번씩 도전해 보는 유명한 데이터 중 하나

이미지 인식의 꽃, 컨볼루션 신경망(CNN)

▼ 그림 16-1 | MNIST 손글씨 데이터 이미지

1 이미지를 인식하는 원리

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - MNIST 데이터는 텐서플로의 케라스 API를 이용해 간단히 불러올 수 있음
  - 함수를 이용해서 사용할 데이터를 불러옴

```
from tensorflow.keras.datasets import mnist
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 이때 불러온 이미지 데이터를 X로, 이 이미지에 0~9를 붙인 이름표를 y로 구분해 명명하겠음
  - 또한, 7만 개 중 학습에 사용될 부분은 train으로, 테스트에 사용될 부분은 test라는 이름으로 불러오겠음

    - 학습에 사용될 부분: X_train, y_train
    - 테스트에 사용될 부분: X_test, y_test

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 케라스의 MNIST 데이터는 총 7만 개 이미지 중 6만 개를 학습용으로, 1만 개를 테스트용으로 미리 구분해 놓고 있음
  - 이를 다음과 같이 확인할 수 있음

```
print("학습셋 이미지 수: %d개" % (X_train.shape[0]))
print("테스트셋 이미지 수: %d개" % (X_test.shape[0]))
```

1 이미지를 인식하는 원리

- 이미지를 인식하는 원리

실행 결과

```
학습셋 이미지 수: 60000개
테스트셋 이미지 수: 10000개
```

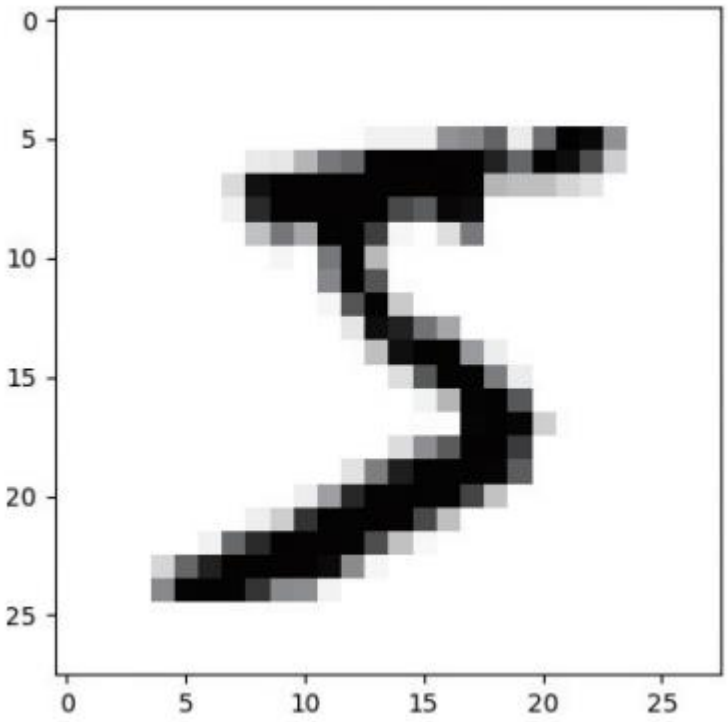
1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 불러온 이미지 중 한 개만 다시 불러와 보자
  - 이를 위해 먼저 맷플롯립 라이브러리를 불러옴
  - imshow() 함수를 이용해 이미지를 출력할 수 있음
  - 모든 이미지가 X_train에 저장되어 있으므로 X_train[0]으로 첫 번째 이미지를, cmap='Greys' 옵션을 지정해 흑백으로 출력되게 함

```
import matplotlib.pyplot as plt

plt.imshow(X_train[0], cmap='Greys')
plt.show()
```

1 이미지를 인식하는 원리

▼ 그림 16-2 | MNIST 손글씨 데이터의 첫 번째 이미지



1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 이 이미지를 컴퓨터는 어떻게 인식할까?
  - 이 이미지는 가로 28×세로 28 = 총 784개의 픽셀로 이루어져 있음
  - 각 픽셀은 밝기 정도에 따라 0부터 255까지 등급을 매김
  - 흰색 배경이 0이라면 글씨가 들어간 곳은 1~255의 숫자 중 하나로 채워져 긴 행렬로 이루어진 하나의 집합으로 변환

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 다음 코드로 이를 확인할 수 있음

```
for x in X_train[0]:
    for i in x:
        sys.stdout.write("%-3s" % i)
    sys.stdout.write('\n')
```

1 이미지를 인식하는 원리

▼ 그림 16-3 | 그림의 각 좌표를 숫자로 표현해 보기

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0
0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0
0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0	0	0
0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	35	241	225	160	108	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 바로 이렇게 이미지는 다시 숫자의 집합으로 바뀌어 학습셋으로 사용
  - 우리가 앞서 배운 여러 예제와 마찬가지로 속성을 담은 데이터를 딥러닝에 집어넣고 클래스를 예측하는 문제로 전환시키는 것
  - 28×28 = 784개의 속성을 이용해 0~9의 클래스 열 개 중 하나를 맞히는 문제가 됨

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 이제 주어진 가로 28, 세로 28의 2차원 배열을 784개의 1차원 배열로 바꾸어 주어야 함
  - 이를 위해 reshape() 함수를 사용
  - reshape(총 샘플 수, 1차원 속성의 개수) 형식으로 지정
  - 총 샘플 수는 앞서 사용한 X_train.shape[0]을 이용하고, 1차원 속성의 개수는 이미 살펴본 대로 784개

```
X_train = X_train.reshape(X_train.shape[0], 784)
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 케라스는 데이터를 0에서 1 사이의 값으로 변환한 후 구동할 때 최적의 성능을 보임
  - 현재 0~255 사이의 값으로 이루어진 값을 0~1 사이의 값으로 바꾸어야 함
  - 바꾸는 방법은 각 값을 255로 나누는 것
  - 이렇게 데이터의 폭이 클 때 적절한 값으로 분산의 정도를 바꾸는 과정을 데이터 정규화(normalization)라고 함

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 현재 주어진 데이터 값은 0부터 255까지의 정수로, 정규화를 위해 255로 나누어 주려면 먼저 이 값을 실수형으로 바꾸어야 함
  - 다음과 같이 astype() 함수를 이용해 실수형으로 바꾼 후 255로 나눔

```
X_train = X_train.astype('float64')
X_train = X_train / 255
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - X_test에도 마찬가지로 이 작업을 적용
  - 다음과 같이 한 번에 적용

```
X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 이제 숫자 이미지에 매겨진 이름을 확인해 보자
  - 우리는 앞서 불러온 숫자 이미지가 5라는 것을 눈으로 보아 짐작할 수 있음
  - 실제로 이 숫자의 레이블이 어떤지 불러오고자 y_train[0]을 다음과 같이 출력해 보자

```
print("class : %d " % (y_train[0]))
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 이 숫자의 레이블 값인 5가 출력되는 것을 볼 수 있음

실행 결과

```
class : 5
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 12장에서 아이리스 품종을 예측할 때 딥러닝의 분류 문제를 해결하려면 원-핫 인코딩 방식을 적용해야 한다고 배웠음
  - 즉, 0~9의 정수형 값을 갖는 현재 형태에서 0 또는 1로만 이루어진 벡터로 값을 수정

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 지금 우리가 열어 본 이미지의 클래스는 [5]였음
  - 이를 [0,0,0,0,0,1,0,0,0,0]으로 바꾸어야 함
  - 이를 가능하게 해 주는 함수가 바로 np_utils.to_categorical( ) 함수
  - to_categorical(클래스, 클래스의 개수) 형식으로 지정

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 이제 변환된 값을 출력해 보자

```
print(y_train[0])
```

1 이미지를 인식하는 원리

- **이미지를 인식하는 원리**
  - 다음과 같이 원-핫 인코딩이 적용된 것을 확인할 수 있음

실행 결과

```
[ 0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
```

  - 이제 딥러닝을 실행할 준비를 모두 마쳤음

1 이미지를 인식하는 원리

- 이미지를 인식하는 원리

실습 | MNIST 손글씨 인식하기: 데이터 전처리

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
import sys

# MNIST 데이터셋을 불러와 학습셋과 테스트셋으로 저장합니다.
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

1 이미지를 인식하는 원리

- 이미지를 인식하는 원리

```
# 학습셋과 테스트셋이 각각 몇 개의 이미지로 되어 있는지 확인합니다.
print("학습셋 이미지 수: %d개" % (X_train.shape[0]))
print("테스트셋 이미지 수: %d개" % (X_test.shape[0]))

# 첫 번째 이미지를 확인해 봅시다.
plt.imshow(X_train[0], cmap='Greys')
plt.show()

# 이미지가 인식되는 원리를 알아봅시다.
for x in X_train[0]:
    for i in x:
        sys.stdout.write("%-3s" % i)
    sys.stdout.write('\n')
```

1 이미지를 인식하는 원리

- 이미지를 인식하는 원리

```
# 차원 변환 과정을 실습해 봅니다.
X_train = X_train.reshape(X_train.shape[0], 784)
X_train = X_train.astype('float64')
X_train = X_train / 255

X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255

# 클래스 값을 확인해 봅니다.
print("class : %d " % (y_train[0]))

# 바이너리화 과정을 실습해 봅니다.
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

1 이미지를 인식하는 원리

- 이미지를 인식하는 원리

```
print(y_train[0])
```

2 딥러닝 기본 프레임 만들기

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 이제 불러온 데이터를 실행할 차례
  - 총 6만 개의 학습셋과 1만 개의 테스트셋을 불러와 속성 값을 지닌 X, 클래스 값을 지닌 y로 구분하는 작업을 다시 한 번 정리하면 다음과 같음

```
from tensorflow.keras.datasets import mnist

# MNIST 데이터를 불러옵니다.
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 차원 변환 후, 테스트셋과 학습셋으로 나눕니다.
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 이제 딥러닝을 실행하고자 프레임을 설정
  - 총 784개의 속성이 있고 열 개의 클래스가 있음
  - 다음과 같이 딥러닝 프레임을 만들 수 있음

```
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

  - 입력 값(input_dim)이 784개, 은닉층이 512개, 출력이 열 개인 모델
  - 활성화 함수로 은닉층에서는 relu를, 출력층에서는 softmax를 사용

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 딥러닝 실행 환경을 위해 오차 함수로 categorical_crossentropy, 최적화 함수로 adam을 사용

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 모델 실행에 앞서 먼저 성과를 저장하고, 모델의 최적화 단계에서는 학습을 자동 중단하게끔 설정
  - 열 번 이상 모델 성능이 향상되지 않으면 자동으로 학습을 중단

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# 모델 최적화를 위한 설정 구간입니다.
modelpath = "./MNIST_MLP.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)
```

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 주피터 노트북에서 실습한다면 다음 코드가 modelpath 코드 윗부분에 추가
  - 예제 파일을 참고

```
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)
```

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 샘플 200개를 모두 30번 실행하게끔 설정
  - 테스트셋으로 최종 모델의 성과를 측정해 그 값을 출력

```
# 모델을 실행합니다.
history = model.fit(X_train, y_train, validation_split=0.25, epochs=30,
batch_size=200, verbose=0, callbacks=[early_stopping_callback,
checkpointer])

# 테스트 정확도를 출력합니다.
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))
```

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 실행 결과를 그래프로 표현해 보자
  - 이번에는 학습셋의 정확도 대신 학습셋의 오차를 그래프로 표현
  - 학습셋의 오차는 1에서 학습셋의 정확도를 뺀 값
  - 좀 더 세밀한 변화를 볼 수 있게 학습셋의 오차와 테스트셋의 오차를 그래프 하나로 나타내겠음

```
import matplotlib.pyplot as plt

y_vloss = history.history['val_loss']

# 학습셋의 오차
y_loss = history.history['loss']
```

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기

```
# 그래프로 표현합니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시합니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 지금까지 내용을 스크립트 하나로 정리하면 다음과 같음

실습 | MNIST 손글씨 인식하기: 기본 프레임

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt
import numpy as np
import os
```

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기

```
# MNIST 데이터를 불러옵니다.
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 차원 변환 후, 테스트셋과 학습셋으로 나눕니다.
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 모델 구조를 설정합니다.
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기

```
# 모델 실행 환경을 설정합니다.
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# 모델 최적화를 위한 설정 구간입니다.
modelpath = "./MNIST_MLP.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델을 실행합니다.
history = model.fit(X_train, y_train, validation_split=0.25, epochs=30,
batch_size=200, verbose=0, callbacks=[early_stopping_callback,
checkpointer])
```

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기

```
# 테스트 정확도를 출력합니다.
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))

# 검증셋과 학습셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프로 표현해 봅니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')
```

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기

```
# 그래프에 그리드를 주고 레이블을 표시해 보겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기

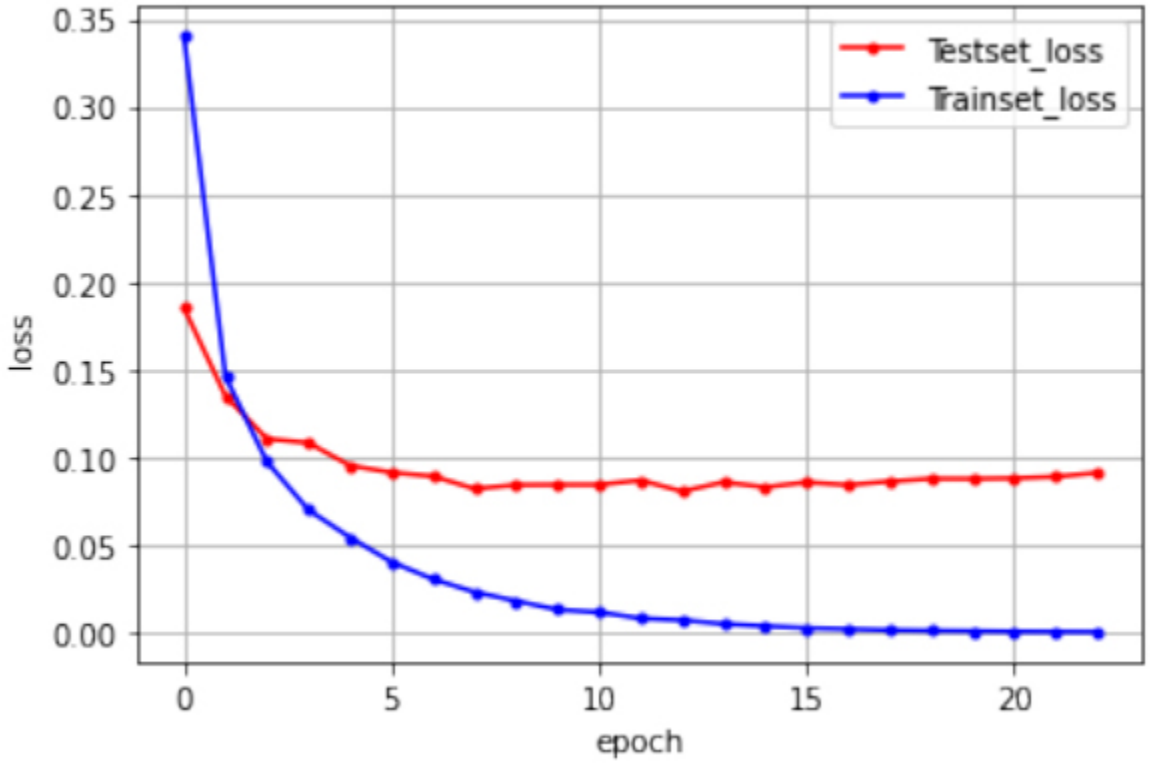
실행 결과

```
Epoch 00001: val_loss improved from inf to 0.18529, saving model to ../
data/model\MNIST_MLP.hdf5
... (중략) ...
Epoch 00013: val_loss improved from 0.08235 to 0.08088, saving model to ../
data/model\MNIST_MLP.hdf5
... (중략) ...
Epoch 00023: val_loss did not improve from 0.08088
313/313 [==============================] - 1s 1ms/step - loss: 0.0711 -
accuracy: 0.9816

 Test Accuracy: 0.9816
```

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기

▼ 그림 16-4 | 학습이 진행될 때 학습셋과 테스트셋의 오차 변화



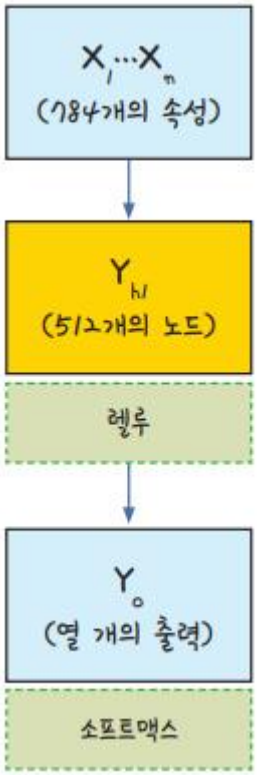
2 딥러닝 기본 프레임 만들기

- **딥러닝 기본 프레임 만들기**
  - 23번째 실행에서 멈춘 것을 확인할 수 있음
  - 베스트 모델은 13번째 에포크일 때이며, 이 모델의 테스트셋에 대한 정확도는 98.16%
  - 함께 출력되는 그래프로 실행 내용을 확인할 수 있음
  - 학습셋에 대한 오차는 계속해서 줄어듦
  - 테스트셋의 과적합이 일어나기 전 학습을 끝낸 모습

2 딥러닝 기본 프레임 만들기

- 딥러닝 기본 프레임 만들기
  - 앞서 98.16%의 정확도를 보인 딥러닝 프레임은 하나의 은닉층을 둔 아주 단순한 모델

▼ 그림 16-5 | 은닉층이 하나인 딥러닝 모델의 도식



3 컨볼루션 신경망(CNN)

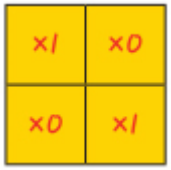
3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 컨볼루션 신경망은 입력된 이미지에서 다시 한 번 특징을 추출하기 위해 커널(슬라이딩 윈도)을 도입하는 기법
  - 예를 들어 입력된 이미지가 다음과 같은 값을 가지고 있다고 하자

1	0	1	0
0	1	1	0
0	0	1	1
0	0	1	0

3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 여기에 2×2 커널을 준비
  - 각 칸에는 가중치가 들어 있음
  - 샘플 가중치를 다음과 같이 ×1, ×0이라고 하겠음



3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 이제 커널을 맨 왼쪽 윗칸에 적용시켜 보자

1×1	0×0	1	0
0×0	1×1	1	0
0	0	1	1
0	0	1	0

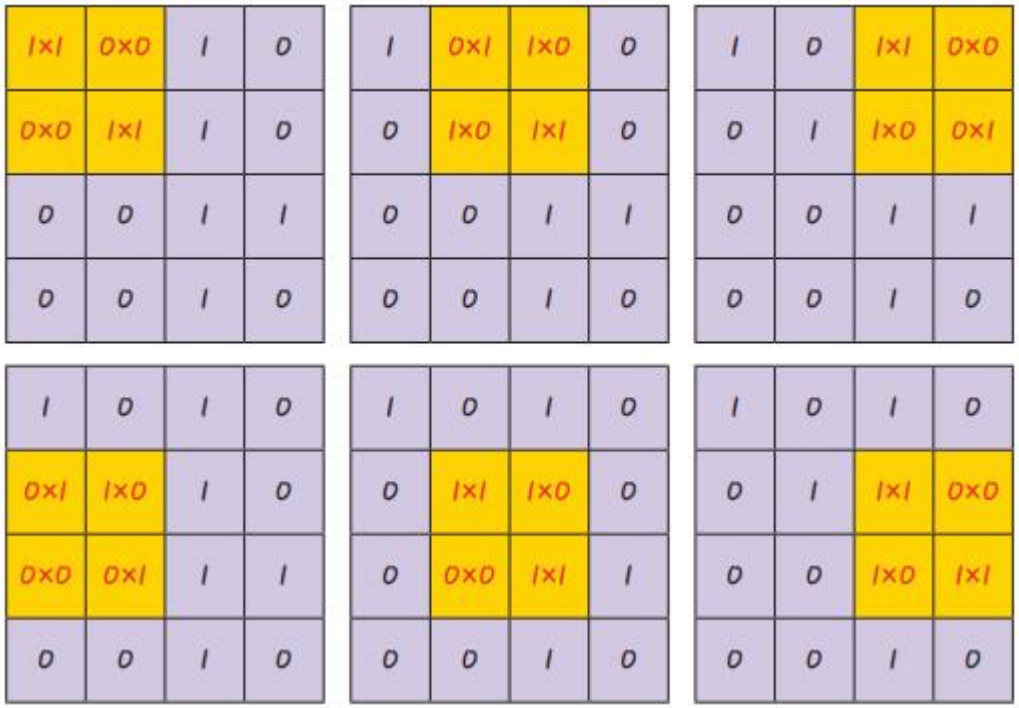
3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 적용된 부분은 원래 있던 값에 가중치의 값을 곱함
  - 그 결과를 합하면 새로 추출된 값은 2가 됨

$$(1\times 1)+(0\times 0)+(0\times 0)+(1\times 1)=2$$

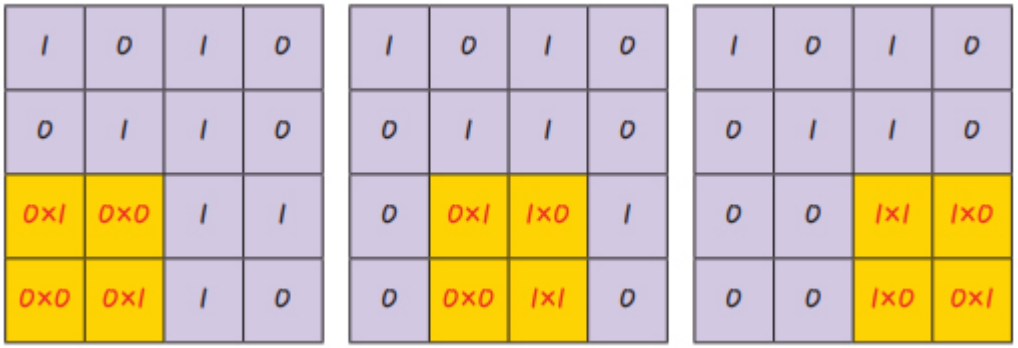
3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 이 커널을 한 칸씩 옮겨 모두 적용해 보자



3 컨볼루션 신경망(CNN)

- 컨볼루션 신경망(CNN)



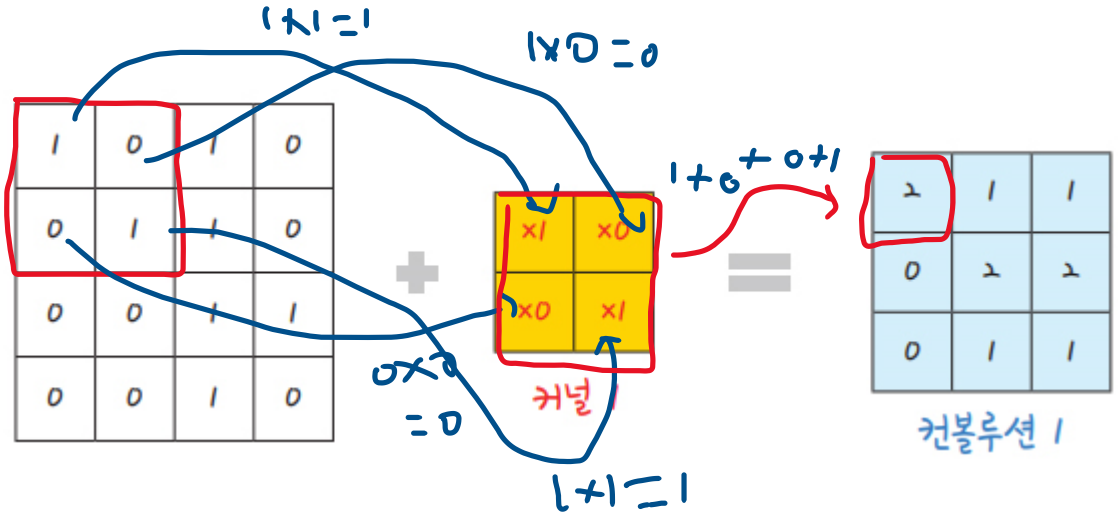
3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 그 결과를 정리하면 다음과 같음

2	1	1
0	2	2
0	1	1

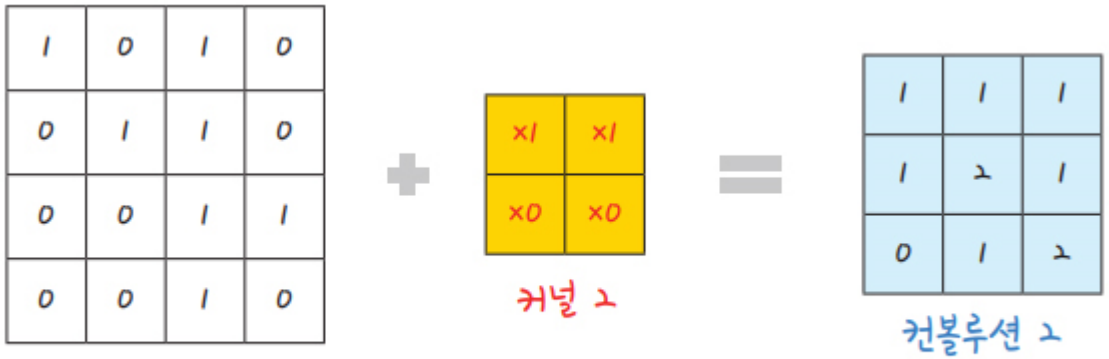
3 컨볼루션 신경망(CNN)

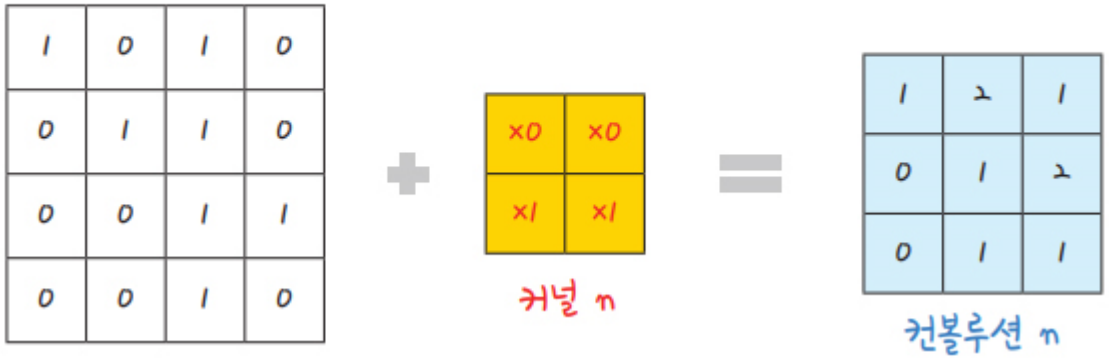
- **컨볼루션 신경망(CNN)**
  - 이렇게 해서 새롭게 만들어진 층을 컨볼루션(합성곱) 층이라고 함
  - 컨볼루션 층을 만들면 입력 데이터가 가진 특징을 대략적으로 추출해서 학습을 진행할 수 있음
  - 이러한 커널을 여러 개 만들 경우 여러 개의 컨볼루션 층이 만들어짐



3 컨볼루션 신경망(CNN)

- 컨볼루션 신경망(CNN)





3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 케라스에서 컨볼루션 층을 추가하는 함수는 Conv2D()
  - 다음과 같이 컨볼루션 층을 적용해 MNIST 손글씨 인식률을 높여 보자

```
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(28,28,1),
activation='relu'))
```

3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 여기에 입력된 네 가지 인자는 다음과 같음

1 | 첫 번째 인자: 커널을 몇 개 적용할지 정함
여기서는 **32개의 커널**을 적용

2 | kernel_size: 커널의 크기를 정함
kernel_size=(행, 열) 형식으로 정하며, 여기서는 **3×3 크기의 커널**을 사용하게끔 정했음

3 | input_shape: Dense 층과 마찬가지로 맨 처음 층에는 입력되는 값을 알려 주어야 함
input_shape=(행, 열, 색상 또는 흑백) 형식으로 정함
만약 입력 이미지가 색상이면 3, 흑백이면 **1**을 지정 여기서는 **28×28 크기의 흑백** 이미지를 사용하도록 정했음

4 | activation: 사용할 활성화 함수를 정의

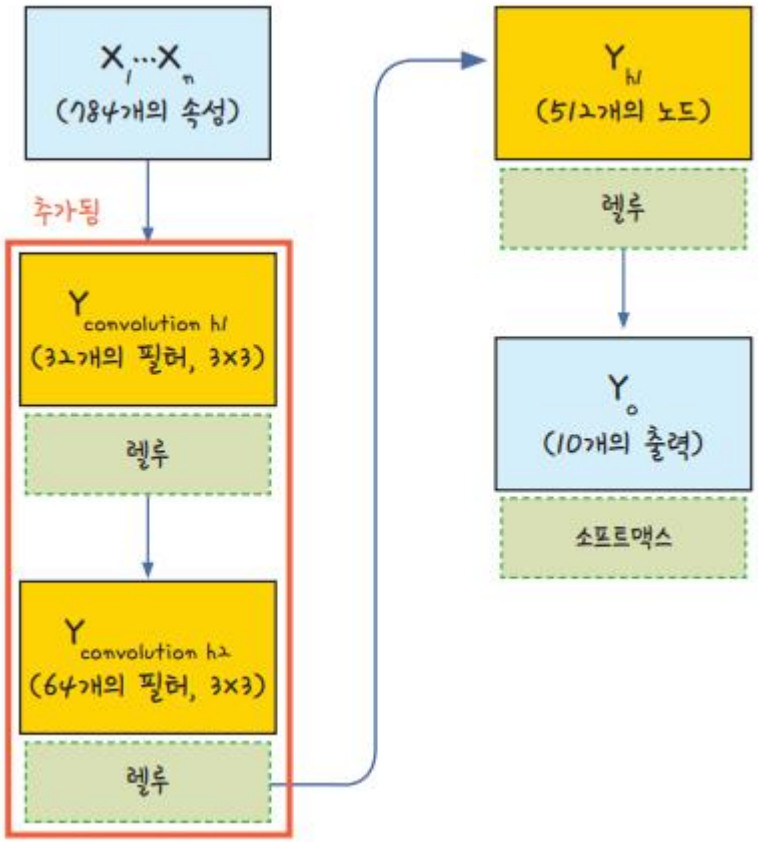
3 컨볼루션 신경망(CNN)

- **컨볼루션 신경망(CNN)**
  - 이어서 컨볼루션 층을 하나 더 추가해 보자
  - 다음과 같이 커널의 수는 64개, 커널의 크기는 3×3으로 지정하고 활성화 함수로 렐루를 사용하는 컨볼루션 층을 추가

```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

3 컨볼루션 신경망(CNN)

▼ 그림 16-6 | 컨볼루션 층의 적용



4 맥스 풀링, 드롭아웃, Flatten

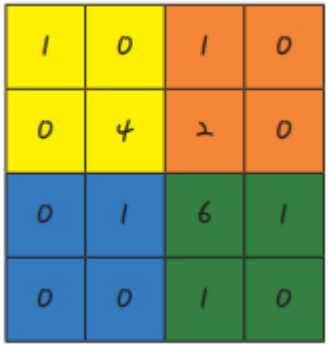
4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링, 드롭아웃, 플래튼**
  - 앞서 구현한 컨볼루션 층을 통해 이미지 특징을 도출
  - 그 결과가 여전히 크고 복잡하면 이를 다시 한 번 축소해야 함
  - 이 과정을 풀링(pooling) 또는 서브 샘플링(sub sampling)이라고 함
  - 이러한 풀링 기법에는 정해진 구역 안에서 최댓값을 뽑아내는 **맥스 풀링**(max pooling)과 평균 값을 뽑아내는 **평균 풀링**(average pooling) 등이 있음
  - 이 중 보편적으로 사용되는 맥스 풀링의 예를 들어 보자
  - 다음과 같은 이미지가 있다고 하자

1	0	1	0
0	4	2	0
0	1	6	1
0	0	1	0

4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링, 드롭아웃, 플래튼**
  - 맥스 풀링을 적용하면 다음과 같이 구역을 나눔



4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링, 드롭아웃, 플래튼**
  - 각 구역에서 가장 큰 값을 추출

4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링, 드롭아웃, 플래튼**
  - 이 과정을 거쳐 불필요한 정보를 간추림
  - 맥스 풀링은 MaxPooling2D( ) 함수를 사용해서 다음과 같이 적용할 수 있음

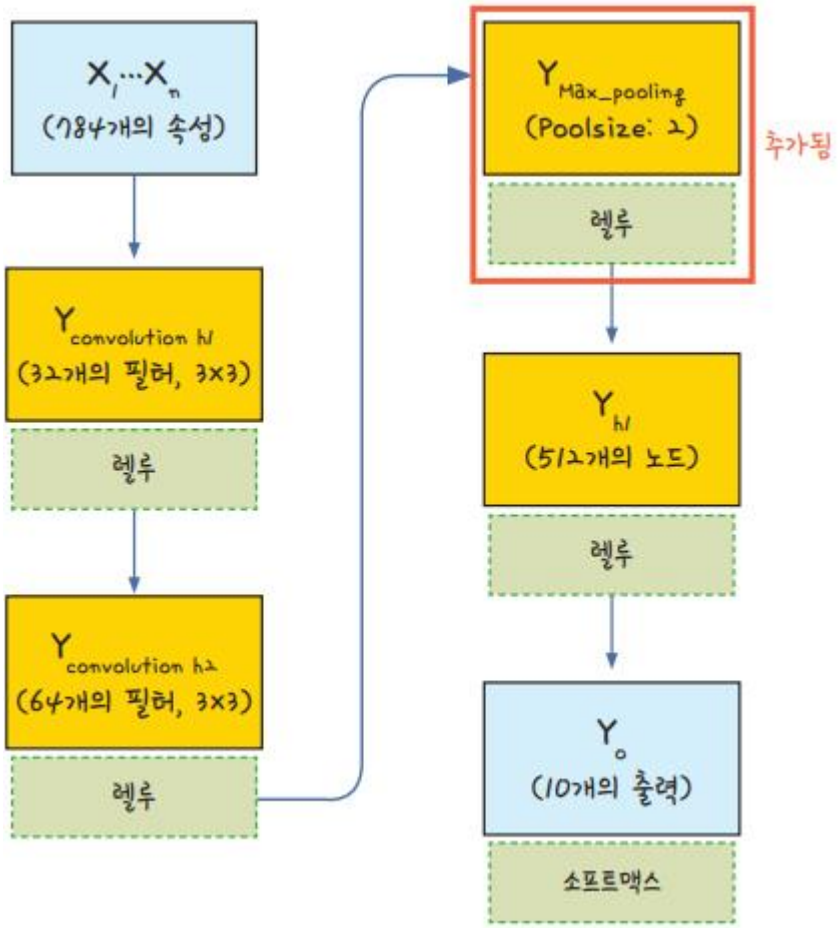
```
model.add(MaxPooling2D(pool_size=(2,2)))
```

4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링, 드롭아웃, 플래튼**
  - pool_size를 통해 풀링 창의 크기를 정하게 됨
  - (2,2)는 가로 2, 세로 2 크기의 풀링 창을 통해 맥스 풀링을 진행하라는 의미

4 맥스 풀링, 드롭아웃, 플래튼

▼ 그림 16-7 | 맥스 풀링 층 추가

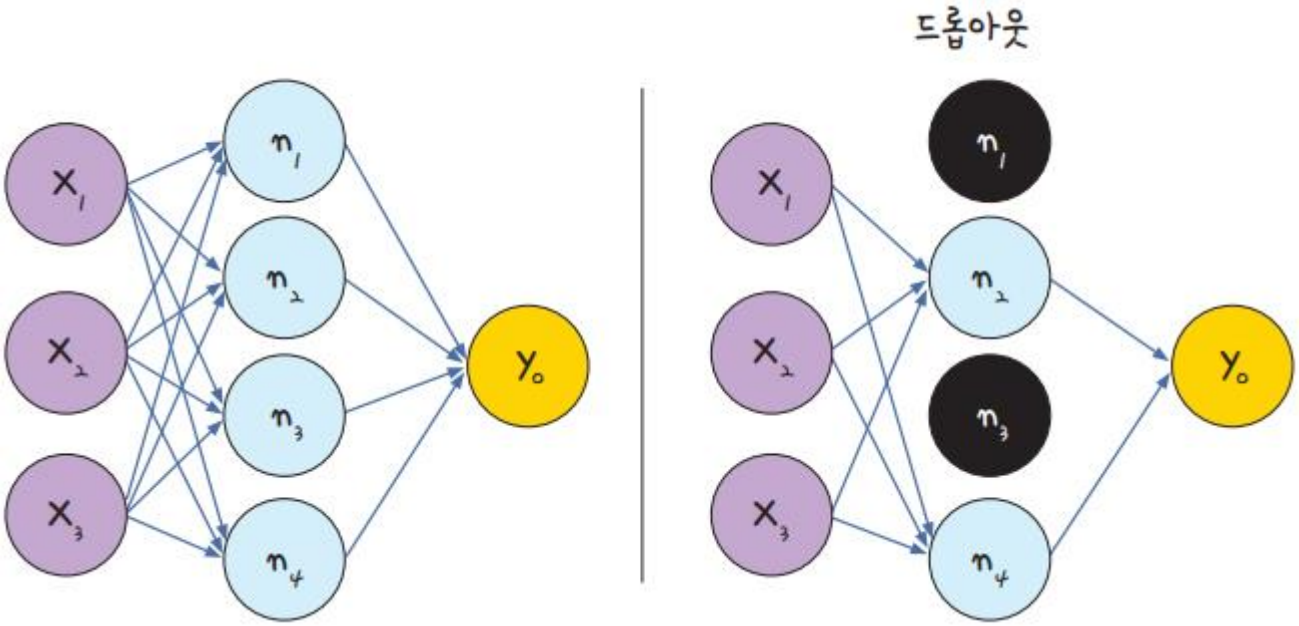


4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링(MaxPooling), 드롭아웃(Dropout), 플래튼(flatten)**
  - **드롭아웃, 플래튼**
  - 노드가 많아지거나 층이 많아진다고 해서 학습이 무조건 좋아지는 것이 아니라는 점을 과적합 의미를 공부하며 배웠음
  - 딥러닝에서 학습을 진행할 때 가장 중요한 것은 과적합을 얼마나 효과적으로 피해 가는지에 달려 있다고 해도 과언이 아님
  - 그동안 이러한 과정을 도와주는 기법이 연구되어 왔음
  - 그중 간단하지만 효과가 큰 기법이 바로 **드롭아웃**(drop out) 기법
  - 드롭아웃은 은닉층에 배치된 노드 중 일부를 임의로 꺼 주는 것

4 맥스 풀링, 드롭아웃, 플래튼

▼ 그림 16-8 | 드롭아웃의 개요, 검은색으로 표시된 노드는 계산하지 않는다



4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링, 드롭아웃, 플래튼**
  - 이렇게 랜덤하게 노드를 꺼 주면 학습 데이터에 지나치게 치우쳐서 학습되는 과적합을 방지할 수 있음
  - 케라스는 이러한 드롭아웃을 손쉽게 적용하도록 도와줌
  - 예를 들어 25%의 노드를 끄려면 다음과 같이 코드를 작성하면 됨

```
model.add(Dropout(0.25))
```

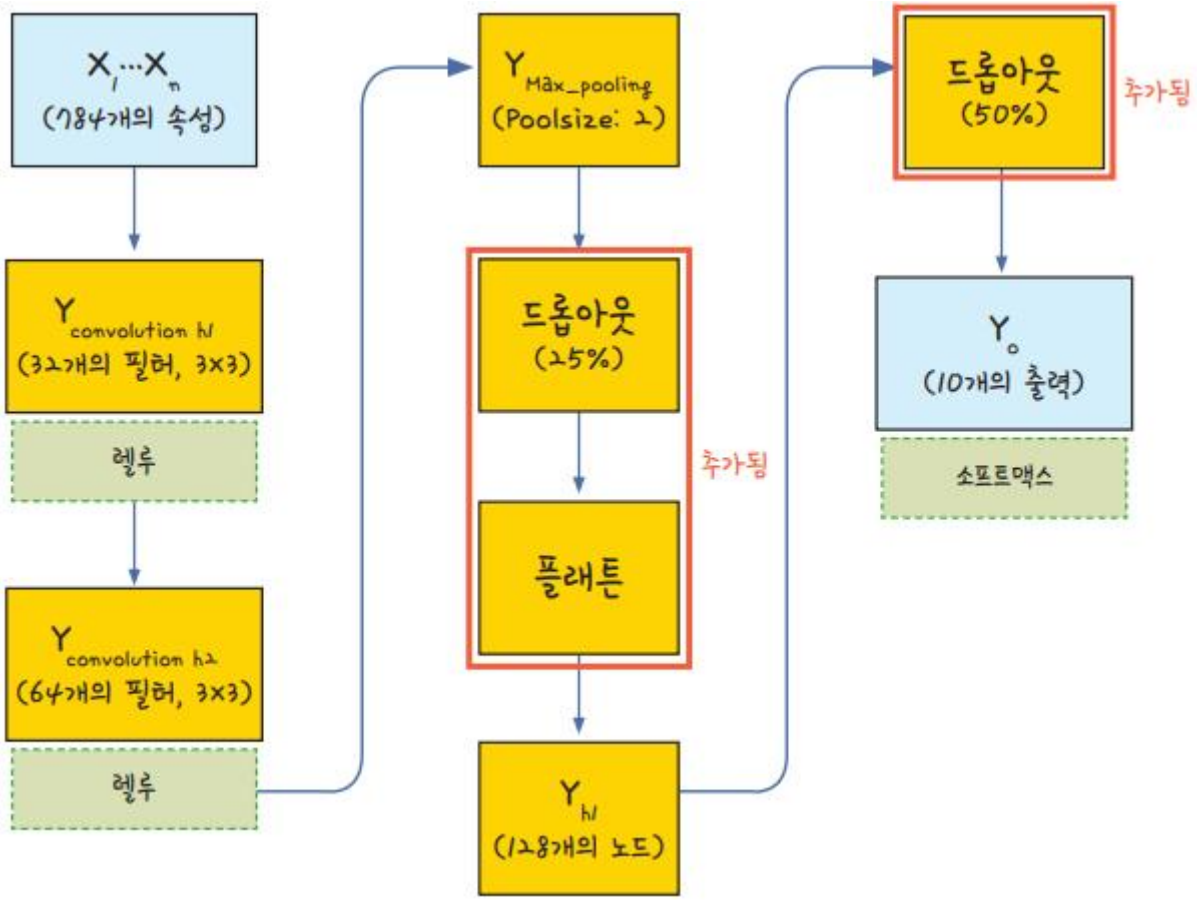
4 맥스 풀링, 드롭아웃, 플래튼

- **맥스 풀링, 드롭아웃, 플래튼**
  - 이제 이러한 과정을 지나 다시 앞에서 Dense() 함수를 이용해 만들었던 기본 층에 연결해 볼까?
  - 이때 주의할 점은 컨볼루션 층이나 맥스 풀링은 주어진 이미지를 2차원 배열인 채로 다룬다는 것
  - 이를 1차원 배열로 바꾸어 주어야 활성화 함수가 있는 층에서 사용할 수 있음
  - Flatten() 함수를 사용해 2차원 배열을 1차원으로 바꾸어 줌

```
model.add(Flatten())
```

4 맥스 풀링, 드롭아웃, 플래튼

▼ 그림 16-9 | 드롭아웃과 플래튼 추가하기



5 컨볼루션 신경망 실행하기

5 컨볼루션 신경망 실행하기

- **컨볼루션 신경망 실행하기**
  - 지금까지 살펴본 내용을 코드로 작성해 보자
  - 앞서 16.2절에서 만든 딥러닝 기본 프레임 코드를 그대로 이용하되 model 설정 부분만 지금까지 나온 내용으로 바꾸어 주면 됨

실습 | MNIST 손글씨 인식하기: 컨볼루션 신경망 적용

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, Max
Pooling2D
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

5 컨볼루션 신경망 실행하기

- 컨볼루션 신경망 실행하기

```
import matplotlib.pyplot as plt
import numpy as np

# 데이터를 불러옵니다.
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32') / 255
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

5 컨볼루션 신경망 실행하기

- 컨볼루션 신경망 실행하기

```
# 컨볼루션 신경망의 설정
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), input_shape=(28,28,1),
activation='relu'))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

5 컨볼루션 신경망 실행하기

- 컨볼루션 신경망 실행하기

```
# 모델의 실행 옵션을 설정합니다.
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# 모델 최적화를 위한 설정 구간입니다.
modelpath = "./MNIST_CNN.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss',
verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 모델을 실행합니다.
history = model.fit(X_train, y_train, validation_split=0.25, epochs=30,
batch_size=200, verbose=0, callbacks=[early_stopping_callback,
checkpointer])
```

5 컨볼루션 신경망 실행하기

- 컨볼루션 신경망 실행하기

```
# 테스트 정확도를 출력합니다.
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))

# 검증셋과 학습셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프로 표현해 봅니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')
```

5 컨볼루션 신경망 실행하기

- 컨볼루션 신경망 실행하기

```
# 그래프에 그리드를 주고 레이블을 표시하겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

5 컨볼루션 신경망 실행하기

- 컨볼루션 신경망 실행하기

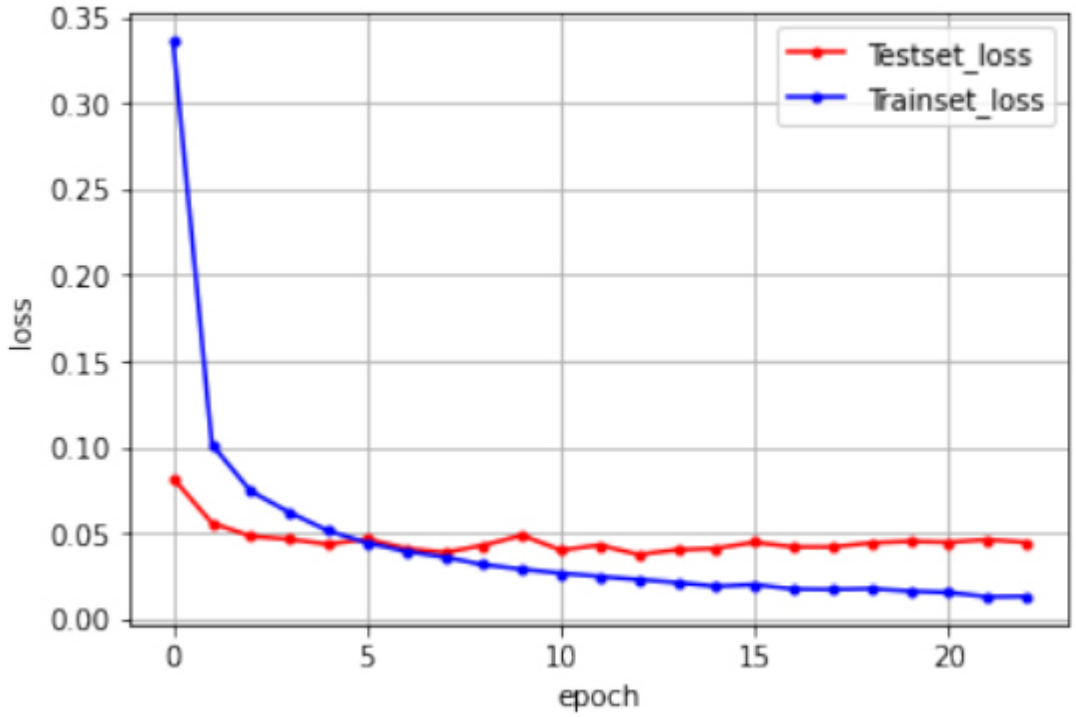
실행 결과

```
Epoch 00001: val_loss improved from inf to 0.08178, saving model to ../
data/model\MNIST_CNN.hdf5
... (중략) ...
Epoch 00008: val_loss improved from 0.04101 to 0.03858, saving model to ../
data/model\MNIST_CNN.hdf5
... (중략) ...

Epoch 00023: val_loss did not improve from 0.03742
313/313 [==============================] - 2s 6ms/step - loss: 0.0341 -
accuracy: 0.9917

 Test Accuracy: 0.9917
```

5 컨볼루션 신경망 실행하기

- 컨볼루션 신경망 실행하기

▼ 그림 16-10 | 학습 진행에 따른 학습셋과 테스트셋의 오차 변화



5 컨볼루션 신경망 실행하기

- **컨볼루션 신경망 실행하기**
  - 8번째 에포크에서 베스트 모델을 만들었고 23번째 에포크에서 학습이 중단
  - 테스트 정확도는 99.17%로 향상
  - 학습 과정을 그래프로 확인해 본 결과, 컨볼루션 신경망 모델로 만든 학습셋과 테스트셋의 오차가 이전의 얕은 구조로 만든 딥러닝 모델보다 작아졌음을 볼 수 있음
  - 99.17%의 정확도는 1만 개의 테스트 이미지 중 9,917개를 맞혔다는 의미
  - 바로 앞의 'MNIST 손글씨 인식하기: 기본 프레임' 실습에서는 정확도가 98.16%였으므로 101개의 이미지를 더 맞혔음
  - 100% 다 맞히지 못한 이유는 데이터 안에 다음과 같이 확인할 수 없는 글씨가 있었기 때문임

5 컨볼루션 신경망 실행하기

▼ 그림 16-11 | 알아내지 못한 숫자의 예