AI 라이브러리 활용

16장 CNN 익히기 2

이 찬 우

학습내용 : *이미지 인식의 꽃, CNN 익히기 2*

1 | 데이터 전처리

2 | 딥러닝 기본 프레임 만들기

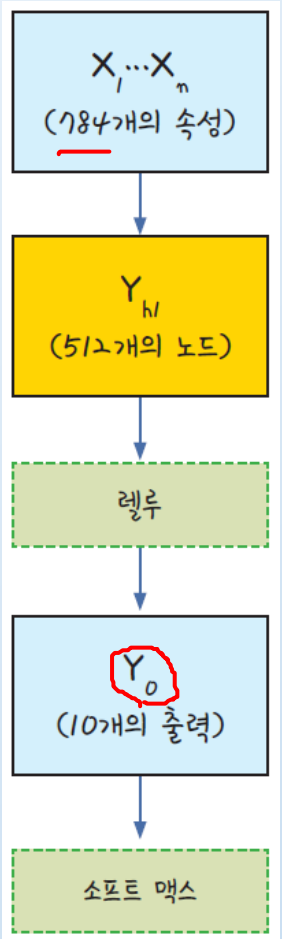
3 | 더 깊은 딥러닝

4 | 컨볼루션 신경망(CNN)

5 | 맥스 풀링

6 | 컨볼루션 신경망 실행하기

3 | 더 깊은 딥러닝



- 앞서 98.21%의 정확도를 보인 딥러닝 프레임은 하나의 은닉층을 둔 아주 단순한 모델
- 딥러닝은 이러한 기본 모델을 바탕으로,
  → 프로젝트에 맞춰서 어떤 옵션을 더하고 어떤 층을 추가하느냐에 따라 성능이 좋아질 수 있음

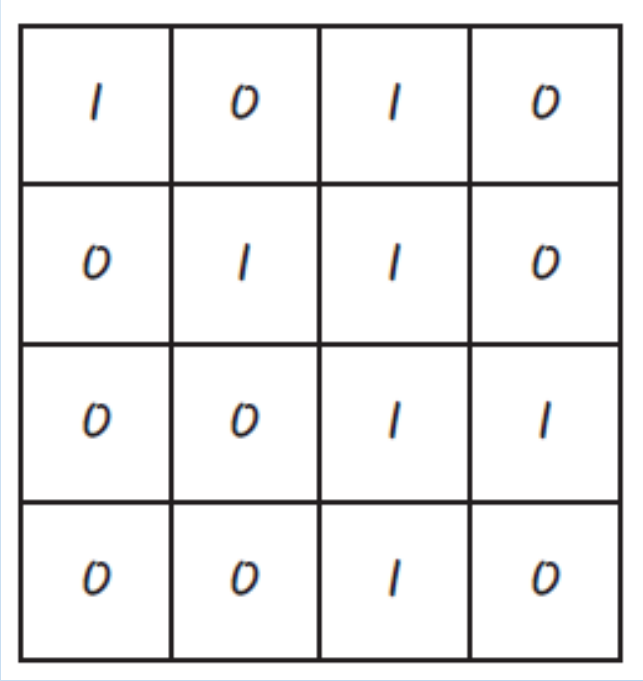
그림 16-5 은닉층이 하나인 딥러닝 모델의 도식

4 | 컨볼루션 신경망(CNN)

- 컨볼루션 신경망 :

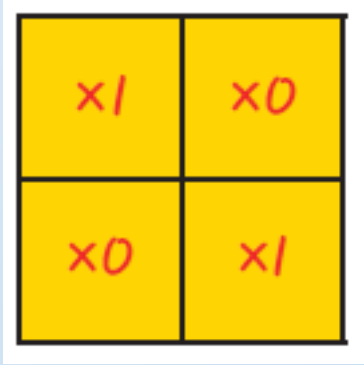
  입력된 이미지에서 다시 한번 특징을 추출하기 위해 마스크

  (필터, 윈도 또는 커널이라고도 함)를 도입하는 기법



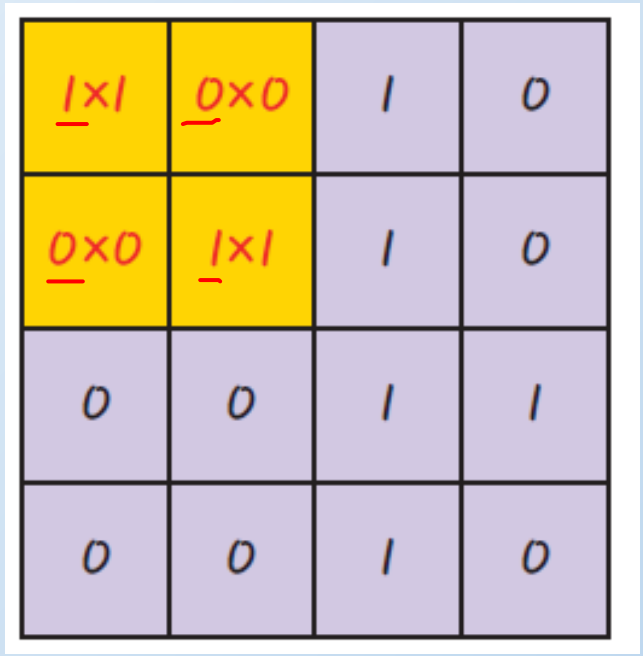
4 | 컨볼루션 신경망(CNN)

- 여기에 2×2 마스크를 준비함
- 각 칸에는 가중치가 들어있음
- 샘플 가중치를 다음과 같이 ×1, ×0라고 함



4 | 컨볼루션 신경망(CNN)

- 이제 마스크를 맨 왼쪽 위칸에 적용시켜 보자



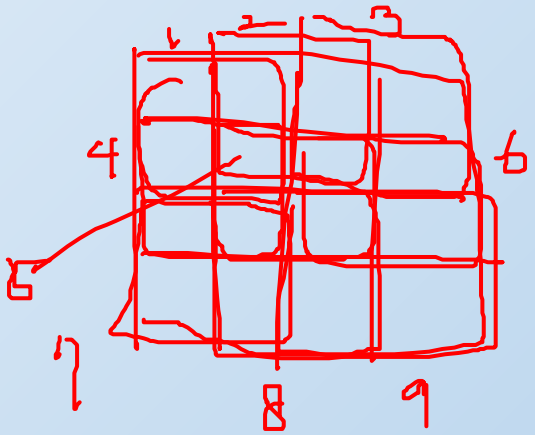
⇒ 2

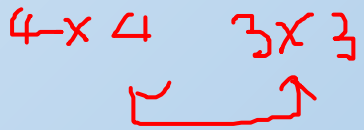
4 | 컨볼루션 신경망(CNN)

- 적용된 부분은 원래 있던 값에 가중치의 값을 곱해 줌
- 그 결과를 합하면 새로 추출된 값은 2가 됨

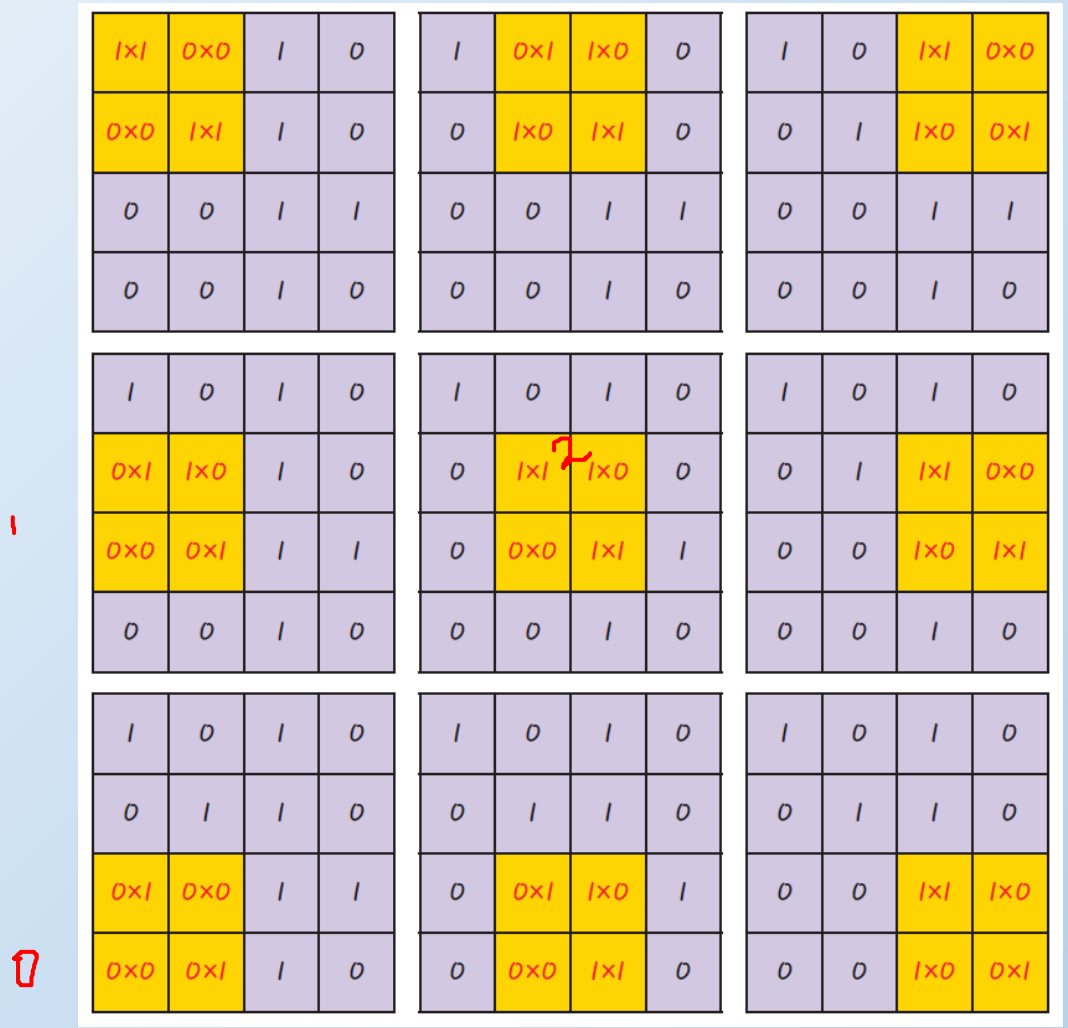
$$(1\times 1)+(0\times 0)+(0\times 0)+(1\times 1)=2$$

- 이 마스크를 한 칸씩 옮겨 모두 적용해 보자



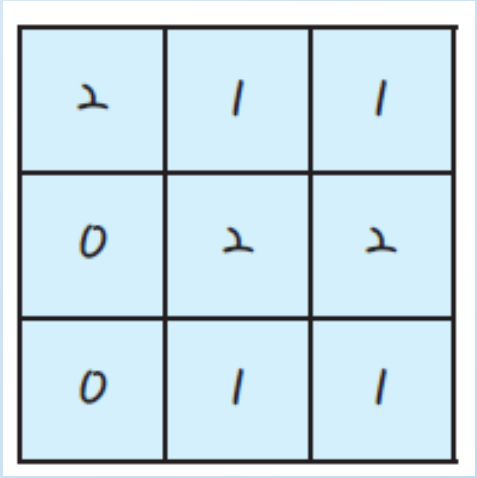


4 | 컨볼루션 신경망(CNN)



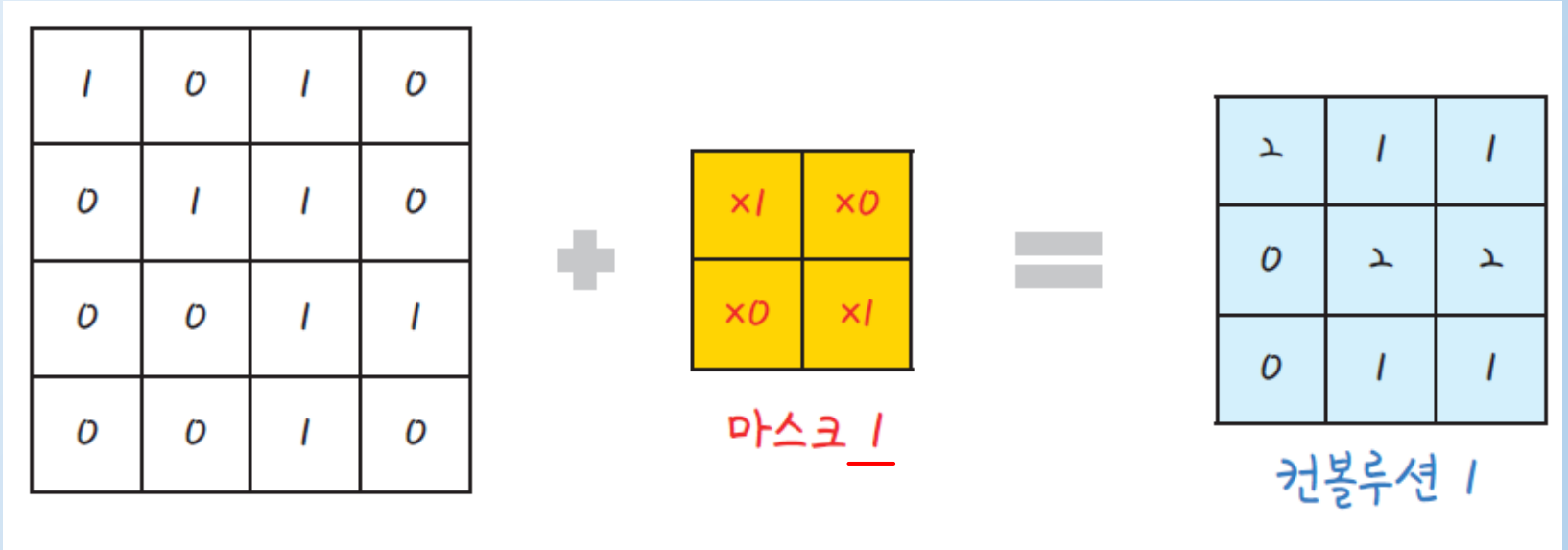
4 | 컨볼루션 신경망(CNN)

- 그 결과를 정리하면 다음과 같음

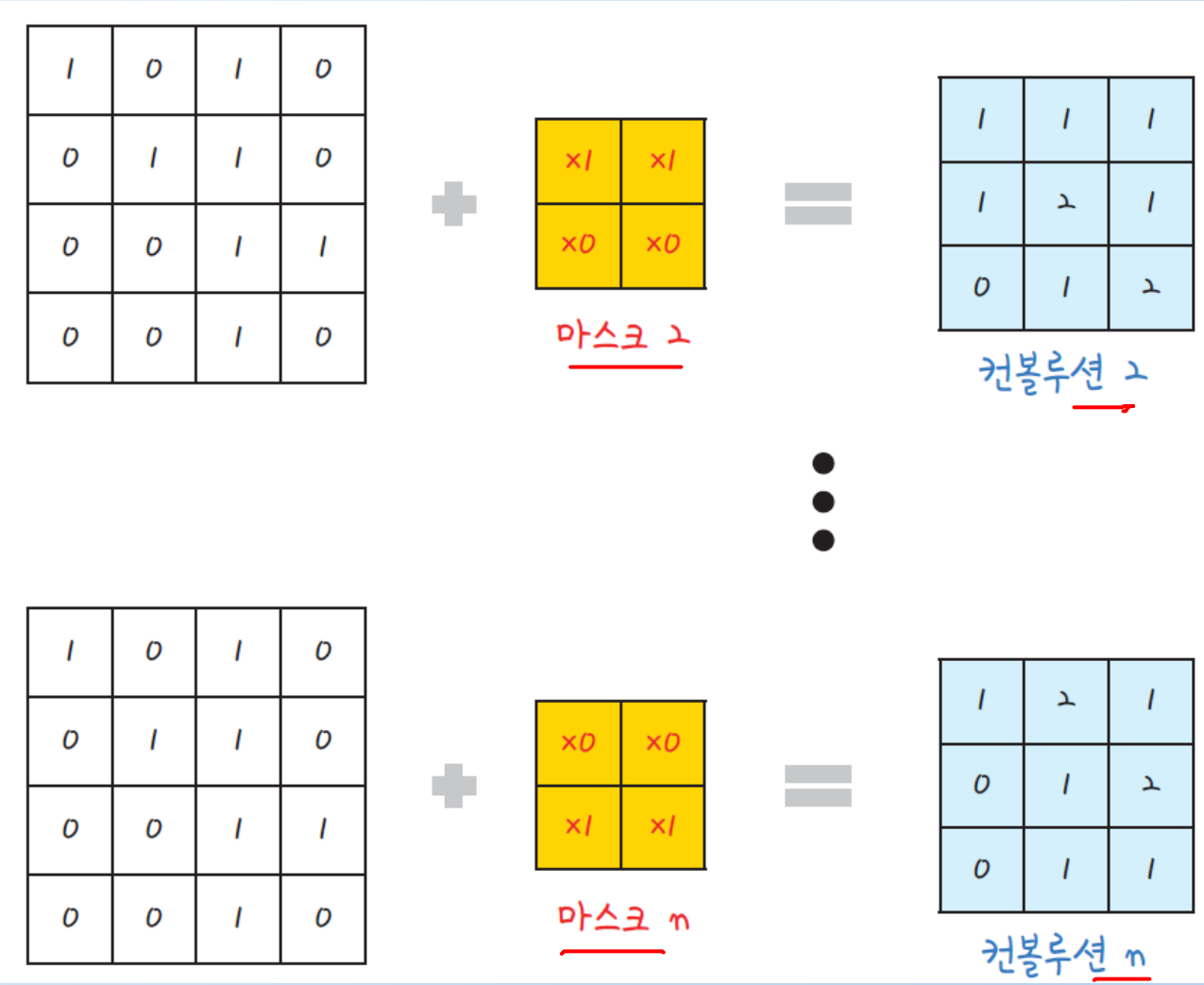


4 | 컨볼루션 신경망(CNN)

- 새롭게 만들어진 층을 컨볼루션(합성곱)이라고 부름
- 컨볼루션을 만들면 입력 데이터로부터 더욱 정교한 특징을 추출할 수 있음
- 이러한 마스크를 여러 개 만들 경우 여러 개의 컨볼루션이 만들어짐



4 | 컨볼루션 신경망(CNN)



4 | 컨볼루션 신경망(CNN)

- 케라스에서 컨볼루션 층을 추가하는 함수는 Conv2D( )임
- 다음과 같이 컨볼루션 층을 적용하여 MNIST 손글씨 인식률을 높여봄

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

4 | 컨볼루션 신경망(CNN)

- 여기에 입력된 4가지 인자는 다음과 같음

  1 | 첫 번째 인자: 마스크를 몇 개 적용할지 정함

  여러 개의 마스크를 적용하면 서로 다른 컨볼루션이 여러 개 나옴

  여기서는 32개의 마스크를 적용함

  2 | kernel_size: 마스크(커널)의 크기를 정함

  kernel_size=(행, 열) 형식으로 정하며, 여기서는 3×3 크기의 마스크를 사용하게끔 정함

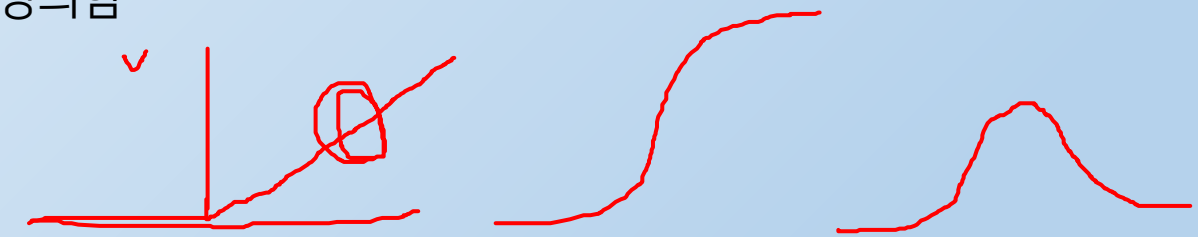
4 | 컨볼루션 신경망(CNN)

3 | input_shape: Dense 층과 마찬가지로 맨 처음 층에는 입력되는 값을 알려주어야 함

input_shape=(행, 열, 색상 또는 흑백) 형식으로 정함

만약 입력 이미지가 색상이면 3, 흑백이면 1을 지정함

4 | activation: 활성화 함수를 정의함



4 | 컨볼루션 신경망(CNN)

- 컨볼루션 층을 하나 더 추가함
- 다음과 같이 마스크 64개를 적용한 새로운 컨볼루션 층을 추가할 수 있음

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

4 | 컨볼루션 신경망(CNN)

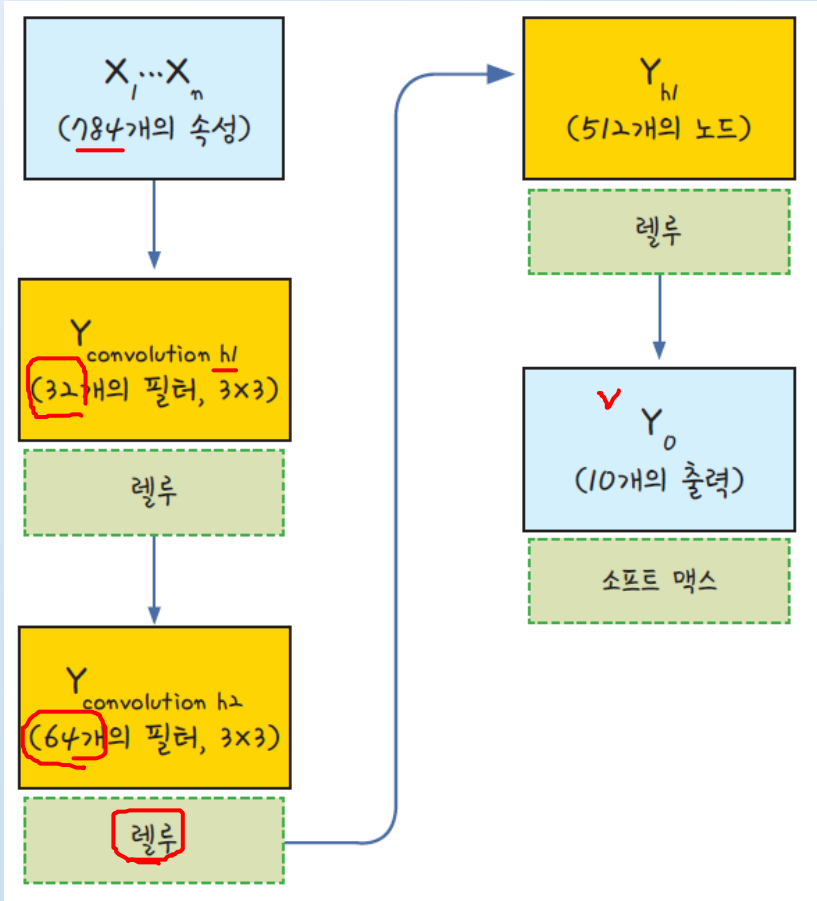


그림 16-6 컨볼루션 층의 적용

5 | 맥스 풀링

- 풀링(pooling) 또는 서브 샘플링(sub sampling) :

  앞서 구현한 컨볼루션 층을 통해 이미지 특징을 도출함

  그 결과가 여전히 크고 복잡하면 이를 다시 한번 축소해야 함

- 맥스 풀링(max pooling) :

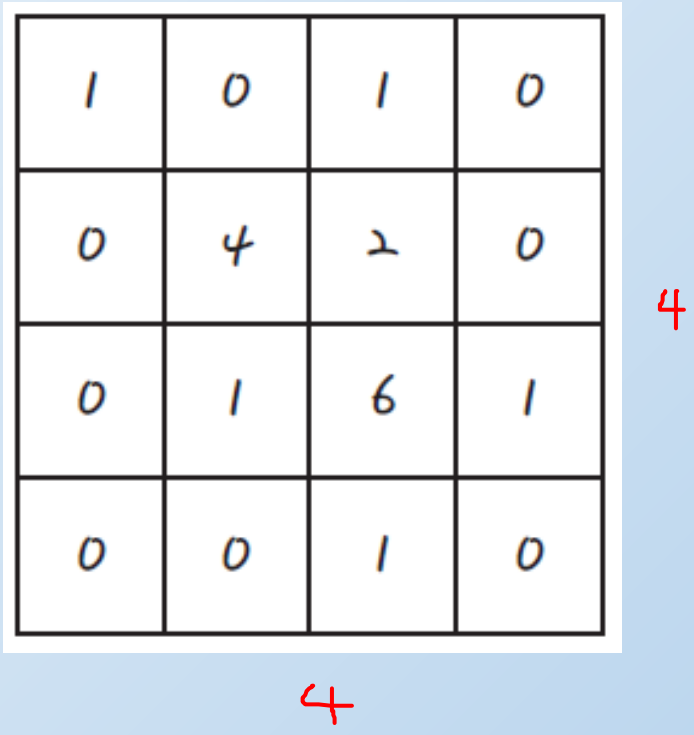
  풀링 기법에는 정해진 구역 안에서 최댓값을 뽑아내는 것

- 평균 풀링(average pooling) :

  평균값을 뽑아내는 것

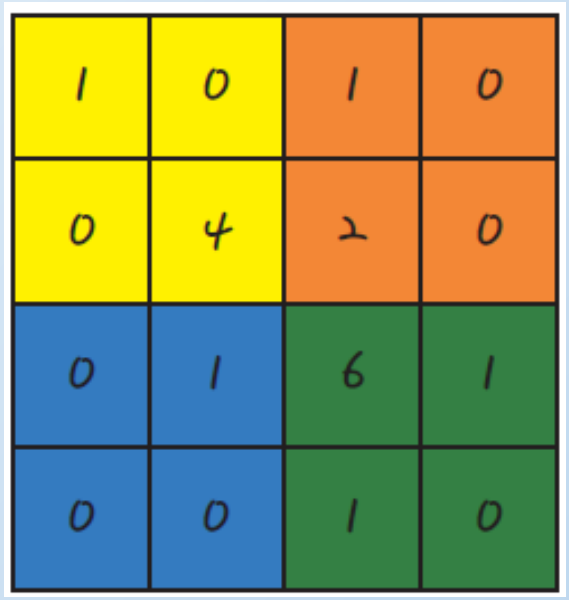
5 | 맥스 풀링

- 이중 보편적으로 사용되는 맥스 풀링의 예를 들어 보자



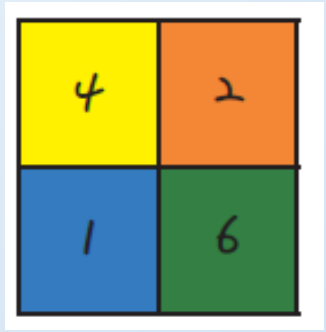
5 | 맥스 풀링

- 맥스 풀링을 적용하면 다음과 같이 구역을 나눔



5 | 맥스 풀링

- 각 구역에서 가장 큰 값을 추출함



max

5 | 맥스 풀링

- 이 과정을 거쳐 불필요한 정보를 간추림

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

5 | 맥스 풀링

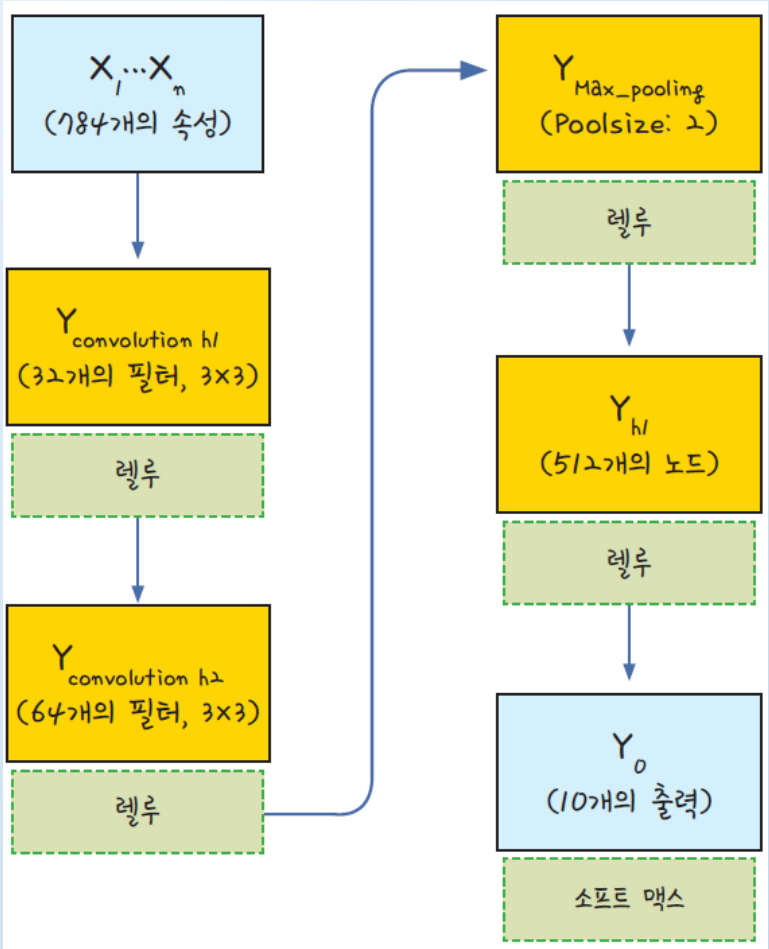


그림 16-5 은닉층이 하나인 딥러닝 모델의 도식

- 여기서 pool_size는 풀링 창의 크기를 정하는 것으로,

  → 2로 정하면 전체 크기가 절반으로 줄어듦

5 | 맥스 풀링

드롭아웃, 플래튼

- 딥러닝 학습을 실행할 때 가장 중요한 것은 과적합을 얼마나 효과적으로 피해가는 지에 달려 있다고 해도 과언이 아님
- 그동안 이러한 과정을 도와주는 기법이 연구되어 옴
- 그 중 간단하지만 효과가 큰 기법이 바로 드롭아웃(drop out) 기법임
- 드롭아웃은 은닉층에 배치된 노드 중 일부를 임의로 꺼주는 것

5 | 맥스 풀링

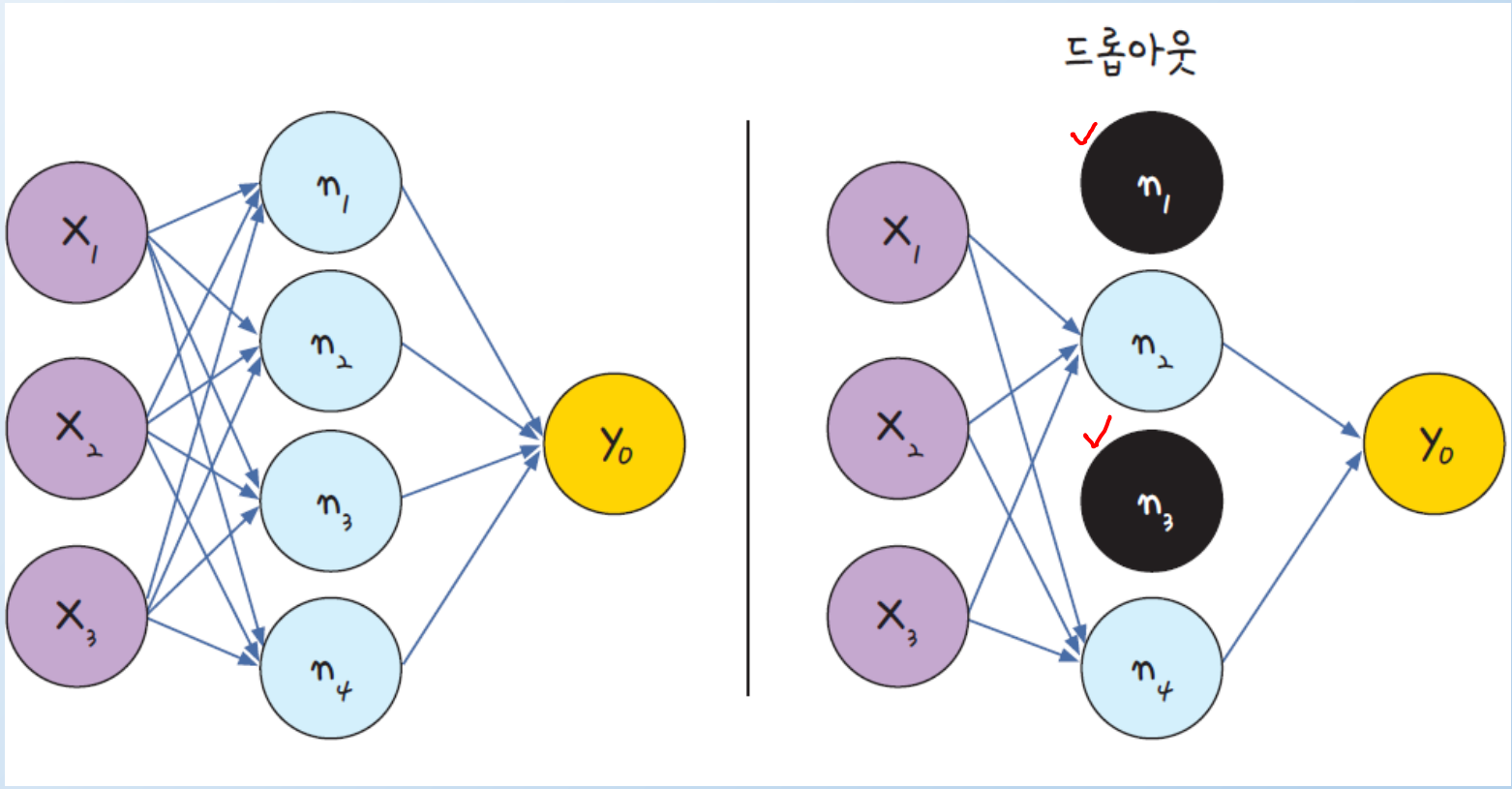


그림 16-8 드롭아웃의 개요, 검은색으로 표시된 노드는 계산하지 않는다.

5 | 맥스 풀링

- 랜덤하게 노드를 끔으로써 학습 데이터에 지나치게 치우쳐서 학습되는 과적합을 방지할 수 있음
- 케라스는 이를 손쉽게 적용하도록 도와줌

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

5 | 맥스 풀링

- 이제 이러한 과정을 지나 다시 앞에서 Dense( ) 함수를 이용해 만들었던 기본 층에 연결해 볼 때

  → 이때 주의할 점은 컨볼루션 층이나 맥스 풀링은 주어진 이미지를 2차원 배열인 채로 다룬다는 점임

  → 이를 1차원 배열로 바꿔주어야 활성화 함수가 있는 층에서 사용할 수 있음

  → Flatten( ) 함수를 사용해 2차원 배열을 1차원으로 바꿔줌

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

5 | 맥스 풀링

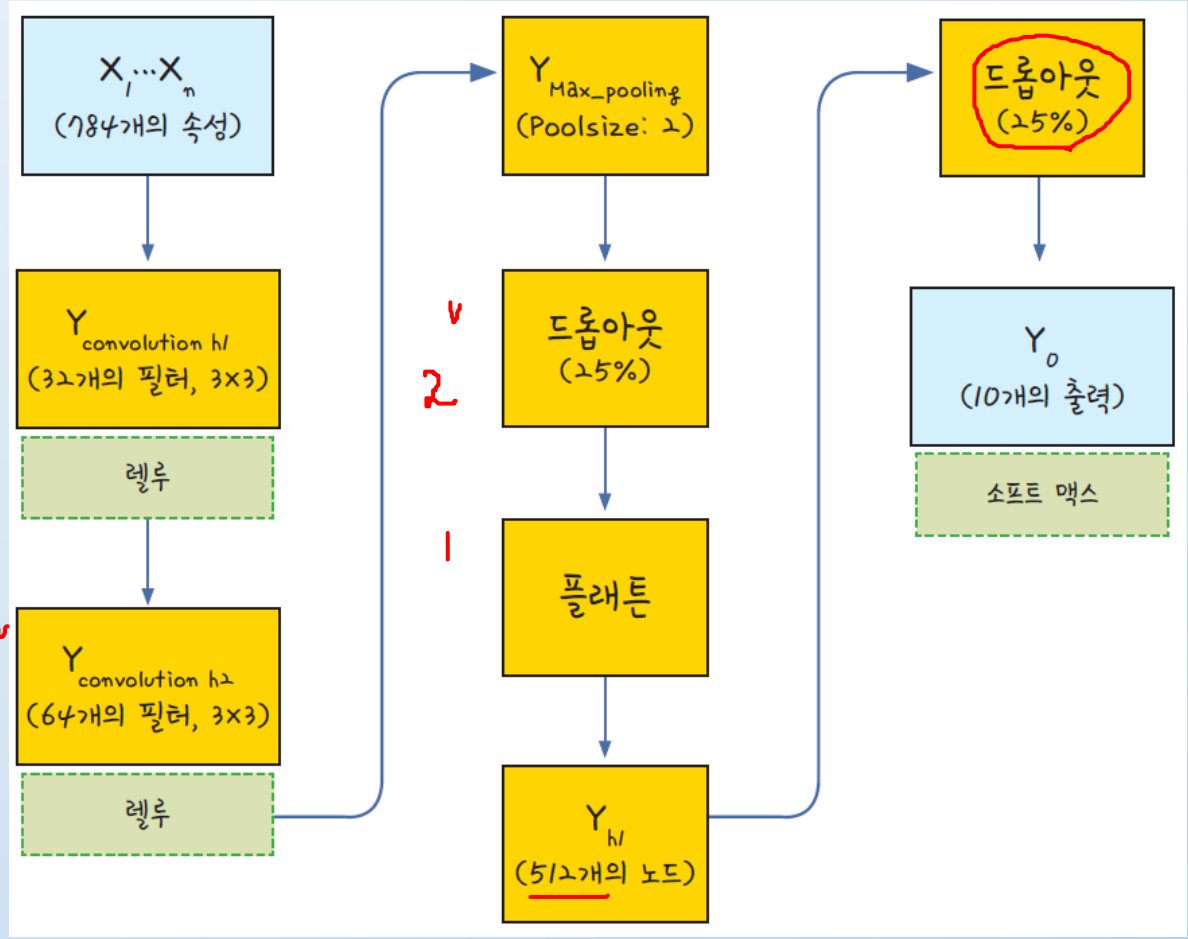


그림 16-9 드롭아웃과 플래튼 추가하기

6 | 컨볼루션 신경망 실행하기

- 앞서 코드 16-2에서 만든 딥러닝 기본 프레임을 그대로 이용하되 model 설정 부분만 지금까지 나온 내용으로 바꿔주면 됨

코드 16-3 MNIST 손글씨 인식하기: 컨볼루션 신경망 적용

- **예제 소스**: run_project/16_MNIST_Deep.ipynb

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D,
MaxPooling2D
from keras.callbacks import ModelCheckpoint,EarlyStopping
```

6 | 컨볼루션 신경망 실행하기

```
import matplotlib.pyplot as plt
import numpy
import os
import tensorflow as tf

# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)
```

6 | 컨볼루션 신경망 실행하기

```
# 데이터 불러오기
(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 = np_utils.to_categorical(Y_train)
Y_test = np_utils.to_categorical(Y_test)
```

6 | 컨볼루션 신경망 실행하기

```
# 컨볼루션 신경망 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))
model.add(Dropout(0.25))
model.add(Flatten( ))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
```

6 | 컨볼루션 신경망 실행하기

```
model.add(Dense(10, activation='softmax'))

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

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

6 | 컨볼루션 신경망 실행하기

```
modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_
loss', verbose=1, save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss',
patience=10)

# 모델의 실행
history = model.fit(X_train, Y_train, validation_data=(X_test, Y_
test), epochs=30, batch_size=200, verbose=0, callbacks=[early_
stopping_callback,checkpointer])
```

6 | 컨볼루션 신경망 실행하기

```
# 테스트 정확도 출력
print("\n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test)
[1]))

# 테스트셋의 오차
y_vloss = history.history['val_loss']

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

6 | 컨볼루션 신경망 실행하기

```
# 그래프로 표현
x_len = numpy.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 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

6 | 컨볼루션 신경망 실행하기

실행 결과

```
Epoch 00000: val_loss improved from inf to 0.06068, saving model to ./
model/00-0.0607.hdf5
Epoch 00001: val_loss improved from 0.06068 to 0.04409, saving model to ./
model/01-0.0441.hdf5
Epoch 00002: val_loss improved from 0.04409 to 0.03909, saving model to ./
model/02-0.0391.hdf5
Epoch 00003: val_loss improved from 0.03909 to 0.03188, saving model to ./
model/03-0.0319.hdf5
Epoch 00004: val_loss improved from 0.03188 to 0.02873, saving model to ./
model/04-0.0287.hdf5
Epoch 00005: val_loss did not improve
Epoch 00006: val_loss did not improve
Epoch 00007: val_loss did not improve
Epoch 00008: val_loss improved from 0.02873 to 0.02678, saving model to ./
model/08-0.0268.hdf5
```

6 | 컨볼루션 신경망 실행하기

```
Epoch 00009: val_loss did not improve
Epoch 00010: val_loss improved from 0.02678 to 0.02617, saving model to ./model/10-0.0262.hdf5
Epoch 00011: val_loss did not improve
Epoch 00012: val_loss did not improve
Epoch 00013: val_loss improved from 0.02617 to 0.02454, saving model to ./model/13-0.0245.hdf5
Epoch 00014: val_loss did not improve
Epoch 00015: val_loss did not improve
Epoch 00016: val_loss did not improve
Epoch 00017: val_loss did not improve
```

6 | 컨볼루션 신경망 실행하기

```
Epoch 00018: val_loss did not improve
Epoch 00019: val_loss did not improve
Epoch 00020: val_loss did not improve
Epoch 00021: val_loss did not improve
Epoch 00022: val_loss did not improve
Epoch 00023: val_loss did not improve
Epoch 00024: val_loss did not improve

Test Accuracy: 0.9928
```

TIP

CPU 기반으로 코드 16-3을 실행하면 앞서 다룬 예제보다 시간이 더 오래 걸릴 것입니다. 이 책에서 소개한 작은 프로젝트 정도면 CPU에서 실습해도 큰 상관이 없지만, 더 큰 딥러닝 프로젝트는 반드시 GPU 환경을 갖추어 시행할 것을 추천합니다. 또한, 결과는 실행할 때마다 다를 수도 있습니다.

6 | 컨볼루션 신경망 실행하기

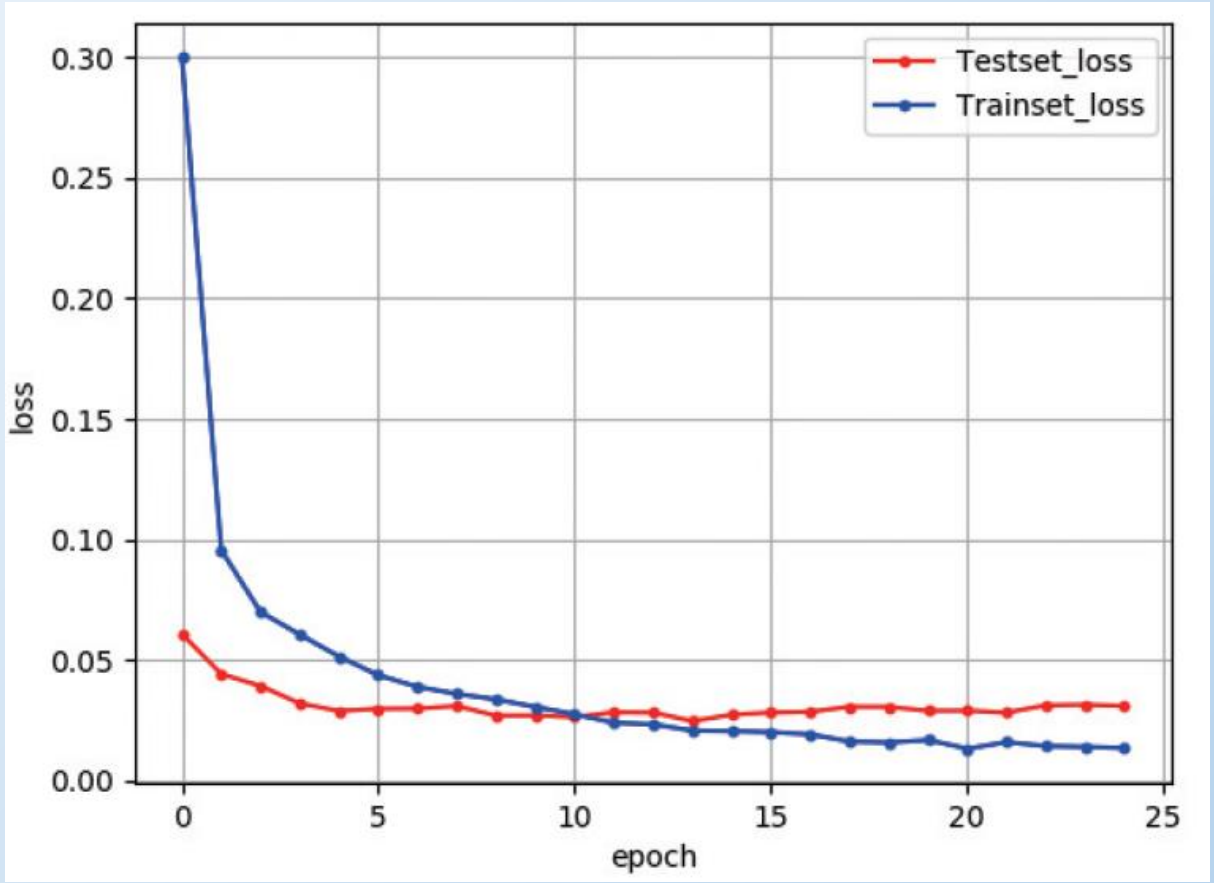


그림 16-10 학습의 진행에 따른 학습셋과 테스트셋의 오차 변화

6 | 컨볼루션 신경망 실행하기

- 100% 다 맞히지 못한 이유는 데이터 안에 다음과 같이 확인할 수 없는 글씨가 있었기 때문임

그림 16-11 알아내지 못한 숫자의 예

정리 학습 : *이미지 인식의 꽃, CNN 익히기 2*

1 | 데이터 전처리

2 | 딥러닝 기본 프레임 만들기

3 | 더 깊은 딥러닝

4 | 컨볼루션 신경망(CNN)

5 | 맥스 풀링

6 | 컨볼루션 신경망 실행하기

다음 시간

딥러닝을 이용한 자연어 처리