

1. 데이터 전처리



- **Mnist 데이터셋**
70,000개의 글자
이미지에 0~9까지의
이름표를 붙인
데이터셋

1. 데이터 전처리

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

```
import numpy
import sys
import tensorflow as tf
```

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

```
# MNIST데이터셋 불러오기
(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_data()
```

```
print("학습셋 이미지 수 : %d 개" % (X_train.shape[0])) # 총 샘플수
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))
```

```
# 이미지 출력
import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='Greys')
plt.show()
```

1. 케라스로 MNIST데이터 호출

- 학습에 사용될 부분: X_train, Y_class_train
- 테스트에 사용될 부분: X_test, Y_class_test

2. MNIST데이터는 총 70,000개로
60,000 -> 학습용 / 10,000 -> 테스트용
으로 미리 구분되어 있다.

3. matplotlib라이브러리 호출하여
imshow()함수로 이미지 출력
- X_train[0] : 첫번째 이미지
- cmap='Greys' : 흑백

1. 데이터 전처리

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

import numpy
import sys
import tensorflow as tf

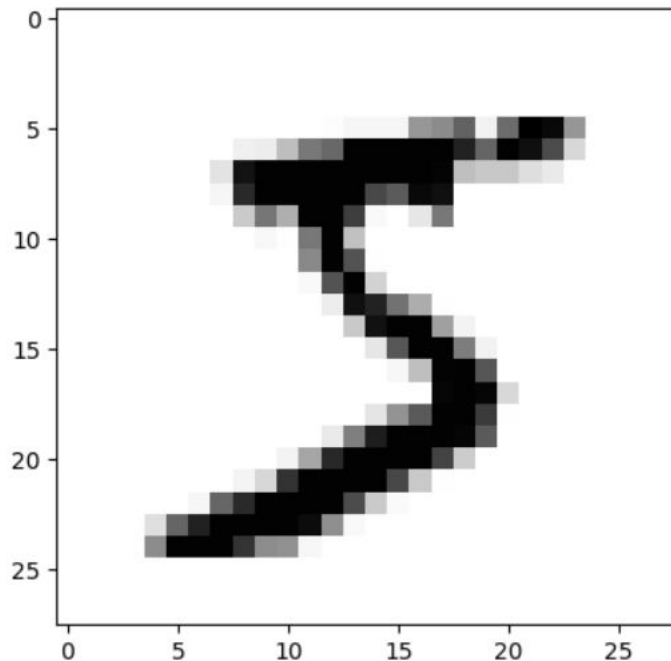
# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)

# MNIST데이터셋 불러오기
(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_data()

print("학습셋 이미지 수 : %d 개" % (X_train.shape[0])) # 총 샘플수
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))

# 이미지 출력
import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='Greys')
plt.show()
```

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



1. 데이터 전처리

```
# 2차원 배열의 숫자 집합으로 출력
for x in X_train[0]:
    for i in x:
        sys.stdout.write('%d\t' % i)
    sys.stdout.write('\n')

# X_train 1차원 배열로 변환 과정
X_train = X_train.reshape(X_train.shape[0], 784)
X_train = X_train.astype('float64') # 실수형으로 변환하여
X_train = X_train / 255 # 정규화

# X_test도 마찬가지로 1차원 배열로 변환 (한번에 고고)
X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255

# 클래스 값 확인
print("class : %d " % (Y_class_train[0]))

# 바이너리화 과정
Y_train = to_categorical(Y_class_train, 10)
Y_test = to_categorical(Y_class_test, 10)

print(Y_train[0])
```

1. 흰색배경 : 0 / 그외 : 1~255 로
채워져 긴 행렬로 이루어진 하나의
숫자 집합으로 변환

2. 28x28의 이차원 배열을 784개의
1차원 배열로 reshape()함수를
사용하여 변환
- reshape(총 샘플 수, 1차원 속성의 수)
- 정수형 -> 실수형
- 0~1사이로 정규화

3. 클래스 값 확인후 딥러닝의 분류
문제를 해결하기 위해 바이너리 형태로
바꿔줘야함 -> 원-핫 인코딩
- to_categorical(클래스, 클래스의 개수).

3

[illegible]

```
class : 5
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

2. 딥러닝 기본 프레임

```
# 모델 프레임 설정
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu')) #은닉층
model.add(Dense(10, activation='softmax')) # 출력층
```

- 입력값 784개
- 은닉층 512개 / relu
- 출력층 10개 / softmax

```
# 모델 실행 환경 설정
```

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

- 오차함수 categorical_crossentropy
- 최적화함수 adam

```
# 모델 최적화 설정
```

```
MODEL_DIR = './model/'
```

```
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)
```

```
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)
```

- EarlyStopping()함수로 10회 이상
모델의 성과향상이 없으면 자동
중단하도록 설정

```
# 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])
```

- 샘플 200개를 총 30번 실행하도록
설정

```
# 테스트 정확도 출력
```

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

- 테스트셋의 최종 모델의 성과
측정하여 출력

2. 딥러닝 기본 프레임

```
# 테스트 셋의 오차
y_vloss = history.history['val_loss']

# 학습셋의 오차 (1 - 학습셋 정확도)
y_loss = history.history['loss']

# 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()
```

Epoch 13: val_loss improved from 0.06102 to 0.05823, saving model to ./model/13-0.0582.hdf5

Epoch 14: val_loss did not improve from 0.05823

Epoch 15: val_loss did not improve from 0.05823

Epoch 16: val_loss did not improve from 0.05823

Epoch 17: val_loss did not improve from 0.05823

Epoch 18: val_loss did not improve from 0.05823

Epoch 19: val_loss did not improve from 0.05823

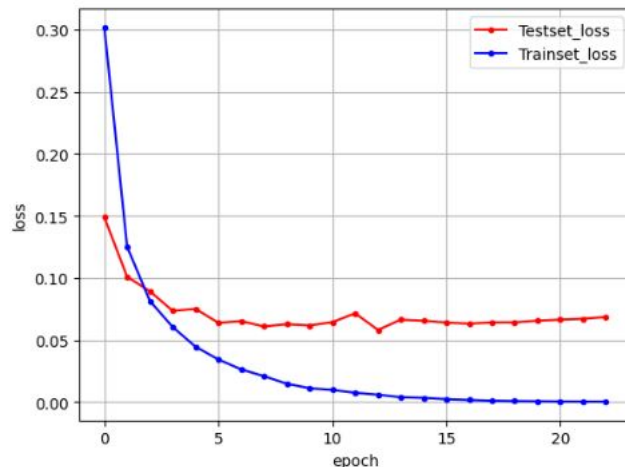
Epoch 20: val_loss did not improve from 0.05823

Epoch 21: val_loss did not improve from 0.05823

Epoch 22: val_loss did not improve from 0.05823

Epoch 23: val_loss did not improve from 0.05823
313/313 [=====] - 1s 3ms/step - loss: 0.0687 - accuracy: 0.9832

Test Accuracy: 0.9832



Convolution Neural Networks (합성곱 신경망)

[illegible]

인간의 뇌신경망적 특징도 따왔다.

1차원이라면 위치 정보는 어디...??

이미지/비디오 와 같은 고차원 데이터에 효율적

이미지/비디오 와 같은 고차원 데이터에 효율적

Convolution(합성곱) 이 뭐였지?

Convolution 연산 (*이산변수는 인테그랄이 아닌 시그마*)

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau$$

사실은.. Cross Correlation 연산이래요

공업수학(1-2)
신호및시스템(2-2)

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t + \tau) d\tau$$

<https://en.wikipedia.org/wiki/Cross-correlation>

<https://www.youtube.com/watch?v=O9-HN-yzsFQ>


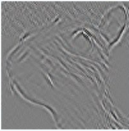
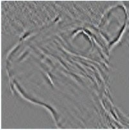

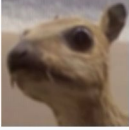
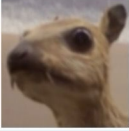
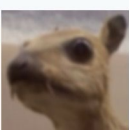
2D Convolution

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3×3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5×5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

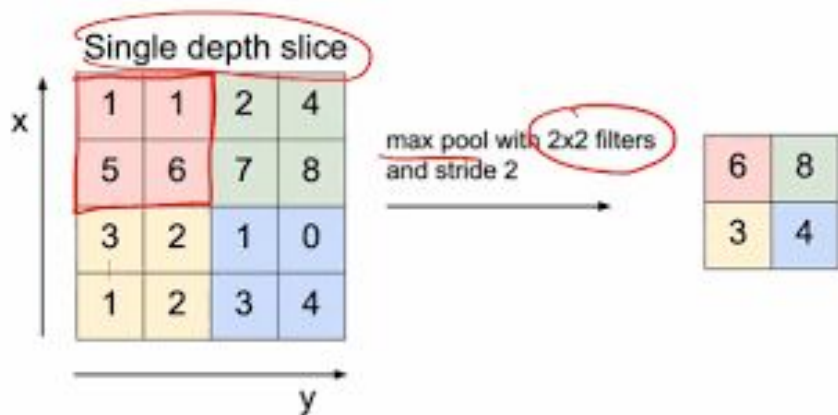
원본 * 커널 =
원본이미지들의 특징이 추출됨

커널의 종류에 따라 다양한 특징들이 추출됨

CNN 은 이 커널의 값들을 학습하는 것
(가중치들이 이 커널, 커널을 학습->특징을 학습)

[https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

Stride

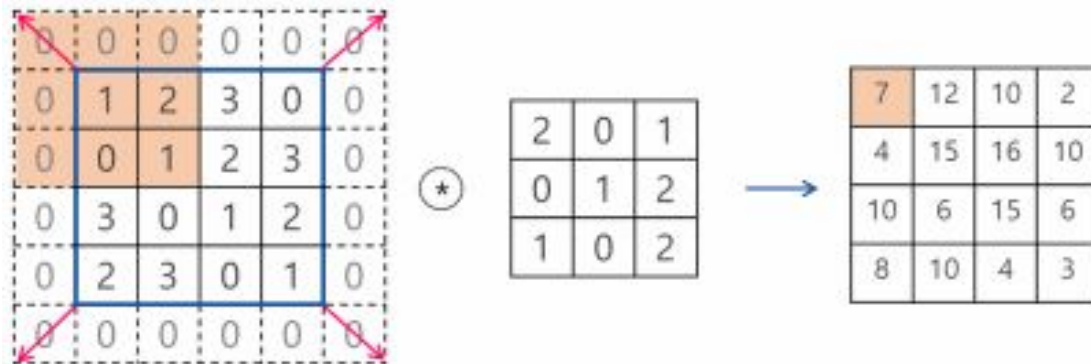
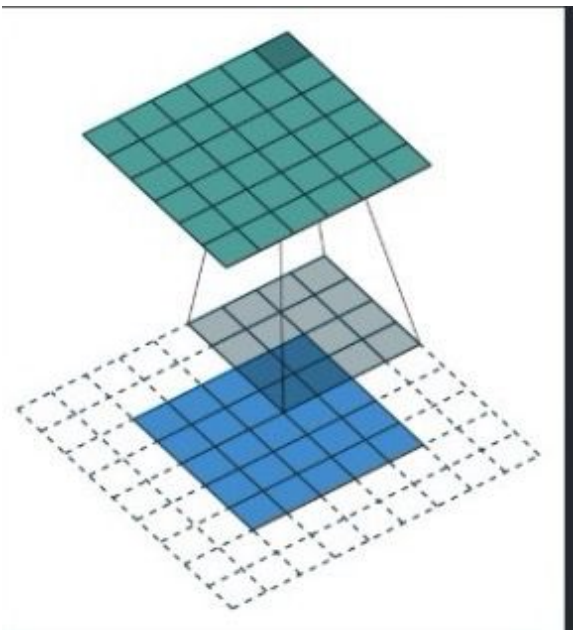


필터를 한칸씩 움직이지
않고
N칸씩 움직여보자

stride N

띄엄띄엄봐도 정보가
나오니까.. + 연산량 감소

Padding



0으로 채우면 zero padding

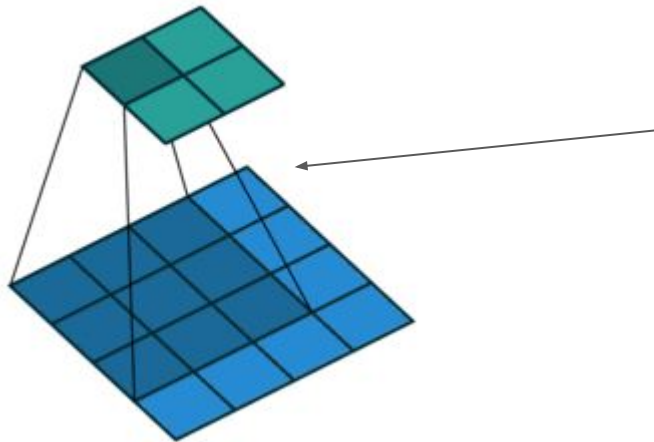
합성곱 연산시 차원 감소 \rightarrow 정보량 손실

특히 모서리 부분의 픽셀은 한번만 사용되어 윤곽 정보 손실

합성곱 연산 처리 중의 차원 축소를 방지하고

특히 윤곽자리의 이미지 정보를 보존하기 위해서 경계를 덧대는
전처리 방식.

차원 계산

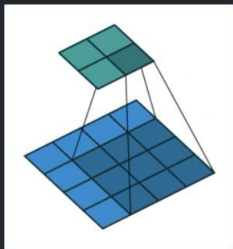


$$(OH, OW) = \left(\frac{H + 2P - FH}{S} + 1, \frac{W + 2P - FW}{S} + 1 \right)$$

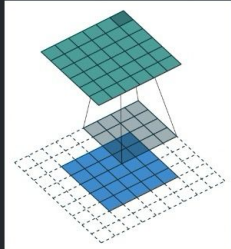
- (H, W) : 입력크기
- (FH, FW) : 필터크기
- (OH, OW) : 출력크기
- P : 패딩
- S : 스트라이드

Convolution animations

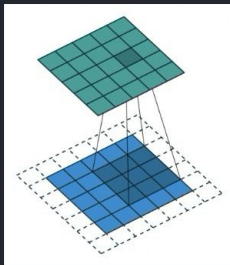
N.B.: Blue maps are inputs, and cyan maps are outputs.



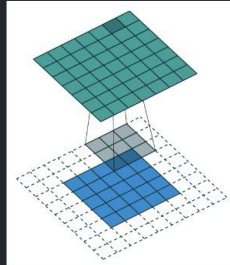
No padding, no strides



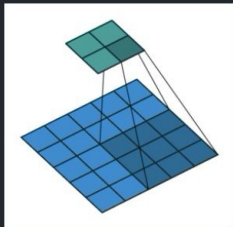
Arbitrary padding, no strides



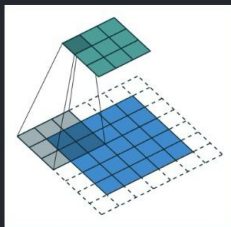
Half padding, no strides



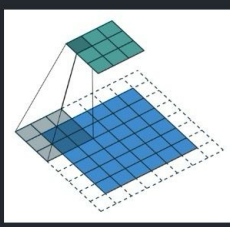
Full padding, no strides



No padding, strides



Padding, strides



Padding, strides (odd)

파이썬과 케라스로 딥러닝 인공지능을 배우고 있어.

이때 CNN 에 대한 개념을 설명하려고 하고, 합성곱 연산에 대하여 예시를 들어 설명하려고 해

이미지의 외곽선을 따주는 커널을 이미지 배열 행렬에 합성곱 연산을 하여

인풋 이미지와 아웃풋 이미지를 시각화해서 띄워 비교해줄 수 있는 예시 코드를 파이썬으로 작성해줘

이때 라이브러리에 내장되어있는 MNIST 데이터를 사용해주고, 합성곱 연산의 원리를 이해할 수 있도록

라이브러리 내장 함수를 바로 이용하는 것이 아닌, 어느정도 numpy 등을 이용해서 풀어서 코드를 작성해줘

5

ChatGPT

좋습니다. 파이썬과 케라스를 이용하여 딥러닝, 특히 CNN(합성곱 신경망)에 대해 배우고 계시다니 매우 흥미로운 학습 주제입니다. CNN의 핵심 개념 중 하나인 합성곱 연산에 대해 설명하고, 이를 통해 이미지의 외곽선을 감지하는 예시를 만들어보겠습니다.

우선, MNIST 데이터셋을 불러오고, NumPy를 사용하여 합성곱 연산을 직접 구현해보겠습니다. 이 과정에서 이미지의 외곽선을 감지하기 위한 커널을 정의하고, 이 커널을 사용하여 인풋 이미지에 합성곱 연산을 수행한 후, 결과를 시각화하여 인풋 이미지와 비교해 보겠습니다.

합성곱 연산을 구현하기 위해 다음과 같은 단계를 거칩니다:

1. MNIST 데이터셋 불러오기
2. 외곽선 감지를 위한 커널 정의
3. 합성곱 연산 수행
4. 원본 이미지와 처리된 이미지 비교 시각화

GPT 와 함께하는 Conv2D 구현해보기

<https://colab.research.google.com/drive/1d9dwpbflKmJCB8HG7OBFmod2ufhOs9ew?usp=sharing>

<https://chat.openai.com/share/a39eeca8-583e-4035-9105-4f404982c574>

3. 케라스에서의 CNN 구조

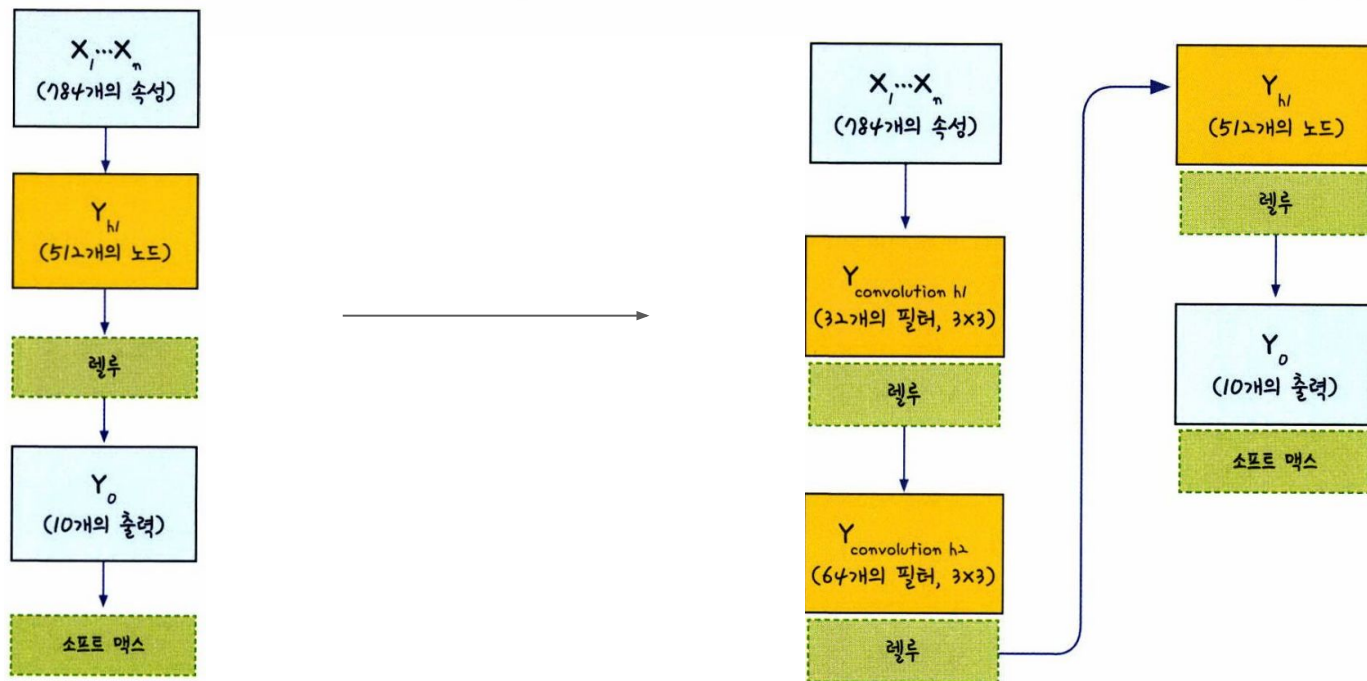
Conv2D : 케라스에서 컨볼루션 층을 추가하는 함수

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

1. 첫 번째 인자 : 커널을 몇 개 적용할지 정함
2. `kernel_size` : 커널의 크기를 정함 `kernel_size = (행, 열)`
3. `input_shape` : 맨 처음층에는 입력되는 값을 알려주어야함
`input_shape = (행, 열, 색상 또는 흑백)`,
입력 이미지가 색상이면 3, 흑백이면 1
4. `activation` : 활성화 함수 정의

3. 케라스에서의 CNN 구조

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



맥스 풀링

풀링 or 서브 심플링 : 컨볼루션 층을 통해 이미지 특징을 도출하였을 때 여전히 크고 복잡하여 이를 다시한번 축소하는 과정

맥스 풀링 : 정해진 구역 안에서
최댓값을 뽑아내는 풀링

평균 풀링 : 정해진 구역 안에서
평균값을 뽑아내는 풀링

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

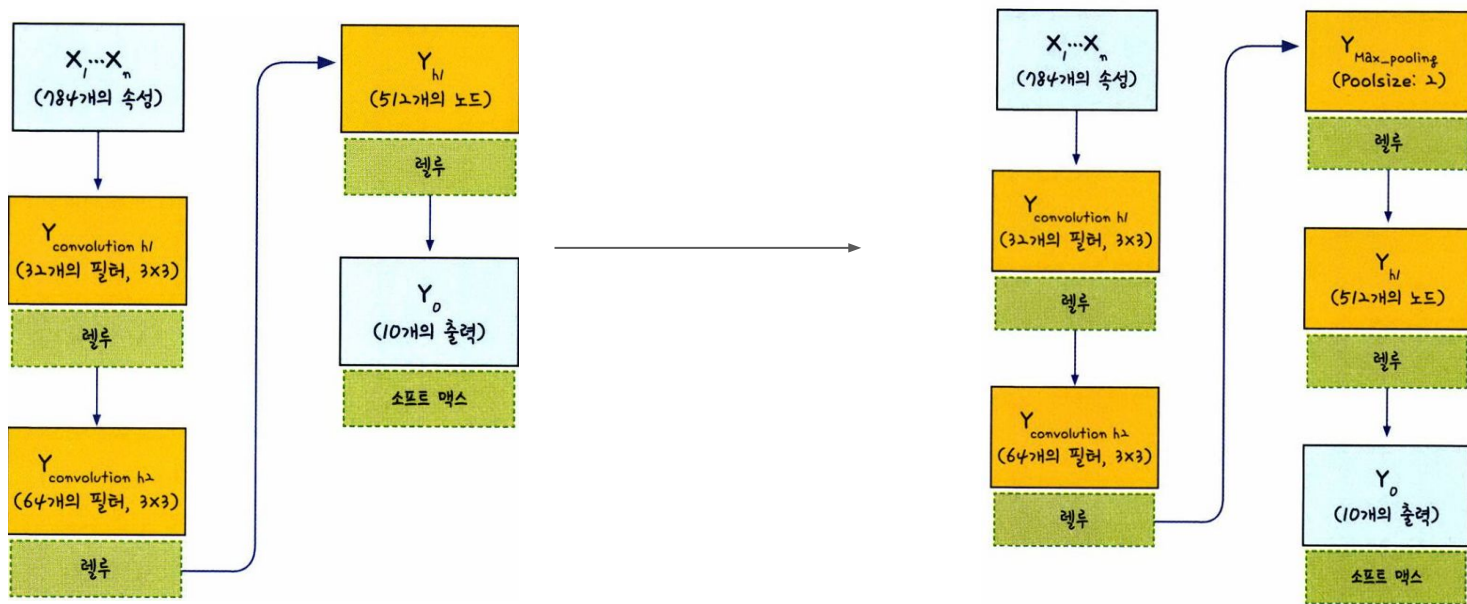
1	0	1	0
0	4	2	0
0	1	6	1
0	0	1	0

4	2
1	6

맥스 풀링

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

pool_size : 풀링 창을 크기를 정하는 것 / 2로 정하면 전체 크기가 절반으로 줄어듬

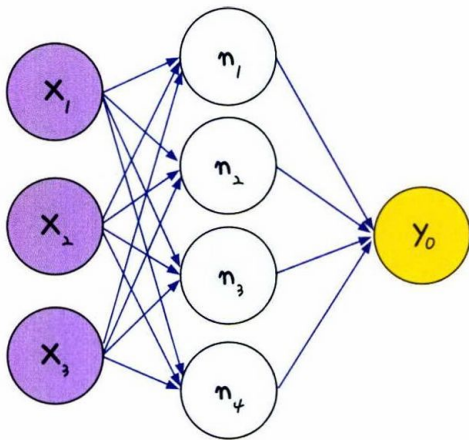


드롭아웃

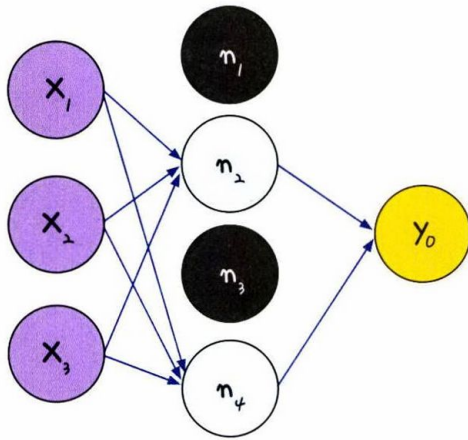
드롭아웃 : 은닉층에 배치된 노드 중 일부를 꺼주는 것

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

25%의 노드를 끄는 코드



드롭아웃



플래튼

플래튼 : 2차원 배열을 1차원 배열로 바꾸는것

이유 : 활성화 함수가 있는 층에서 사용하기 위해

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

