

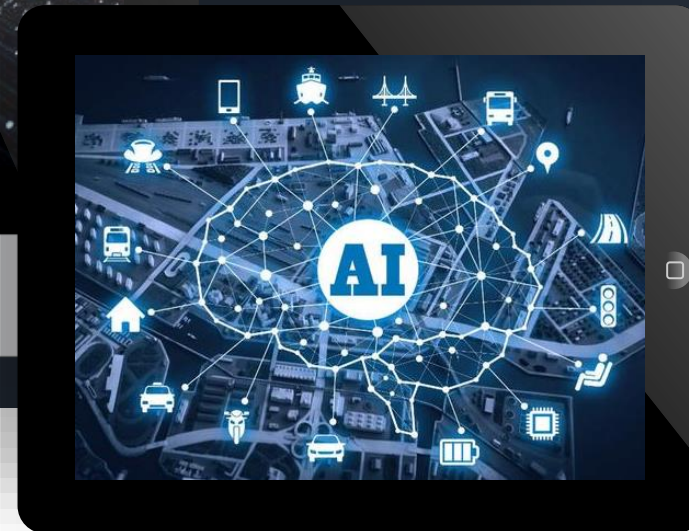
# 인공신경망 CNN 아키텍처



인공신경망 CNN 아키텍처  
Inception, Residual block  
전이학습

이 슬라이드에서 사용한 서체 :

- Open Sans(<https://ko.cooltext.com/Download-Font-Open+Sans>)
- KoPubWorld돋움체(<http://www.kopus.org/biz/electronic/font.aspx>)



# 과정 안내

인공신경망 이론과 CNN 아키텍처

## ✓ 인공지능망 이론과 CNN 아키텍처

- ▶ 이미지 합성곱을 이용한 인공지능망을 이해하고 구현할 수 있다.
- ▶ CNN 구조와 원리를 이해하고 구현할 수 있다.
- ▶ Inception Module, Residual Block을 이해하고 구현할 수 있다.
- ▶ 전이학습을 통해 이미지 분류 모델을 재학습할 수 있다.

## ✓ 주요 내용의 흐름



## ✓ 선수지식

- ▶ 파이썬, 넘파이(또는 판다스), 텐서플로우(케라스), 인공지능망 이론, 딥러닝 구현

# 1장. 합성곱 신경망

인공신경망 이론과 CNN 아키텍처

# 1절. 합성곱 신경망

1장. 합성곱 신경망

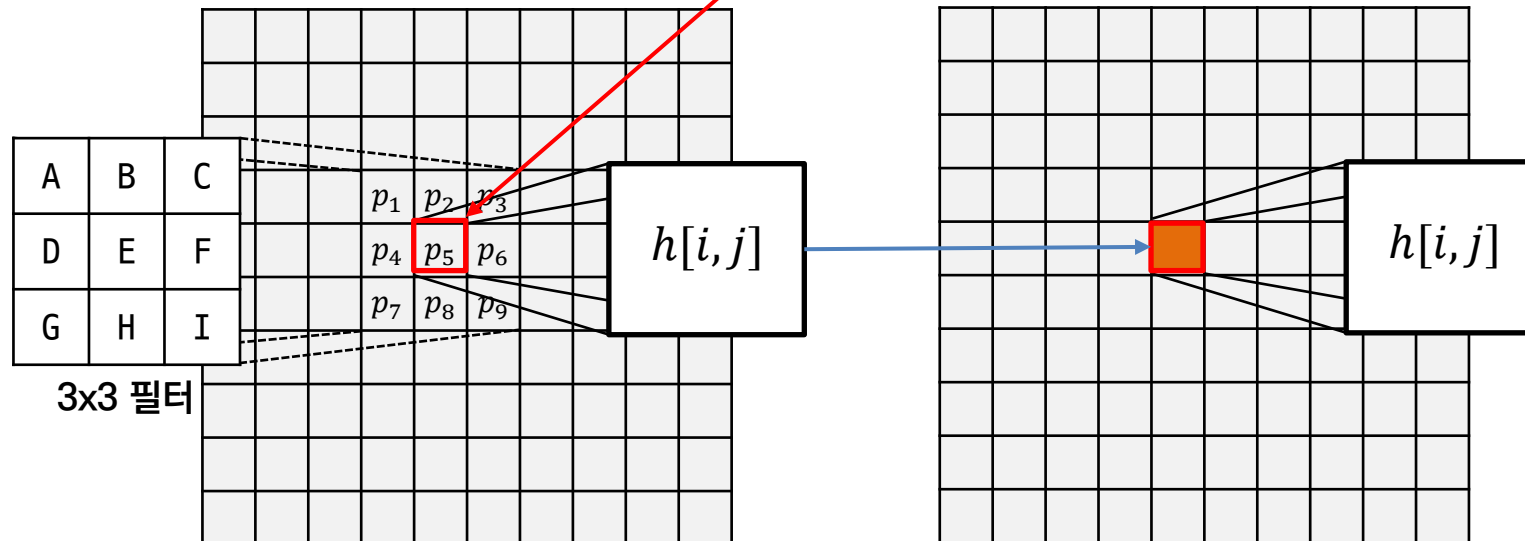


# 이미지 합성곱

1절. 합성곱 신경망 > 1.2. 이미지 합성곱

이미지는 2차원 이므로 적용하는 필터도 2차원

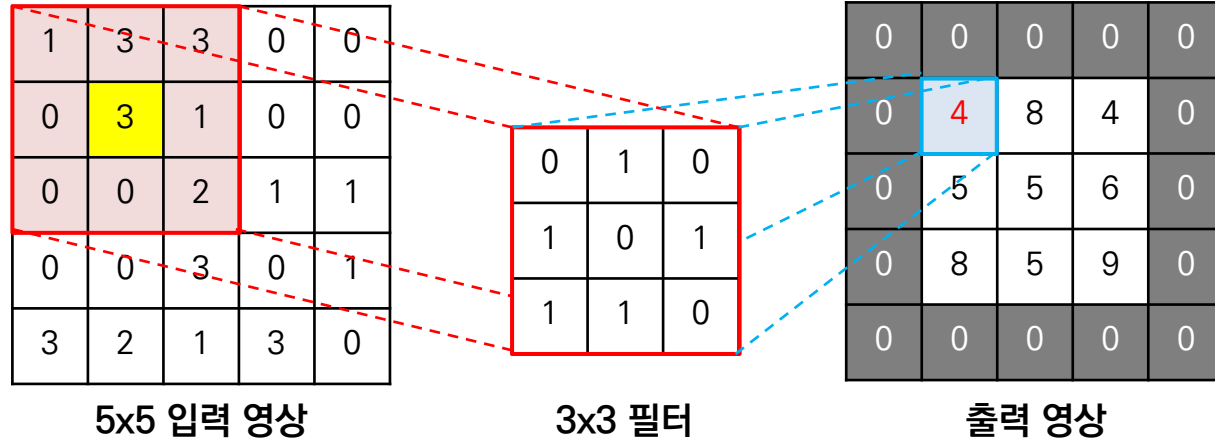
필터를 적용할 수 있는 영역에 대해서 **현재 주목화소**와 주위 화소의 배열을 각 요소끼리 모두 곱한 후 모든 항목을 더함



$$h[i, j] = A \times p_1 + B \times p_2 + C \times p_3 + D \times p_4 + E \times p_5 + F \times p_6 + G \times p_7 + H \times p_8 + I \times p_9$$

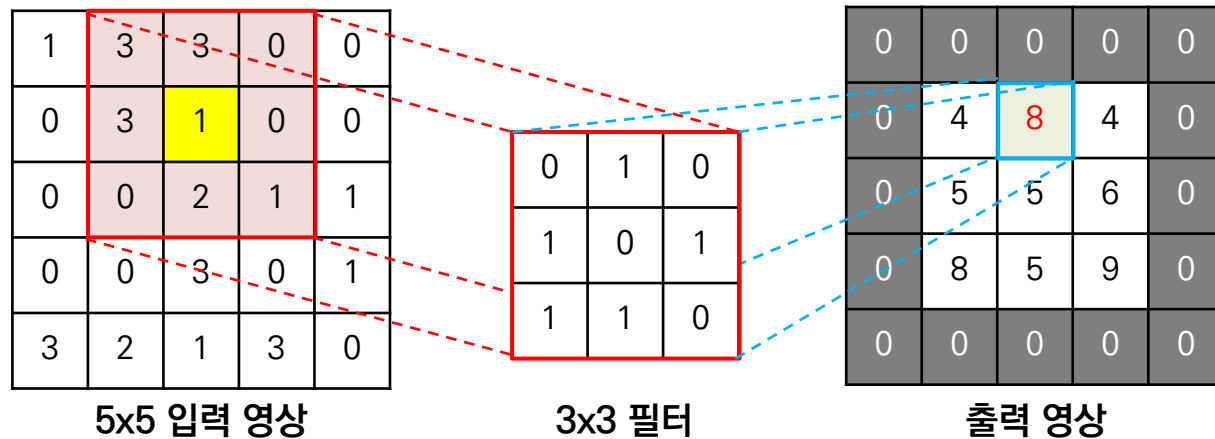
# 필터 적용

1절. 합성곱 신경망 > 1.2. 이미지 합성곱



Zero padding : 5x5 이미지의  
바깥 화소는 필터를 적용할 수  
없으므로 0으로 채움

$$\text{img}[1,1] = (1 \times 0) + (3 \times 1) + (3 \times 0) + (0 \times 1) + (3 \times 0) + (1 \times 1) + (0 \times 1) + (0 \times 1) + (2 \times 0) = 4$$



$$\text{img}[1,2] = (3 \times 0) + (3 \times 1) + (0 \times 0) + (3 \times 1) + (1 \times 0) + (0 \times 1) + (0 \times 1) + (2 \times 1) + (1 \times 0) = 8$$



# 이미지 필터 적용 예

1절. 합성곱 신경망 > 1.2. 이미지 합성곱

필터 커널에 따라 원본 이미지로부터 특성이 강조된 이미지를 얻을 수 있음

0	0	0
0	1	0
0	0	0



원 영상

0	-1	0
-1	5	-1
0	-1	0



Sharpen 필터

1	1	1
1	1	1
1	1	1

$$\frac{1}{9}$$



Box blur

1	2	1
2	4	2
1	2	1

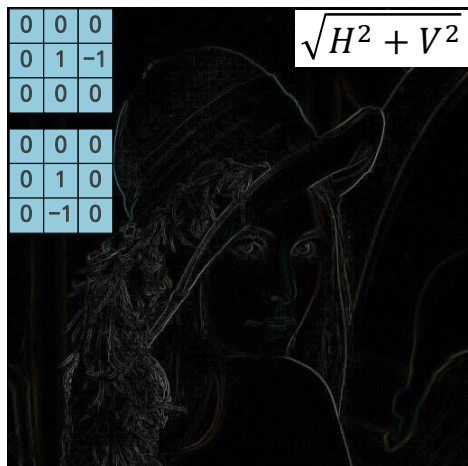
$$\frac{1}{16}$$



Gaussian blur

0	0	0
0	1	-1
0	0	0
0	0	0
0	1	0
0	-1	0

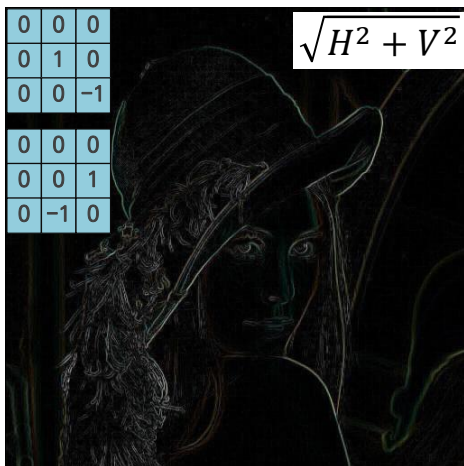
$$\sqrt{H^2 + V^2}$$



차분 필터

0	0	0
0	1	0
0	0	-1
0	0	0
0	0	1
0	-1	0

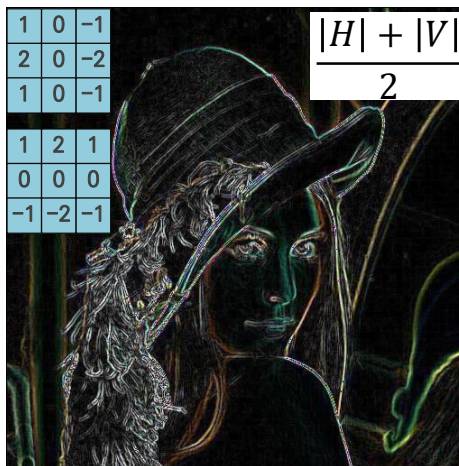
$$\sqrt{H^2 + V^2}$$



Roberts 필터

1	0	-1
2	0	-2
1	0	-1
1	2	1
0	0	0
-1	-2	-1

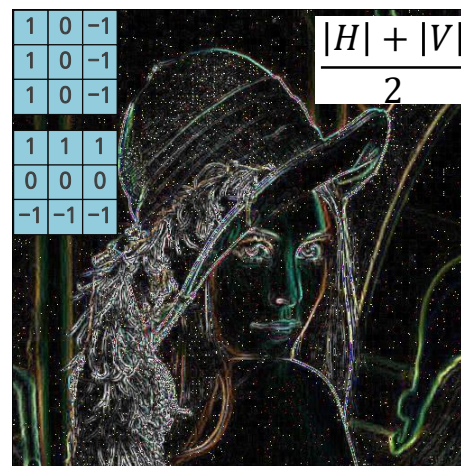
$$\frac{|H| + |V|}{2}$$



Sobel 필터

1	0	-1
1	0	-1
1	0	-1
1	1	1
0	0	0
-1	-1	-1

$$\frac{|H| + |V|}{2}$$



Prewitt 필터

## 1절. 합성곱 신경망 > 1.2. 이미지 합성곱

## 1절. 합성곱 신경망 > 1.2. 이미지 합성곱

[illegible]

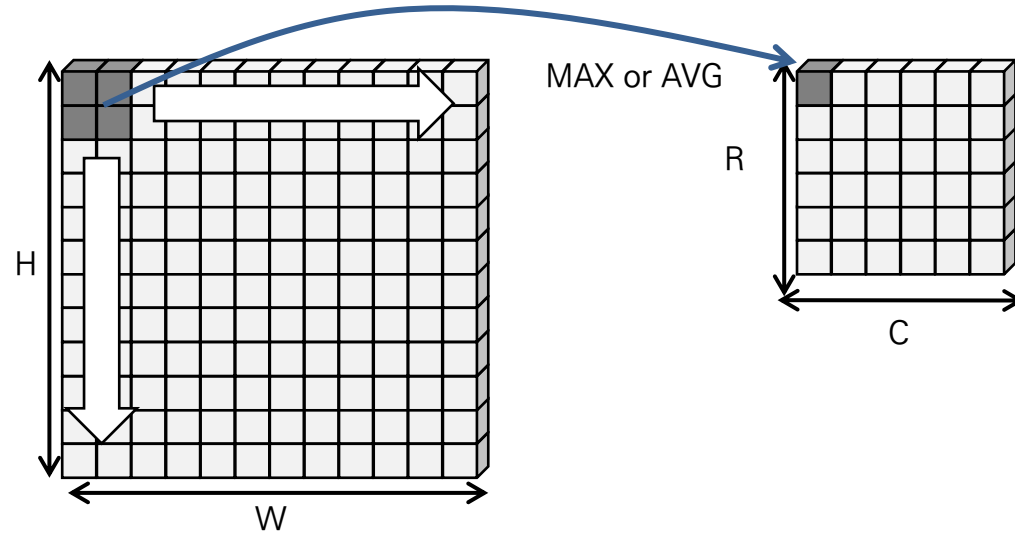
8



# 풀링

1절. 합성곱 신경망 > 1.3. 풀링

입력 이미지를 차원 축소해서 크기를 줄이는 것

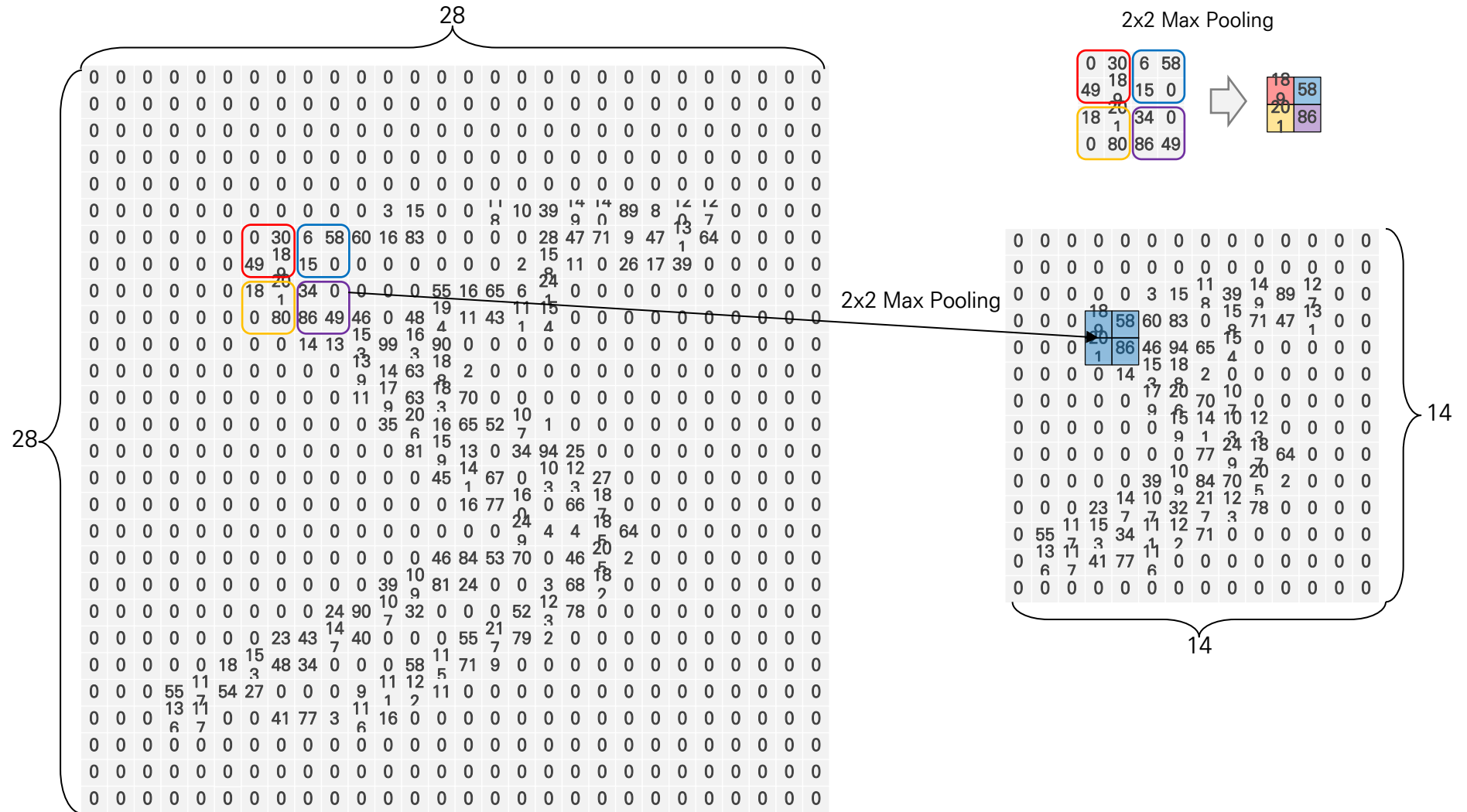


풀링(Pooling) 방법

- Max Pooling : 대상 화소 중에서 가장 큰 화소 값을 선택
- Mean Pooling(Average Pooling) : 대상 화소들의 평균값을 선택
- Min Pooling : 대상 화소 중에서 가장 작은 값을 선택
- Overlapped Pooling : 풀링 사이즈보다 스트라이드가 작을 경우

# Max Pooling

1절. 합성곱 신경망 > 1.3. 풀링

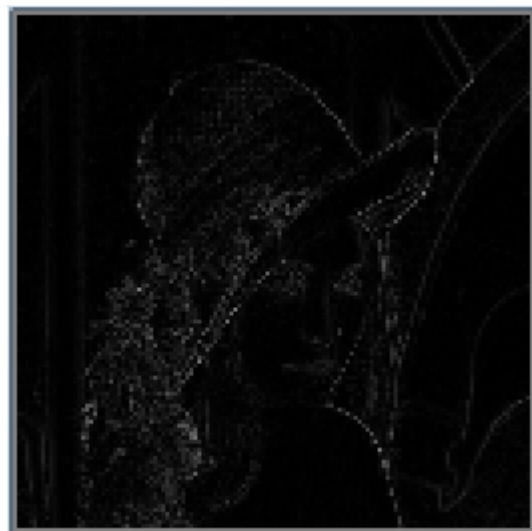


# Resize vs. Max pooling

1절. 합성곱 신경망 > 1.3. 풀링



Resize 1/2



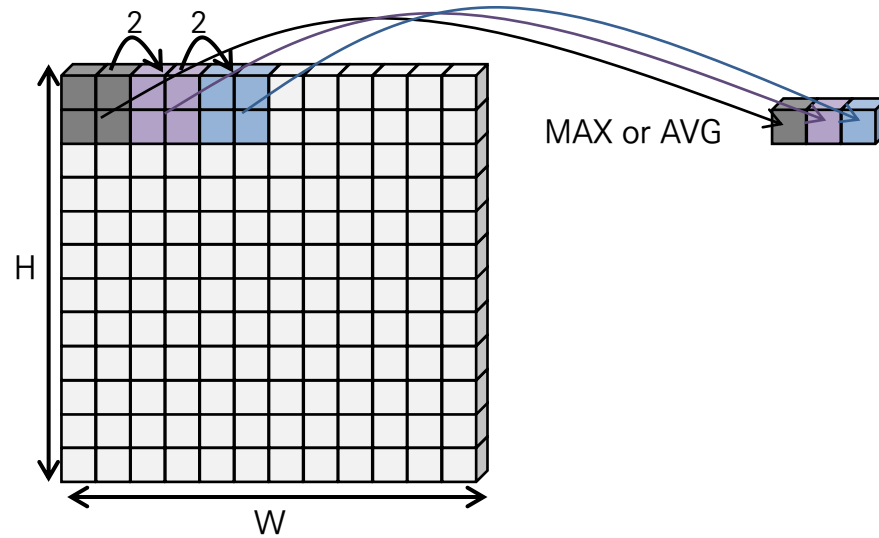
2x2 Max pooling



# 스트라이드

1절. 합성곱 신경망 > 1.4. 스트라이드와 제로 패딩

**스트라이드** : 합성곱 또는 풀링 과정에서 필터가 이미지에 적용된 후 움직이는 크기



## 1절. 합성곱 신경망 > 1.4. 슬라이드와 제로 패딩

- |   |   |    |
|---|---|----|
| 0 | 0 | 0  |
| 0 | 1 | -1 |
| 0 | 0 | 0  |

[illegible]

### 5x5 Median 필터 적용 후

[illegible]

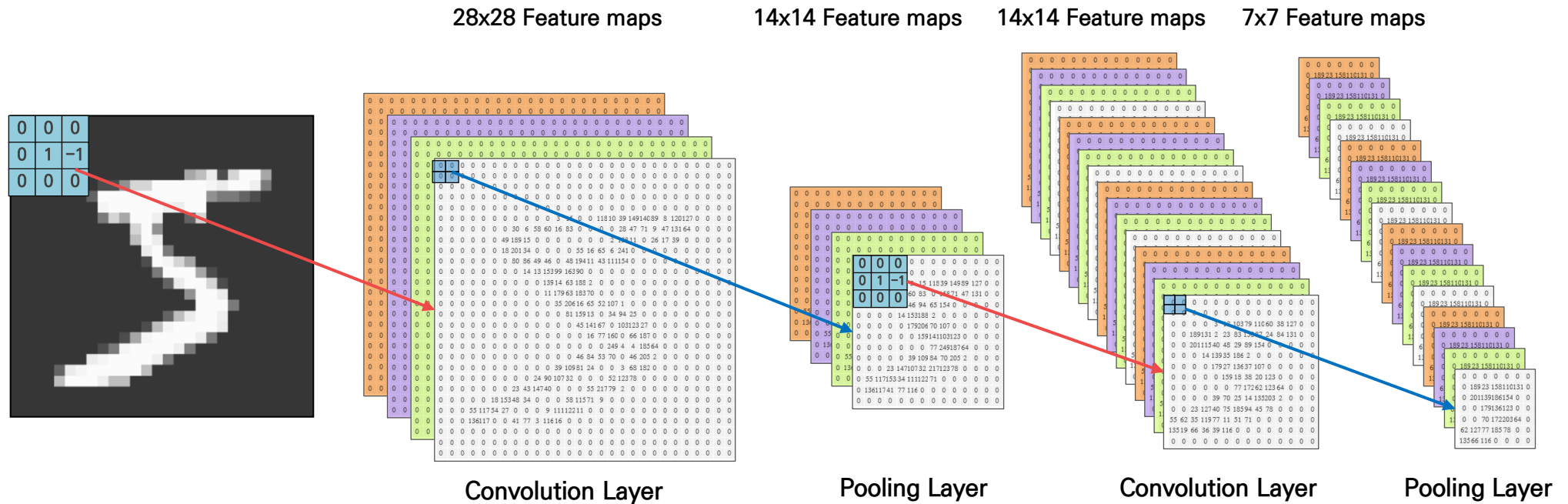
13



# 합성곱 계층과 풀링 계층

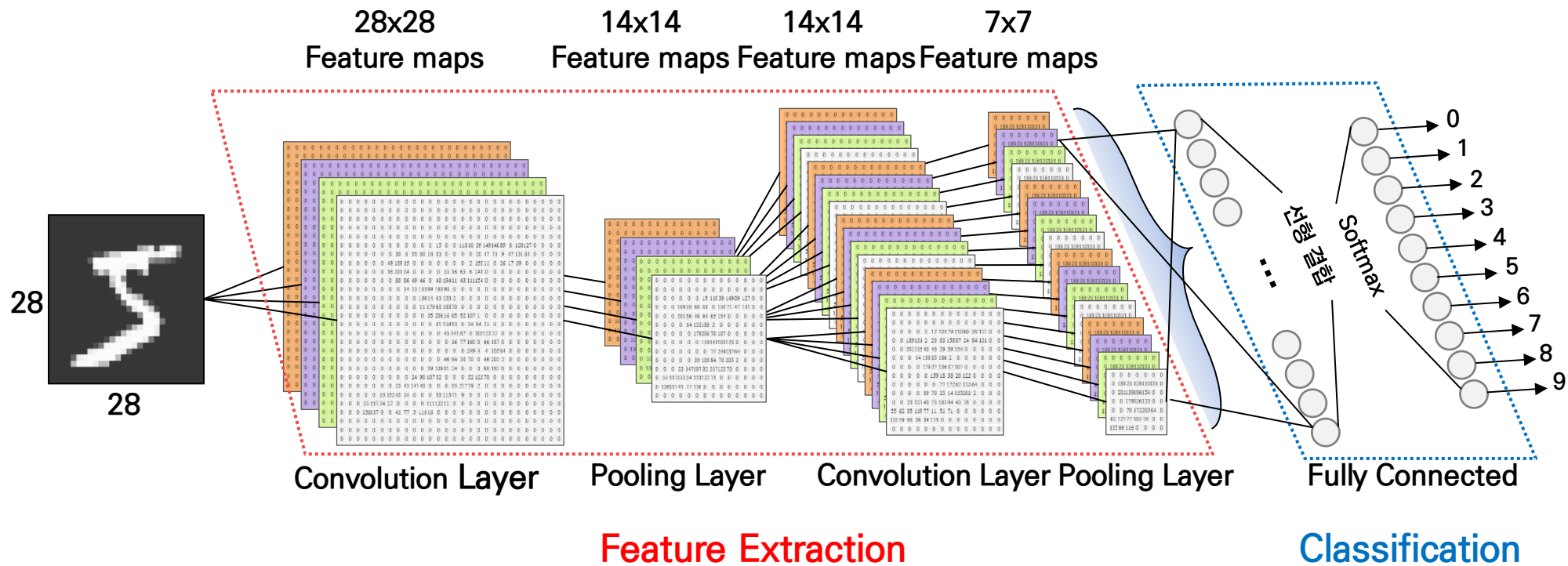
1절. 합성곱 신경망 > 1.5. CNN 모델

입력 영상의 크기가 28x28일 경우 Feature Extraction Layer



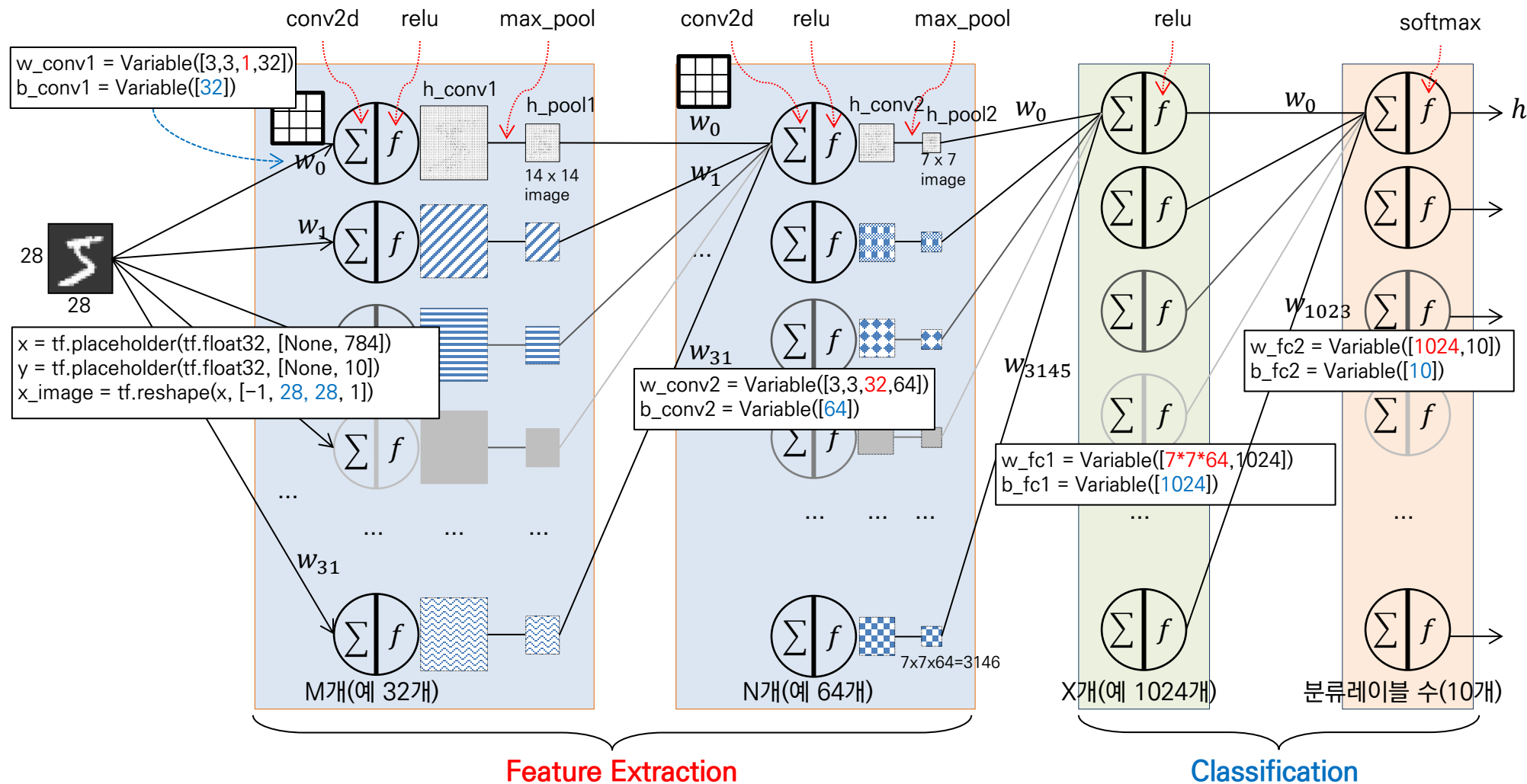
# CNN 모델

## 1절. 합성곱 신경망 > 1.5. CNN 모델



# CNN 모델링 구성도(입력 이미지 크기 28x28)

1절. 합성곱 신경망 > 1.5. CNN 모델



특징이 더 도드라지도록 Weight(Filter)가 학습됨

분류를 잘 하도록 Weight가 학습됨

## 2절. 손글씨 숫자 인식을 위한 CNN 구현



1장. 합성곱 신경망

# 손글씨 숫자 인식하기

## 2절. 손글씨 숫자 인식을 위한 CNN 구현 > 2.1. 손글씨 숫자 인식하기

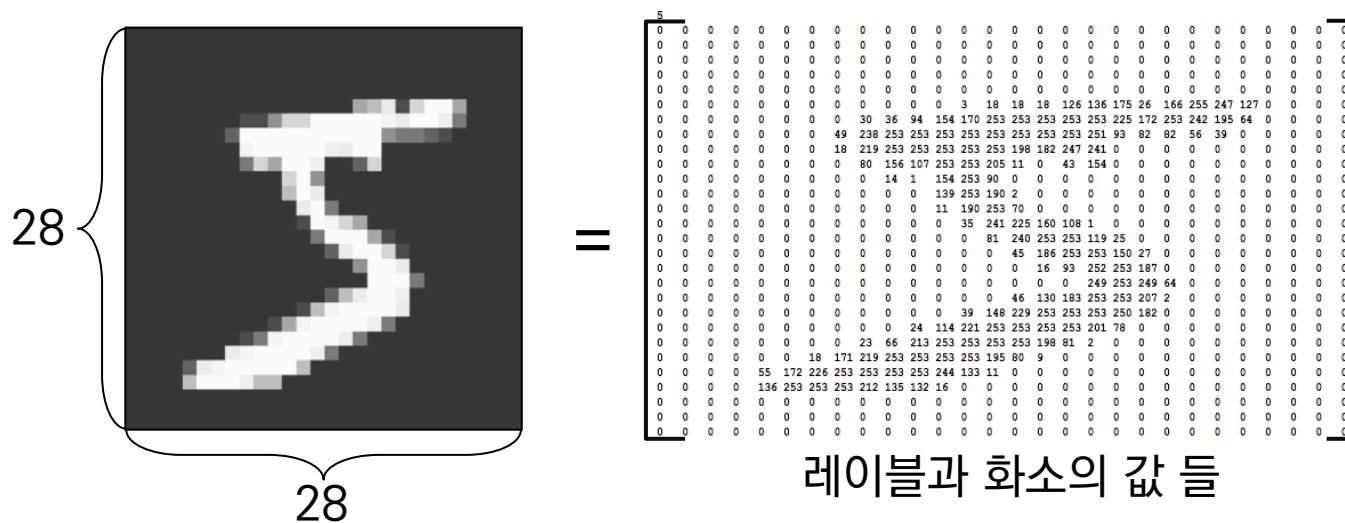
- 우편번호 자동 분류기처럼 숫자를 분류할 수 있도록 만들기 위해 필요한 것
  - 딥러닝 구현 알고리즘
  - 손으로 쓴 숫자 데이터
- 알고리즘은?
  - 인공신경망 CNN 사용
- 데이터는?
  - 종이에 0~9까지 숫자를 쓴다. → 대략 1000개 이상은 써야 하지 않을까?
  - 숫자를 이미지로 저장한다.
  - 개별 숫자를 잘라내어 파일로 저장한다.
  - 전처리 한다.
    - 숫자의 크기를 일정한 크기로 자르고, 밝기 값 조정 등...
  - 레이블을 지정한다.



# MNIST 손글씨 숫자 데이터

2절. 손글씨 숫자 인식을 위한 CNN 구현 > 2.2. MNIST 손글씨 숫자 데이터

- MNIST(Modified National Institute of Standards and Technology)
- 미국의 NIST에서 이미지 처리 시스템을 위해 모은 0부터 9까지의 숫자 이미지 데이터셋
- 각 이미지는 28x28 크기, 이것을 펼치면 784 차원의 벡터

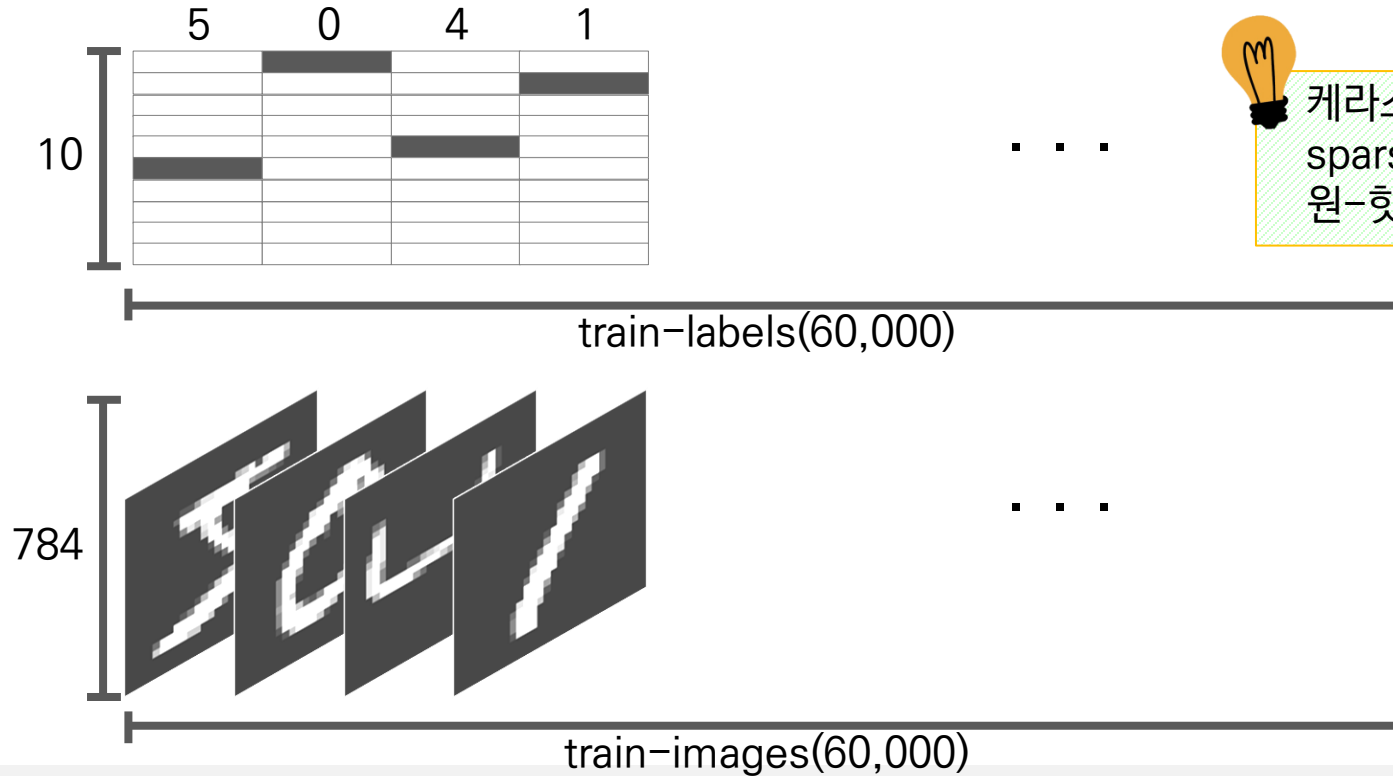


28 x 28 pixels image    이미지를 784개 화소(Pixel)의 숫자로 표현

# 인공신경망에서 원-핫 인코딩

## 2절. 손글씨 숫자 인식을 위한 CNN 구현 > 2.2. MNIST 손글씨 숫자 데이터


- 원-핫 인코딩(One-Hot Encoding)은 범주형 데이터(카테고리 데이터)를 수치 데이터로 변환하는 데 사용되는 기술
- 원-핫 인코딩의 핵심 아이디어는 각 카테고리 값을 이진 벡터로 표현하는 것
  - 각 카테고리는 해당하는 인덱스 위치의 원소만 1이고 나머지 원소는 모두 0으로 표현됨
  - 이렇게 하면 모든 카테고리가 수치 데이터로 변환되며, 각 카테고리 간에 상호간섭이 없게 됨
- 인공신경망은 멀티클래스 분류일 경우 출력층의 뉴런의 개수가 클래스 라벨의 수와 일치해야 하는데, 그러면 인공신경망의 출력은 분류 클래스 레이블의 수 만큼 출력되고, 이 값이 정답과 비교되려면 정답을 원-핫 인코딩 해야 함



💡 케라스에서는 멀티클래스 분류를 위한 손실함수에 `sparse_categorical_crossentropy`를 사용하면 원-핫 인코딩 하지 않아도 됩니다.

# 합성곱 층을 위한 클래스 Conv2D

2절. 손글씨 숫자 인식을 위한 CNN 구현 > 2.3. 손글씨 숫자 인식을 위한 케라스 CNN 모델



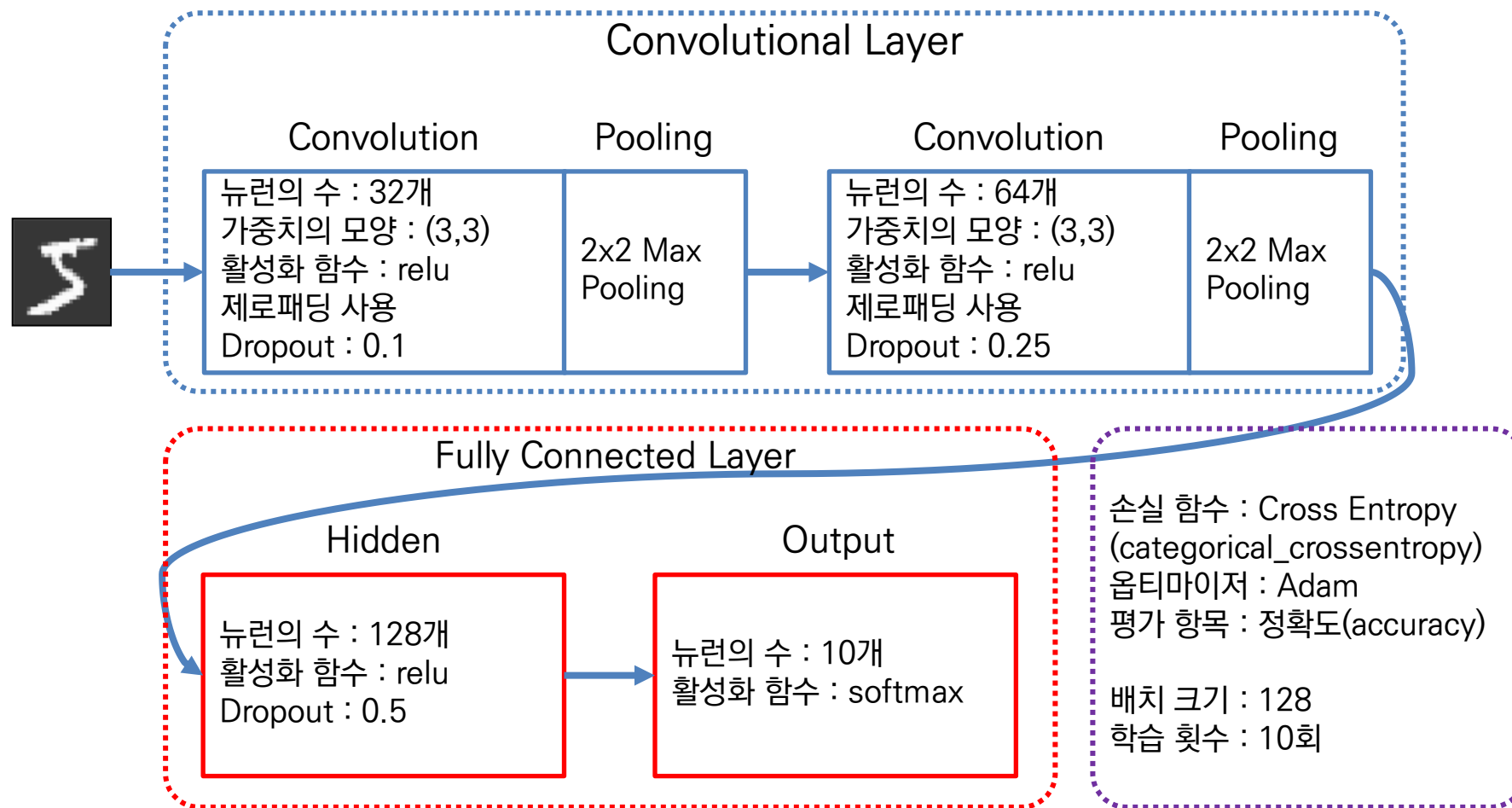
```
tf.keras.layers.Conv2D(
```

```
    filters, kernel_size,  
    strides=(1, 1), padding='valid',  
    data_format=None, dilation_rate=(1, 1),  
    activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform',  
    bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None
```

```
)
```

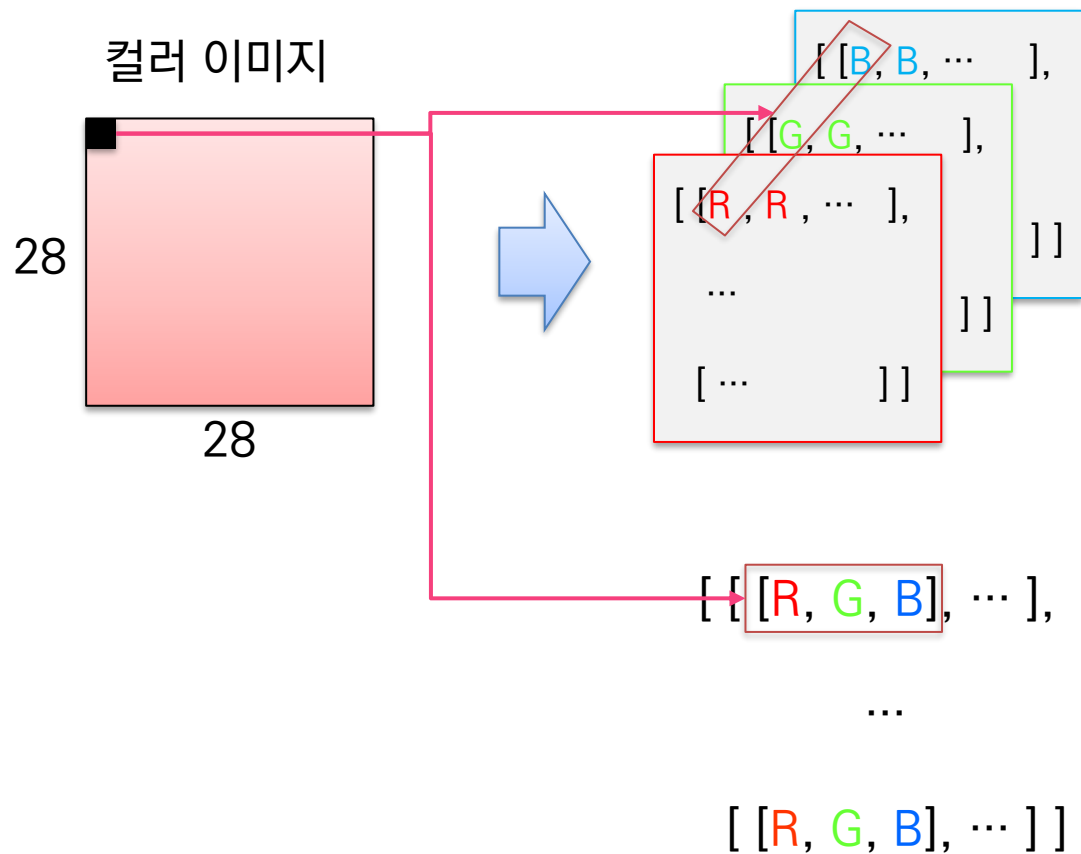
# 손글씨 숫자 인식을 위한 케라스 CNN 모델

2절. 손글씨 숫자 인식을 위한 CNN 구현 > 2.3. 손글씨 숫자 인식을 위한 케라스 CNN 모델



# Channel First와 Channel Last

2절. 손글씨 숫자 인식을 위한 CNN 구현



Channel First  
(3, 28, 28)



텐서플로우는 Channel last,  
파이토치는 Channel first 구조를  
사용합니다.

Channel Last  
(28, 28, 3)



# CNN 코드 (1/2) – 모델 정의하기

2절. 손글씨 숫자 인식을 위한 CNN 구현 > 2.3. 손글씨 숫자 인식을 위한 케라스 CNN 모델

## ✓ 모델 정의

```
1. from tensorflow.keras import Sequential, layers
2. model = Sequential([
3.     layers.Input(shape=(28,28,1)),
4.     layers.Conv2D(32, (3,3), activation='relu'),
5.     layers.MaxPooling2D(), # pool_size=(2,2)가 기본값
6.     layers.Dropout(0.1),
7.     layers.Conv2D(64, 3, activation='relu'),
8.     layers.MaxPooling2D(),
9.     layers.Dropout(0.25),
10.    layers.Flatten(),
11.    layers.Dense(128, activation='relu'),
12.    layers.Dropout(0.5),
13.    layers.Dense(10, activation='softmax')
14. ])
15. model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d (MaxPooling2D)	(None, 12, 12, 32)	0
dropout (Dropout)	(None, 12, 12, 32)	0
conv2d_1 (Conv2D)	(None, 10, 10, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204928
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

=====  
Total params: 225546 (881.04 KB)  
Trainable params: 225546 (881.04 KB)  
Non-trainable params: 0 (0.00 Byte)  
=====

# CNN 코드 (2/2) – 데이터 불러오고 학습한 후 평가하기

2절. 손글씨 숫자 인식을 위한 CNN 구현 > 2.3. 손글씨 숫자 인식을 위한 케라스 CNN 모델

## ✓ 훈련 정의

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

## ✓ 데이터 불러오기

```
1. from tensorflow.keras.datasets import mnist  
2. (X_train, y_train), (X_test, y_test) = mnist.load_data()  
3. X_train = X_train.reshape(-1, 28, 28, 1) / 255.0  
4. X_test = X_test.reshape(-1, 28, 28, 1) / 255.0
```

## ✓ 데이터 학습

```
1. model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1)
```

## ✓ 모델 평가

```
1. loss, accuracy = model.evaluate(test_X, test_y, verbose=0)  
2. print(f'Test loss: {loss}, Test accuracy: {accuracy}')
```

## 3절. CNN 아키텍처

### 1장. 합성곱 신경망



# CNN 알고리즘

## 3절. CNN 아키텍처

LeNet : 최초의 CNN

AlexNet : 2012년 ILSVRC 대회 우승

VGGNet : 2014년 ILSVRC 준우승

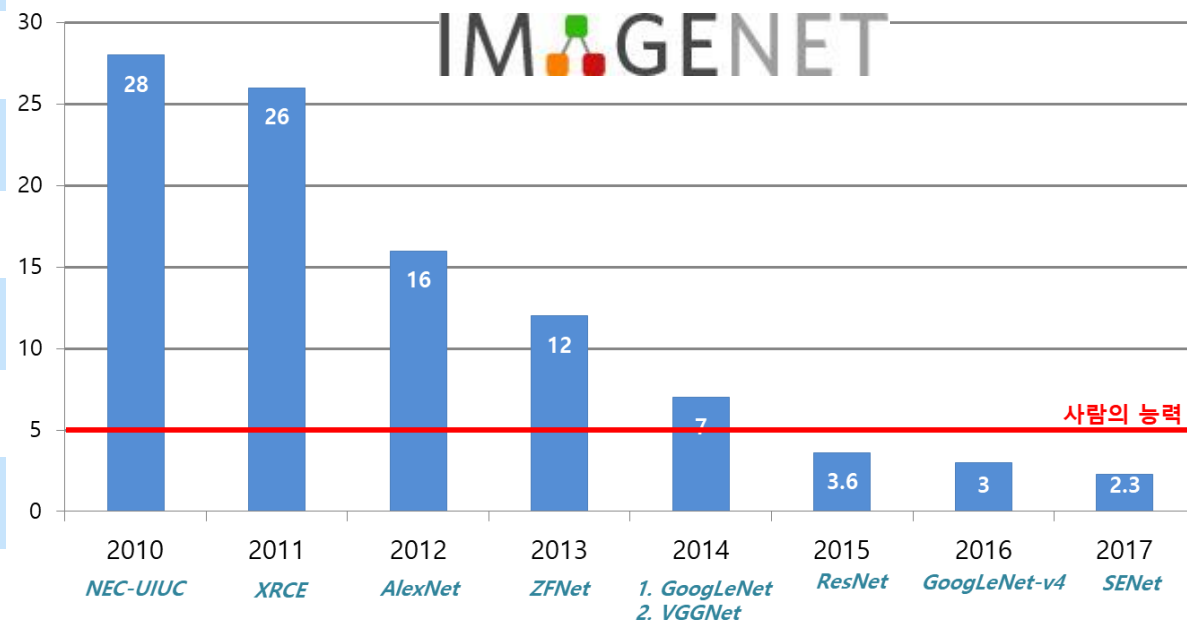
GoogLeNet : 2014년 ILSVRC 대회 우승

ResNet : 2015년 ILSVRC 대회 우승(top-5 error 3.57%)

SENet : 2017년 ILSVRC 대회 우승(top-5 error 2.251%)

<http://www.image-net.org/challenges/LSVRC/>

우승 알고리즘의 분류 에러율(%)



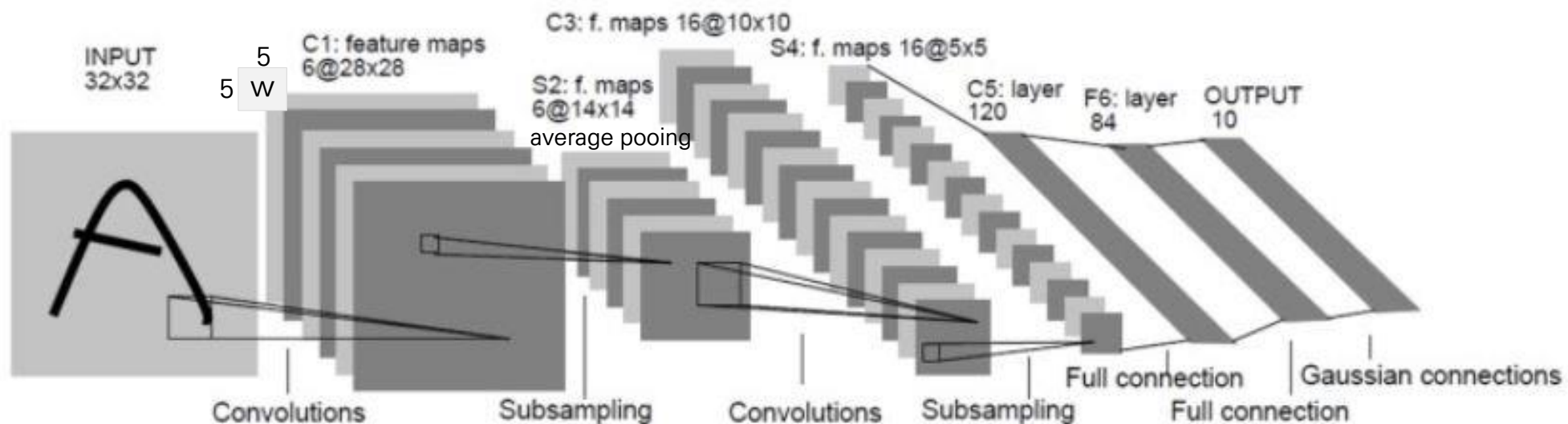
# LeNet

## 3절. CNN 아키텍처 > 3.1. LeNet

LeNet은 CNN을 처음 개발한 Yann Lecun연구팀이 1998년에 개발한 알고리즘

논문명 : Gradient-based learning applied to document recognition

Convolution과 Subsampling을 반복적으로 거치면서, 마지막에 Fully-connected Multi-layered Neural Network로 Classification을 수행

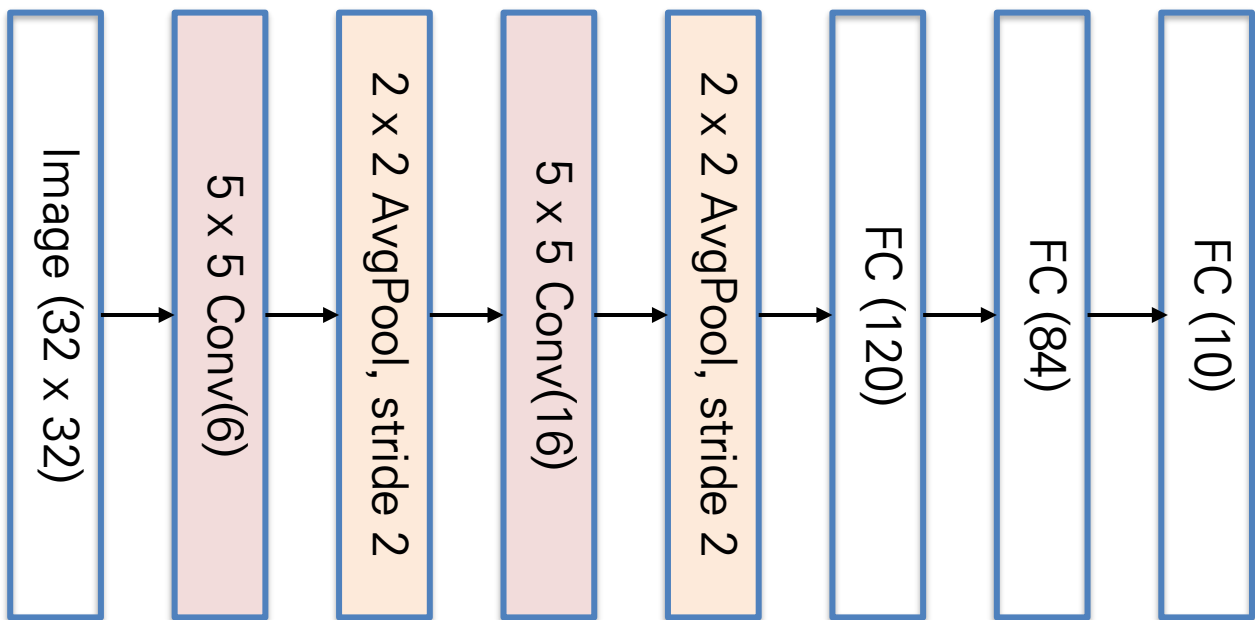


LeNet-5 구조



# 케라스로 LeNet 구현하기

3절. CNN 아키텍처 > 3.1. LeNet



# 케라스로 LeNet 구현하기

## 3절. CNN 아키텍처 > 3.1. LeNet

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D
from tensorflow.keras.layers import Flatten, Dense, Input

class LeNet:

    @staticmethod
    def build(input_shape=(32,32,1), activation='sigmoid'):
        model = Sequential()
        model.add(Input(shape=input_shape))

        # 첫 번째 CONV => ACTIVATION => POOL 계층
        model.add(Conv2D(6, 5, activation=activation,
                        kernel_initializer='random_uniform'))
        model.add(AveragePooling2D(pool_size=(2,2)))

        # 두 번째 CONV => ACTIVATION => POOL 계층
        model.add(Conv2D(16, 5, activation=activation,
                        kernel_initializer='random_uniform'))
        model.add(AveragePooling2D(pool_size=(2,2)))

        # 첫 번째 FC 계층
        model.add(Flatten())
        model.add(Dense(120, activation=activation))
        # 두 번째 FC 계층
        model.add(Dense(84, activation=activation))
        # 출력층 soft-max 활성화 함수 사용
        model.add(Dense(10, activation='softmax'))

        return model
```

```
model = LeNet.build(input_shape=(28, 28, 1), activation="relu")
model.summary()

model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=['accuracy'])

from tensorflow.keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 28, 28, 1) / 255.0
X_test = X_test.reshape(-1, 28, 28, 1) / 255.0

hist = model.fit(X_train, y_train, batch_size=200, epochs=20)

loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Accuracy: {accuracy * 100:.2f}%")
```



LeNet 아키텍처를 생각하며  
구현해 보세요.

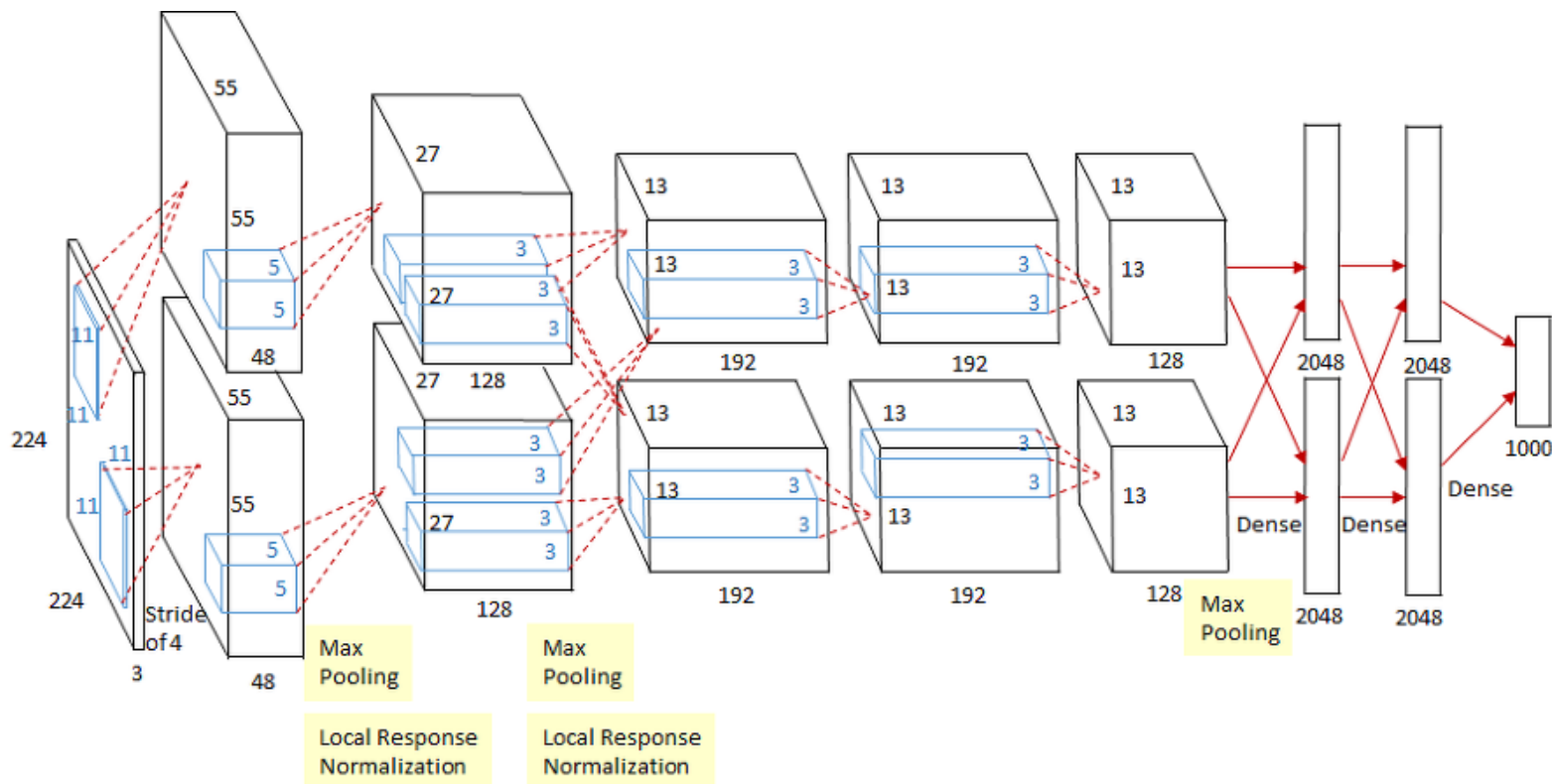
# AlexNet

3절. CNN 아키텍처 > 3.2. AlexNet

ImageNet 영상 데이터 베이스를 기반으로 한 화상 인식 대회인 “ILSVRC 2012”에서 우승

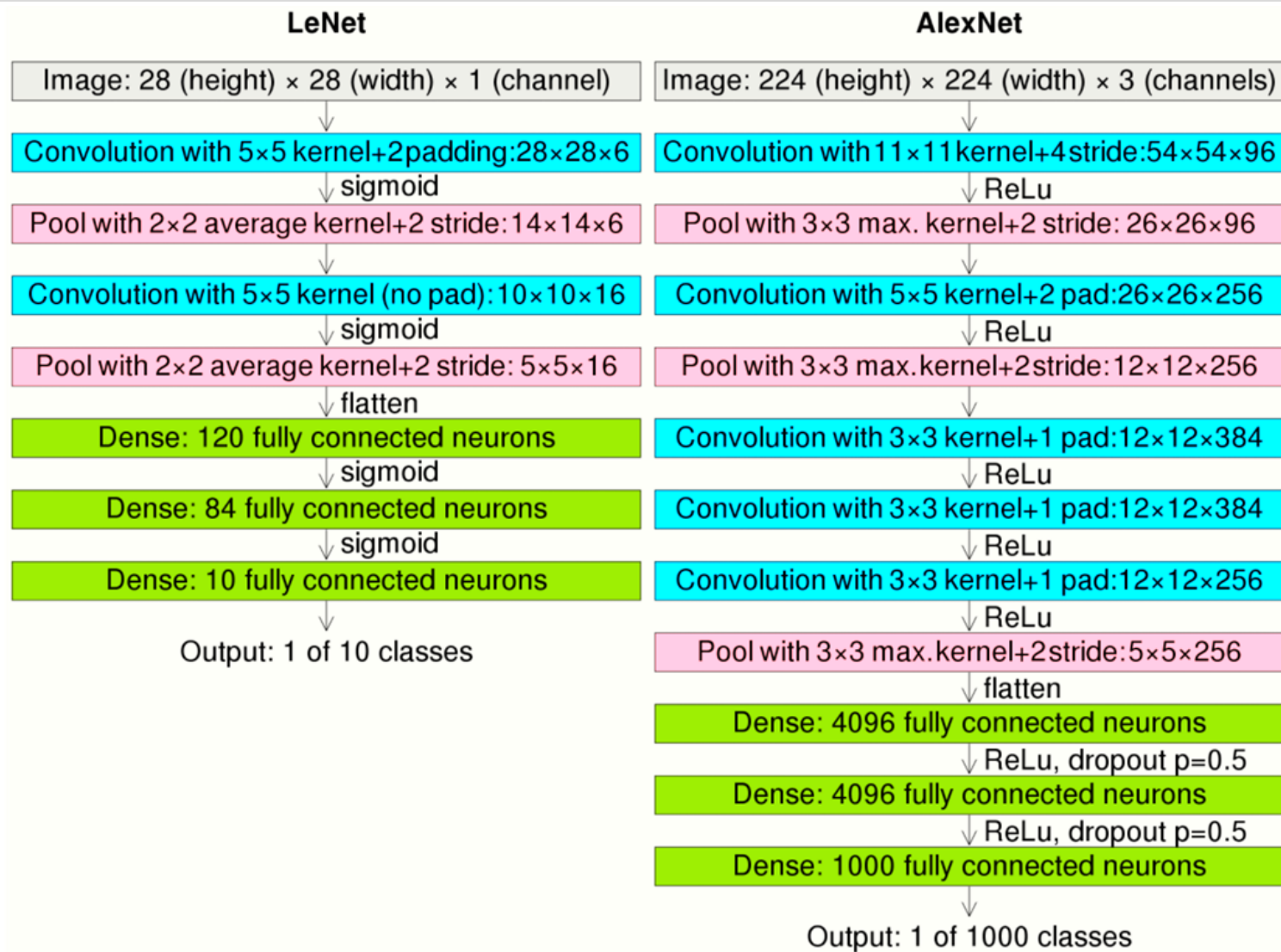
논문명 : ImageNet Classification with Deep Convolutional Neural Networks

5개의 convolution layers와 3개의 fully-connected layers로 구성되어 있음



# LeNet과 AlexNet 비교

3절. CNN 아키텍처 > 3.2. AlexNet



# AlexNet의 주요 특징

3절. CNN 아키텍처 > 3.2. AlexNet

ReLU

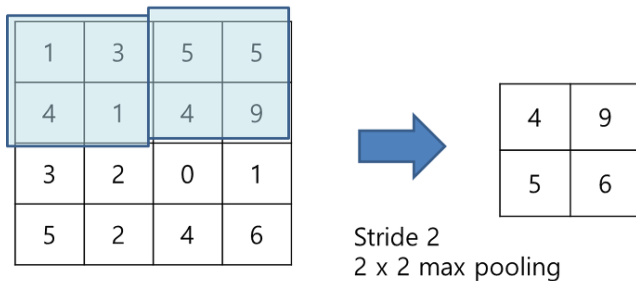
dropout

Overlapping pooling

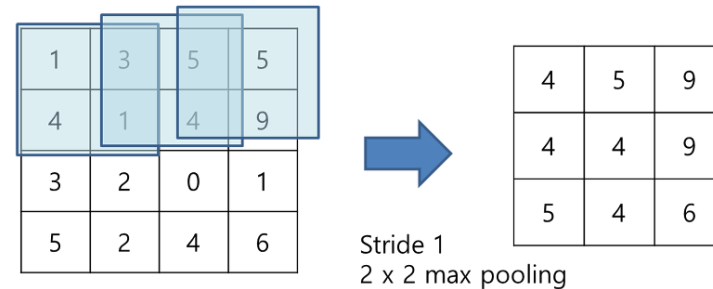
LRN(local response normalization)

Image augmentation

Non-overlapping pooling



Overlapping pooling



과적합을 방지하기 위해서 사용한 LRN은 주로 CNN에서 사용되며, 뉴런의 출력을 정규화하고 지역 정규화를 수행합니다.

- 이는 뉴런의 주변 뉴런에 의한 활성화를 정규화하며, 특정 지역 창 내에서 가중치가 있는 뉴런을 포함한 피쳐 맵 값을 정규화합니다.
- 이 방법은 주변 뉴런에 의존하며 하이퍼 파라미터로 조절됩니다.
- 반면, 배치 정규화(Batch Normalization)는 미니배치 내 각 입력 특성을 독립적으로 정규화하고, 레이어의 모든 뉴런에 대한 평균과 분산을 계산한 후, 이를 사용하여 모든 뉴런에 동일한 정규화 값을 적용합니다. 이 방법은 전체 배치에 대한 통계를 사용하여 뉴런의 활성화 값을 정규화하며, 미니배치 내의 모든 입력에 대해 동작합니다.
- 현재, 배치 정규화가 더 효과적으로 사용되며, LRN은 대부분 사용되지 않습니다.

# AlexNet을 이용한 CIFAR-10 이미지 분류

## 3절. CNN 아키텍처 > 3.2. AlexNet

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

class AlexNet:
    @staticmethod
    def build(input_shape=(224,224,3), activation='relu', class_num=1000):
        model = Sequential()
        model.add(Input(shape=input_shape))

        model.add(Conv2D(96, (11,11), strides=(4,4),
                        activation=activation, padding="same"))
        model.add(MaxPooling2D(pool_size=(3,3), strides=2))
        model.add(BatchNormalization())

        model.add(Conv2D(256, (5,5), activation=activation, padding="same"))
        model.add(MaxPooling2D(pool_size=(3,3), strides=2))
        model.add(BatchNormalization())

        model.add(Conv2D(384, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(384, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(256, (3,3), activation=activation, padding="same"))

        model.add(Flatten())
        model.add(Dense(4096, activation=activation))
        model.add(Dense(4096, activation=activation))
        model.add(Dense(class_num, activation='softmax'))

        return model
```

```
model = AlexNet.build(input_shape=(32,32,3), class_num=10)
```

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

```
from tensorflow.keras.datasets import cifar10
(train_X, train_y), (test_X, test_y) = cifar10.load_data()
```

```
model.fit(train_X, train_y, validation_data=(test_X, test_y),
          batch_size=200, epochs=10, verbose=1)
```

```
loss, accuracy = model.evaluate(test_X, test_y, verbose=1)
print(f"Loss: {loss}, Accuracy: {accuracy*100}%")
```



AlexNet 아키텍처를 생각하며  
구현해 보세요.

# VGG(Visual Geometry Group)

3절. CNN 아키텍처 > 3.3. VGGNet

옥스포드 대학의 연구팀 VGG

VGGNet부터 네트워크의 깊이가 깊어지기 시작(VGGNet은 VGG16, VGG19와 같은 모델)

최초의 Very Deep 네트워크, Kernel Size를 줄여 연산량의 부담을 줄이는 방식을 보여줌

옥스포드 대학의 VGG팀은 VGG16, VGG19를 개발하기 전에, 먼저 AlexNet과 거의 유사한 CNN 모델 (VGG-F, VGG-M, VGG-S)들을 개발

논문명 : Return of the Devil in the Details: Delving Deep into Convolutional Nets

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
CNN-F	64x11x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x3 pool	4096 drop- out	4096 drop- out	1000 soft- max

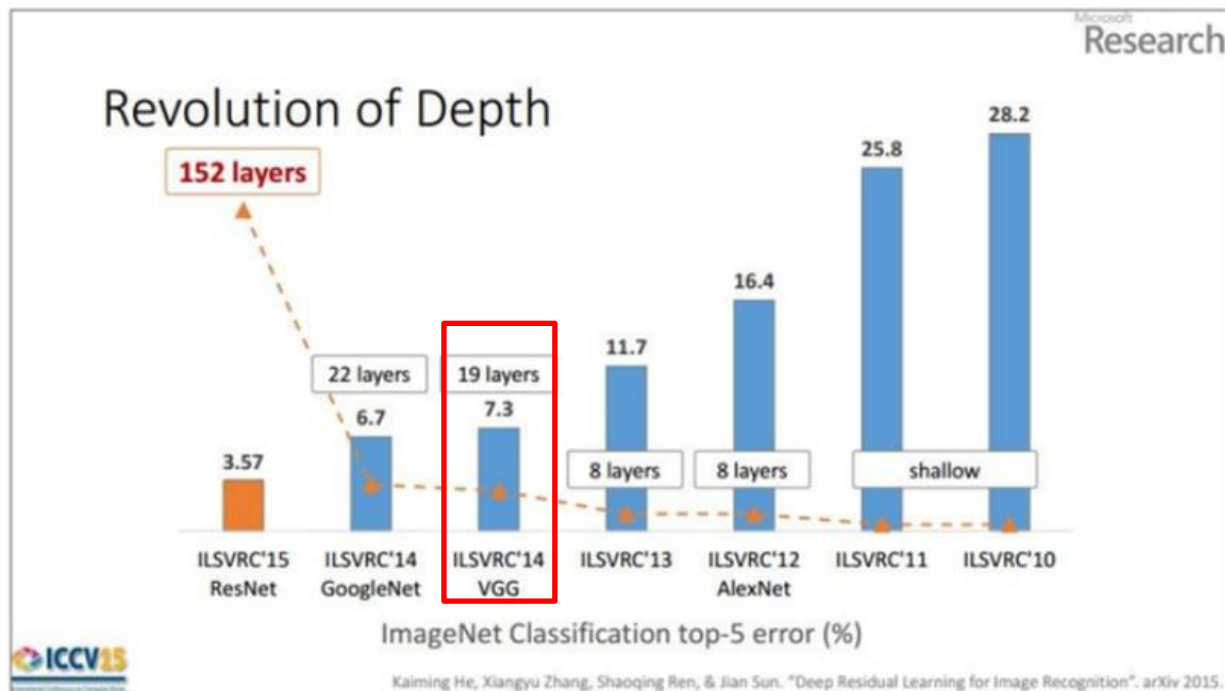


# VGGNet

3절. CNN 아키텍처 > 3.3. VGGNet

옥스포드 대학의 연구팀 VGG에 의해 개발된 VGGNet

ILSVRC-2014 준우승



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

# VGG-19 구현

## 3절. CNN 아키텍처 > 3.3. VGGNet

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

class VGG19:
    @staticmethod
    def build(input_shape=(224,224,3), activation='relu'):
        model = Sequential()

        model.add(Conv2D(64, (3,3), input_shape=input_shape,
                        activation=activation, padding="same"))
        model.add(Conv2D(64, (3,3), activation=activation, padding="same"))
        model.add(MaxPooling2D(pool_size=(2,2)))

        model.add(Conv2D(128, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(128, (3,3), activation=activation, padding="same"))
        model.add(MaxPooling2D(pool_size=(2,2)))

        model.add(Conv2D(256, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(256, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(256, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(256, (3,3), activation=activation, padding="same"))
        model.add(MaxPooling2D(pool_size=(2,2)))

        model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
        model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
        model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
model.add(Conv2D(512, (3,3), activation=activation, padding="same"))
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(Flatten())
model.add(Dense(4096, activation=activation))
model.add(Dense(4096, activation=activation))
model.add(Dense(1000, activation='softmax'))
```

```
return model
```

```
from tensorflow.keras.applications.vgg19 import VGG19
vgg = VGG19() # model & weights
weights = vgg.get_weights()
model.set_weights(weights)
```



모델을 내려받아  
weight를 바꿔  
사용합니다.

```
from tensorflow.keras.preprocessing import image
img = image.load_img("sample.jpg", target_size=(224,224))
img_data = image.img_to_array(img)
print("before reshape:", img_data.shape)
import numpy as np
img_data = img_data[np.newaxis, ...]
print("after reshape:", img_data.shape)

pred = model.predict(img_data)
from tensorflow.keras.applications.vgg19 import decode_predictions
print(decode_predictions(pred, top=3))
```

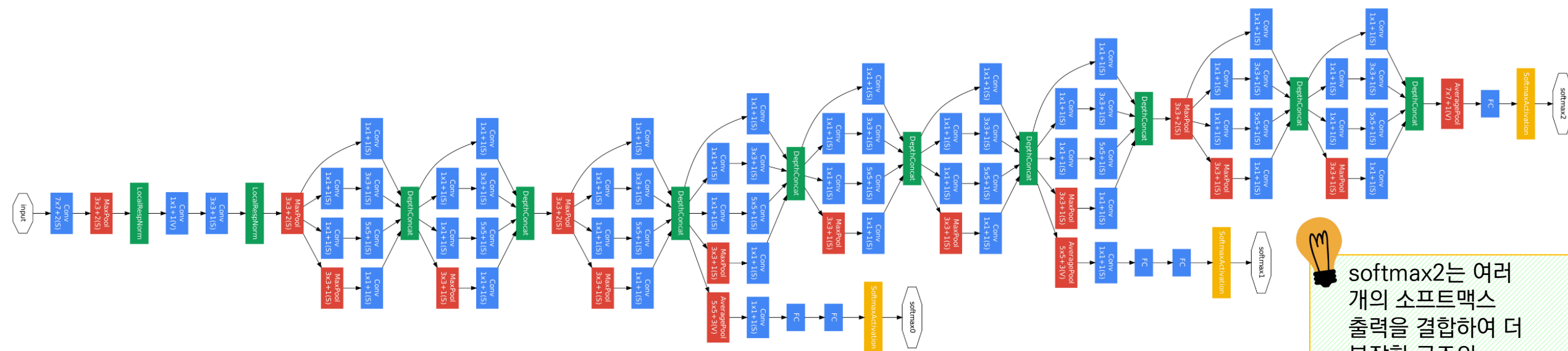
# GoogLeNet

3절. CNN 아키텍처 > 3.4. GoogLeNet

ILSVRC-2014에서 Top-5 test accuracy 93.3%로 우승

논문명 : Going Deeper with Convolutions(<https://arxiv.org/abs/1409.4842>)

GoogLeNet은 19층의 VGG19보다 좀 더 깊은 22층으로 구성되어 있음



GoogLeNet의 구조

softmax2는 여러 개의 소프트맥스 출력을 결합하여 더 복잡한 구조의 네트워크를 지원합니다.

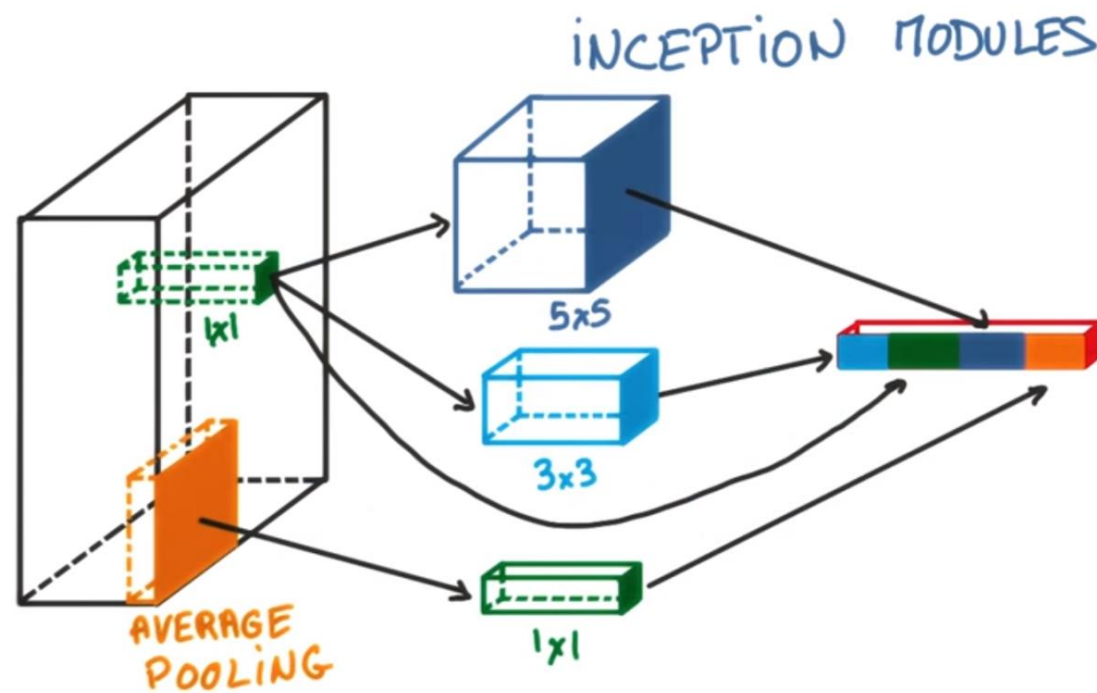
# GoogLeNet

3절. CNN 아키텍처 > 3.4. GoogLeNet

22 레이어로 더 깊어졌고, Inception이라는 모듈이 도입되었음

VGGNet보다 구조가 복잡해 널리 쓰이진 않았지만 아키텍처 면에서 주목을 받음

보통 하나의 conv layer에는 한 가지의 conv filter가 사용했었는데, GoogLeNet 연구진들은 한 가지의 conv filter를 적용한 conv layer를 단순히 깊게 쌓는 방법도 있지만, 하나의 layer에서도 다양한 종류의 filter나 pooling을 도입함으로써 개별 layer를 두텁게 확장시킬 수 있다는 창조적인 아이디어로 후배 연구자들에게 많은 영감을 줌



# GoogLeNet 특징

3절. CNN 아키텍처 > 3.4. GoogLeNet

1 x 1 컨볼루션 : 특성맵의 개수를 줄임(연산량이 줄어듦)

Inception 모듈 : 이전 층에서 생성된 특성맵을 1x1 컨볼루션, 3x3 컨볼루션, 5x5 컨볼루션, 3x3 최대풀링해준 결과 얻은 특성맵들을 모두 함께 쌓아줌(더 다양한 종류의 특성이 도출됨)

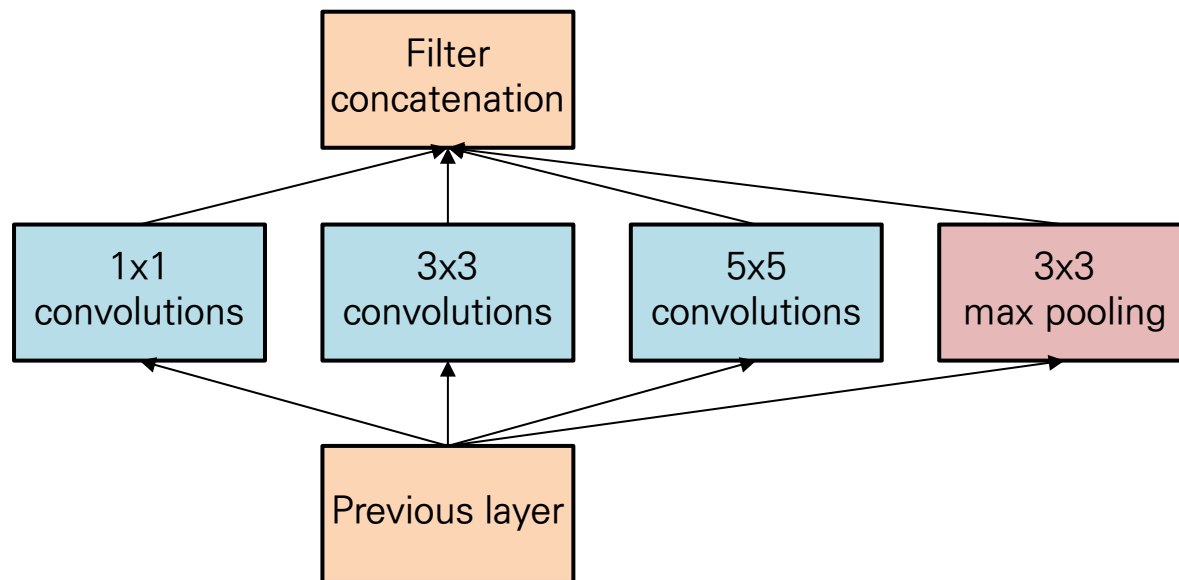
global average pooling : 전 층에서 산출된 특성맵을 각각 평균낸 것을 이어서 1차원 벡터를 만들어주는 것

- 1차원 벡터를 만들어줘야 최종적으로 이미지 분류를 위한 softmax 층을 연결해줄 수 있기 때문

auxiliary classifier : 네트워크 중간에 두 개의 보조 분류기(auxiliary classifier)를 달아줌.

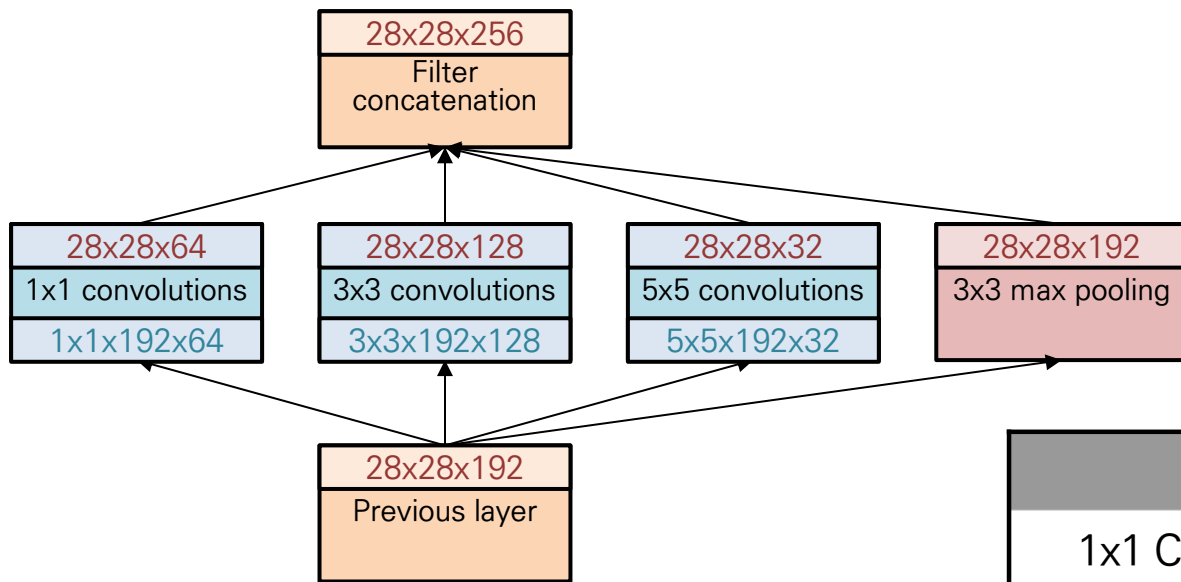
# Inception 모듈, naïve version

3절. CNN 아키텍처 > 3.4. GoogLeNet



# 가중치 개수와 연산 횟수는?

3절. CNN 아키텍처 > 3.4. GoogLeNet



	채널 수	파라미터 수	연산횟수
1x1 Conv.	64	12,288(+64)	9.6M
3x3 Conv.	128	221,184(+128)	173.4M
5x5 Conv.	32	153,600(+32)	120.4M
3x3 Pool.	192		
계	256(출력 채널)	387,072(+224)	303.4M

\* 괄호 안의 수는 bias(뉴런)의 수



# 인셉션 모듈 구현

3절. CNN 아키텍처 > 3.4. GoogLeNet

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Conv2D
from tensorflow.keras.layers import MaxPooling2D, concatenate

model_input = Input(shape=(28,28,3))
pre = Dense(192)(model_input)
conv1 = Conv2D(64, 1, padding='same')(pre)
conv2 = Conv2D(128, 3, padding='same')(pre)
conv3 = Conv2D(32, 5, padding='same')(pre)
pool = MaxPooling2D(pool_size=(3,3), strides=1, padding='same')(pre)
model_output = concatenate([ conv1, conv2, conv3, pool])

model = Model(inputs=model_input, outputs=model_output)
model.summary()
```



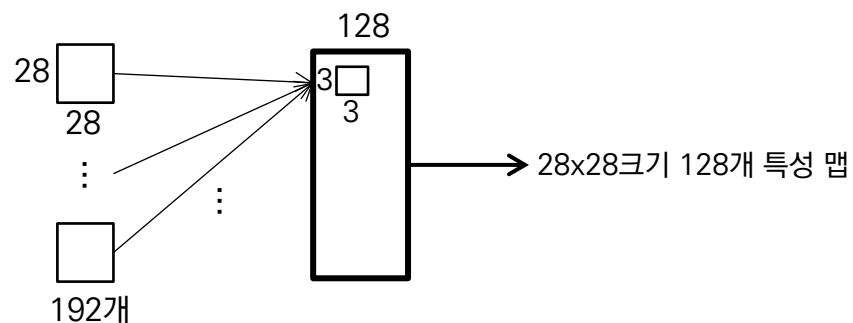
함수형 API를 사용해서  
구현할 수 있습니다.

Model: "model"

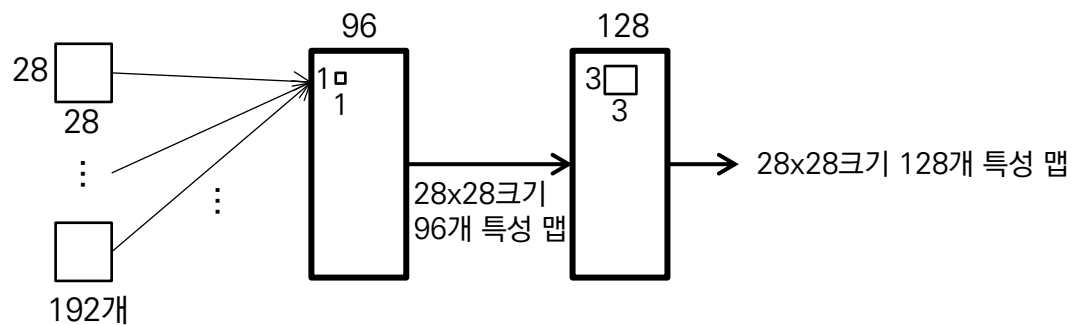
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 28, 28, 3)]	0	[]
dense (Dense)	(None, 28, 28, 192)	768	['input_1[0][0]']
conv2d (Conv2D)	(None, 28, 28, 64)	12352	['dense[0][0]']
conv2d_1 (Conv2D)	(None, 28, 28, 128)	221312	['dense[0][0]']
conv2d_2 (Conv2D)	(None, 28, 28, 32)	153632	['dense[0][0]']

# 1x1 컨볼루션층의 추가

3절. CNN 아키텍처 > 3.4. GoogLeNet



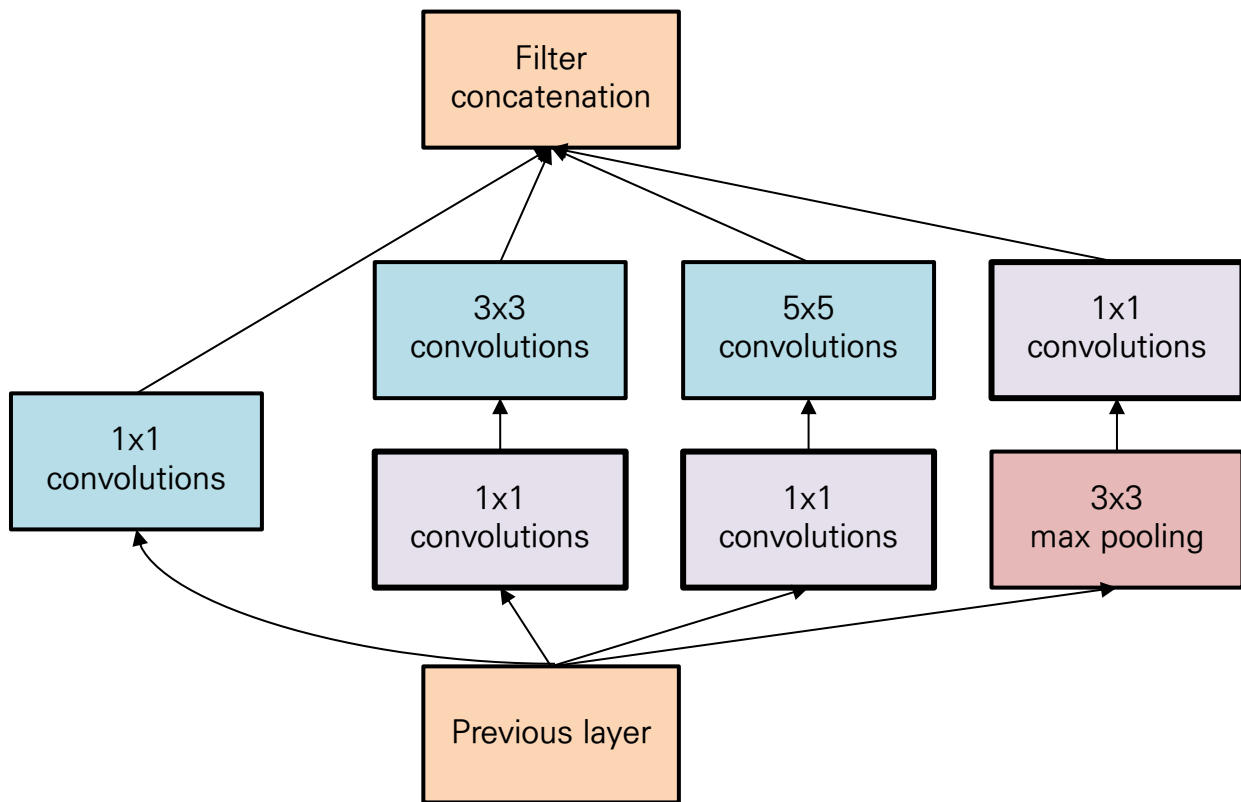
$$(28 \times 28 \times 192) \times (3 \times 3 \times 128) = 173,408,256$$



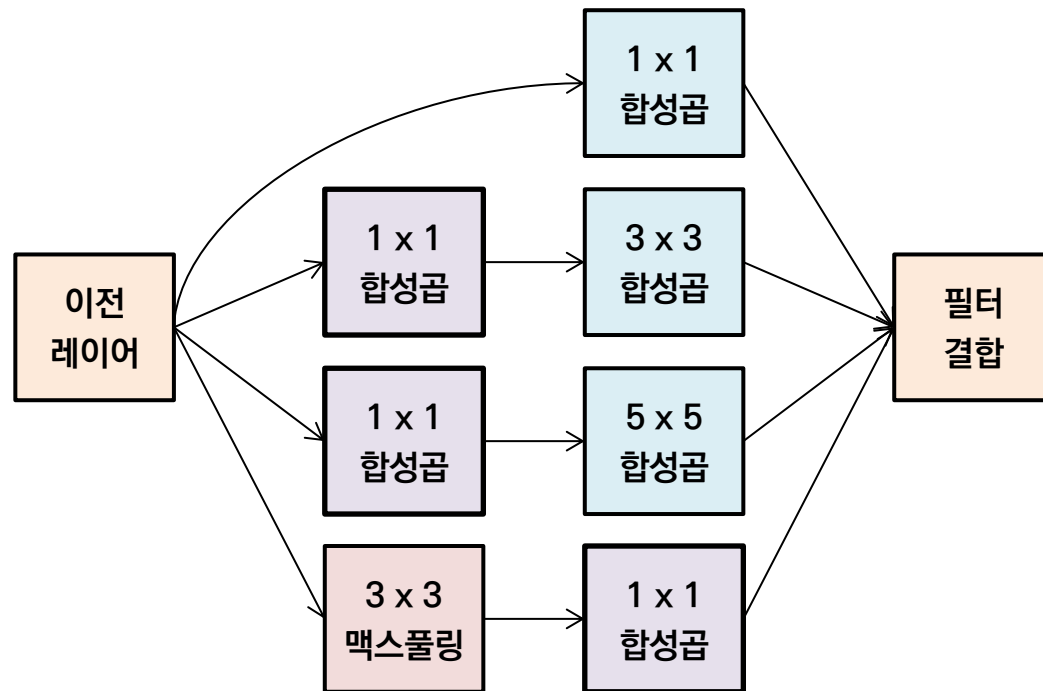
$$(28 \times 28 \times 192) \times (1 \times 1 \times 96) + (28 \times 28 \times 96) \times (3 \times 3 \times 128) = 14,450,688 + 86,704,128 = 101,154,816$$

# Inception 모듈, Bottle Neck 구조

3절. CNN 아키텍처 > 3.4. GoogLeNet

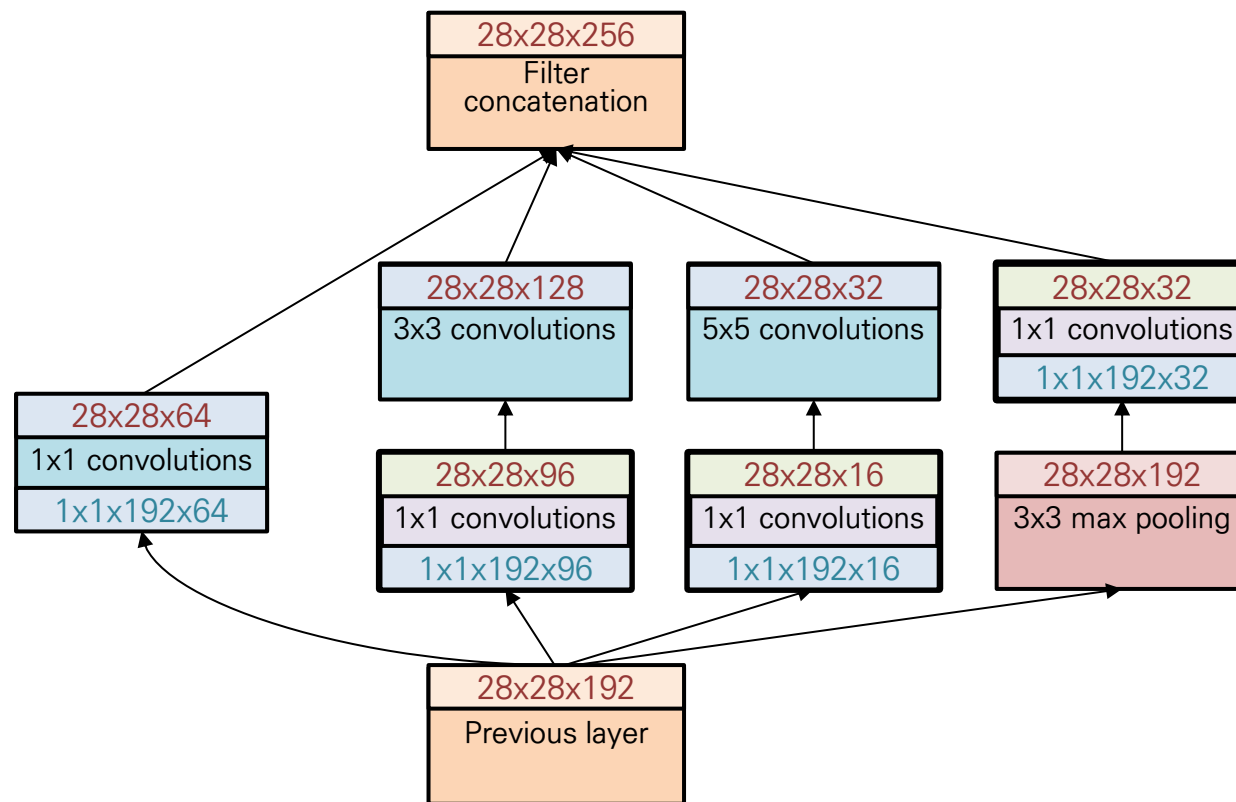


왼쪽 -> 오른쪽 방향으로 그리면...



# Inception 모듈, Bottle Neck구조

3절. CNN 아키텍처 > 3.4. GoogLeNet



# 나이프 버전과 보틀넥 구조 비교

3절. CNN 아키텍처 > 3.4. GoogLeNet

	naïve version			Bottle Neck 구조		
	채널 수	파라미터 수	연산횟수	채널 수	파라미터 수	연산횟수
1x1 conv	64	12,288(+64)	9.6M	64	12,288(+64)	9.6M
1x1 → 3x3				96	18,432(+96)	14.5M
3x3 conv	128	221,184(+128)	173.4M	128	110,592(+128)	86.7M
1x1 → 5x5				16	3,072(+16)	2.4M
5x5 conv	32	153,600(+32)	120.4M	32	12,800(+32)	10.0M
3x3 pool	192			192		
pool → 1x1				32	6,144(+32)	4.8M
계	256	387,072(+224)	303.4M	256	163,328(+368)	128.0M

# 1x1 Conv 층을 추가한 인셉션 모듈 구현

3절. CNN 아키텍처 > 3.4. GoogLeNet

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Conv2D
from tensorflow.keras.layers import MaxPooling2D, concatenate
```

```
model_input = Input(shape=(28,28,3))
pre = Dense(192)(model_input)
```

```
conv1 = Conv2D(64, 1, padding='same')(pre)
```

```
conv1_2 = Conv2D(96, 1, padding='same')(pre)
conv2 = Conv2D(128, 3, padding='same')(conv1_2)
```

```
conv1_3 = Conv2D(16, 1, padding='same')(pre)
conv3 = Conv2D(32, 5, padding='same')(conv1_3)
```

```
pool = MaxPooling2D(pool_size=(3,3), strides=1,
padding='same')(pre)
conv1_4 = Conv2D(32, 1, padding='same')(pool)
model_output = concatenate([ conv1, conv2, conv3, conv1_4])
```

```
model = Model(inputs=model_input, outputs=model_output)
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 28, 28, 3)]	0	[]
dense (Dense)	(None, 28, 28, 192)	768	['input_1[0][0]']
conv2d_1 (Conv2D)	(None, 28, 28, 96)	18528	['dense[0][0]']
conv2d_3 (Conv2D)	(None, 28, 28, 16)	3088	['dense[0][0]']
max_pooling2d (MaxPooling2D)	(None, 28, 28, 192)	0	['dense[0][0]']
conv2d (Conv2D)	(None, 28, 28, 64)	12352	['dense[0][0]']
conv2d_2 (Conv2D)	(None, 28, 28, 128)	110720	['conv2d_1[0][0]']
conv2d_4 (Conv2D)	(None, 28, 28, 32)	12832	['conv2d_3[0][0]']
conv2d_5 (Conv2D)	(None, 28, 28, 32)	6176	['max_pooling2d[0][0]']
concatenate (Concatenate)	(None, 28, 28, 256)	0	['conv2d[0][0]', 'conv2d_2[0][0]', 'conv2d_4[0][0]', 'conv2d_5[0][0]']

Total params: 164464 (642.44 KB)  
Trainable params: 164464 (642.44 KB)  
Non-trainable params: 0 (0.00 Byte)

# ResNet

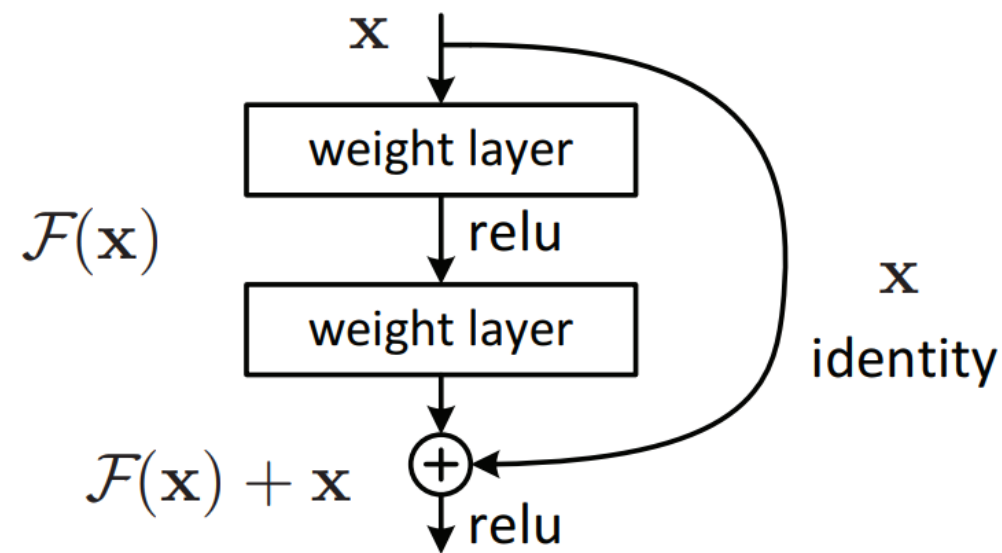
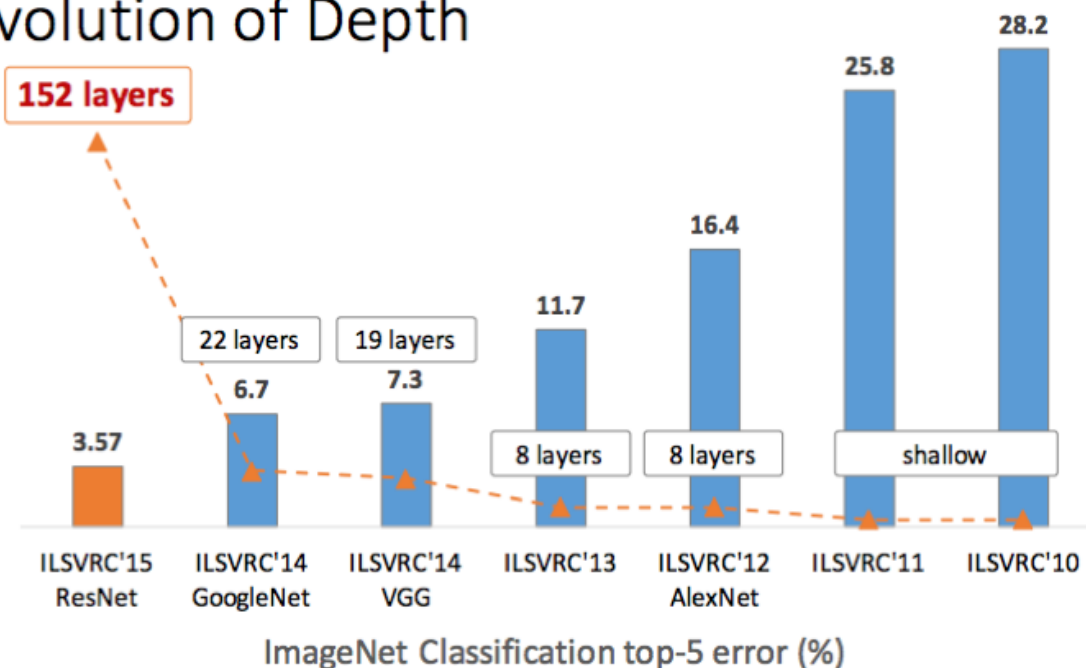
3절. CNN 아키텍처 > 3.5. ResNet

## 2015년 ILSVRC에서 우승

논문명 : Deep Residual Learning for Image Recognition

## 마이크로소프트에서 개발(Kamming He의 작품)

### Revolution of Depth

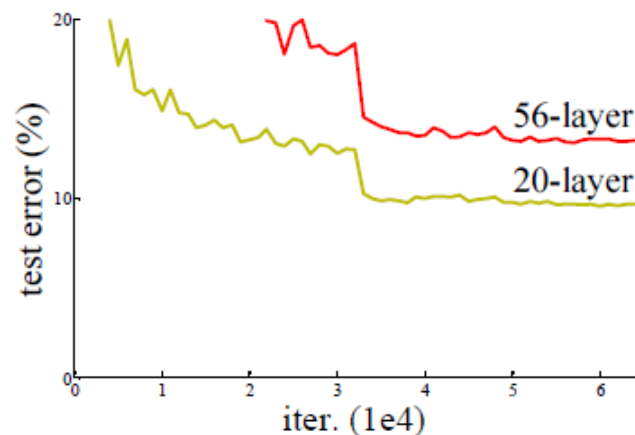
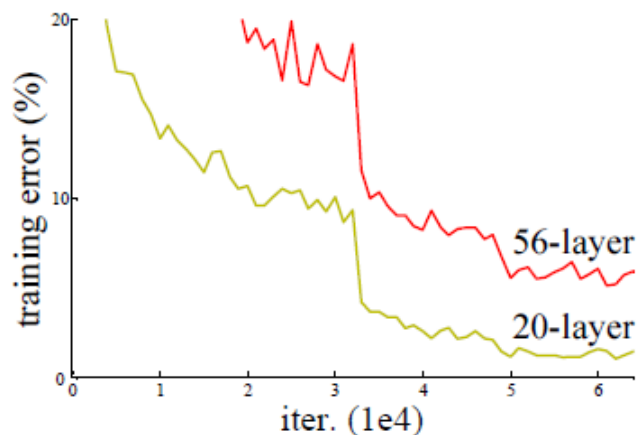




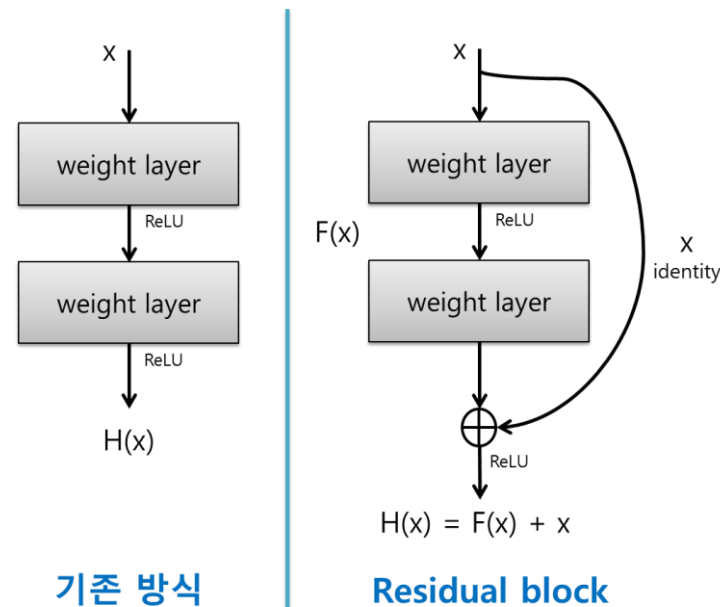
# ResNet

## 3절. CNN 아키텍처 > 3.5. ResNet

ResNet 저자들은 기존의 방식으로 망을 무조건 깊게 한다고 좋은 것이 아님을 증명

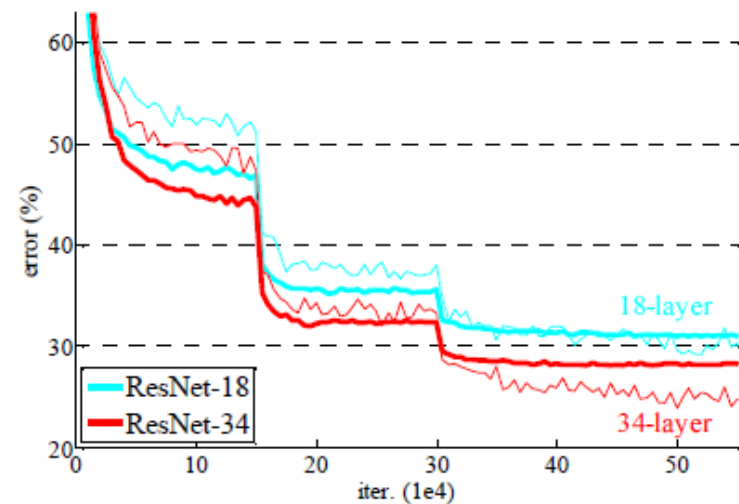
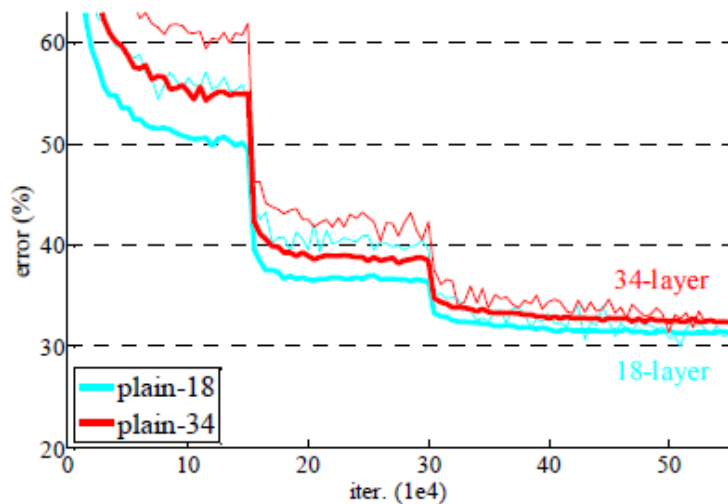
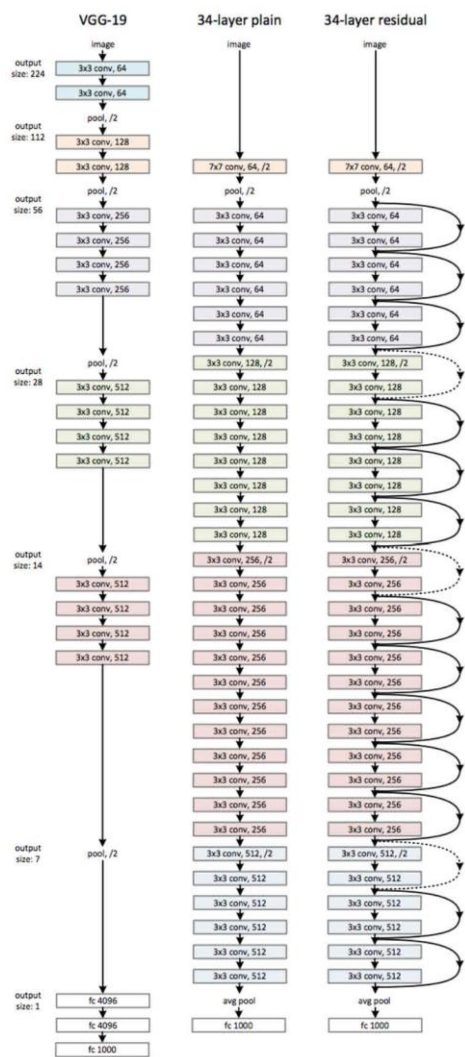


- 기존의 신경망은 입력값  $x$ 를 타겟값  $y$ 로 매핑하는 함수  $H(x)$ 를 얻는 것이 목적
- ResNet은  $F(x) + x$ 를 최소화하는 것을 목적으로 함



# ResNet

## 3절. CNN 아키텍처 > 3.5. ResNet

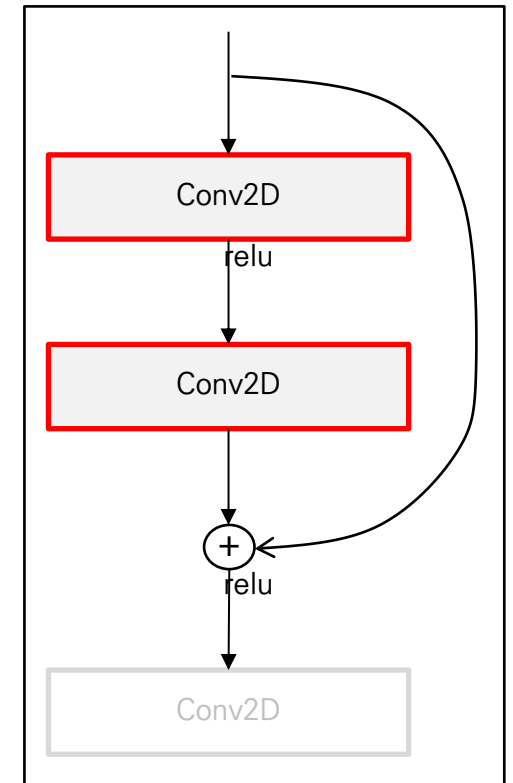


layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

# Residual block 구현

3절. CNN 아키텍처 > 3.5. ResNet

```
1. from tensorflow.keras.models import Model
2. from tensorflow.keras.layers import Input, Conv2D, Activation, add
3.
4. model_input = Input(shape=(28, 28, 3))
5.
6. x = Conv2D(128, 3, padding='same', activation='relu')(model_input)
7.
8. conv = Conv2D(64, 3, padding='same', activation='relu')(x)
9.
10. conv = Conv2D(128, 3, padding='same')(conv)
11.
12. y = add([conv, x])
13. y = Activation('relu')(y)
14. model = Model(inputs=model_input, outputs=y)
15. model.summary()
```



# Residual block 구현

3절. CNN 아키텍처 > 3.5. ResNet

Model: "functional\_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 28, 28, 3)	0	-
conv2d (Conv2D)	(None, 28, 28, 128)	3,584	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 28, 28, 64)	73,792	conv2d[0][0]
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856	conv2d_1[0][0]
add (Add)	(None, 28, 28, 128)	0	conv2d_2[0][0], conv2d[0][0]
activation (Activation)	(None, 28, 28, 128)	0	add[0][0]

Total params: 151,232 (590.75 KB)

Trainable params: 151,232 (590.75 KB)

Non-trainable params: 0 (0.00 B)

## 4절. 케라스의 모델

1장. 합성곱 신경망



# 케라스에서 사용 가능 한 모델

## 4절. 케라스의 모델

- 케라스 어플리케이션은 선행학습된 가중치와 함께 사용할 수 있도록 한 딥러닝 모델
- 이 모델로 예측, 특성추출, 파인튜닝을 할 수 있음
- 가중치는 모델을 인스턴스화 할 때 자동으로 다운로드되어 ~/.keras/models/에 저장

모델	설명	input size
Xception	ImageNet에 대해 가중치가 선행학습된 Xception V1 모델	299x299
VGG16	ImageNet에 대해 가중치가 선행학습된 VGG16 모델	224x224
VGG19	ImageNet에 대해 가중치가 선행학습된 VGG19 모델.	224x224
ResNet, ResNetV2, ResNeXt	ImageNet에 대해 가중치가 선행학습된 ResNet 모델, ResNetV2 모델, ResNeXt 모델.(ResNet50, ResNet101, ResNet152, ResNet50V2, ResNet50V2, ResNet101V2, ResNet152V2, ResNeXt50, ResNeXt101)	224x224
InceptionV3	ImageNet에 대해 가중치가 선행학습된 Inception V3.	299x299
InceptionResNetV2	ImageNet에 대해 가중치가 선행학습된 Inception-ResNet V2 모델.	299x299
MobileNet	ImageNet에 대해 가중치가 선행학습된 MobileNet 모델.	224x224
MobileNetV2	ImageNet에 대해 가중치가 선행학습된 MobileNetV2 model.	224x224
DenseNet	ImageNet에 대해 가중치가 선행학습된 DenseNet.	224x224
NASNet	ImageNet에 대해 가중치가 선행학습된 Neural Architecture Search Network (NASNet) 모델	NASNetLarge: 331x331 NASNetMobile: 224x224

# ResNet50 불러와 예측하기

4절. 케라스의 모델

## 클래스의 객체를 생성하고 이미지를 이용해서 predict() 수행

```
1. from tensorflow.keras.applications import ResNet50
2. from tensorflow.keras.preprocessing import image
3. from tensorflow.keras.applications.resnet50 import decode_predictions
4. import numpy as np
5.
6. model = ResNet50(weights='imagenet') # 첫 실행 시 가중치 다운로드됨
7.
8. # https://commons.wikimedia.org/wiki/File:YellowLabradorLooking_new.jpg
9. img = image.load_img('YellowLabradorLooking_new.jpg', target_size=(224, 224))
10. x = image.img_to_array(img) # x.shape=(224,224,3)
11. x = np.expand_dims(x, axis=0) # x.shape=(1,244,244,3)
12. pred = model.predict(x, verbose=0)
13. print('Predicted:', decode_predictions(pred, top=3))
```

내려받은 모델은 .keras/models/ 폴더 아래에 저장됨

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
102967424/102967424 [=====] - 3s 0us/step
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
35363/35363 [=====] - 0s 0us/step
Predicted: [[('n02099712', 'Labrador_retriever', 0.26897743), ('n02108089', 'boxer', 0.15149544), ('n02099849', 'Chesapeake_Bay_retriever', 0.10756537)]]
```

1000개 클래스 레이블은 .keras/models/ 폴더 아래에 imagenet\_class\_index.json 파일로 저장됨

### 케라스의 CNN 모델을 불러와 새로운 모델에 적용할 수 있음

- CNN 클래스를 이용해서 객체 생성시 `include_top=False` 속성을 추가하고, trainable 속성을 True로 변경

최상위 층(출력층)을 제외함

```
1. from tensorflow.keras.applications import ResNet50
2. from tensorflow.keras.models import Sequential
3. from tensorflow.keras.layers import Dense, Flatten
4.
   resnet_model = ResNet50(input_shape=(224,224,3), include_top=False)
5. resnet_model.trainable = True
6.
   model = Sequential()
7. model.add(resnet_model)
8. model.add(Flatten())
9. model.add(Dense(1024, activation='relu')) # FC 층 추가
10. model.add(Dense(100, activation='softmax')) # 100 클래스 분류
11. model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23587712
flatten (Flatten)	(None, 100352)	0
dense (Dense)	(None, 1024)	102761472
dense_1 (Dense)	(None, 100)	102500

Total params: 126451684 (482.37 MB)  
Trainable params: 126398564 (482.17 MB)  
Non-trainable params: 53120 (207.50 KB)

Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_notop.h5](https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5)  
94765736/94765736 [=====] - 3s 0us/step