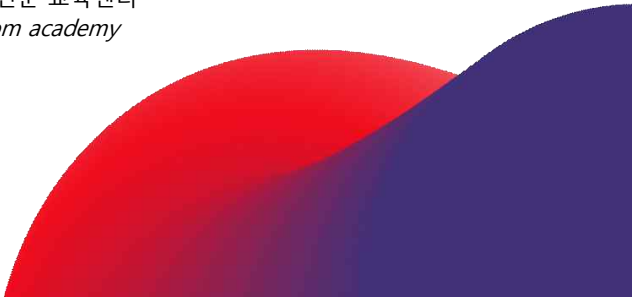


# 머신러닝 및 딥러닝 이해

"IT 융합 전문 교육센터"  
*Hancom academy*



## Copyright Note

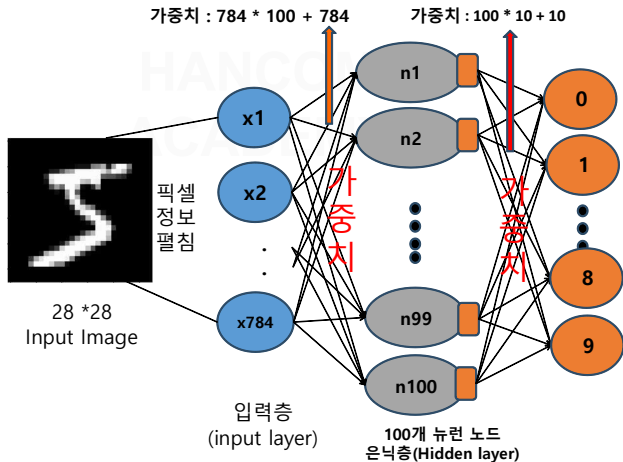
- Issue Date: 2021.03.01
- Homepage: [www.mdsacademy.co.kr](http://www.mdsacademy.co.kr)
- email: [edu@hancom.com](mailto:edu@hancom.com)
- author: shin seungcheol
- Copyright© 2021 *Hancom academy*, Co., Ltd. All rights reserved.
- 본 저작물의 저작권은 Hancom academy 및 저작권자에게 있으므로 저작권자의 사전 서면 허가 없이 무단으로 전재 혹은 복제를 금합니다.

## \* CNN( Convolution Neural Network )

**합성곱 신경망** : 합성곱층(convolution layer)을 하나 이상 사용한 신경망

- 이미지를 분석을 위한 패턴(특징)을 찾는데 유용한 알고리즘으로 **이미지 특징을 자동으로 학습하고 이미지를 분류하는 대표적인 딥러닝 모델**
- 기존 이미지인식에 사용된 FCNN ( Fully Connected Neural Network ) 한계를 개선하기 위해 개발 되었으며 이미지 인식과 분류에 탁월한 성능을 발휘
- CNN 메커니즘은 입력층과 가까운 층에서는 가장자리(edge),곡선,직선등 기초적인 저수준(low level)특징을 학습하고 점차 높은 후반부 층으로 갈수록 질감(texture), 물체 일부분(object parts)과 같은 고수준(high level)특징을 학습(인식)한다. 마지막으로 찾아낸 모든 특징을 출력층(Fully connected layer)의 입력값으로 사용해 물체(이미지)의 종류를 분류하는 추론 과정을 수행
- CNN 은 크게 세 가지 종류의 층으로 구성
  - **컨볼루션층(convolution layer)** : 이미지로부터 특징을 추출 ( Filter -> Relu(활성화함수) )
  - **풀링층(pooling layer)** : 이미지에서 대표 표본을 추출하는 방식으로 이미지 크기를 적절히 줄이면서 특정 Feature를 강조(강화)할 수 있는 장점
  - **FC 층(Fully Connected layer)** : 최종적인 이미지 분류 작업 진행

# \* FCNN ( Fully Connected Neural Network ) 문제점

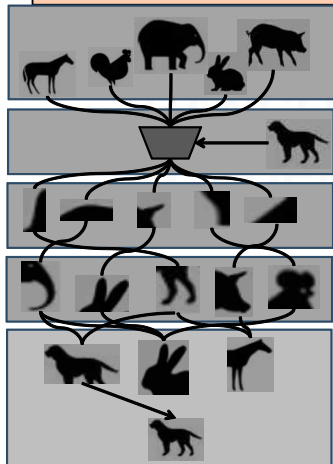


- FCNN은 input image 를 픽셀의 행인 1차원 배열로 직렬화 한 후, 입력 데이터로 사용하는 방식
- 1차원 배열로 펼치는(직렬화) 과정에서 데이터의 형상이 무시되어 형상에 담긴 정보(상관관계)를 잃게 되고 데이터 전체 관계를 고려하지 못하는 문제점을 갖음
- 엄청나게 많은 가중치 학습으로 학습 시간이 오래 걸림

↓ CNN 모델 설계로 해결

- Fully connected layer 앞부분에 합성곱층(Convolution layer)과 pooling layer를 추가하여 데이터의 형상을 유지하도록 함, 결과적 이미지 분류의 성능이 향상

## \* CNN 예측 메커니즘



90%확률로  
개 예측

### 1. 학습(training)

동물 라벨이 붙여진 수 많은 이미지 데이터를 가지고 **분류 방법을 모델이 학습**

### 2. 새로운 이미지 데이터 입력

학습된 신경망 모델에 라벨이 없는 **새로운 이미지 데이터 주입**

### 3. 전반부 레이어(층)

가장자리(edge), 직선, 곡선 등 단순하고 **기초적인 특징에 반응**

### 4. 후반부 레이어(층)

물체 일부분 등 좀 더 **고차원적인 특징에 반응**

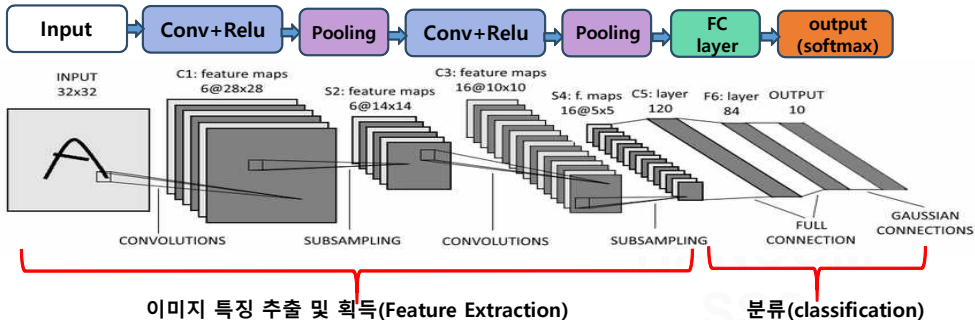
### 5. 출력층

동물을 구별할 수 있는 **추상화 개념에 반응**

### 6. 최종 예측

학습된 신경망에 근거해 이미지 속 **동물을 예측**

# \* CNN 모델 블록 구조

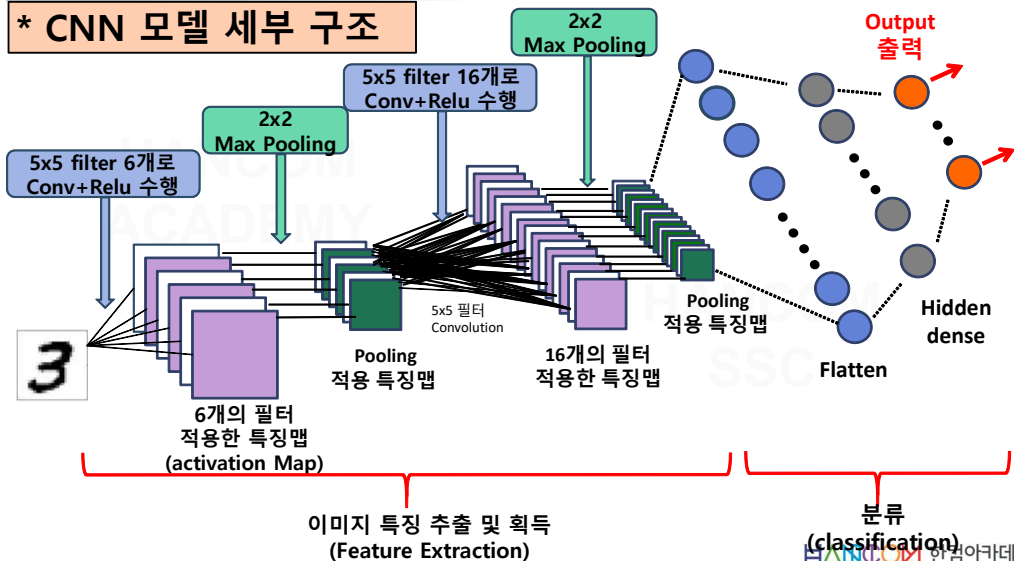


이미지 특징 추출 및 획득 (Feature Extraction)

분류 (classification)

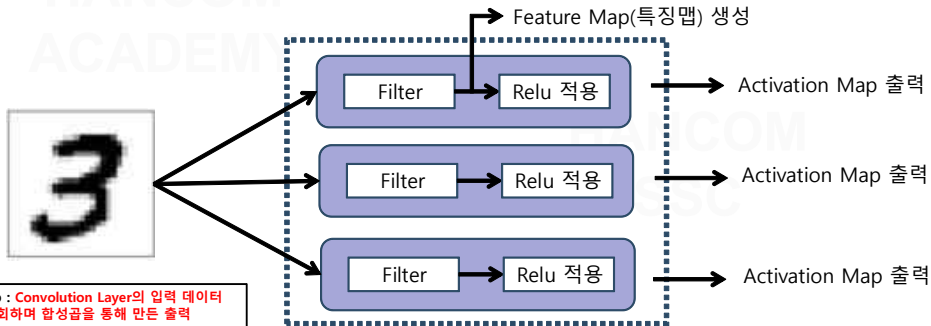
- Input --> C1 --> S2 --> C3 --> S4 --> C5 --> F6 --> output 순으로 진행
- C1 layer : 입력이미지(32x32)를 6개의 (5x5) 필터와 convolution 연산, 결과로 6장의 28x28 특징맵 생성
- S2 layer : pooling 작업, 2x2 필터를 2 stride 로 설정해 특징맵이 절반 크기로 축소, 6장의 14x14 특징맵 생성
- C3 layer : 6장의 14x14 특징맵에 convolution 연산을 수행, 16장의 10x10 특징맵 생성
- S4 layer : pooling 작업, 16장의 5x5 특징맵으로 축소
- C5 layer : 16장의 5x5 특징맵을 5x5 필터 120개로 convolution 연산을 수행, 1x1 크기의 120개 특징맵 생성

# \* CNN 모델 세부 구조



## \* Convolution layer (합성곱 층) : 입력 데이터로부터 특징을 추출하는 역할을 수행

- 특징을 추출하는 **필터(Filter)** 와 **필터 연산 결과값**을 비선형 값으로 바꾸어주는 **활성화 함수(Activation Function)**으로 구성



\* Feature Map : Convolution Layer의 입력 데이터를 필터가 순회하며 합성곱을 통해 만든 출력

\* Activation Map : Feature Map 행렬에 활성화 함수를 적용한 결과

Convolution layer



## \* 필터(Filter) 정의

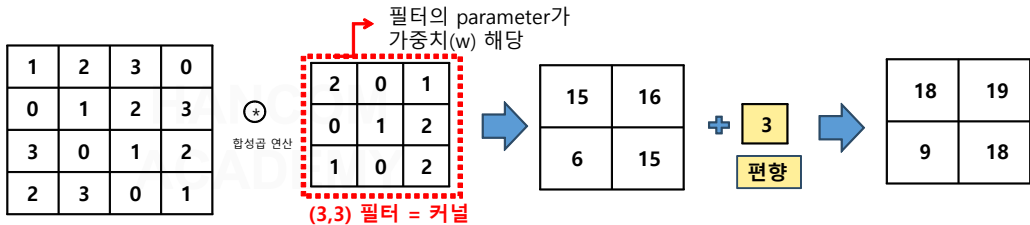
: 추출하려는 이미지의 특징이 대상 이미지 데이터에 있는지 없는지를 **검출해주는 역할**

- 입력 이미지는 행렬 형상을 가졌으며, 필터(Filter) 역시 행렬 차원으로 정의
- 어떠한 필터(Filter)를 사용하느냐에 따라 찾을 수 있는 이미지의 특징이 달라짐
- **필터의 가중치가 CNN 모델이 찾고자하는 목표 가중치**라고 생각하면 됨
- 합성곱 연산은 이미지의 특징을 추출하는 필터를 생성하고 이동해가며 입력 데이터에 적용해 합성곱 연산의 출력을 완성해 나감
- 필터 하나당 feature map이 형성되며, convolution layer는 생성된 feature map을 스택처럼 쌓음
- **결론적으로 학습 과정을 통해 점차 특징을 잘 찾는 필터가 생성되도록 하는게 CNN 딥러닝의 목표**

## \* 필터 크기 설정

: 사용자가 설정 해 주어야할 하이퍼파라미터로서 여러 가지 값을 시도해 봐야 겠지만, 보통 (3, 3) 이나 (5, 5) 크기를 권장

# \* 합성곱 연산 ( 필터 연산 )

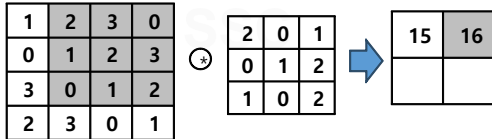
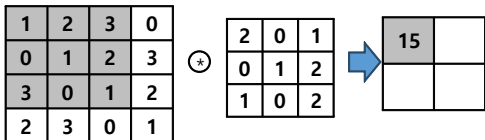


(4,4) 입력데이터

1)

\* 1 stride 필터 순회 예시 )

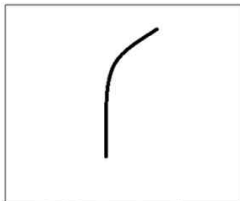
2)



단일곱셈-누산 :

$$1*2+2*0+3*1+0*0+1*1+2*2+3*1+0*0+1*2 = 15$$

## \* 필터 적용 예시)



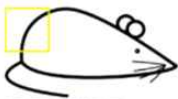
커브 검출 필터 준비



0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

필터 픽셀 표현(행렬 데이터)

\* 쥐 이미지에 곡선이 있는지 검출하기 위해 필터 연산 수행



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

\*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

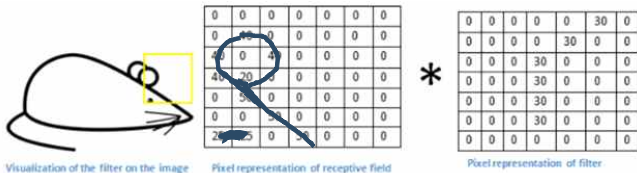
Pixel representation of filter

Filter



곡선이 있는 쥐 엉덩이 부분의 이미지와 필터 연산을 수행하면 **큰 값이 나옴**

## \* 필터 적용 예시)



곡선이 없는 쥐 머리 부분의 이미지와 필터 연산을 수행하면 결과값의 거의 0에 수렴

필터는 입력받은 데이터에서 그 특성을 가지고 있으면 결과값이 크게 나오고, 특성을 가지고 있지 않으면 결과 값이 0에 가까운 값이 나오게 만들어

이미지 데이터가 해당 특성을 가지고 있는지 없는지 여부를 알 수 있게 해주는 특징맵을 생성토록 해준다!!

## \* 패딩

: 합성곱 연산을 수행하기 전, **입력 데이터 주변을 특정값(보통 0)으로 채워 늘리는 것**

- 여러 합성곱 과정에서 가장자리 픽셀 정보등 활용 가능한 픽셀 정보가 유실되는 문제가 발생
- 패딩은 합성곱 진행 과정에서 정보가 소실되는 것을 방지하기 위해 **입력 이미지 행렬 가장자리에 0 값(제로 패딩)을 추가적으로 넣어 값을 키운 다음 필터를 적용하는 방법**

↖	0	0	0	0	↗
0	5	8	9	3	0
0	5	8	9	3	0
0	5	8	9	3	0
0	5	8	9	3	0
↙	0	0	0	0	↘

(4, 4) 입력

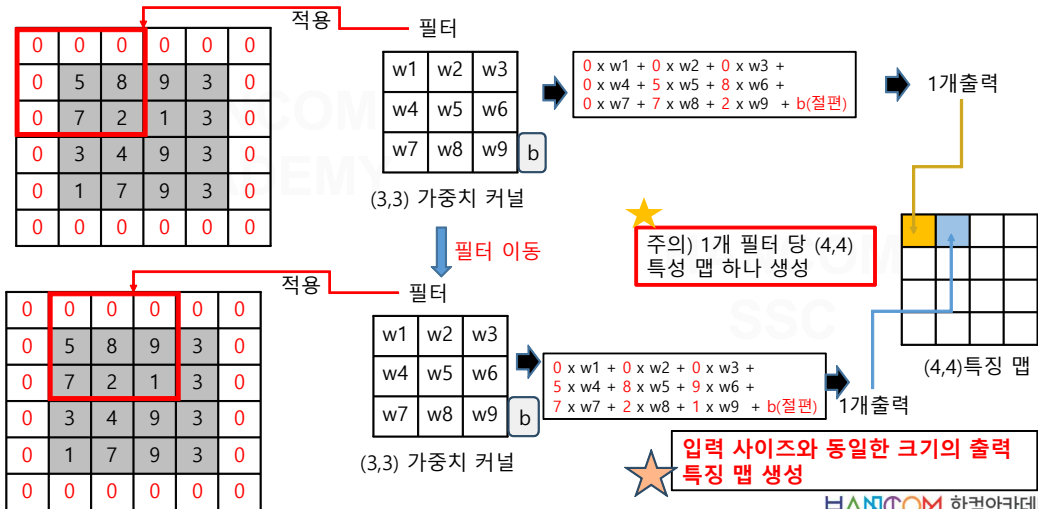
**패딩** : - 세임 패딩 ( same ) : 입력 주변 0으로 채움 ( **zero padding** )  
 - 밸리드 패딩 ( valid ) : 패딩 없이 순수한 입력 배열에서만 합성곱

- **케라스 Conv2D 클래스 경우** "valid" padding, "same" padding으로 지원, 기본값 valid 패딩, padding 매개변수에 'same' 지정하여 same 패딩 적용

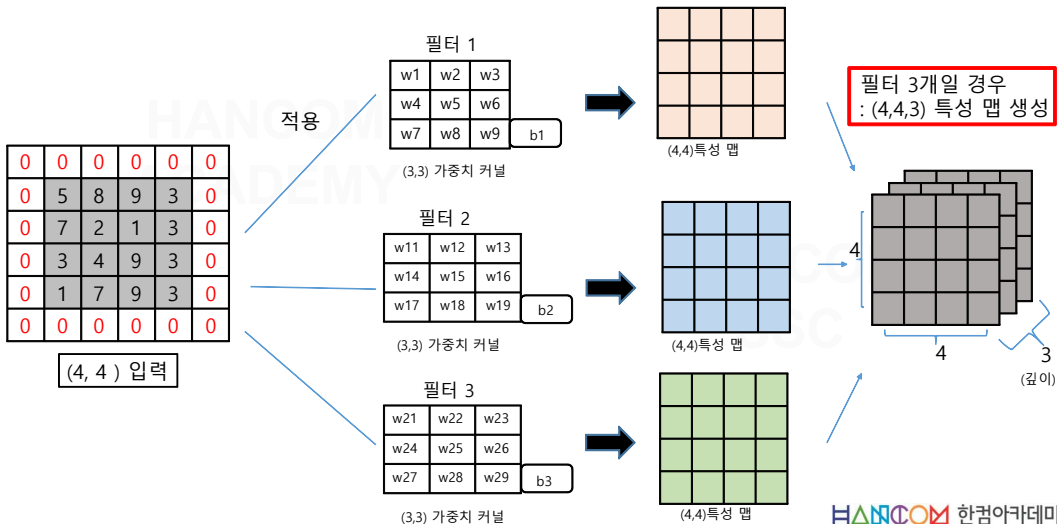
0값으로 1 픽셀 padding(zero padding) 과 1 stride 설정한 합성곱 연산 수행 결과는 **입력 행렬과 동일한 사이즈로 행렬로 출력이 유지**

- **합성곱 신경망에는 대부분 same 패딩 사용**  
 ( 입력 행렬과 출력 행렬의 형태(shape)가 동일하게 유지)

\* same 패딩 적용한 필터 연산 : ( 4 , 4 ) 입력 --> ( 4 , 4 ) 출력



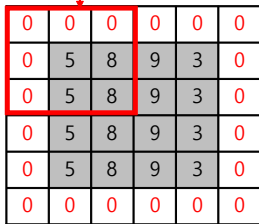
## \* 여러개 필터 커널을 적용한 합성곱 연산



## \* 스트라이드(stride)

- 입력 데이터에 필터를 적용할 때 이동할 간격을 조절하는 것
- **필터가 이동할 간격을 의미**
- zero padding 적용한 (6, 6) 원본 이미지 영상이 있을 때, (3, 3) 필터를 **좌측 상단에서 부터 왼쪽으로 한칸씩 그 다음 줄에서 왼쪽으로 한칸씩 적용해서 특징을 추출**
- 스트라이드(stride)는 보통 1과 같이 작은 값이 더 잘 작동, 디폴트 값 1
- 스트라이드 크기도 연산 결과 출력 크기에 영향을 줌

(3,3) 커널 필터

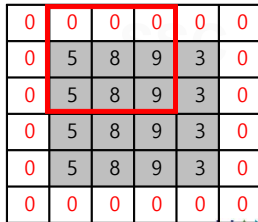


0	0	0	0	0	0
0	5	8	9	3	0
0	5	8	9	3	0
0	5	8	9	3	0
0	5	8	9	3	0
0	0	0	0	0	0

이동



이동할 간격 : 스트라이드(stride) 기본값 1  
strides 매개변수로 이동값 지정 가능  
대부분 기본값 활용



0	0	0	0	0	0
0	5	8	9	3	0
0	5	8	9	3	0
0	5	8	9	3	0
0	5	8	9	3	0
0	0	0	0	0	0



## \* 합성곱 연산-출력크기 계산

- 패딩과 스트라이드 적용, 입력데이터 와 필터의 크기가 지정되었을 때  
출력 데이터의 크기(shape)을 구하는 식

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

- ( H , W ) : 입력 크기
- ( FH, FW ) : 필터 크기
- P : 패딩
- S : 스트라이드
- ( OH, OW ) : 출력크기

- \* 예) **패딩 1, 스트라이드 1** 일 경우 출력 크기 계산

0	0	0	0	0	0
0	5	8	9	3	0
0	7	2	1	3	0
0	3	4	9	3	0
0	1	7	9	3	0
0	0	0	0	0	0

입력 ( 4 , 4 )

⊗

2	0	1
0	1	2
1	0	2

필터 ( 3 , 3 )



7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

( 4 , 4 ) 출력  
특성맵

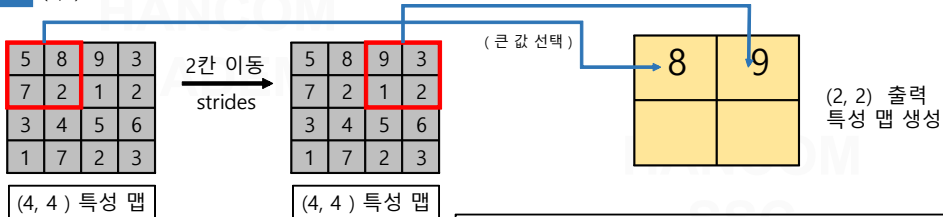
$$= \left( \frac{4 + 2 * 1 - 3}{1} + 1, \frac{4 + 2 * 1 - 3}{1} + 1 \right) = (4, 4)$$

## \* 풀링층 ( Pooling Layer )



(2,2) 풀링 윈도우 예

- **특징맵 크기를 줄여** 계산될 파라미터의 수를 줄임으로서 계산 속도 향상을 기대할 수 있으며 **특정 데이터를 강조하는 효과**
- 입력 데이터의 변화에 강하게(영향을 덜 받게) 만들어 주는 효과, 즉 입력 데이터의 약간의 변화를 풀링이 흡수해 사라지게 함



케라스 사용 예)

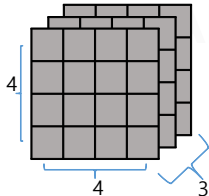
MaxPooling2D(pool\_size=(2,2))  
최대 풀링으로 풀링층 설계 시



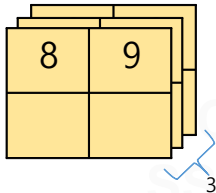
- 최대 풀링 (가장 큰 값 선택)
- **학습할 가중치 없음, 겹치지 않고 이동**
- **pool\_size = (2, 2) 또는 2 정수로 풀링 윈도우의 크기를 설정하면**  
**strides = None 인 경우 pool\_size 값으로 설정되어 (2,2) 풀링 크기시 2 스트라이드로 2칸씩 이동**

\* 평균풀링도 있으나 이미지 인식 분야에서는 주로 **최대 풀링 사용**

필터 3개일 경우  
: (4,4,3) 특성 맵 생성

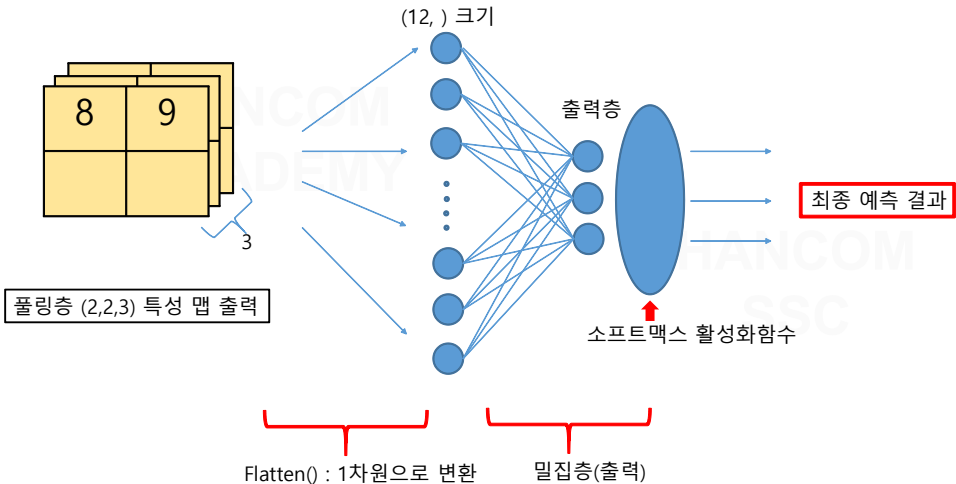


(2,2) 풀링 예

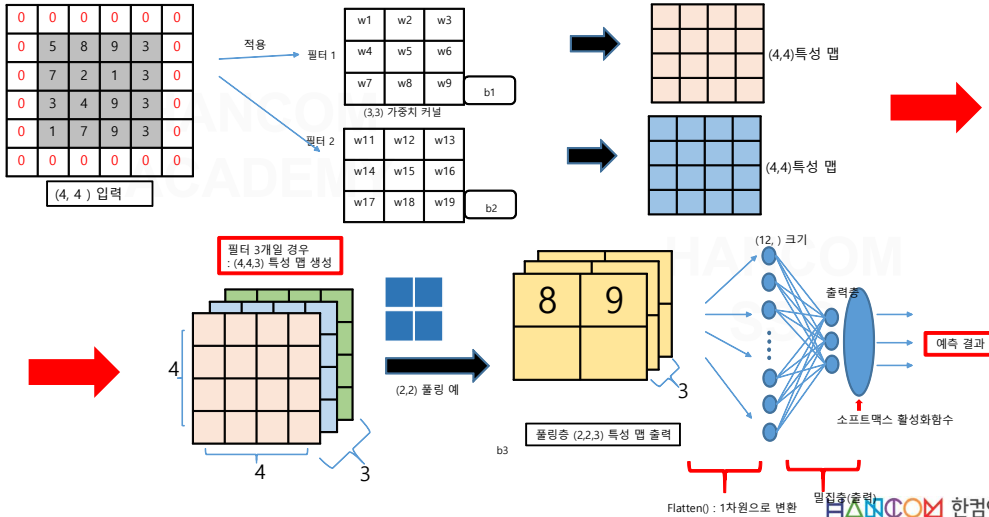


(2,2,3) 특성 맵 출력

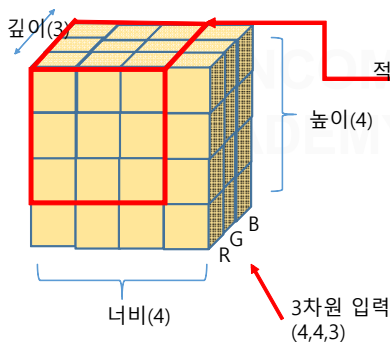
# \* Flatten / 밀집층(출력)



# 합성곱 신경망 구조 - 전체구조

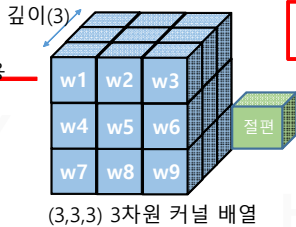


# 합성곱 - 컬러 이미지( 3차원 입력 )

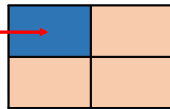


- 컬러 이미지는 RGB 채널로 구성  
3차원 배열로 표시
- 높이(rows), 너비(cols), 깊이(채널)

적용



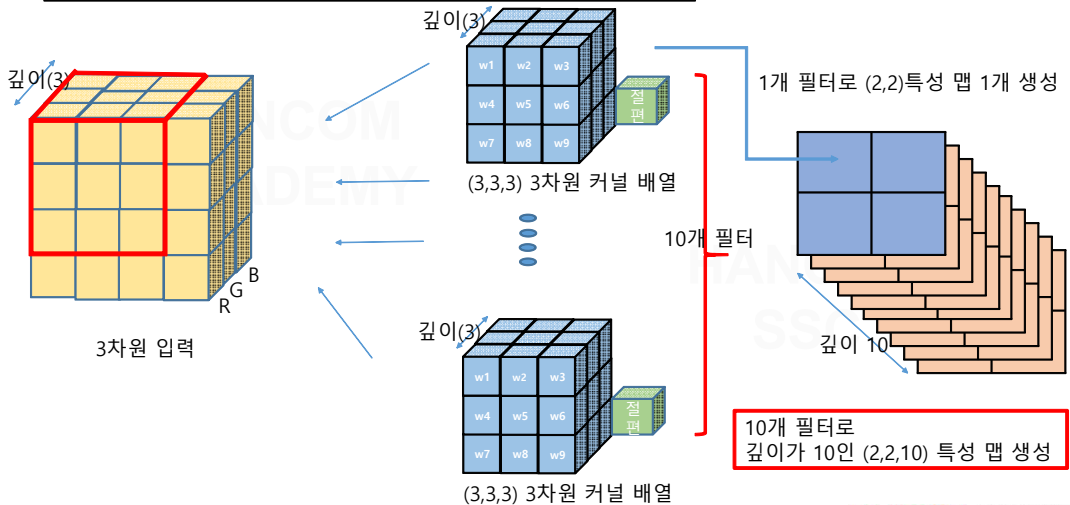
3차원 입력 시 3차원 커널 필요

커널 배열 깊이와  
입력 배열의 깊이는 동일 해야함

(2,2) 크기 특성 맵 출력

(3,3,3)영역에 해당하는 27개의 원소에 27개의  
가중치를 곱하고 1개의 절편을 더해 하나의 출력 생성

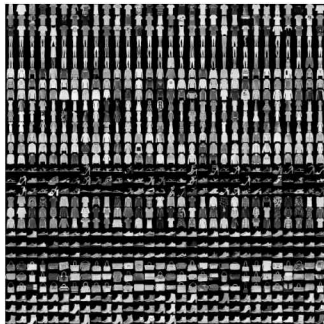
# 합성곱 - 컬러 이미지 ( 10개 필터 예 )



**\* CNN 모델 활용 1번째 예제  
( Fashion MNIST 활용한 이미지 분류)**



## \* 패션 MNIST 이미지 Dataset 분석



패션 MNIST 데이터셋

- tf.keras.datasets 모듈은 Neural Network 훈련에 사용할 수 있는 여러 데이터셋(imdb,mnist등)을 포함하고 있으며 **그중 하나인 fashion\_mnist 데이터셋을 load 해 사용**

- Fashion MNIST 데이터셋 :  
(28,28)픽셀 저해상도 이미지 70000개의 회색조(그레이스케일) 데이터 집합

- Fashion MNIST target label :  
패션 10개의 카테고리를 [ 0 ~ 9 ] 숫자로 구분



카테고리	티셔츠 /탑	바지	스웨터	드레스	코트	샌달	셔츠	스니커즈	가방	앵클부츠
Label	0	1	2	3	4	5	6	7	8	9


## 1) Keras 패션 MNIST 이미지 Dataset load

```
import numpy as np
from tensorflow.keras.datasets import fashion_mnist

# (28,28) train : 60000개 , test : 10000개 데이터 로드
(train_input, train_target) , (test_input, test_target) = fashion_mnist.load_data()

print(len(train_input), train_input.shape) ) # 60000 , (60000, 28, 28)
print(test_input[0].shape, len(test_input)) # (28,28) , 길이 10000

print('target label 체크 : ')
print( np.unique(train_target, return_counts=True) )
print(train_target[0]) # label 9
```



```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000, 6000], dtype=int64))
```

0 ~9 총 10개 카테고리 분류

```
print(train_input[0])
plt.imshow(train_input[0], cmap='gray') # label 9인 input 이미지 체크
plt.show()
```

(28, 28 ) 데이터

target label 9 인  
훈련데이터셋 이미지 출력

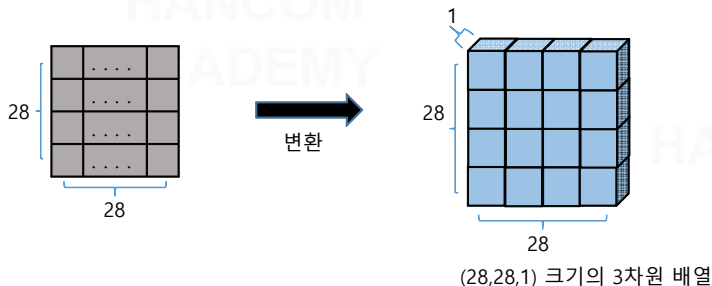
```
np.set_printoptions(linewidth=np.inf)
print(train_input[0])
```

```
[[ 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  0  0 13 73  0  1  4  0  0  0  1  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  3  0 36 136 127 62 54  0  0  0  1  3  4  0  0  3]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  6  0 102 204 176 134 144 123 23  0  0  0  0 12 10  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 155 236 207 178 107 156 161 109 64 23 77 130 72 15]
```

각 픽셀은 0~255 사이의 값을 가지며  
검은색은 0,  
흰색은 255,  
회색은 0~255 사이의 값

## 2) CNN 모델 입력을 위한 데이터 Shape 형태 변환 및 정규화

- 케라스 합성곱층 입력 - (패션 MNIST 이미지 3차원 변환 입력)
- 케라스 합성곱 층은 항상 3차원 입력 기대 ( **rows, cols, channels** )
- 예) 흑백 이미지(28,28) 2차원 배열을 (28,28,1) 3차원 배열로 변환해서 적용



```
train_scaled = train_input.reshape(-1,28,28,1) / 255.0 # 데이터셋 정규화
print(train_scaled.shape)
```

### 3) 훈련 데이터 셋 => 훈련 / 검증 데이터셋으로 분할


```
from sklearn.model_selection import train_test_split
```

```
# 훈련 세트와 검증 세트로 분할
```

```
train_scaled, val_scaled, train_target, val_target = train_test_split(train_scaled,  
train_target, test_size=0.2, random_state=42)
```

```
print(len(train_scaled)) # 48000
```

```
print(len(val_scaled)) # 12000
```



60000 개 샘플 데이터를 20% 비율로 훈련 / 검증 데이터 셋으로 분할

#### 4) 합성곱 CNN 모델 생성을 위한 keras 모듈 추가

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Dropout, Flatten  
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

- layer 층 설계 및 추가 - Sequential, Dense
- 과적합 방지하기 위해 Dropout 층 추가
- 합성곱 연산 - Conv2D
- 풀링층 - MaxPooling 사용

HANCOM  
SSC

## 5) 케라스 합성곱 신경망 - 합성곱층 추가

```
model = Sequential()
```

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

첫 사용 시  
입력 형태 지정

(28, 28, 1) 입력

(3,3,1) 커널 사이즈 32개 필터

이미지 48000 개

1

28

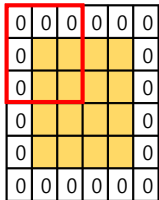
28

same 패딩 적용

( 48000, 28,28,1)

샘플수

입력 데이터 차원



깊이(1)



32개 필터

relu 함수 적용

28

28

(28, 28, 32) 특성맵

Conv2D

## keras.layers.Conv2d – 주요 매개변수(arguments)

filters	필터의 개수(특징 맵의 개수), 정수 형태로 입력
kernel_size	필터의 크기를 지정, 주로 2개의 정수가 들어간 튜플,리스트형태로 입력 받지만, 가로와 세로의 길이가 동일하다면 정수 하나만 입력
strides	스트라이드 값을 지정, 주로 2개의 정수가 들어간 튜플,리스트형태로 입력 받지만, 움직이는 거리가 같다면 정수 하나만 입력, 기본값 1
padding	'valid' 혹은 'same' 값을 입력. 기본값 'valid' , 'same'패딩 많이 사용
data_format	'channels_last' 또는 'channels_first' 값을 입력, 기본값 'channels_last' * channels : 차원의 깊이 , 예 (28, 28, 1)
activation	'relu' 와 같은 활성화 함수 입력
input_shape	이 계층 처음 사용시 input_shape 인자 값을 지정, 튜플 또는 리스트 형식으로 입력, 4차원 데이터 중 batches 값을 뺀 (rows, cols, channels 또는 channels, rows, cols) 형태로 입력

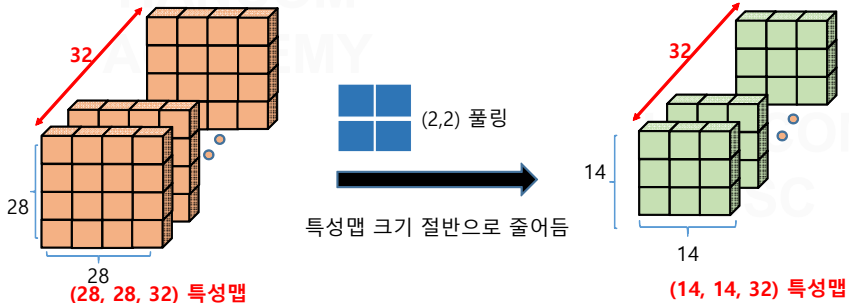
output shape : batch\_size + (new\_rows, new\_cols, filters) OR batch\_size + (filters, new\_rows, new\_cols)



## 6) 케라스 합성곱 신경망 - 풀링층 추가

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

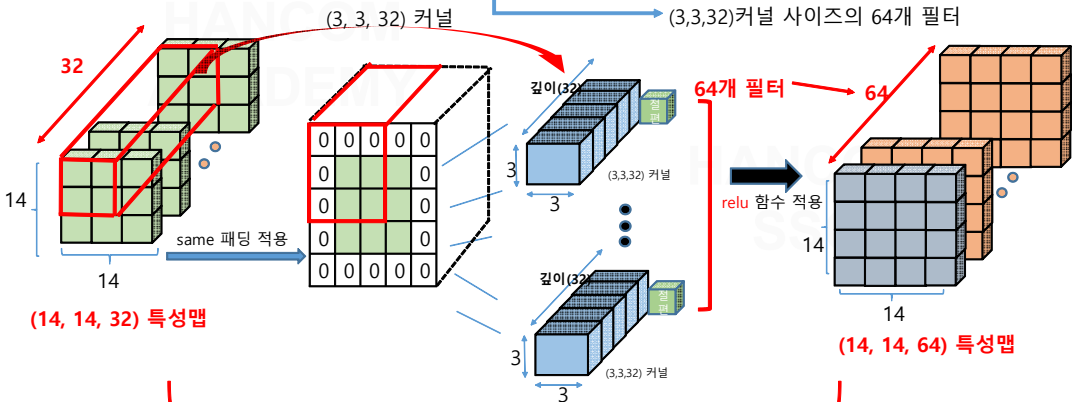
최대값 풀링, (2,2)풀링 크기



MaxPooling2D

## 7) 케라스 합성곱 신경망 - 2번째 합성곱층 추가

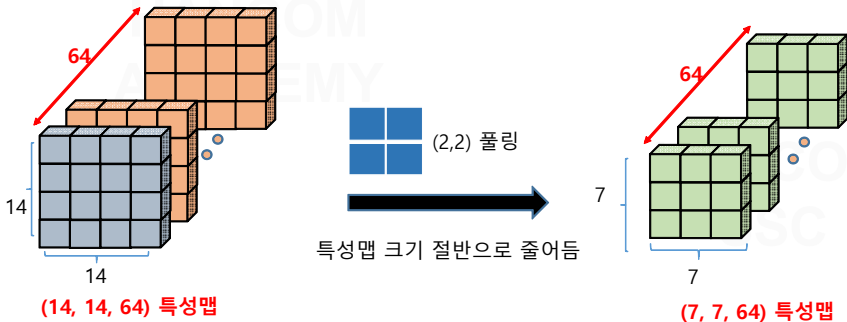
```
model.add( Conv2D(64, kernel_size=(3,3), activation = 'relu', padding='same') )
```



## 8) 케라스 합성곱 신경망 - 2번째 풀링층 추가

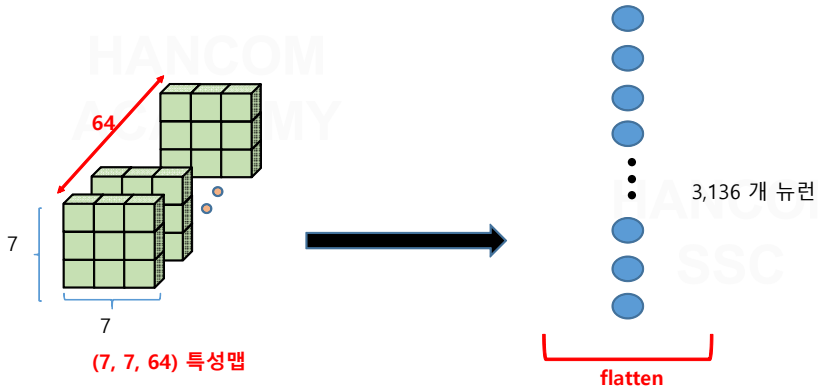
```
model.add( MaxPooling2D(2) )
```

최대값 풀링, (2,2)풀링 크기



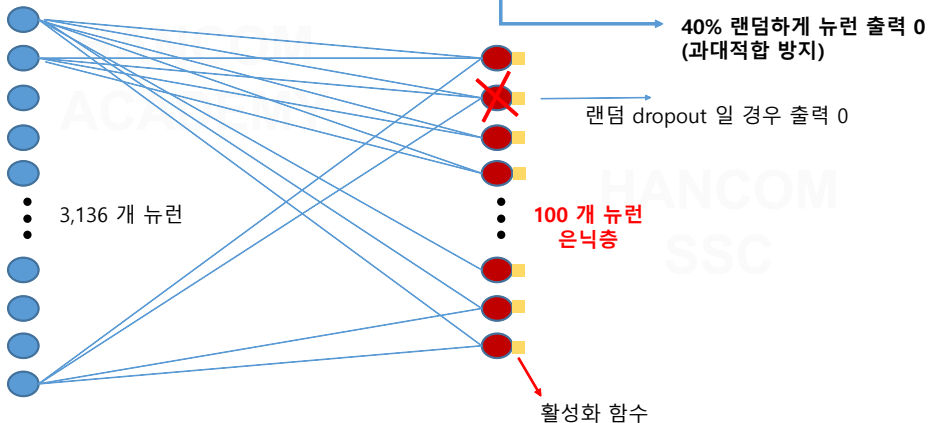
## 9) 케라스 합성곱 신경망 - 1차원 배열로 펼치기(flatten)

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



## 10) 케라스 합성곱 신경망 - 은닉층 , Dropout층 추가

```
model.add( Dense(100, activation='relu') )
model.add( Dropout(0.4) )
```



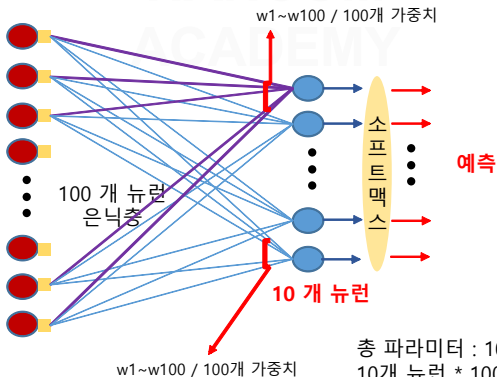
# 11) 합성곱 - 출력층 뉴런 개수 설정 및 다중 분류에 따른 '소프트맥스' 활성화 함수 사용

#출력층, 소프트맥스 활성화 함수

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

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 100)	313700
dropout (Dropout)	(None, 100)	0
<b>dense_1 (Dense)</b>	<b>(None, 10)</b>	<b>1010</b>
=====		



총 파라미터 : 1010

10개 뉴런 \* 100개 가중치 + 10개 절편

## 12) 케라스 합성곱 모델 컴파일

- 모델을 컴퓨터가 이해할 수 있도록 컴파일하고 **효과적으로 구현될 수 있도록 여러 환경을 설정**

\* **필수 설정 옵션(인수)**

- **optimizer** : 훈련 과정 **최적화**( 적응적 학습률 등 ) 방법 설정, 'adam' 또는 'sgd' 처럼 문자열로 지정 가능
- **loss** : 훈련 과정에서 사용할 **손실 함수 설정** ( loss function ), 타깃과 예측값의 오차 계산
- **metrics** : 훈련을 **모니터링** 하기 위한 **평가 지표** 선택 , metrics = ['accuracy']

### 손실함수 + 활성화함수 조합

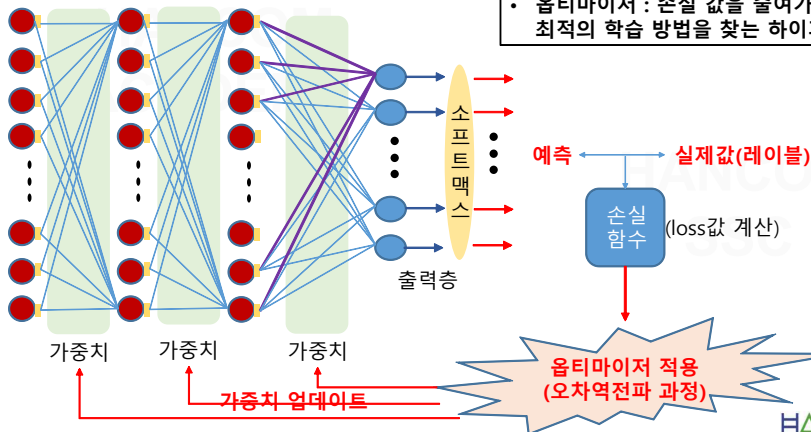
유형	손실 함수 명	출력층의 활성화 함수 명	
회귀 문제	mean_squared_error (평균 제곱 오차)	-	-
다중 클래스 분류	categorical_crossentropy (범주형 교차 엔트로피)	소프트맥스 (softmax)	원-핫 인코딩 상태 사용
다중 클래스 분류	<b>sparse_categorical_crossentropy</b>	소프트맥스 (softmax)	<b>원-핫 인코딩이 된 상태일 필요없이 정수 인코딩 된 상태에서 수행 가능</b>
이진 분류	binary_crossentropy (이진 교차 엔트로피)	시그모이드 (sigmoid)	1, 0 또는 긍정, 부정 등 2진 분류에 사용, 예) 와인 분류

\* 이 외 다양한 함수는 케라스 공식 문서 참조

## 12) 케라스 합성곱 모델 컴파일

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

- 옵티마이저 : 손실 값을 줄여가는(오차역전파) 최적의 학습 방법을 찾는 하이퍼파라미터



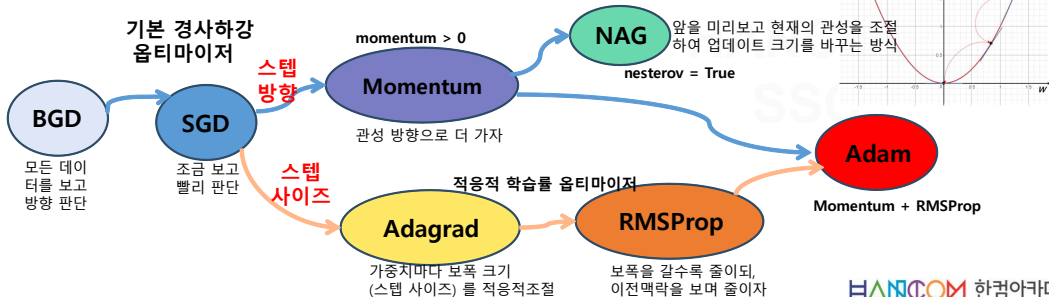


## \* 케라스 모델 컴파일 파라미터 (옵티마이저)

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

→ 모멘텀(관성) 최적화 + RMSprop(보폭) 융합으로 적응적 학습률 최적화

- 손실값의 최저점을 찾는 **경사 하강법에 적용하여 최적의 가중치와 학습률을 찾는 하이퍼파라미터**
- 가중치를 확률적 경사하강법(SGD) 을 사용해 빠르게 업데이트 할지 이동방향(Momentum)에 관성을 적용할지, 학습률(보폭) 크기는 어떻게 할지 결정하는 역할



## \* 케라스 모델 컴파일 파라미터 ( 손실함수, 평가지표 )

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

매 epoch 마다 손실과 함께  
정확도 지표 출력

### • loss ( 손실 함수 )

- 회귀 : 평균 제곱 오차(MSE : Mean Squared Error)
- 분류 : 크로스 엔트로피, 이진 분류 경우 binary\_crossentropy  
다중 분류 경우 categorical\_crossentropy 또는 **sparse\_categorical\_crossentropy** 활용



딥러닝 학습 목표 :  
손실 값을 최소화하는(오차역전파)  
최적의 가중치 W와 편향 b를 찾는게 목표임  
으로 손실 함수 선정은 중요

## 13) 케라스 모델\_콜백 ( ModelCheckpoint, EarlyStopping ) 적용

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping # 모델저장, 조기종료
```

- 콜백(callback) : **훈련 과정 중 어떤 작업을 수행할 수 있게하는 객체**
- keras.callbacks 패키지 아래 있는 클래스 , fit()메서드 callbacks 매개변수에 리스트 형태로 전달

```
checkpoint_cb = ModelCheckpoint('best-cnn-model.h5')
```

```
early_stopping_cb = EarlyStopping(patience=2, restore_best_weights=True)
```

최적 검증 점수를 만드는 모델  
자동 저장

10 epoch 이후 loss가 다시 증가하는 과대적합이  
발생하는데 20 epochs 설정으로 훈련이 진행됨.  
따라서, 필요하지 않는 훈련을 하지 않도록  
**조기 종료**가 필요

## 13) 케라스 모델 콜백 - 조기종료

from tensorflow.keras.callbacks import ModelCheckpoint, **EarlyStopping** # 조기종료 콜백

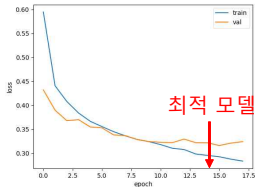
- 조기종료 : 과대적합(손실이 다시 증가되는 시점)을 막기 위해 훈련 미리 중지(에포크 제한)
- 케라스 EarlyStopping 콜백 사용
- patience** 매개변수 : 정확도가 더 이상 향상되지 않더라도 **더 진행될 에포크 회수**
- restore\_best\_weights** 매개변수 : True 지정 시 **가장 낮은 손실을 낸 모델 가중치로 복구**
- ModelCheckpoint 와 **EarlyStopping** 콜백 함께 사용 시 최적 모델 파일로 저장하고 손실이 다시 증가될 때 미리 훈련 중지

\* 최종 콜백 적용한 모델 훈련 코드

checkpoint\_cb = **ModelCheckpoint**('best-cnn-model.h5')  
early\_stopping\_cb = **EarlyStopping**(patience=2, restore\_best\_weights=True)

history = model.fit(train\_scaled, train\_target, epochs=20,  
validation\_data=(val\_scaled, val\_target), **callbacks=[checkpoint\_cb,  
early\_stopping\_cb]**, verbose=1)

'best-cnn-model.h5' 로 최적 모델 저장



## 14) 케라스 합성곱 모델 훈련 - 훈련 과정 모니터링

```
history = model.fit( train_scaled, train_target, epochs=20, validation_data=(val_scaled, val_target)
                    , verbose=0 )
```

\* 케라스 기본 미니배치 경사 하강법 , 기본 batch\_size = 32

1. verbose = 0 옵션 : 훈련과정 출력 되지 않음

2. verbose = 1 옵션 : 훈련과정 진행막대와 손실등의 지표 출력

```
Epoch 1/5
 32/48000 [.....] - ETA: 2:03 - loss: 2.3707 - accuracy: 0.0625
 928/48000 [.....] - ETA: 6s - loss: 1.4286 - accuracy: 0.5399
```

```
47936/48000 [=====>.] - ETA: 0s - loss: 0.5240 - accuracy: 0.8193
48000/48000 [=====] - 3s 59us/step - loss: 0.5239 - accuracy: 0.8193 - val_loss: 0.4190 - val_accuracy: 0.8543
```

매 epoch 마다 훈련데이터 손실, 정확도  
계산 출력

validation\_data 옵션으로 검증데이터 훈련  
시 검증데이터 손실, 정확도 계산 출력

3. verbose = 2 옵션 : 훈련과정 진행막대와 출력을 제외한 매 epoch의 손실, 정확도만 출력

```
Epoch 4/5
 - 3s - loss: 0.3259 - accuracy: 0.8810 - val_loss: 0.3660 - val_accuracy: 0.8679
Epoch 5/5
 - 3s - loss: 0.3052 - accuracy: 0.8884 - val_loss: 0.3296 - val_accuracy: 0.8804
```

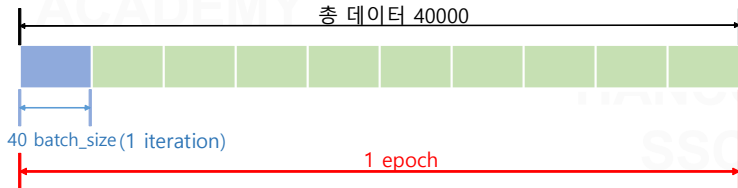
## \* 케라스 모델 fit() 메서드 파라미터

batch\_size(배치 사이즈) : 몇 개의 샘플로 가중치를 갱신할 것인지 지정

batch\_size = 1 일 경우 1데이터 훈련 할 때 마다 가중치 갱신(SGD : 확률적경사하강)

batch\_size = 32 일 경우 32개 데이터 훈련 할 때 마다 가중치 갱신(미니배치경사하강)

epochs(에포크) : 전체 데이터에 대해 순전파와 역전파가 끝난 상태, 즉 모든 데이터를 한번 훈련 끝낸 상태



총 데이터 : 40000 , batch\_size = 40, epochs = 20 일 경우 가정하면  
 1 epoch =>  $40000 / 40$  => 1000번 iteration 학습(가중치 갱신)  
 20 epoch 동안 =>  $20 * 1000$ 번 => 20000번의 학습

## \* 케라스 fit() 메서드 반환값 체크 및 활용

```
history = model.fit( train_scaled, train_target, epochs=20, validation_data=(val_scaled, val_target)
                    , verbose=0 )
```

### \* fit()메서드 반환

훈련결과 반환(측정)값 history객체의 history dict에 저장

```
print(history.history.keys())
```

dict\_keys(['val\_loss', 'val\_accuracy', 'loss', 'accuracy'])

검증데이터 손실

검증데이터 정확도

훈련데이터 손실

훈련데이터 정확도

```
print(history.history['loss'])
print(history.history['accuracy'])
print(history.history['val_loss'])
print(history.history['val_accuracy'])
```

기본적으로 손실을 계산하지만  
metrics 매개변수에 'accuracy' 추가로  
매 epoch 마다 손실과 정확도 계산 저장

```
[0.5287141543875138, 0.39304731270174187, 0.3512934491535028, 0.3238498495121797, 0.30389539121836423]
[0.81535417, 0.8589583, 0.8707917, 0.88189584, 0.88708335]
[0.4673801041841507, 0.38554213058948517, 0.34939442535241444, 0.3470803085764249, 0.33978807347019513]
[0.8370000123977661, 0.8615833520889282, 0.8722500205039978, 0.8711666464805603, 0.8775833249092102]
```

## \* 케라스 훈련 모델 저장 및 복원

```
history = model.fit( train_scaled, train_target, epochs=20, validation_data=(val_scaled, val_target)
                    , verbose=0 )
```

\* 과대/과소적합 해결 ( optimizer, Dropout, epochs 등 조절 ) 후 최적 가중치, 옵티마이저등의 모델 저장

save\_weights : 모델의 가중치 저장

save : 모델의 가중치, 옵티마이저 등, 전체 구조 저장

```
model.save_weights('SimpleNN_model_weights.h5')
model.save('SimpleNN_model.h5')
```

↓  
HDF5 포맷으로 저장

load\_weights : 모델의 가중치 불러오기  
단, save\_weights()메서드로 저장했던 모델과  
정확히 동일 모델 구조일때 가능

```
model.load_weights('SimpleNN_model_weights.h5')
```

또한, load\_weights()메서드로 가중치만 load 한 경우  
모델 compile 한 후 사용해야 함

```
original_keras_version = f.attrs['keras_version'].decode('utf8')
AttributeError: 'str' object has no attribute 'decode'
```

→ h5py : version 2.10.0 으로 낮춰서 재 설치



## \* 케라스 훈련 모델 저장 및 복원

### load\_weights()메서드

```
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

(train_input, train_target) , (test_input, test_target) = fashion_mnist.load_data()

train_scaled = train_input / 255.0

# 훈련 세트와 검증 세트로 분할
train_scaled, val_scaled, train_target , val_target = train_test_split(train_scaled,
train_target, test_size=0.2, random_state=42)

def model_fn(a_layer = None):
    model = Sequential()
    model.add( Flatten(input_shape=(28,28)) )
    model.add( Dense(100, activation='relu'))
    if a_layer:
        model.add(a_layer)
    model.add( Dense(10, activation='softmax') )
    return model

model = model_fn(Dropout(0.4))
model.summary()

model.load_weights('SimpleNN_model_weights.h5')
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

print("acc : %.4f" %model.evaluate(val_scaled,val_target)[1] )
```

### load\_model()메서드 : 모델 가중치, 옵티마이저, 구조 모두 복원

```
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten,
Conv2D, MaxPool2D
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

(train_input, train_target) , (test_input, test_target) =
fashion_mnist.load_data()

train_scaled = train_input / 255.0

# 훈련 세트와 검증 세트로 분할
train_scaled, val_scaled, train_target , val_target =
train_test_split(train_scaled, train_target, test_size=0.2,
random_state=42)

from keras.models import load_model
model = load_model('SimpleNN_model.h5')

print("acc : %.4f" %model.evaluate(val_scaled,val_target)[1] )
```

## 15) 케라스 합성곱 저장 모델 로드 후 모델 예측 / 성능 평가

predict() : 모델을 이용한 특정 데이터 예측 메서드 , evaluate() : 모델 평가 메서드

```
from tensorflow.keras.models import load_model
CNN_NewModel = load_model('best-cnn-model.h5')
#예측
preds = CNN_NewModel.predict(val_scaled) # 검증 데이터 예측
print(preds[0]) # 검증 데이터로 예측한 첫번째 결과값
pred_label = np.argmax(preds[0]) # 예측 결과값에 대한 최대 예측값(약 1) 인덱스 추출
print(pred_label)
print(val_target[0]) # 검증 데이터의 타깃값 확인

# 모델 성능 평가
val_loss , val_accuracy = CNN_NewModel.evaluate(val_scaled, val_target)
print('val_loss : %.4f, val_accuracy : %.4f' % (val_loss, val_accuracy))
```

```
[4.6385484e-10 7.0333528e-13 4.9370260e-12 2.2778895e-12 1.0145110e-10
 6.3032938e-05 1.2948305e-13 6.4105041e-09 9.9993694e-01 1.9423802e-14]
```

(손실값) (정확도)

모델 성능 평가

fit(), predict(), evaluate() 메서드 첫번째 인자의  
입력차원은 데이터의 배치 차원일 것을 기대

```
preds = model.predict(val_scaled[0:1])
print(val_scaled[0:1].shape)
```

하나를 선택할 때 slicing으로  
배치차원으로 전달

```
375/375 [=====] - 1s 3ms/step - loss:
0.3297 - accuracy: 0.8774
```

```
val_loss : 0.3175, val_accuracy : 0.8857]
```

## 16) 임의의 신규 패션 이미지 load 후 CNN 모델 예측 분류

- Opencv (오픈소스 컴퓨터 비전 라이브러리) 라이브러리 활용한 이미지 처리
- 파이썬에서는 opencv-python 패키지 설치 활용 ( opencv-python 4.4.0.46 버전 사용중 )
- import cv2 ( opencv-python 패키지 설치 후 패키지 모듈 name ==> cv2 )

\* 간단히 cv2 모듈이 제공하는 이미지 파일 읽기 / 출력 / 저장 / size 변경 방법 알아보자.

\* 이미지 읽어오기 \_ **cv2.imread( filename, flag )**

```
import cv2
img = cv2.imread( 'sandal.jpg' , cv2.IMREAD_GRAYSCALE )
```

- filename : 이미지파일의 경로, flag(int) : 이미지 파일을 읽을 때의 지정 option
- return : numpy.ndarray 타입의 이미지 객체 행렬

flag 옵션 종류	의미	대체 표기
cv2.IMREAD_COLOR	이미지파일을 color로 읽어들임, 투명부분은 무시, Default 값임	정수 1
<b>cv2.IMREAD_GRAYSCALE</b>	<b>이미지파일을 Grayscale로 읽어들임, 실제 이미지처리시 많이 사용</b>	<b>정수 0</b>
cv2.IMREAD_UNCHANGED	이미지파일을 alpha channel 까지 포함하여 읽어 들임	정수 -1

### \* 이미지 출력하기 \_ cv2.imshow( title, image )

```
import cv2 # opencv-python 4.5.4.60 버전 설치
img = cv2.imread( 'sandal.jpg' , cv2.IMREAD_GRAYSCALE )
```

```
cv2.imshow('sandal img', img)
cv2.waitKey( 0 )
cv2.destroyAllWindows()
```

#### \* 파라미터

- title : 출력 윈도우창의 title,
- image(np.ndarray) : cv2.imread() 의 return 값

### \* cv2.waitKey( ) 메서드

: keyboard 입력을 대기하는 함수로 0 이면 key 입력때까지 무한 대기

: 특정 시간 동안 대기하려면 milisecond 값을 넣어주면 됨

사용법 1) cv2.waitKey(500) : 500 ms 대기

사용법 2)

```
k = cv2.waitKey(0) # 이미지 출력 상태로 임의의 키 입력할때 까지 무한 대기
```

```
if k == 27: # esc key 입력
```

```
cv2.destroyAllWindows()
```

### \* cv2.destroyAllWindows( ) 메서드

: 이미지 출력한 화면 윈도우 종료,

일반적으로 cv2.imshow(), cv2.waitKey() , cv2.destroyAllWindows() 함께 사용

**\* 이미지 저장하기 \_ cv2.imwrite( filename, image )**

```
import cv2  # opencv-python 4.5.4.60 버전 설치  
img = cv2.imread( 'sandal.jpg' , cv2.IMREAD_GRAYSCALE )
```

```
cv2.imshow('sandal img', img)  
k = cv2.waitKey( 0 )
```

```
if k == 27 : # esc key 입력  
    cv2.imwrite('sandal_gray.jpg', img)  
    cv2.destroyAllWindows()
```

**\* 파라미터**

- filename : 저장될 이미지 파일명
- image : 저장할 이미지

esc 키 입력시 출력한 이미지를 sandal\_gray.jpg 파일로 저장하면서  
이미지 출력 윈도우 종료

## \* 이미지 size 변경하기 \_ cv2.resize( )

1 번째 방법) 이미지 크기를 직접 입력 + 보간법을 활용한 픽셀 조정 방법

cv2.resize(변경할 원본이미지, 결과 이미지 크기, 보간법)

```
img_resize = cv2.resize(img, dsize=(28,28),interpolation=cv2.INTER_AREA)
cv2.imshow('img_resize' , img_resize)
```

2 번째 방법) 원본 이미지 크기 비율 입력을 통한 이미지 크기 조정

cv2.resize(변경할 원본이미지, dsize=(0, 0), 가로비,세로비, 보간법)

```
img_resize = cv2.resize(img, dsize=(0,0), fx=0.5, fy=0.5,
interpolation=cv2.INTER_LINEAR) # 원본 이미지 가로,세로 0.5배로 변경
cv2.imshow('img_resize' , img_resize)
```

- 이미지를 확대할 경우 : cv2.INTER\_CUBIC(바이큐빅보간법), cv2.INTER\_LINEAR(쌍 선형보간법) 가장 많이 사용
- 이미지를 축소할 경우 : cv2.INTER\_AREA (영역 보간법) 을 가장 많이 사용

## 16) 임의의 신규 패션 이미지 load 후 CNN 모델 예측 분류

```
# opencv-python cv2 모듈 활용 이미지 처리
import cv2 # opencv-python 4.5.4.60 버전 설치
import numpy as np
img_path = 'sandal_1.jpg'
img = cv2.imread(img_path,0) # flag 0 일 경우 Grayscale로 이미지 읽어들이
```



```
# 예측 입력을 위한 (28, 28) 크기로 이미지 사이즈 조정
img_resize = cv2.resize(img, dsize=(28,28),interpolation=cv2.INTER_AREA)
```

```
# 이미지 gray색상을 훈련 데이터셋과 일치시키기 위해
# gray색상을 반전 시켜줘야함
# (이미지 바탕색인 흰색(255)를 --> 바탕 검정색(0)으로 반전 )
```

```
img_reverted = cv2.bitwise_not(img_resize)
np.set_printoptions(suppress=True, linewidth=500)
print(img_reverted)
```

```
[[ 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  8  75  49  2  0  21  62  76  66  12  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  115  203  203  176  106  178  222  216  214  215  201  37  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  29  200  202  201  202  204  208  220  216  214  216  219  146  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  145  202  202  203  205  204  207  208  195  112  114  166  187  0  0]
.....
```

```
new_img = new_img.reshape(1,28,28,1) # 예측 입력을 위한 shape 변환
```

[[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.	0.03137255	0.29411765	0.19215686	0.00784	
314	0.	0.		0.08235294	0.24313725	0.29803922	0.25882353	0.04705882	0.	0.	0.	]								
[0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.									
0.	0.		0.45098039	0.79607843	0.79607843	0.69019608	0.41568627	0.69803922	0.87058824	0.84705882	0.83921569	0.84313725								
	0.78823529	0.14509804	0.	0.	]															
[0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.11372549	0.78431373	0.79215686	0.78823529	0.79			
215686	0.8		0.81568627	0.8627451	0.84705882	0.83921569	0.84705882	0.85882353	0.57254902	0.	0.	]								
[0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.00392157	0.56862745	0.79215686	0.79215686	0.7960784				
3	0.80392157	0.8		0.81176471	0.81568627	0.76470588	0.43921569	0.44705882	0.65098039	0.73333333	0.	0.	]							



## 16) 임의의 신규 패션 이미지 load 후 CNN 모델 예측 분류

```
preds = CNN_NewModel.predict( new_img[0:1] )
```

```
# numpy 데이터 소숫점 이하 3자리까지 출력 및 과학적 표기법 억제
np.set_printoptions(precision=3, suppress=True)
print(preds)
```

예측 결과값

```
[[0.014 0.001 0.001 0.001 0.003 0.667 0.004 0.206 0.041 0.064]]
```

```
classes = ['티셔츠', '바지', '스웨터', '드레스', '코트', '샌달', '셔츠', '스니커즈', '가방', '앵클부츠']
# preds 가장 큰 인덱스 찾아 리스트 인덱스 색인
```

```
print(classes[np.argmax(preds)])
```

샌달

카테고리	티셔츠 /탑	바지	스웨터	드레스	코트	샌달	셔츠	스니커즈	가방	앵클부츠
Label	0	1	2	3	4	5	6	7	8	9

# Thank You



HANCOM

Template Visual Guide, version 1.0

© Hancorn Inc. / Pangyo, February 2019