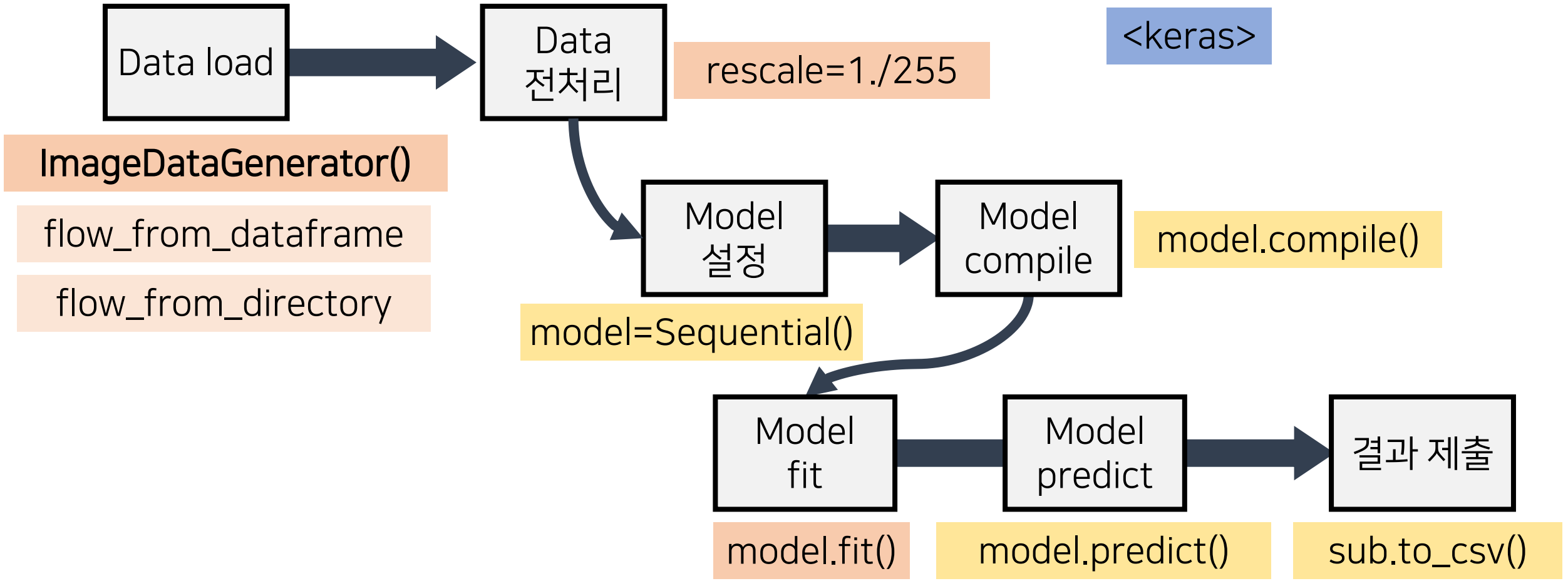
Contents

6장 이미지 분류

이미지 분류 과정


Data load
Data
전처리
rescale=1./255
<keras>
ImageDataGenerator()
flow_from_dataframe
flow_from_directory
Model
설정
Model
compile
model.compile()
model=Sequential()
Model
fit
Model
predict
결과 제출
model.fit()
model.predict()
sub.to_csv()

Contents

6장 이미지 분류

ImageDataGenerator

<keras>

ImageDataGenerator

이미지 파일 불러오기

https://keras.io/ko/preprocessing/image/

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
datagen = ImageDataGenerator(rescale=1./255)
```

flow_from_dataframe

flow_from_directory

```
model.fit(train, epochs= 5)
```

flow_from_dataframe

```
datagen = ImageDataGenerator(rescale=1./255)
```

```
train_data = datagen.flow_from_dataframe(
    train, directory=train_dir,
    x_col=train.columns[0],
    y_col=train.columns[1],
    target_size=(256, 256),
    class_mode="categorical",
    batch_size=32)
```

〈keras〉

```
train=pd.DataFrame()
```

	x	y
0	0.png	4
1	1.Png	2
2	2.png	1

class_mode

class_mode

다중분류(one-hot): "categorical"

이진분류: "binary "

다중분류: "sparse "

np array : " raw "

autoencoder : "input"

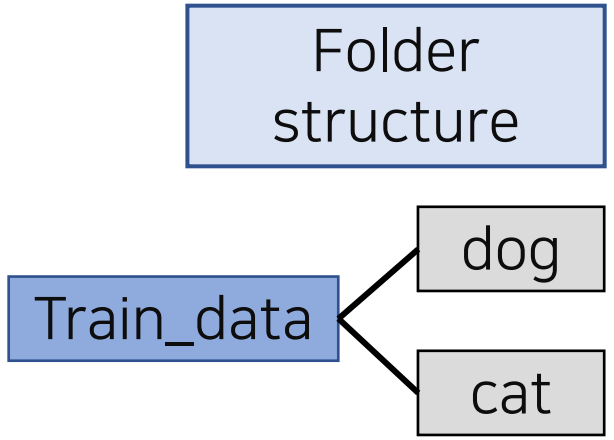
"other" 혹은 None 중 하나.

디폴트 값: "categorical"

flow_from_directory

```
datagen = ImageDataGenerator(rescale=1./255)
```

```
train_data = datagen.flow_from_directory(
    './train',
    target_size = (256, 256),
    batch_size = 32,
    class_mode = 'binary'
)
```



Contents

6장 이미지 분류

CNN 과정

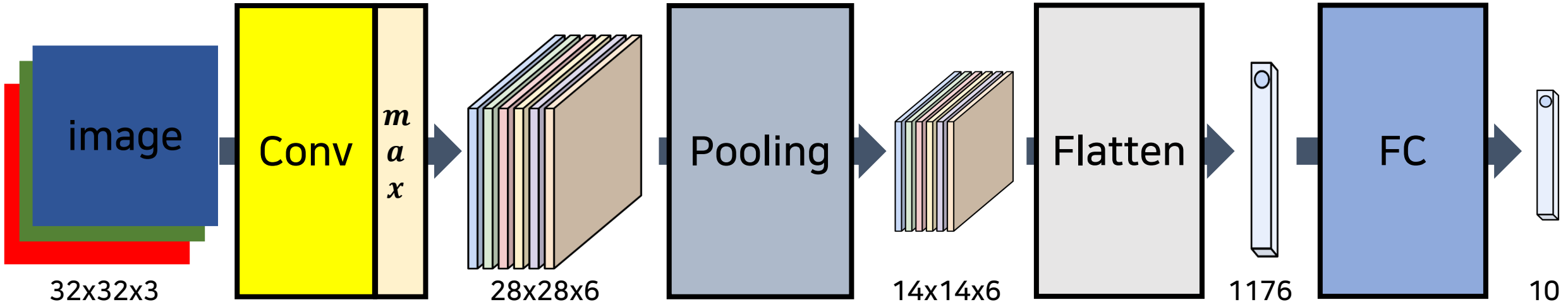
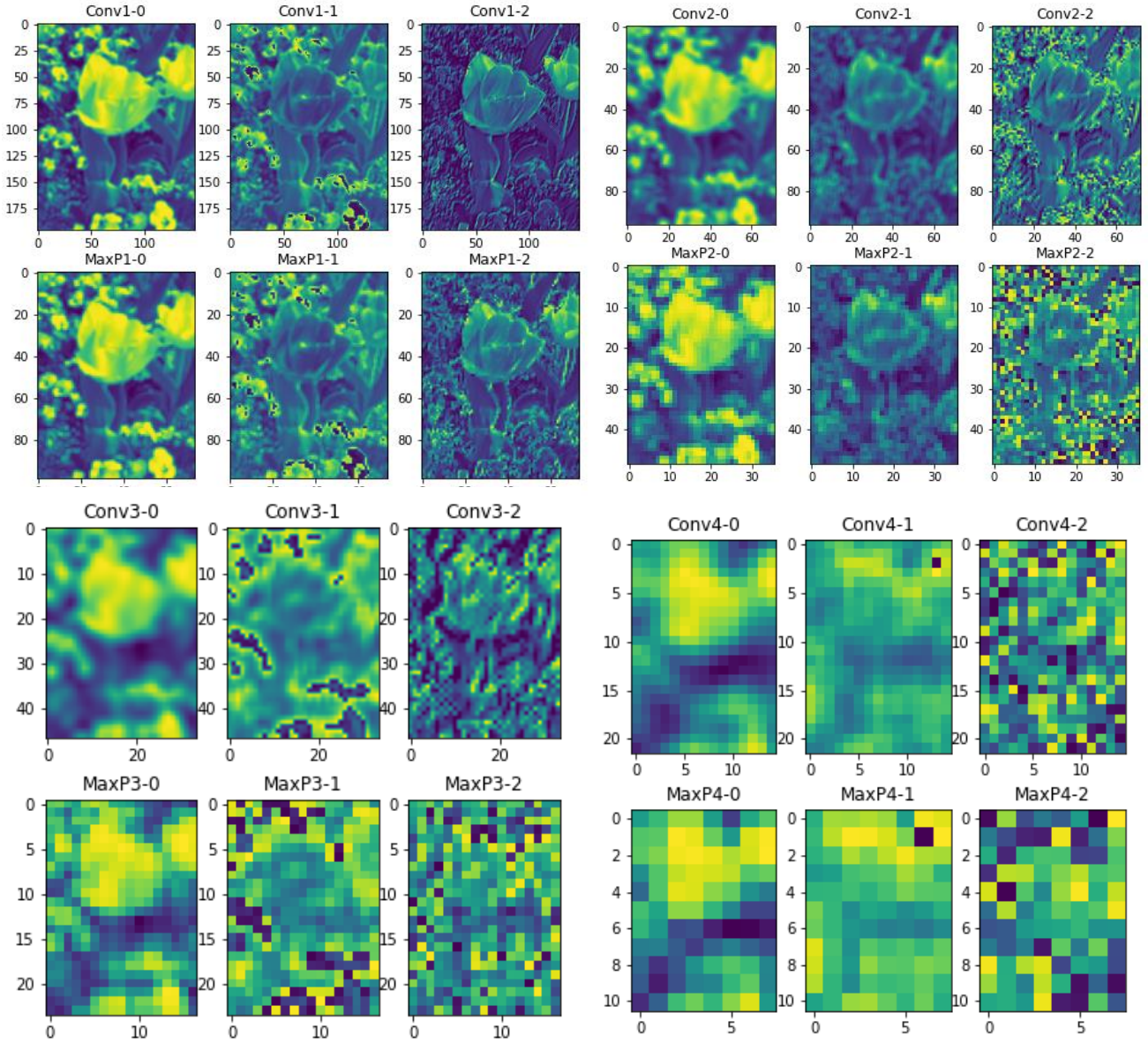


image
Conv
max
Pooling
Flatten
FC
32x32x3
28x28x6
14x14x6
1176
10

CNN 특징

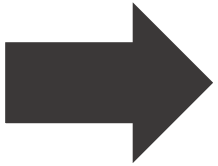

Conv1-0
Conv1-1
Conv1-2
Conv2-0
Conv2-1
Conv2-2
MaxP1-0
MaxP1-1
MaxP1-2
MaxP2-0
MaxP2-1
MaxP2-2
Conv3-0
Conv3-1
Conv3-2
Conv4-0
Conv4-1
Conv4-2
MaxP3-0
MaxP3-1
MaxP3-2
MaxP4-0
MaxP4-1
MaxP4-2

CNN 특징

첫번째 층

에지 감지기

초기 사진에 있는 거의 모든 정보가 유지



상위 층

추상적, 시각적 이해 어려움

'고양이 귀'와 '고양이 눈'과 같이 고수준의 개념을 인코딩

시각적 콘텐츠에 관한 정보가 점점 줄어들고 이미지의 클래스에 관한 정보가 점점 증가

비어 있는 활성화가 층이 깊어짐에 따라 늘어남
활성화 되지 않는 필터 증가

Contents

6장 이미지 분류

CNN 구성요소

Activation

Convolution Layer

Pooling Layer

FC Layer

Filter
Stride
Padding

Max pooling
GAP

1. Convolution 층

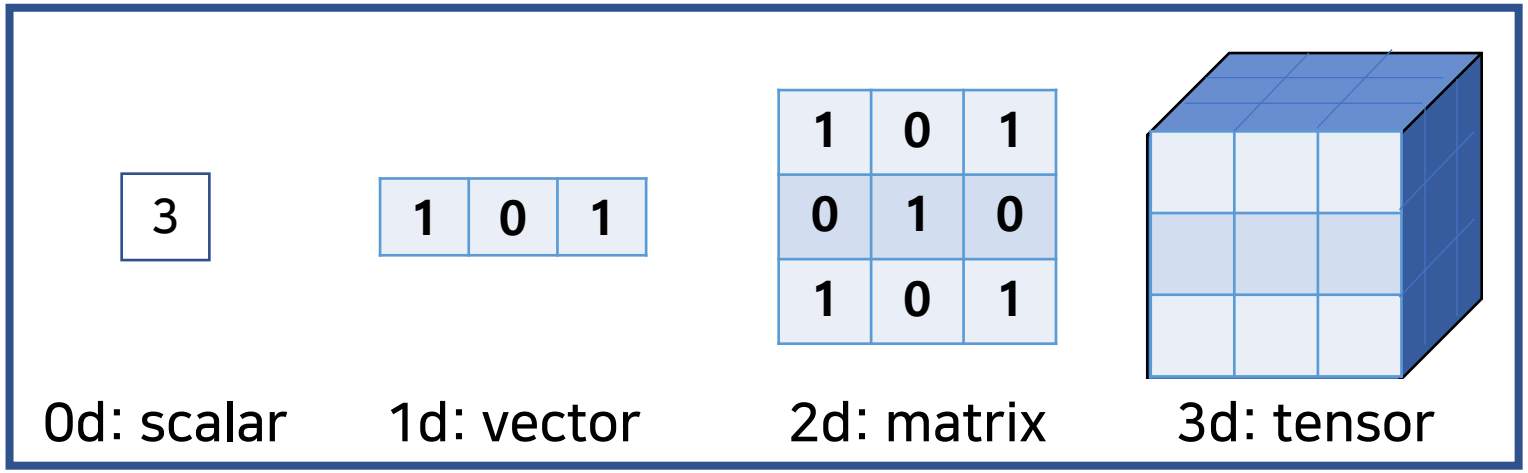
Convolution Layer

Convolution

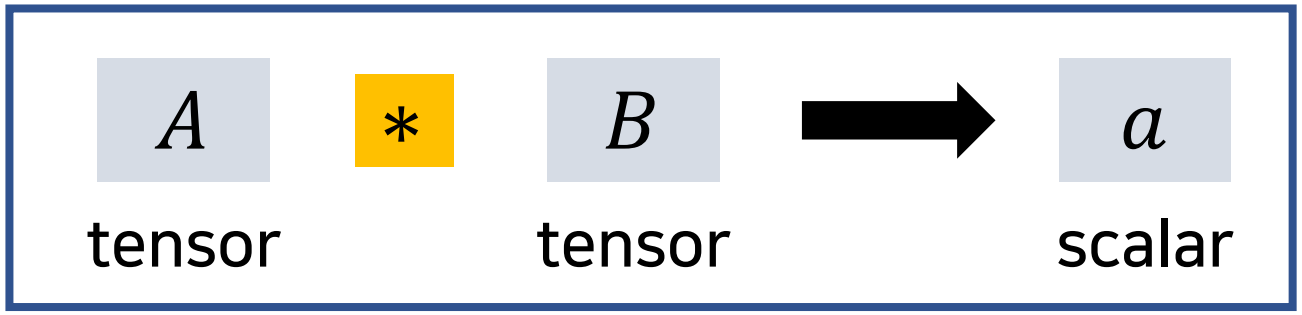
Filter
Stride
Padding

Convolution 정의

Tensor



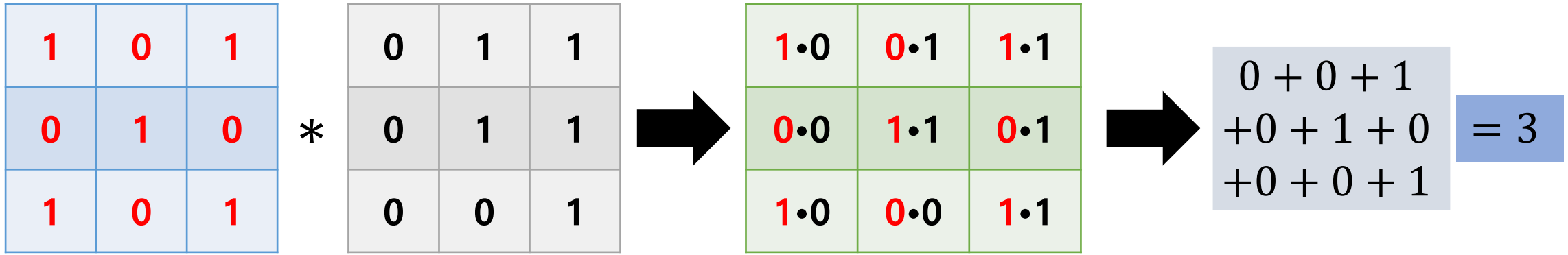
Convolution



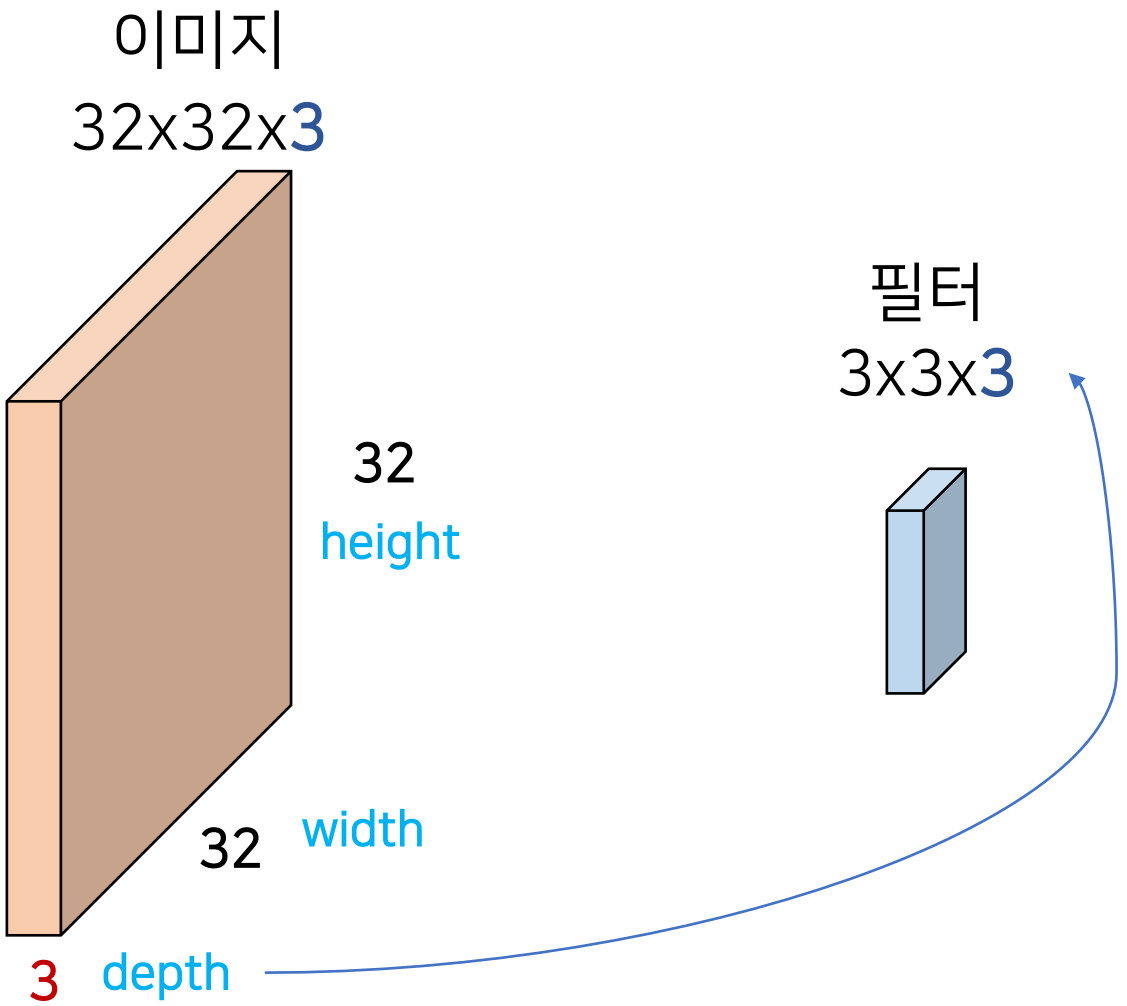
Convolution 계산

1D Convolution

2D Convolution

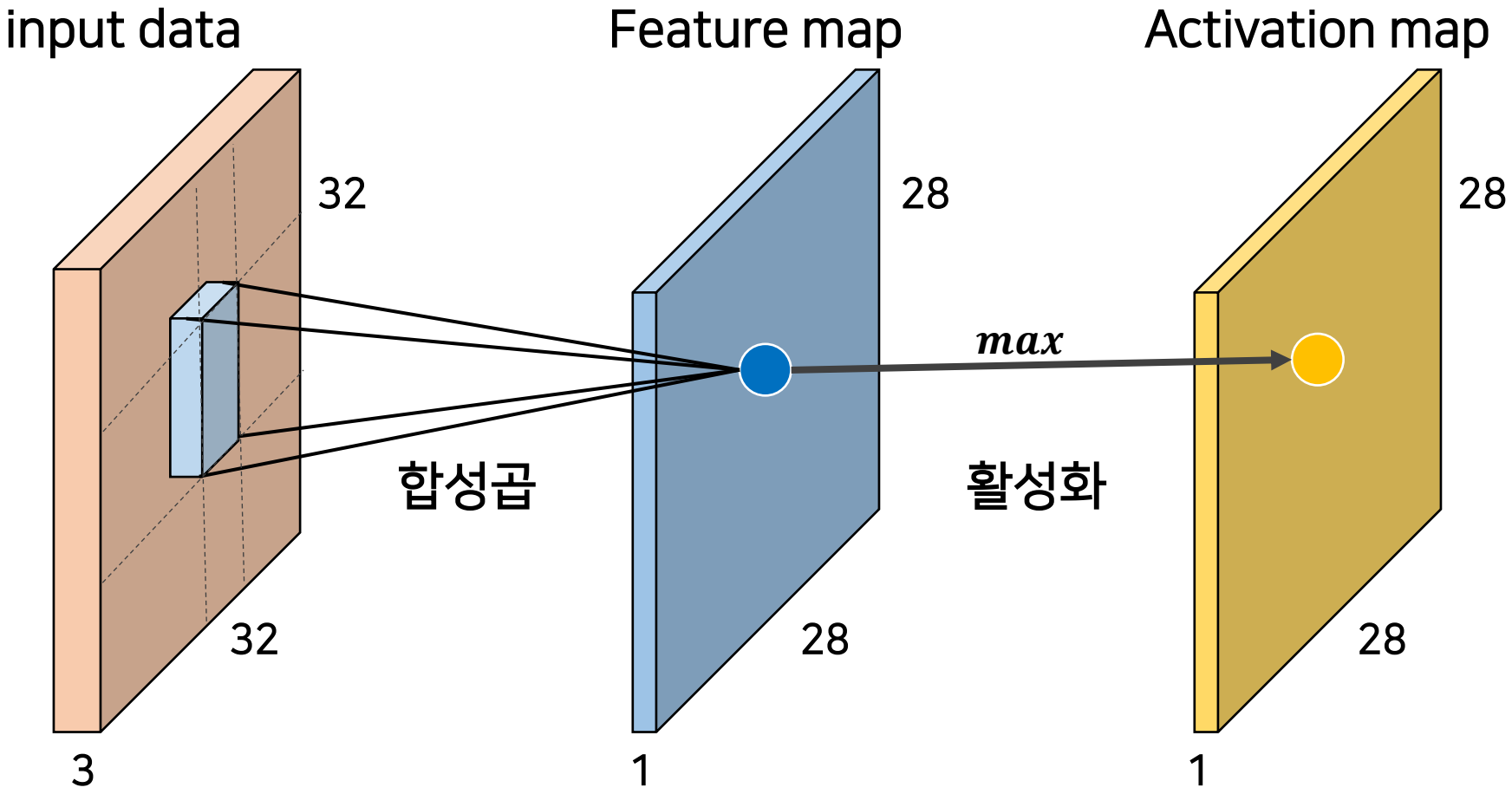


Convolution 층 : Filter

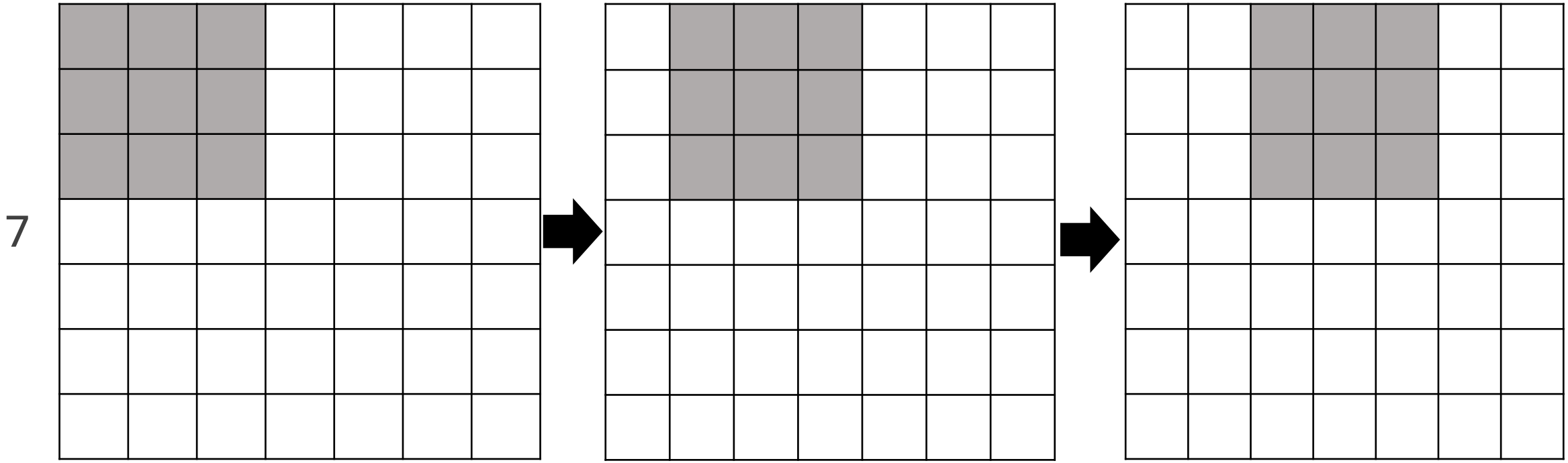


필터와 이미지의 depth는 항상 일치

Convolution 층 : 활성화



Stride

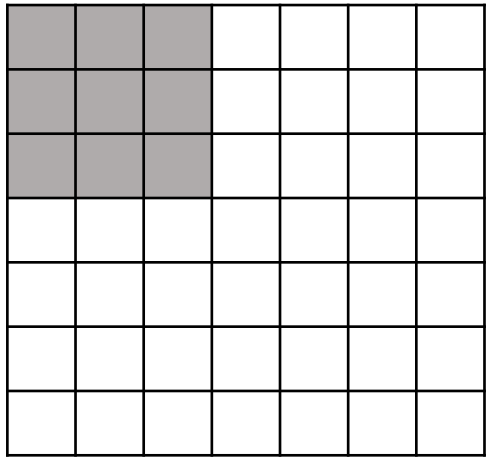


입력: 7x7

필터: 3x3

Stride 1

Zero pad

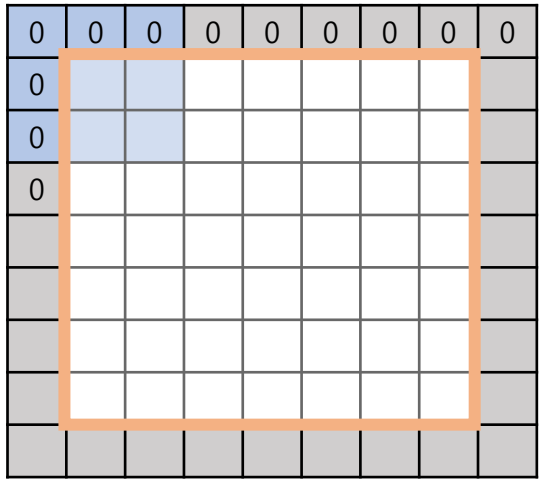


출력: 5x5

필터: 3x3

입력: 7x7

Pad 1



출력: 7x7

필터: 3x3

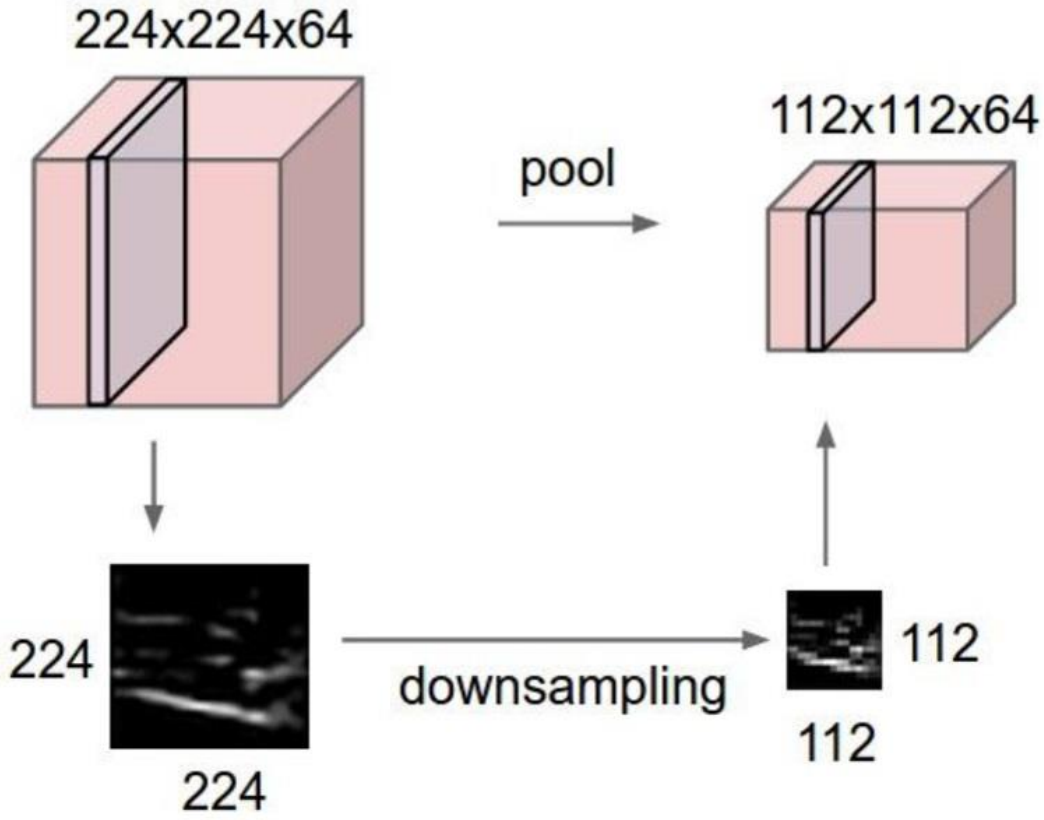
입력: 7x7

2. Pooling 층

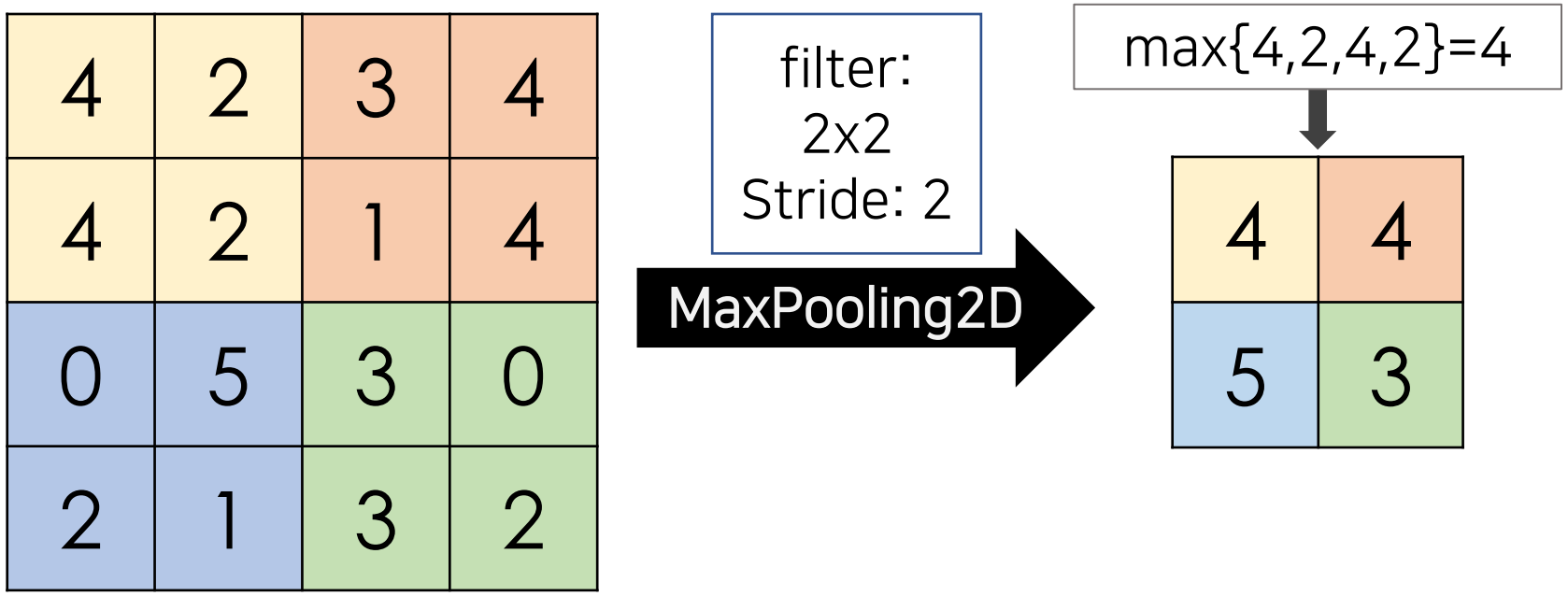
이미지 끌어 당기기

종류

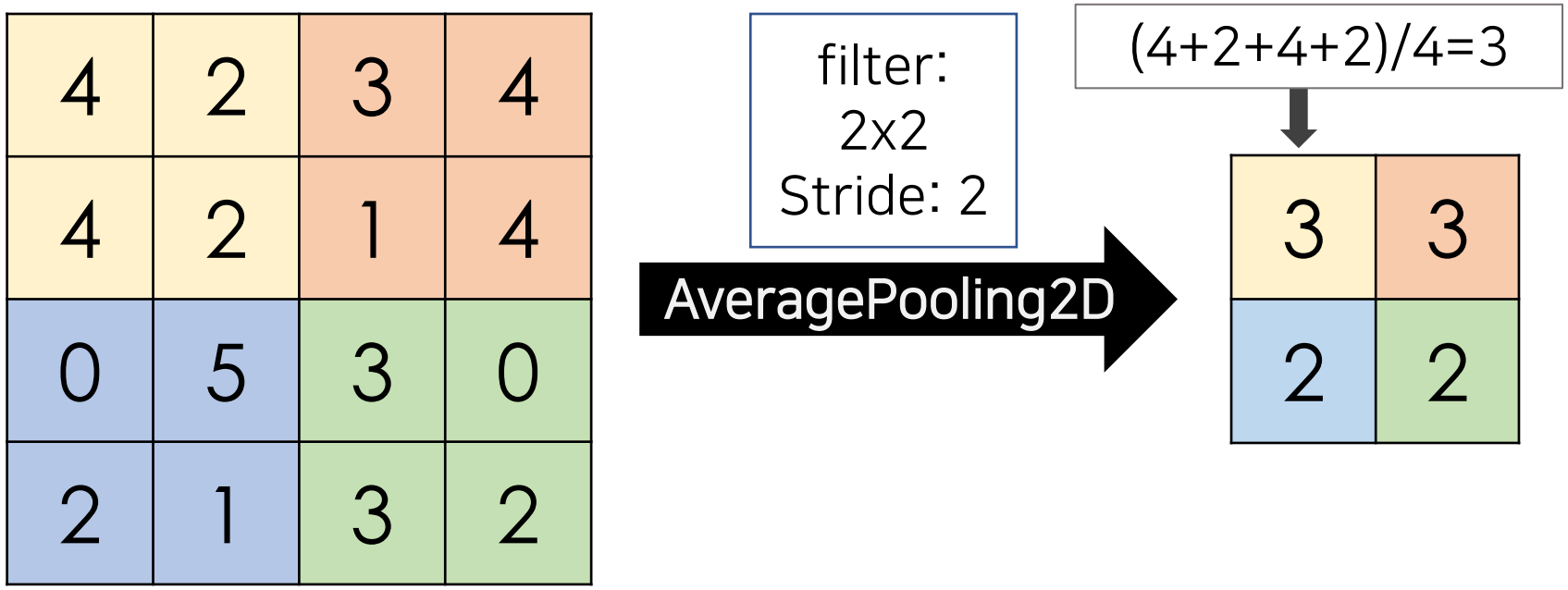
Max Pooling
Average Pooling
GAP



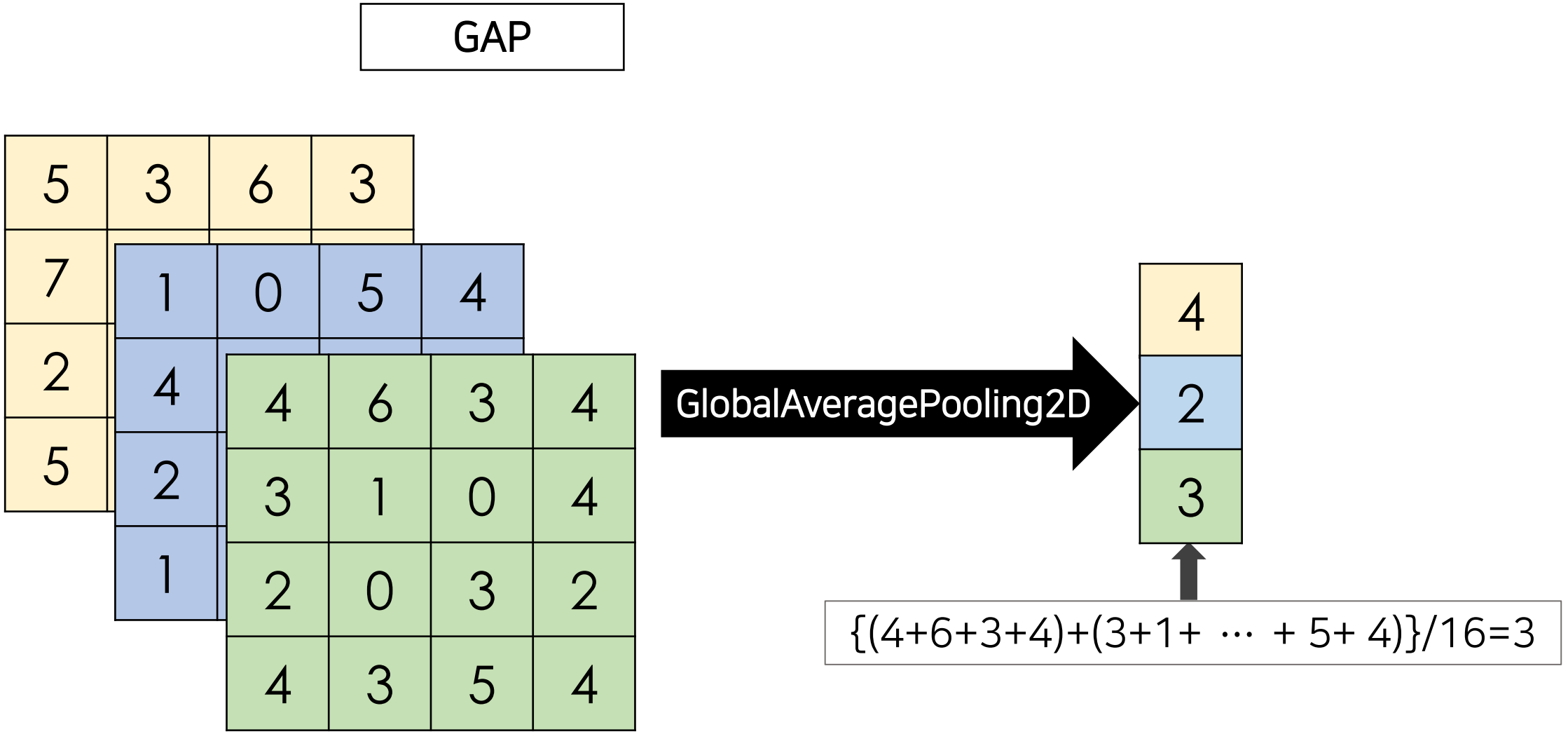
MaxPooling2D



AveragePooling2D



GlobalAveragePooling2D

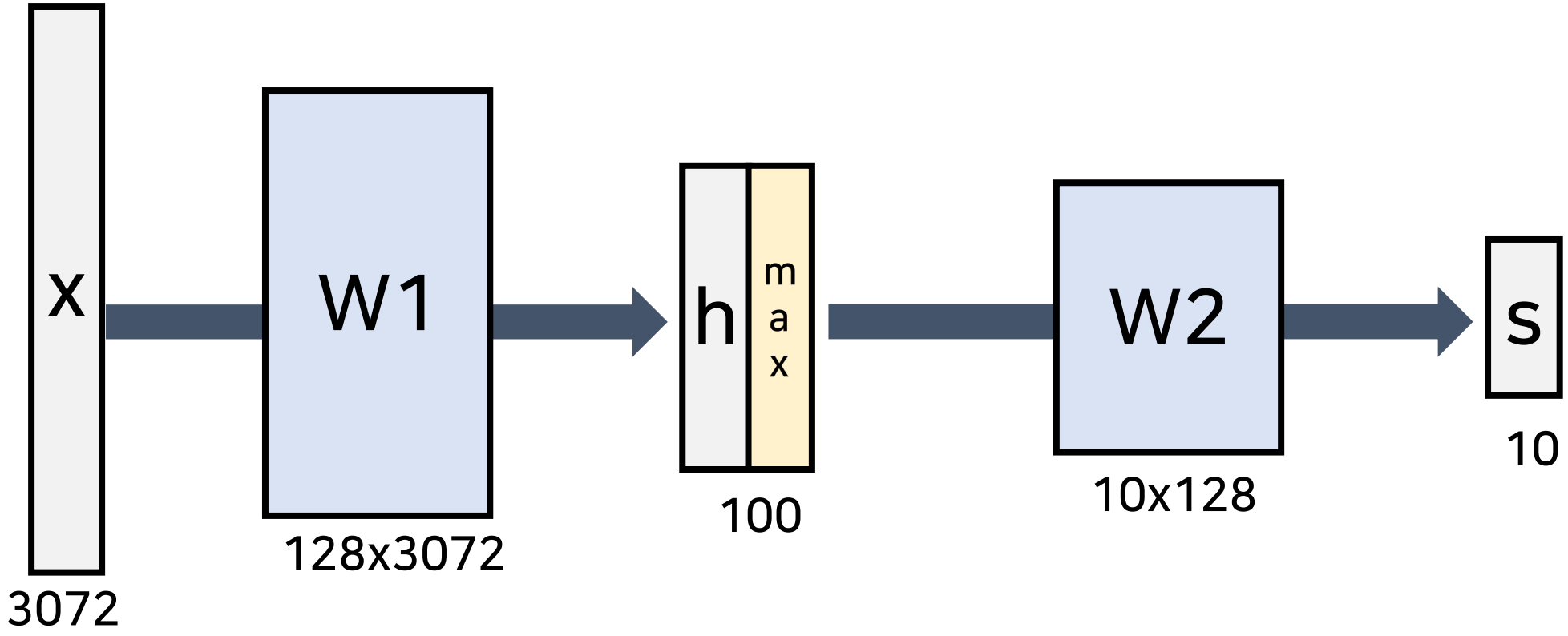


3. Fully Connected 층

신경망
2-layer

$$f = W_2 \max(W_1 x, 0)$$

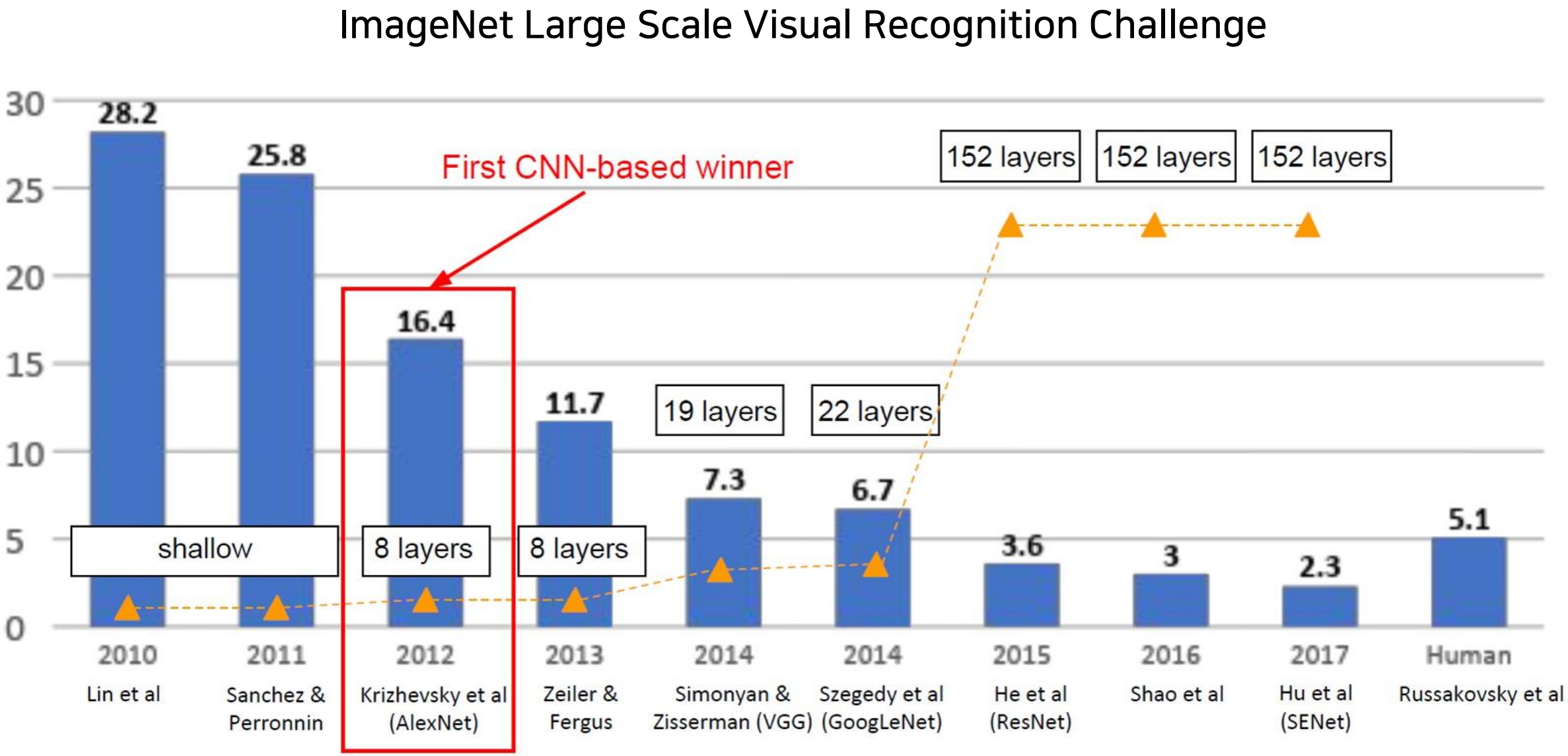
$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D},$$
$$W_2 \in \mathbb{R}^{C \times H}$$



Contents

6장 이미지 분류

ImageNet


ImageNet Large Scale Visual Recognition Challenge
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
First CNN-based winner
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
152 layers
152 layers
2010
Lin et al
2011
Sanchez & Perronnin
2012
Krizhevsky et al (AlexNet)
2013
Zeiler & Fergus
2014
Simonyan & Zisserman (VGG)
2014
Szegedy et al (GoogLeNet)
2015
He et al (ResNet)
2016
Shao et al
2017
Hu et al (SENet)
Human
Russakovsky et al

AlexNet

Convolution

1층	11 x11 x 3	96
2층	5 x 5 x 48	256
3층	3 x 3 x 256	384
4층	3 x 3 x 192	384
5층	3 x 3 x 192	256

FC

6 x 6 x 256 특성맵 flatten
6 x 6 x 256 = 9216차원의 벡터

6층	9216
7층	4096
8층	1000

Output

결과 : 1000개 클래스 분류

Loss Function

softmax

VGGNet

Convolution

층	필터크기	개수
1층(conv1_1)	3 x 3	64
2층(conv1_2)	3 x 3	64
3층(conv2_1)	3 x 3	128
4층(conv2_2)	3 x 3	128
5층(conv3_1)	3 x 3	256
6층(conv3_2)	3 x 3	256
7층(conv3_3)	3 x 3	256
8층(conv4_1)	3 x 3	512
9층(conv4_2)	3 x 3	512
10층(conv4_3)	3 x 3	512
11층(conv5_1)	3 x 3	512
12층(conv5_2)	3 x 3	512
13층(conv5_3)	3 x 3	512

FC

특성맵 flatten

7 x 7 x 512 = 25088개의 뉴런

14층(fc1)	25088
15층(fc2)	4096
16층(fc3)	1000

Output

Loss Function

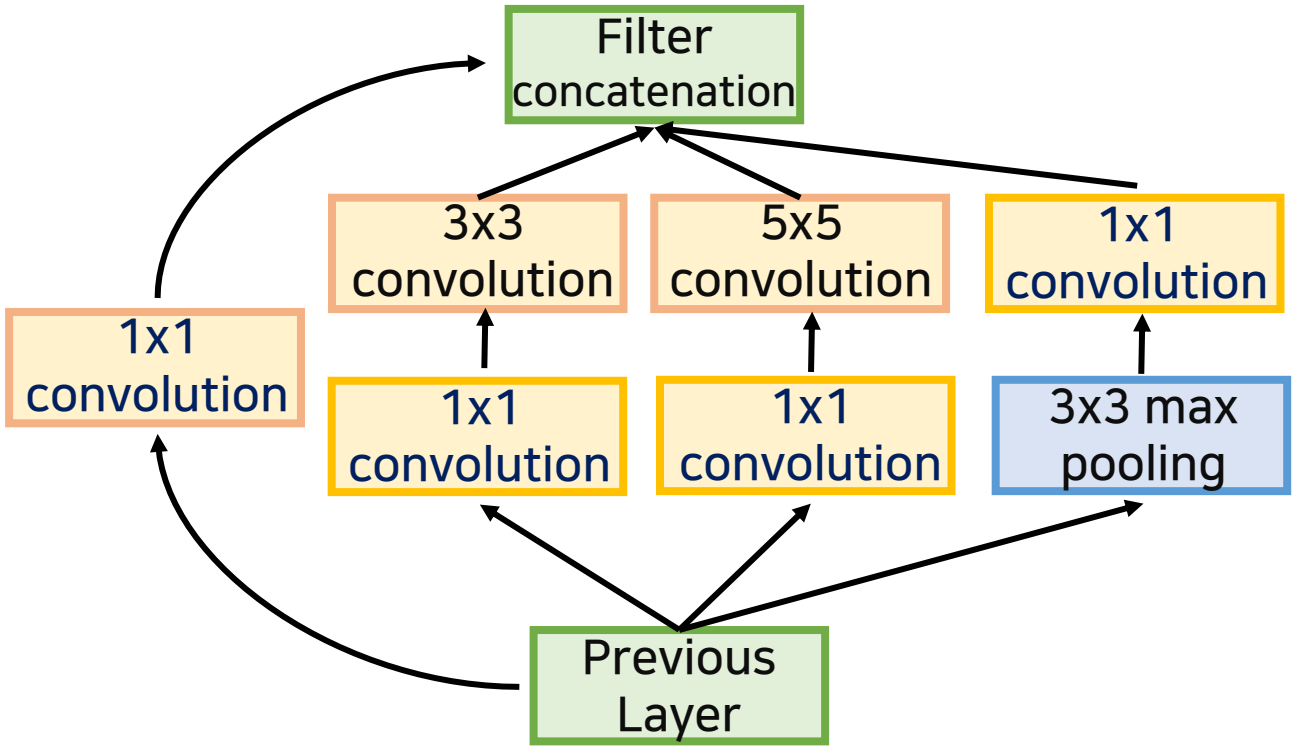
softmax

결과 : 1000개 클래스 분류

[Simonyan and Zisserman, 2014]

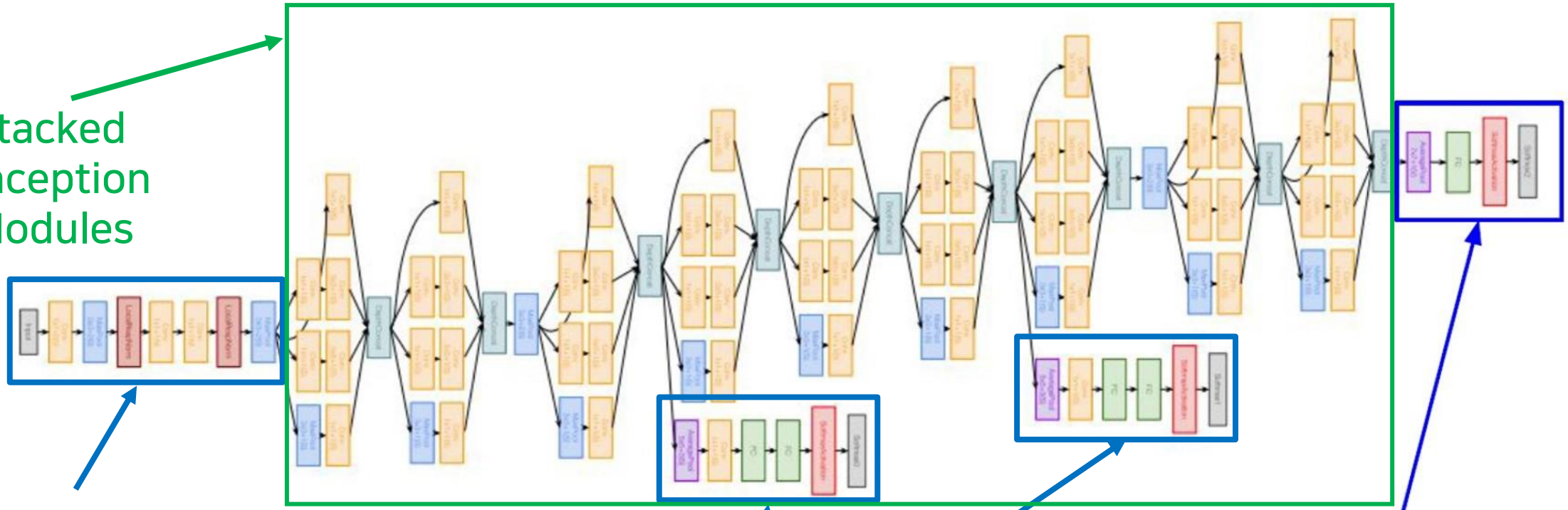
GoogLeNet : Inception Module

With Dimension Reduction



과도한 계산량을 줄이기 위해
1x1 Conv 사용

GoogLeNet



Stacked Inception Modules

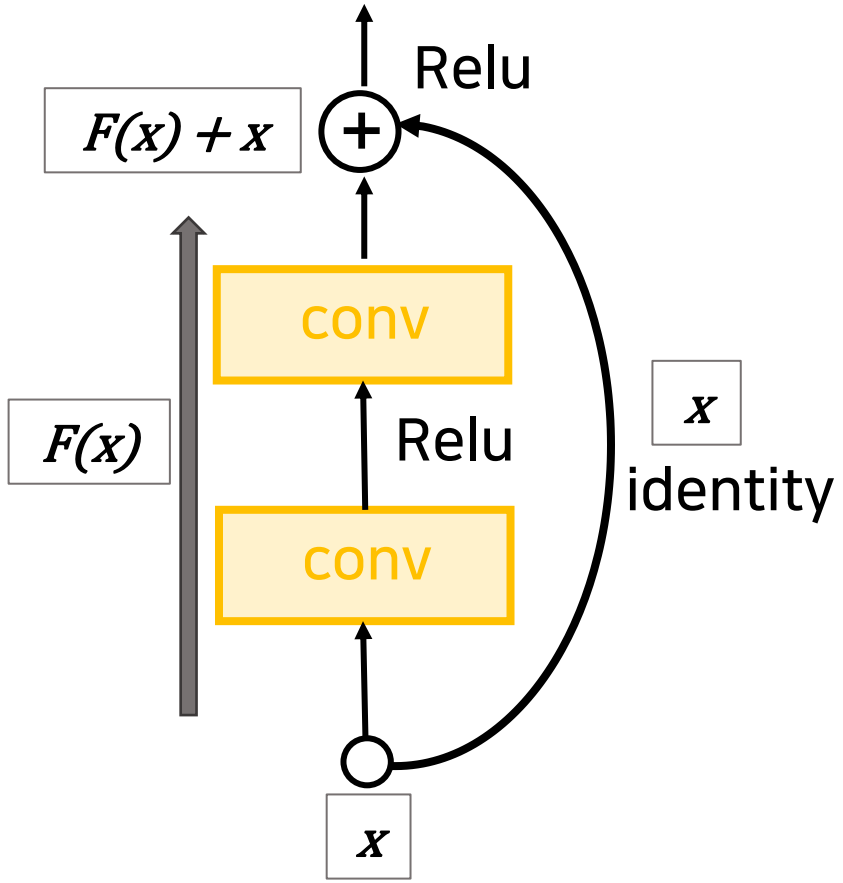
Stem Network: Conv-Pool-2xConv-Pool

Auxiliary classification 출력
낮은 층에 추가적인 미분 삽입
AvgPool-1x1Conv-FC-FC-Softmax

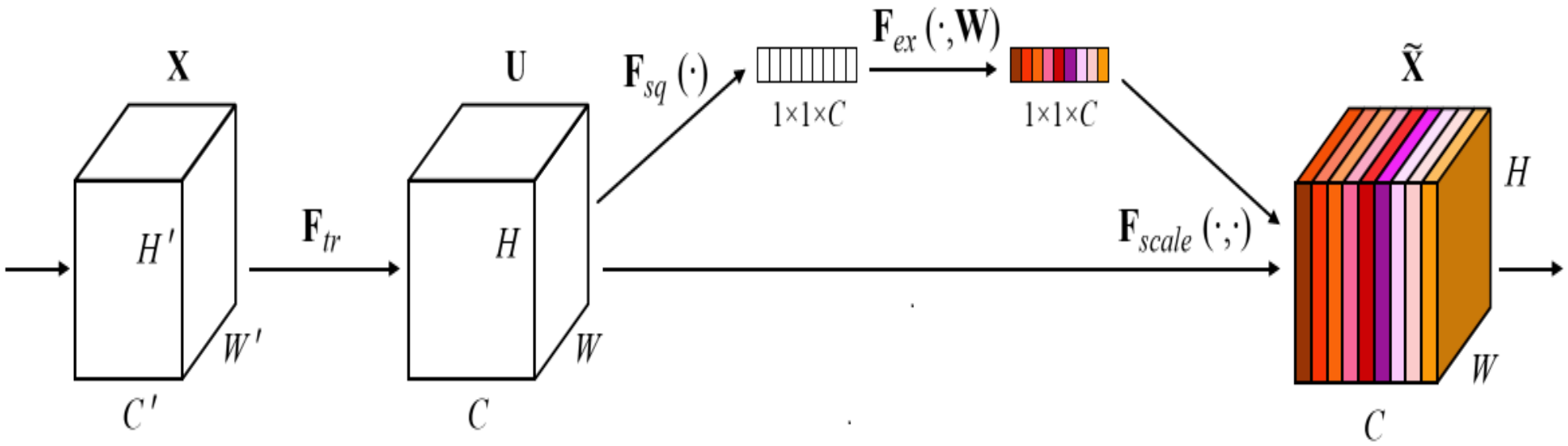
Classifier Output

ResNet

Residual block



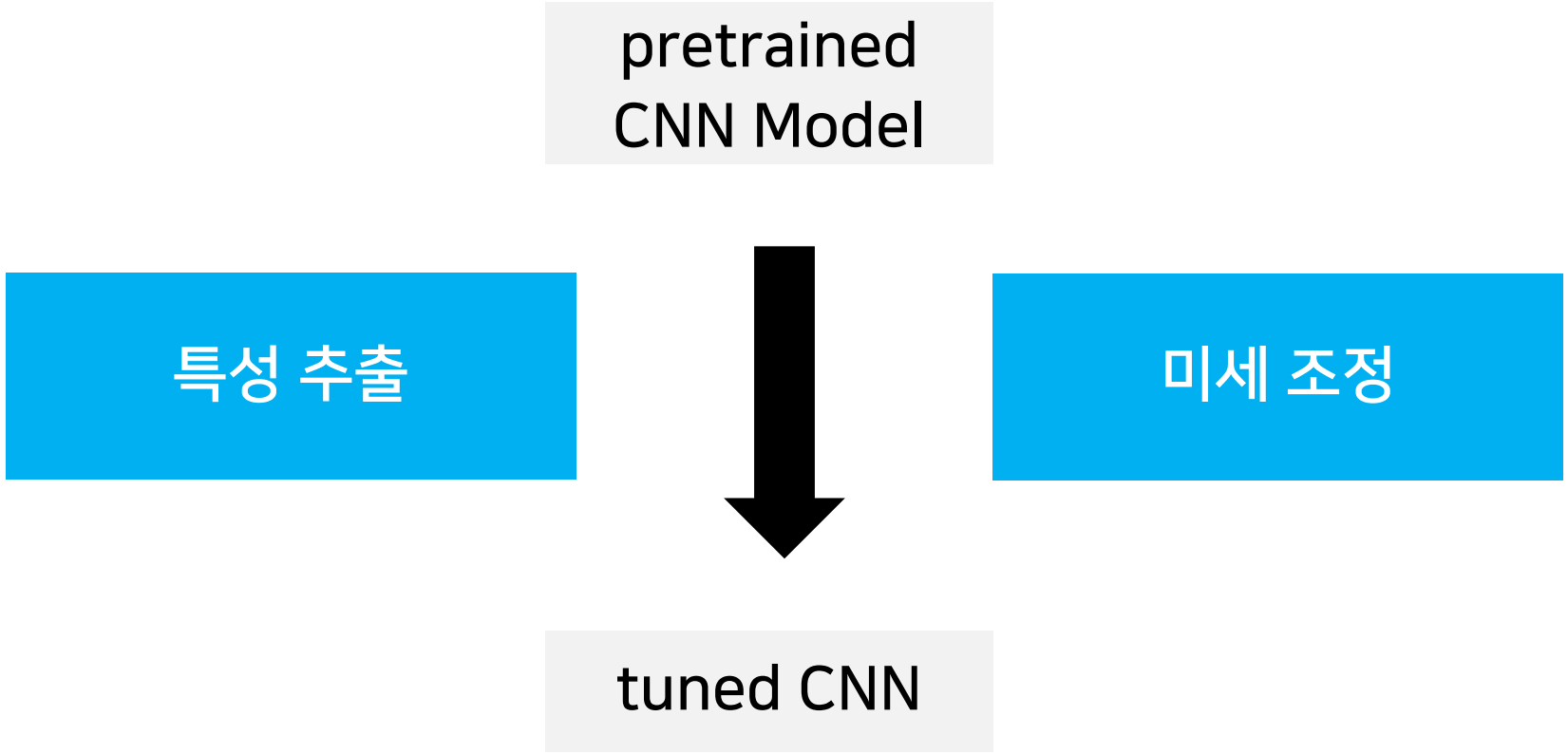
SENet：SE Block



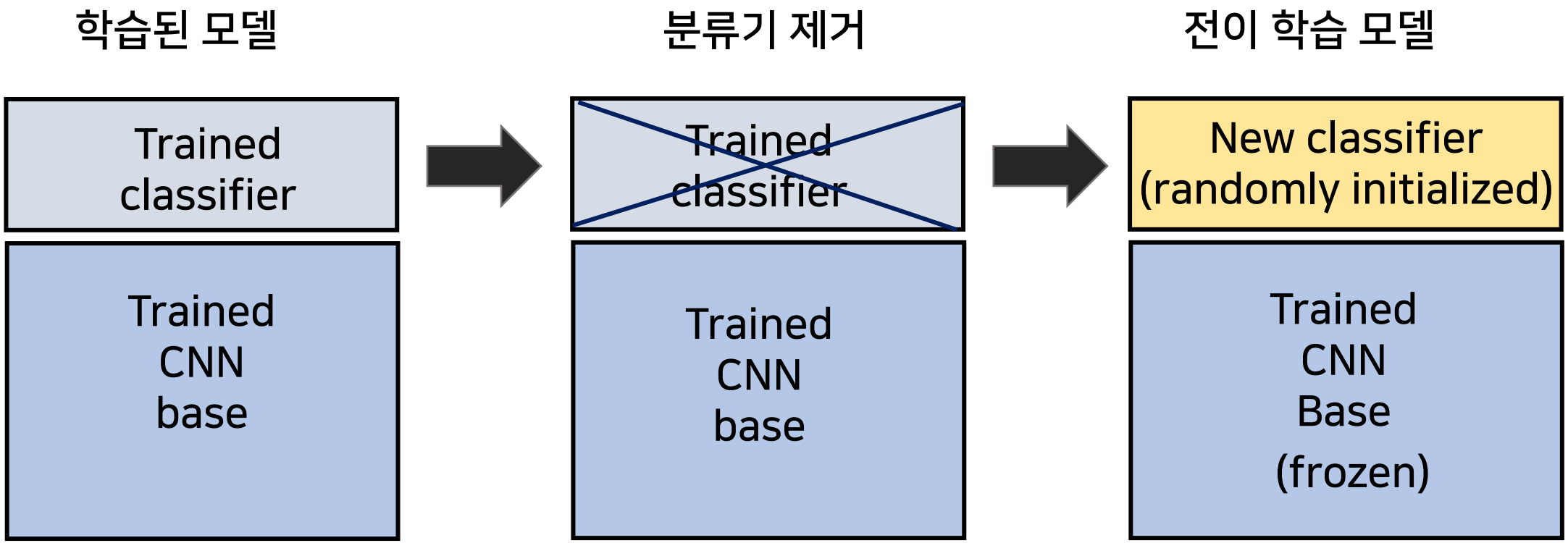
Contents

6장 이미지 분류

전이 학습



전이학습 모델


학습된 모델
분류기 제거
전이 학습 모델
Trained classifier
Trained CNN base
Trained classifier
Trained CNN base
New classifier (randomly initialized)
Trained CNN Base (frozen)

Contents

6장 이미지 분류

실습예제

1. Conv Layer
2. CNN CIFAR10

함수 구현

Convolution 함수

```
def conv(a, b):
    c=np.array(a)*np.array(b)
    return np.sum(c)
```

ReLU 함수

```
def ReLU(f0):
    f0=np.array(f0)
    f0=(f0>0)*f0
    return f0
```

flow_from_directory

flow_from_directory()

```
flow_from_directory(dir, batch_size, target_size, class_mode)
```

-dir : 데이터가 있는 경로

-batch_size : batch_size 설정

-target_size : 불러온 이미지의 사이즈 설정

-class_mode : 데이터 종류 설정

```
train = datagen.flow_from_directory(train_dir, batch_size=32,
        target_size=(32,32), # 데이터 사이즈를 (32,32)로 설정
        class_mode='categorical') #분류용 데이터로 설정
```

Test data 설정

```
test = datagen.flow_from_directory(test_dir, #데이터가 있는 경로
       batch_size=bs, #batch_size 설정 (bs를 사전에 정의해 둠)
       target_size=(32,32), #불러온 이미지의 size 설정
       class_mode='categorical', #분류 데이터로 설정
       shuffle=False) #데이터 순서를 섞지 않고 호출
```

flow_from_dataframe

flow_from_dataframe()

```
flow_from_dataframe(df, directory, x_col, y_col, batch_size,
                    target_size, class_mode)
```

-df : dataframe 파일 경로

-directory : 데이터가 있는 경로

-x_col : dataframe에서 input data가 있는 열

-y_col : dataframe에서 label이 있는 열

-batch_size : batch_size 설정

-target_size : 불러온 이미지의 사이즈 설정

-class_mode : 데이터 종류 설정

```
train = datagen.flow_from_dataframe(train_df,
      directory=train_dir, x_col=train.columns[0],
      y_col=train.columns[1], batch_size=32,
      target_size=(256, 256), class_mode="categorical")
```

Data 확인

```
x, y = train.next() #한 번 데이터 로딩


#로딩한 이미지 그리기
for i in range(0,bs):
        plt.imshow((x[i]*255).astype('int'))
          #datagen에서 255를 나누었으므로 다시 곱한 후 정수로 변환
        plt.show()
        print(x[i].shape, y[i])
        break
```

Conv2D

Conv2D()

```
Conv2D(n,kernel_size,stride=(1,1),activation=None)
```

-n : 필터 개수 설정

-kernel_size : 필터의 크기를 설정

-stride : 필터가 움직이는 폭 설정. 기본값은 (1,1)

-activation : 활성화 함수 설정

```
Conv2D(8, (3,3), activation='relu', input_shape=(32,32,3,))
```

#input_shape : input data shape 설정(첫번째 층인 경우에 설정)

MaxPooling2D

MaxPooling2D()

```
MaxPooling2D(pool_size=(2,2),stride=None)
```

-pool_size : pooling 영역(필터) 크기 설정. 기본값은 (2,2)

-stride : pooling 필터가 움직이는 폭 설정. 기본값은 (2,2)

```
MaxPooling2D() #기본 옵션으로 설정
```

CNN 모델 설정

CNN 모델

```
model=Sequential()
model.add(Conv2D(8,(3,3), activation='relu', input_shape=input_sh))
model.add(MaxPooling2D())
model.add(Conv2D(8,(3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(16,(3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

CNN 모델 학습 / 평가

compile/fit

```
model.compile(loss='sparse_categorical_crossentropy’,
      optimizer=‘RMSProp’, metrics='acc')
hist=md.fit(train, epochs=5)
#fit에 입력할 인수가 단순해졌다.
```

evaluate

```
md.evaluate(test)
```