# Convolutional Neural Network (2)

Jehyuk Lee

Department of AI, Big Data & Management

Kookmin University

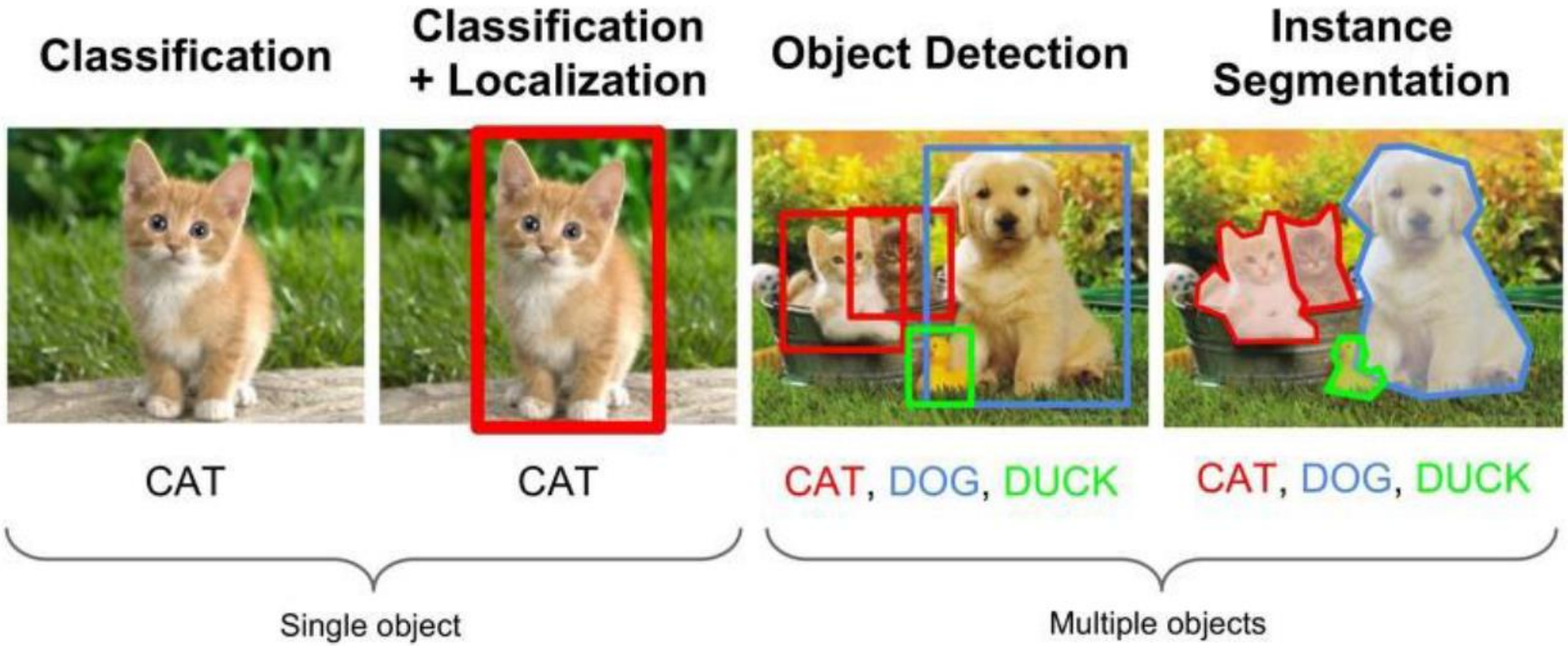
# Contents

- Review
- Advanced CNN model
  - AlexNet, VGGNet, GoogLeNet, ResNet, DenseNet
- Class Activation Map (CAM)
- Transfer Learning

# 1. Review

# Convolutional Neural Network

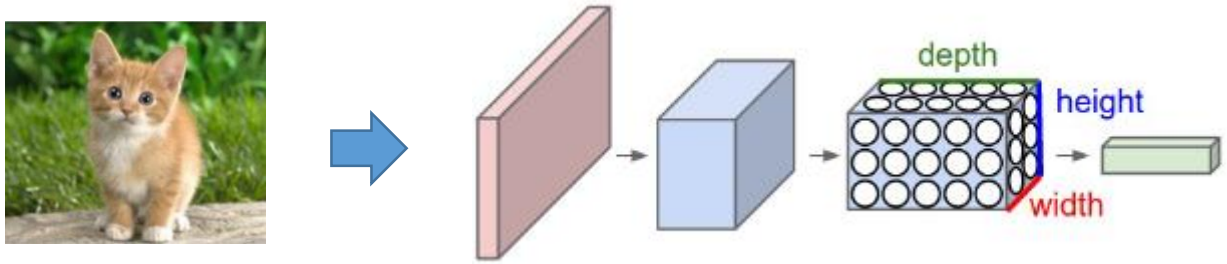
- **합성곱 신경망(Convolutional Neural Network)**
  - Convolutional 연산을 수행하는 Neural Network
  - 이미지로부터 필요한 특징을 스스로 학습하는 능력을 갖춤




(출처: https://analyticsindiamag.com/top-5-image-classification-research-papers-every-data-scientist-should-know/)

# Convolutional Neural Network

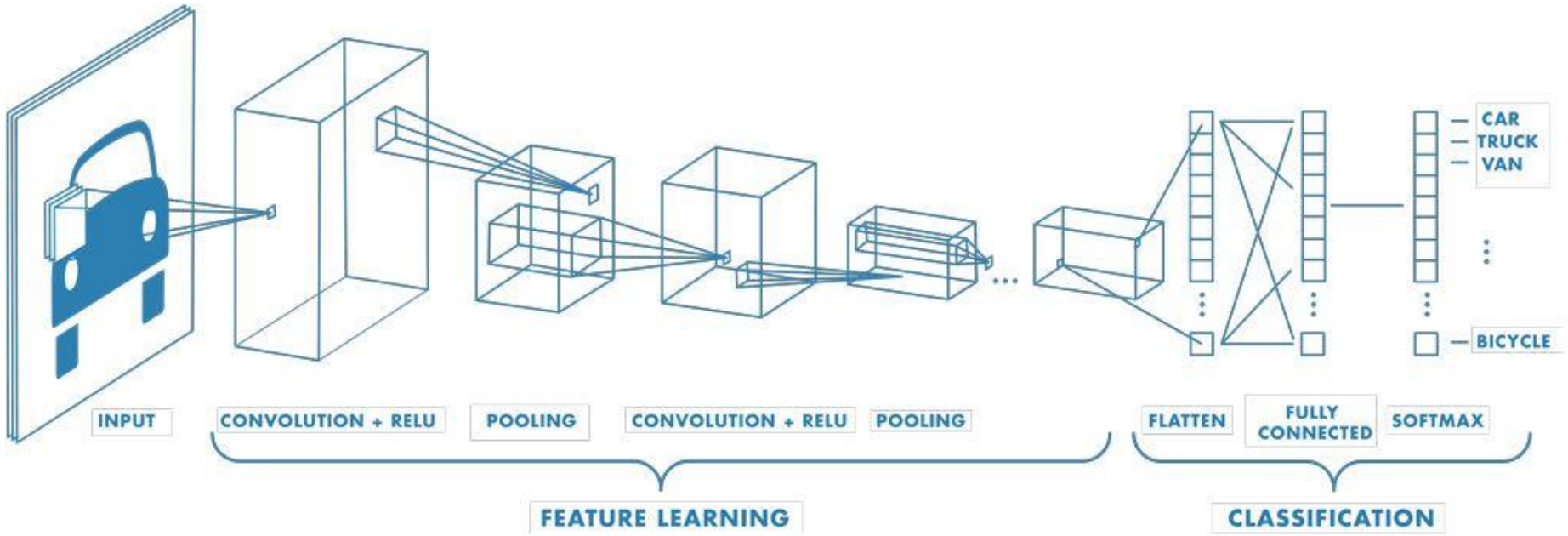
- **Convolutional Neural Network는 이런 한계점을 극복**
  - CNN은 이미지의 형상 정보를 유지
    - CNN은 일반적인 FFNN과 달리 3차원으로 뉴런이 배열되어 있다.
      - Width, Height, Depth
      - (요건 Convolutional Layer를 다룰 때 좀 더 자세히 알아보자)




(출처: https://analyticsindiamag.com/top-5-image-classification-research-papers-every-data-scientist-should-know/, https://cs231n.github.io/convolutional-networks/#architectures)

# CNN Structure

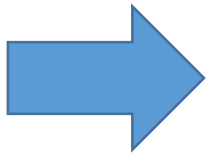
- **Convolutional Neural Network Structure Overview**
  - CNN은 일반적으로 다음과 같은 layer들로 구성되어 있음
    - Convolutional Layer: Input의 일부 region과 filter간의 convolution연산을 수행하는 layer
    - Activation Layer: Convolution 연산을 수행한 결과에 대해 element-wise activation 연산 수행
    - Pooling Layer: 차원별로 Downsampling하는 layer




(출처: https://kr.mathworks.com/discovery/convolutional-neural-network-matlab.html)

# Image Representation

- **이미지를 다음과 같이 컴퓨터에게 인식할 수 있음**
  - 이미지는 다음과 같이 표현 가능
    - RGB 3개의 channel
    - 3차원의 Tensor (RGB)



```
array([[136, 131, 128, ..., 149, 156, 159],
       [ 54,  50,  49, ...,  80,  84,  87],
       [ 40,  38,  39, ...,  70,  74,  76],
       ...,
       [179, 174, 173, ..., 187, 189, 192],
       [170, 169, 170, ..., 191, 192, 193],
       [165, 166, 169, ..., 195, 195, 195]], dtype=uint8)
```

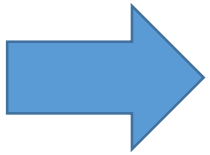
```
array([[173, 165, 159, ..., 168, 173, 176],
       [ 92,  85,  81, ..., 102, 106, 109],
       [ 80,  76,  73, ..., 100, 104, 106],
       ...,
       [167, 162, 161, ..., 177, 179, 182],
       [161, 160, 161, ..., 183, 184, 185],
       [156, 157, 160, ..., 189, 189, 189]], dtype=uint8)
```

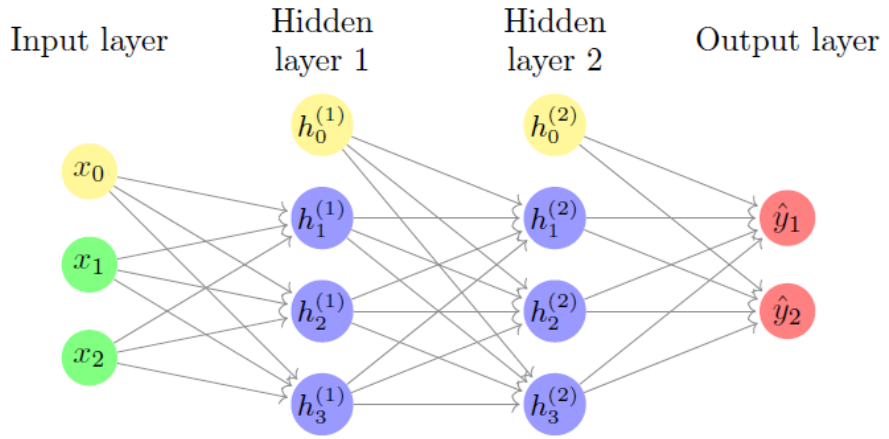
```
array([[155, 148, 141, ..., 136, 139, 142],
       [ 71,  65,  60, ...,  63,  67,  70],
       [ 54,  51,  48, ...,  50,  54,  54],
       ...,
       [151, 146, 145, ..., 165, 167, 170],
       [144, 143, 144, ..., 172, 173, 174],
       [139, 140, 143, ..., 177, 177, 177]], dtype=uint8)
```

(출처: https://analyticsindiamag.com/top-5-image-classification-research-papers-every-data-scientist-should-know/)

# Convolutional Layer

- **가장 간단한 방법은... 각 픽셀 별로 weight 생성**
  - 각 픽셀별로 weight생성
  - 그러나 엄청나게 많은 weight가 필요
    - 아래 이미지 데이터의 크기는 174*203
    - 필요한 weight의 수 : 174 * 203 * 3* (hidden layer의 뉴런 수) = 105K * (hidden layer 뉴런 수)



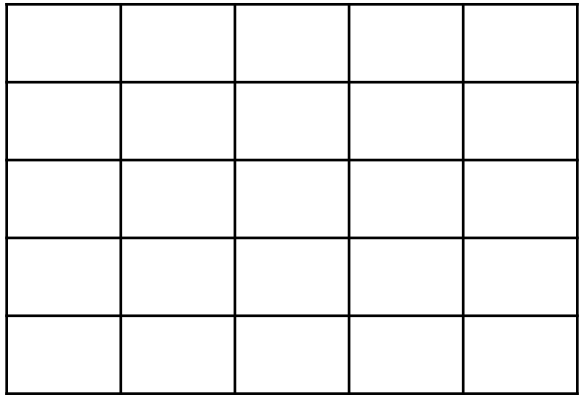


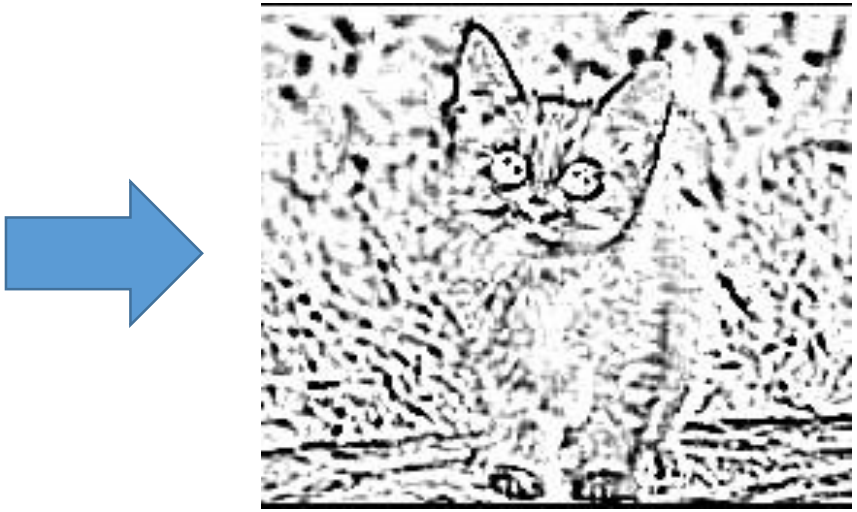

(출처: https://analyticsindiamag.com/top-5-image-classification-research-papers-every-data-scientist-should-know/, https://www.romsoc.eu/artificial-neural-network-and-data-driven-techniques-scientific-computing-in-the-era-of-emerging-technologies/)

# Convolutional Layer

- **Image Convolution**
  - 특정 속성을 탐지하는데 사용하는 tensor
  - Filter, Kernel이라고 부르기도 함
  - 이 Filter를 이미지의 각 region별로 적용 (Parameter sharing)
  - 예시: edge detection

+





(출처: https://analyticsindiamag.com/top-5-image-classification-research-papers-every-data-scientist-should-know/, https://www.romsoc.eu/artificial-neural-network-and-data-driven-techniques-scientific-computing-in-the-era-of-emerging-technologies/)

# Convolutional Layer

- **Convolution Operation (in Image)**
  - 입력 데이터와 필터 간의 연산
    - 필터의 윈도우(window)를 일정 간격으로 이동하면서 입력 데이터에 적용
    - 입력데이터와 필터 간의 element-wise multiply연산 후 add

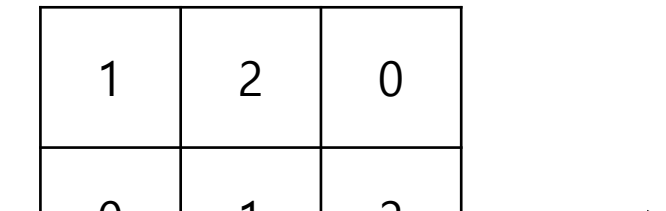
| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 0 |
| 0 | 1 | 2 | 3 | 4 |
| 4 | 0 | 1 | 2 | 3 |
| 3 | 4 | 0 | 1 | 2 |
| 4 | 0 | 1 | 2 | 3 |

**(Input Data)**

$\circledast$

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

**(Filter)**



| | | |
|---|---|---|
| 19 | 18 | 27 |
| 10 | 19 | 18 |
| 17 | 6 | 15 |

**(Output)**

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

# Convolutional Layer

- **Convolution Operation (in Image)**
  - **Padding**
    - Convolutional을 수행하기 전에 데이터 주변을 특정 값(ex: 0)으로 채우기도 함 → **Padding**
    - 주로 출력 크기를 조정할 목적으로 사용

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 0 | 1 | 2 |
| 4 | 0 | 1 |

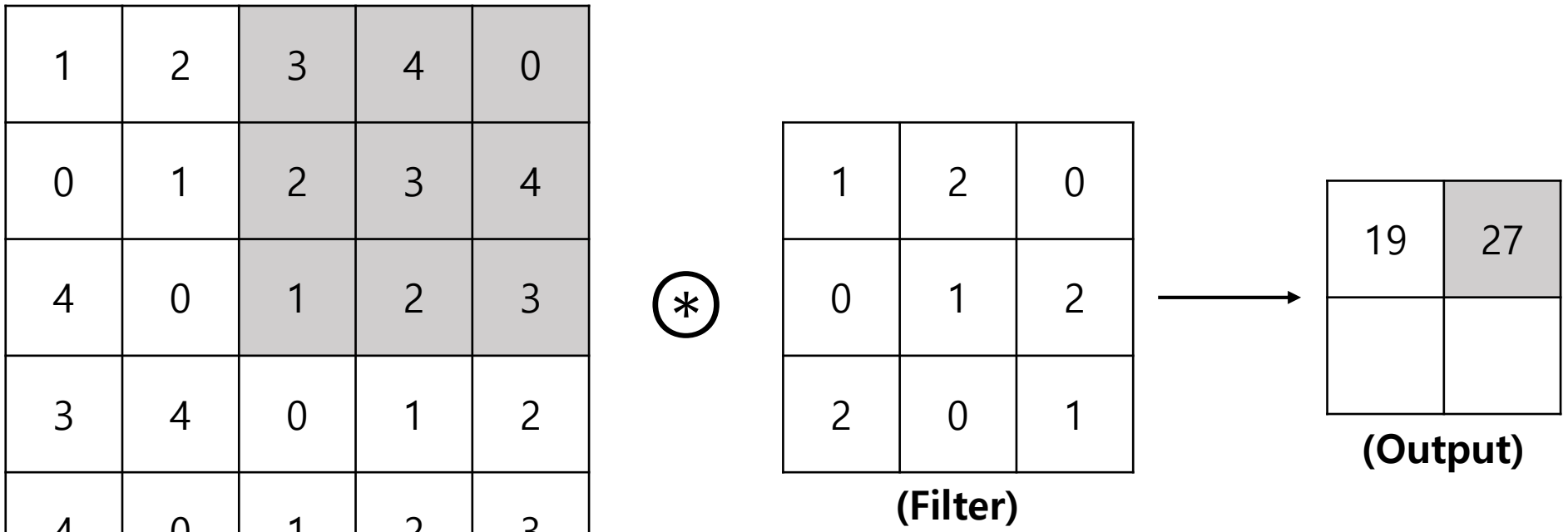
**(Before Padding)**

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 3 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 4 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

**(After Padding)**

# Convolutional Layer

- **Convolution Operation (in Image)**
  - **Stride**
    - 예시: Stride=2 (input data에서 2만큼 이동하여 filter를 적용)

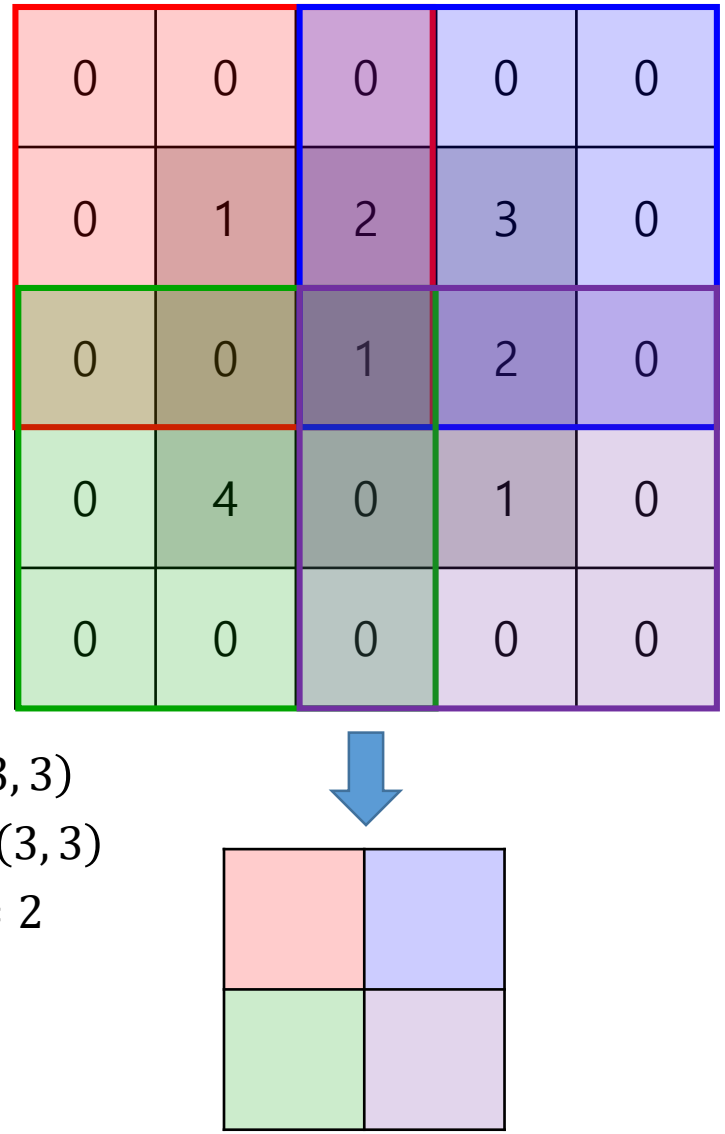


# Convolutional Layer

- **Convolution Operation (in Image)**
  - **Convolution 연산 결과의 shape 예측하기**
    - 입력 데이터의 크기: $(H, W)$
    - 필터의 크기: $(FH, FW)$
    - 패딩의 크기: $P$
    - 스트라이드: $S$
    - 출력 데이터의 크기 $(OH, OW)$는 다음과 같음

$$OH = 1 + \frac{(H + 2P - FH)}{S}$$

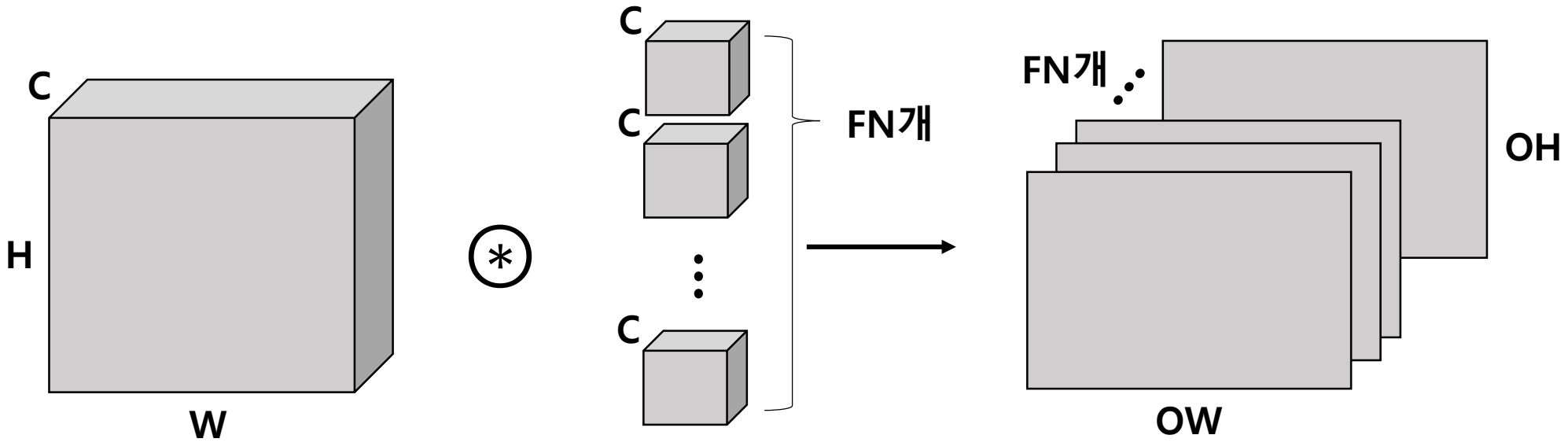
$$OW = 1 + \frac{(W + 2P - FW)}{S}$$




$(H, W) = (3, 3)$
$(FH, FW) = (3, 3)$
$P = 1, S = 2$

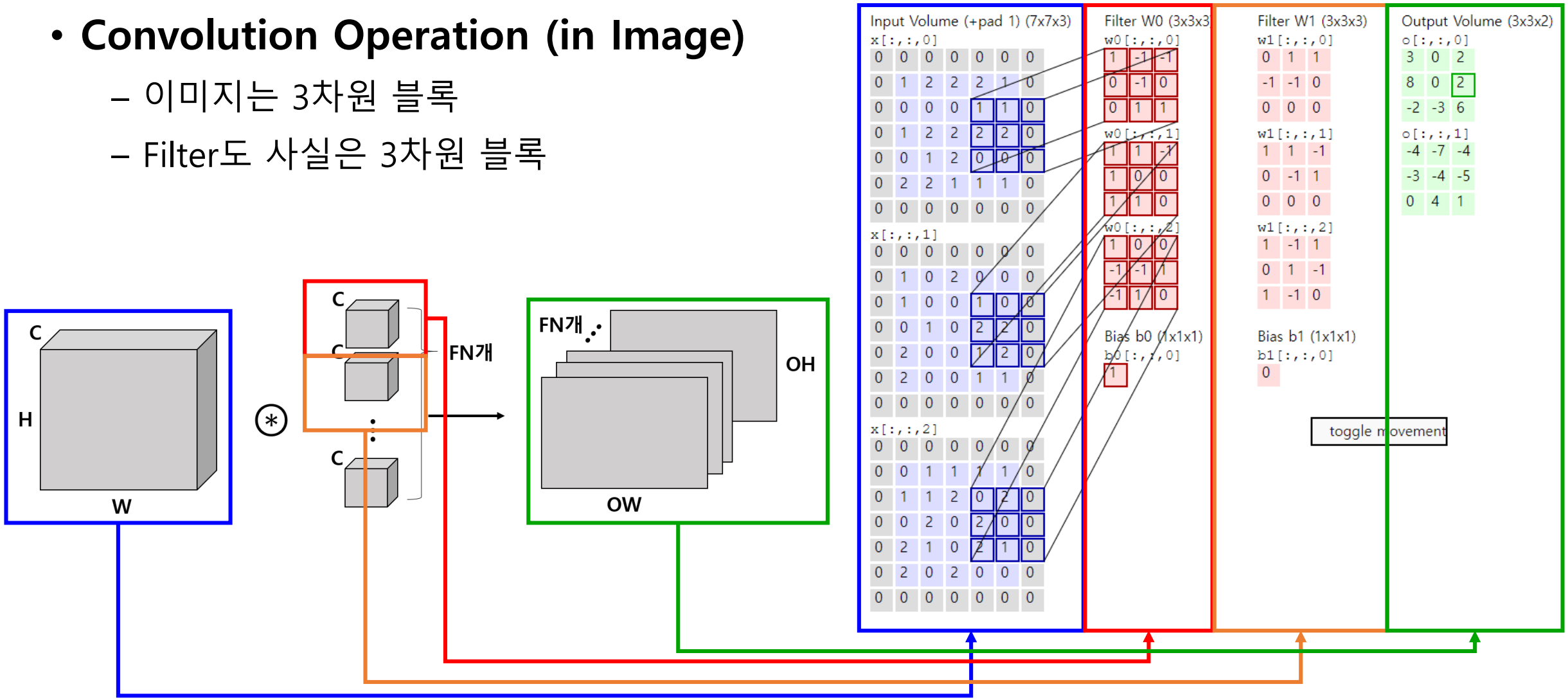
# Convolutional Layer

- **Convolution Operation (in Image)**
  - 이미지는 3차원 블록
    - $(C, H, W)$ = (Channel, Height, Width)
  - Filter도 사실은 3차원 블록
    - $(C, FH, FW)$ = (Channel, Filter Height, Filter Width)
  - 예시: 3차원 이미지와 3차원 Filter로부터 $FN$개의 feature map생성



# Convolutional Layer

- **Convolution Operation (in Image)**
  - 이미지는 3차원 블록
  - Filter도 사실은 3차원 블록




(출처: https://cs231n.github.io/convolutional-networks/#architectures)

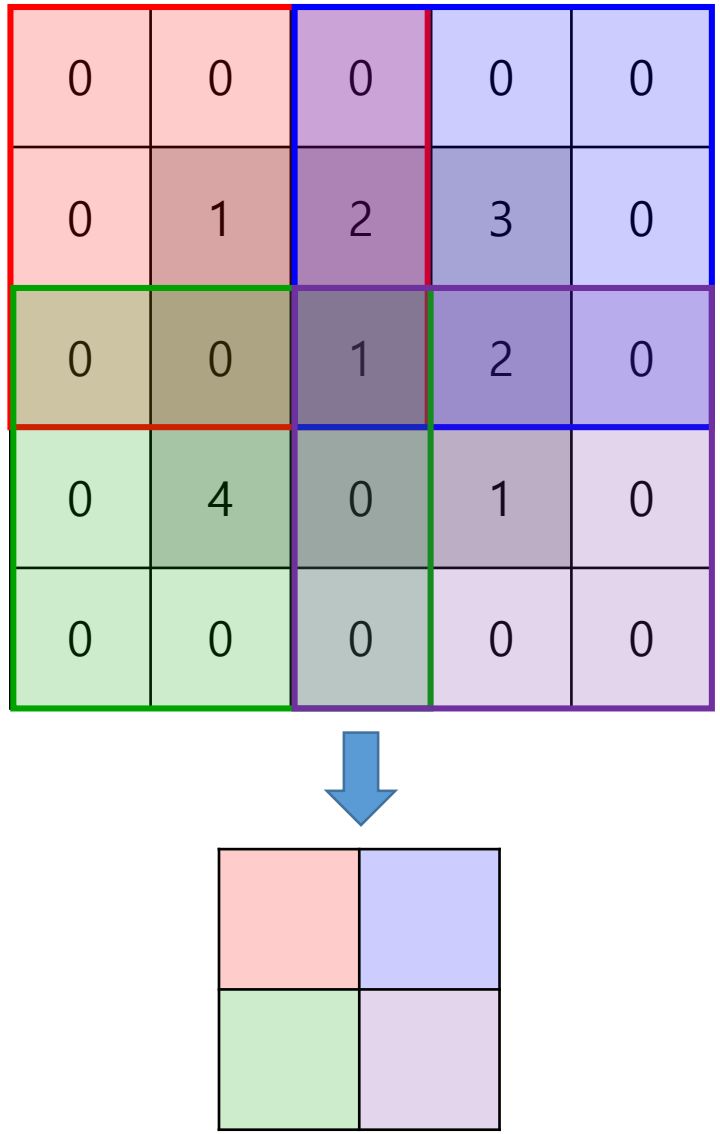
# Convolutional Layer

- **Convolution Operation (in Image)**
  - **Convolution 연산 결과의 3차원 shape 예측하기**
    - 입력 데이터의 크기: $(C, H, W)$
    - 필터의 크기: $(C, FH, FW)$
    - 패딩의 크기: $P$
    - 스트라이드: $S$
    - 필터 수: $FN$
    - 출력 데이터의 크기 $(OC, OH, OW)$는 다음과 같음

$$OC = FN$$

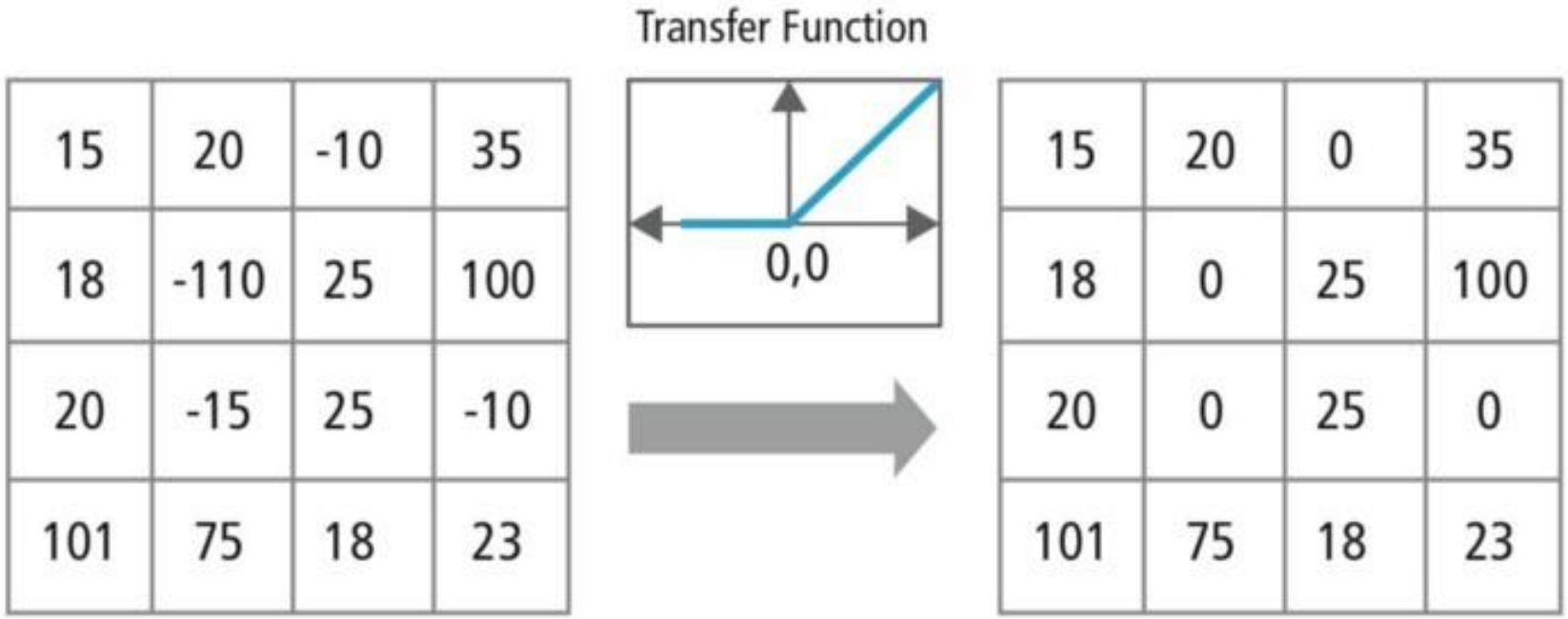
$$OH = 1 + \frac{(H + 2P - FH)}{S}$$

$$OW = 1 + \frac{(W + 2P - FW)}{S}$$



# Convolutional Layer

- **Activation Function**
  - Convolution 연산 결과의 비선형 변환을 수행
  - (대부분 ReLU를 사용)



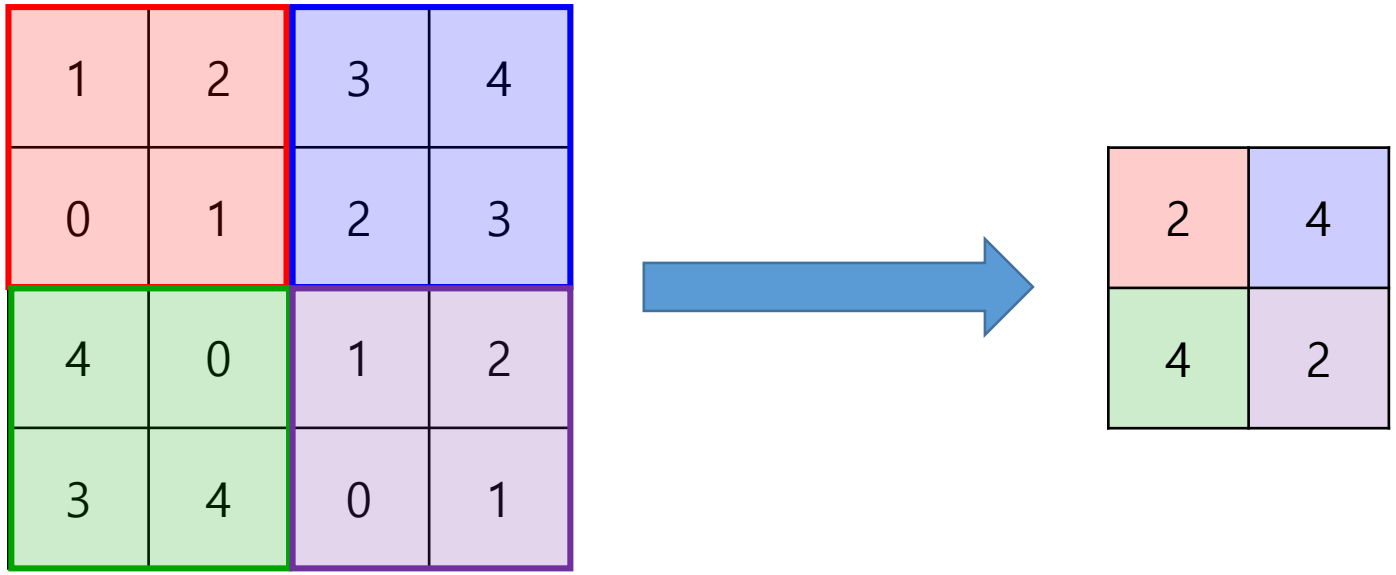

| | | | |
|---|---|---|---|
| 15 | 20 | -10 | 35 |
| 18 | -110 | 25 | 100 |
| 20 | -15 | 25 | -10 |
| 101 | 75 | 18 | 23 |

| | | | |
|---|---|---|---|
| 15 | 20 | 0 | 35 |
| 18 | 0 | 25 | 100 |
| 20 | 0 | 25 | 0 |
| 101 | 75 | 18 | 23 |

(출처: https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117)

# Pooling Layer

- **Pooling**
  - 가로, 세로 방향의 공간을 줄이는 연산
    - 일정 영역의 정보를 축약하는 역할
    - 네트워크의 파라미터의 개수를 줄여 연산 수를 줄임 → 모델의 복잡성 감소 → 오버피팅 방지
    - 예시: Max Pooling with 2 × 2 filters with stride 2



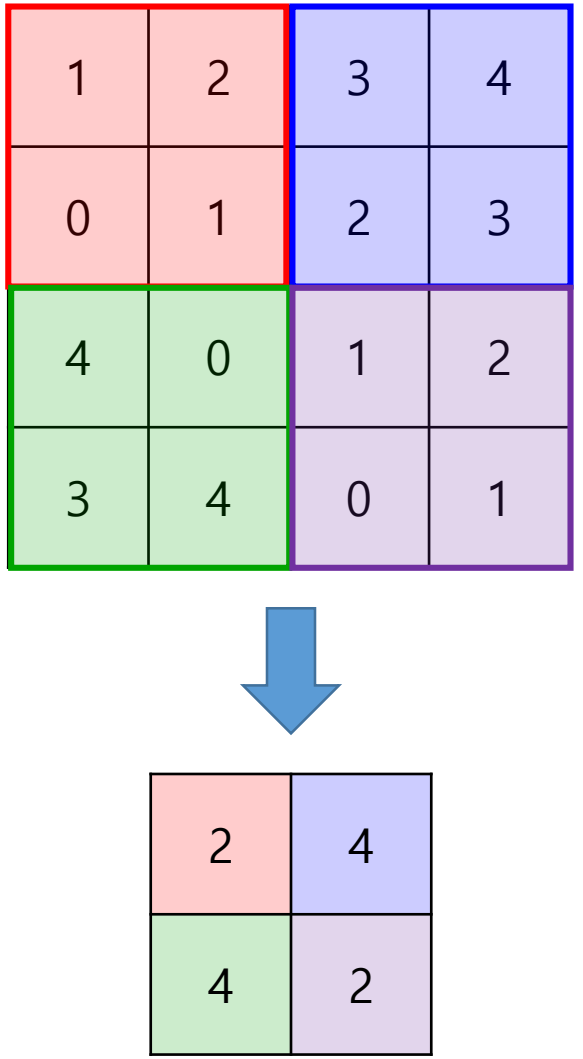
# Pooling Layer

- **Pooling**
  - **Pooling 연산 결과의 shape 예측하기**
    - 입력 데이터의 크기: $(C, H, W)$
    - 필터의 크기: $(C, FH, FW)$
    - 공간 확장: $F$
    - 스트라이드: $S$
    - 출력 데이터의 크기 $(OC, OH, OW)$는 다음과 같음

$$OC = C$$

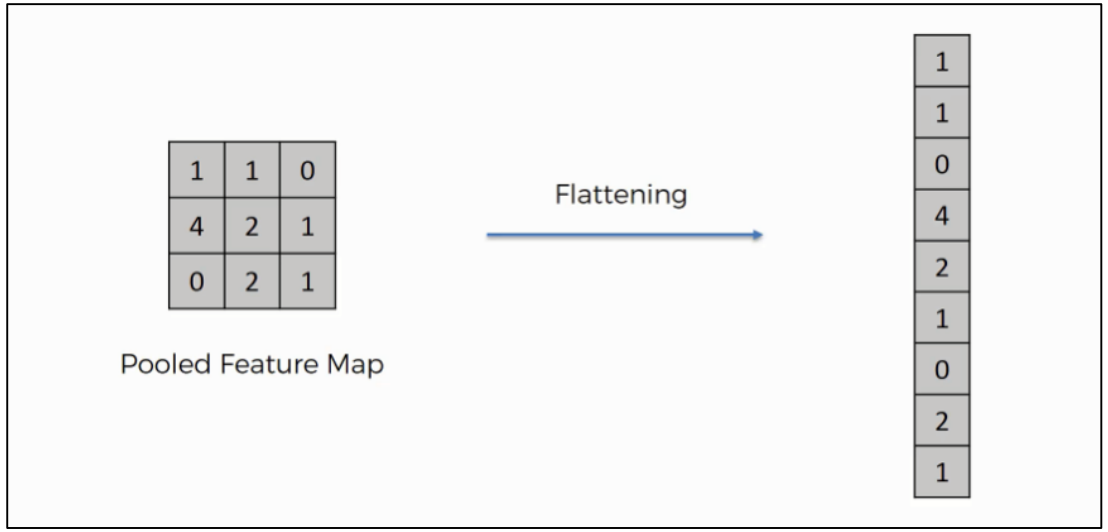
$$OH = 1 + \frac{(H - FH)}{S}$$

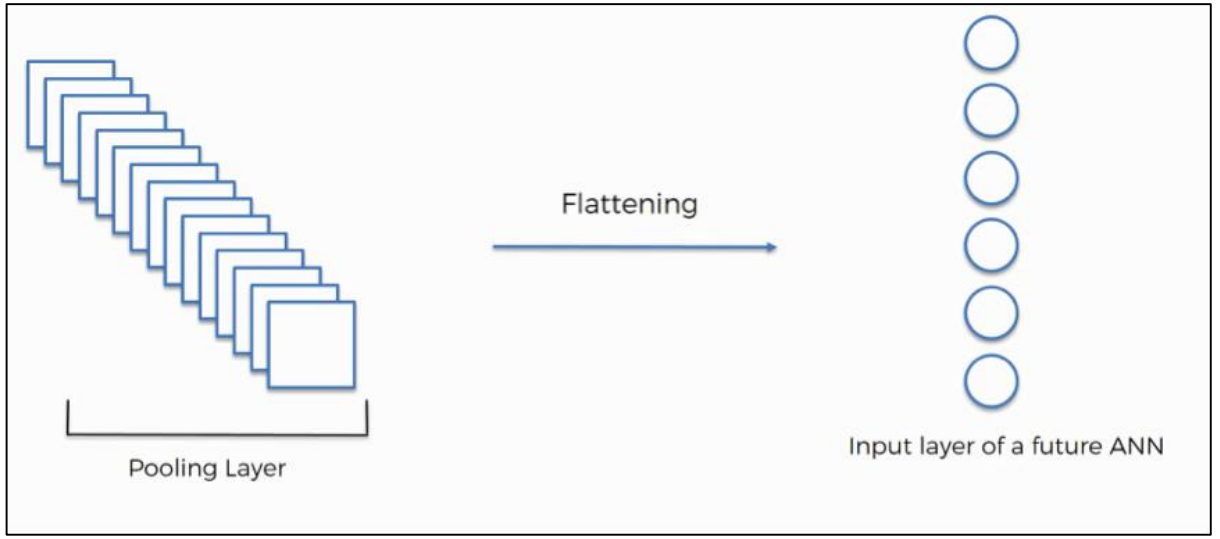
$$OW = 1 + \frac{(W - FW)}{S}$$



# Fully-Connected Layer

- **Fully Connected Layer**
  - Flatten
    - 2차원/3차원의 Matrix/Tensor구조를 1차원 vector로 변환
  - Input for Fully-connected layer
    - Convolutional-Pooling Layers의 연산 결과를 1차원 vector로 변환
    - 변환한 vector를 Fully-connected layer의 input으로




(출처: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening)

# 2. Advanced CNN model

# CNN in Image Classification

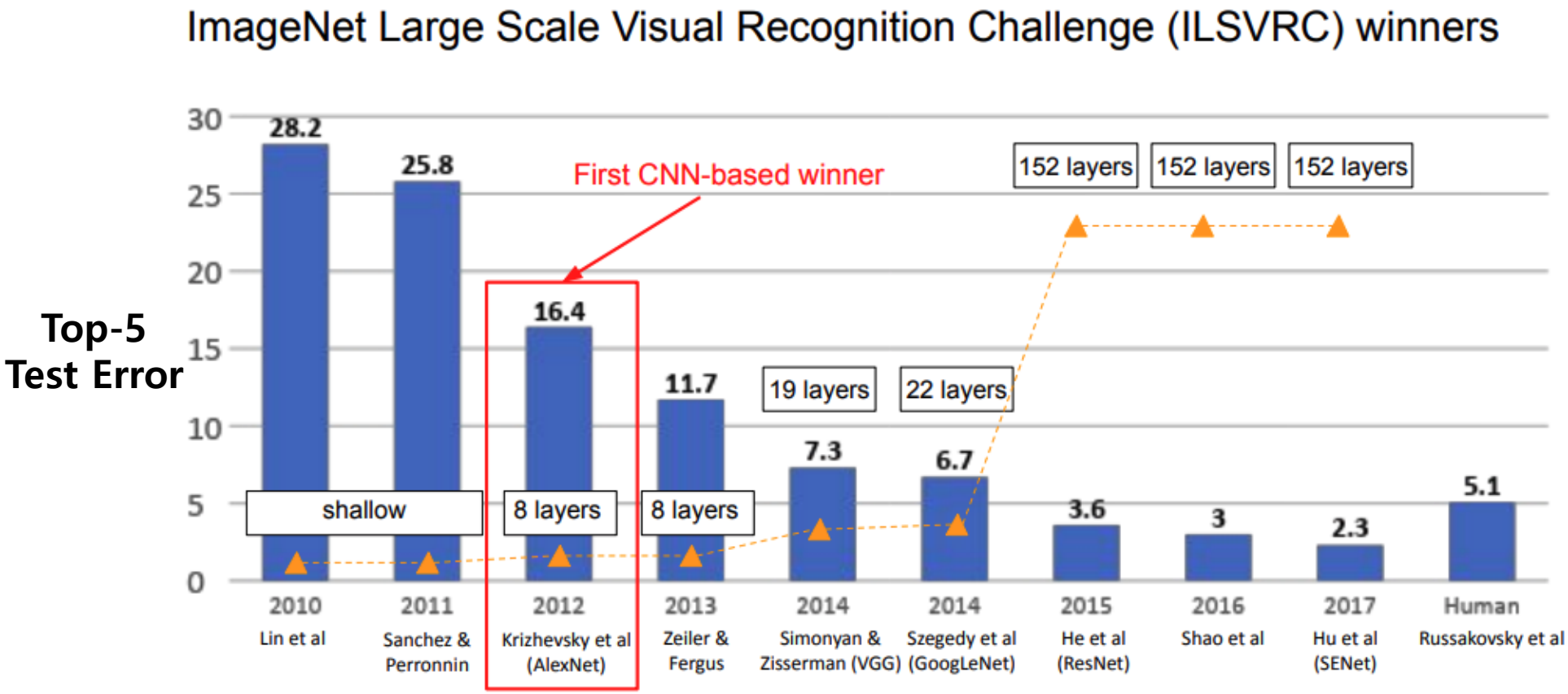
- **ImageNet**
  - 천 가지 범주에서 뽑은 약 1천 4백만 장의 이미지들로 이루어진 거대한 데이터 베이스
  - 이미지 부류가 아주 다양해서, 일상생활에서 볼 수 있는 대부분의 이미지를 포함
  - **ImageNet Large Scale Visual Recognition Challenge(ILSVRC)**에서 이 데이터를 사용
    - Computer Vision 분야에서 권위 있는 Challenge


IMAGENET DATA BASE

(출처: https://supermemi.tistory.com/m/10)

# CNN in Image Classification

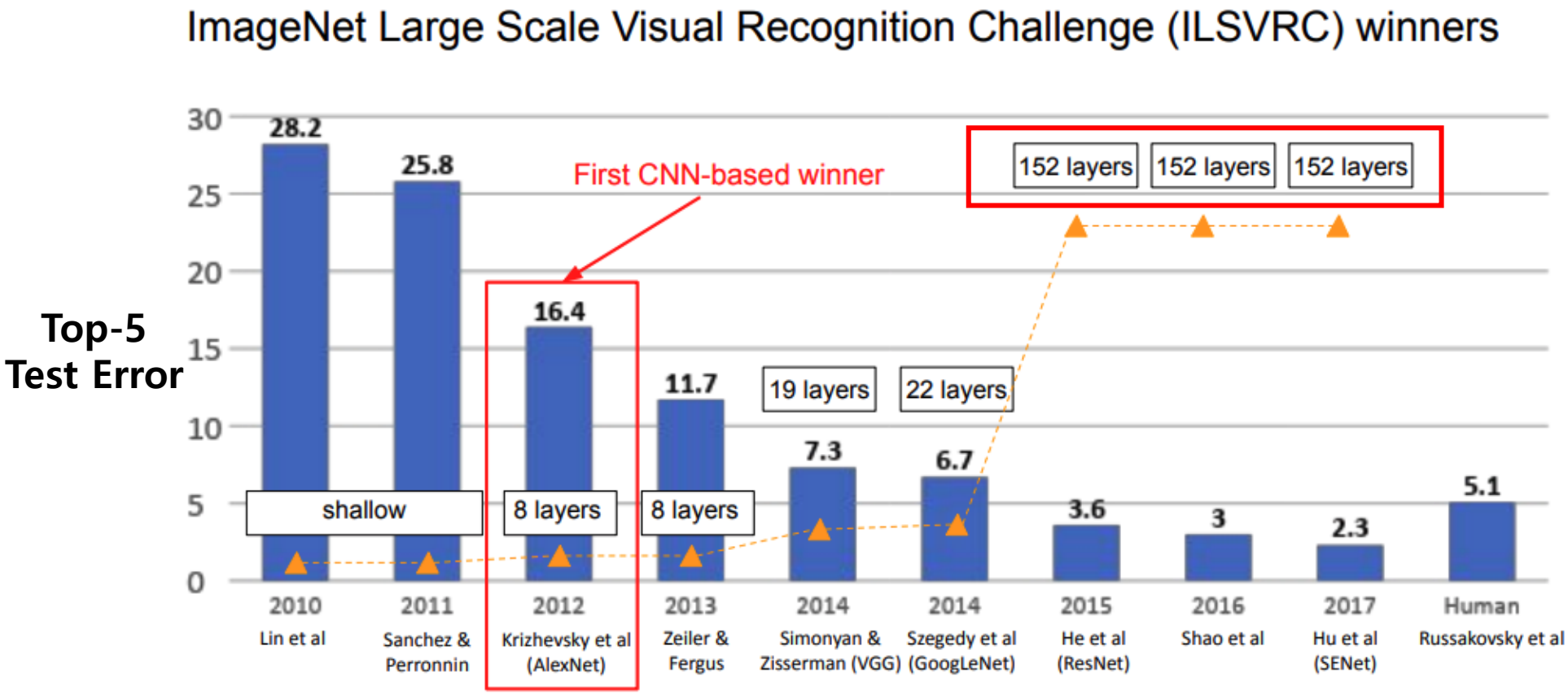
- **CNN이 본격적으로 사용된 건 생각보다 최근의 일**
  - Top-5 Error: 모델이 예측한 최상위 5개 범주 가운데 정답이 없는 경우의 오류율




(출처: http://cs231n.stanford.edu/slides/2022/lecture_6_jiajun.pdf)

# CNN in Image Classification

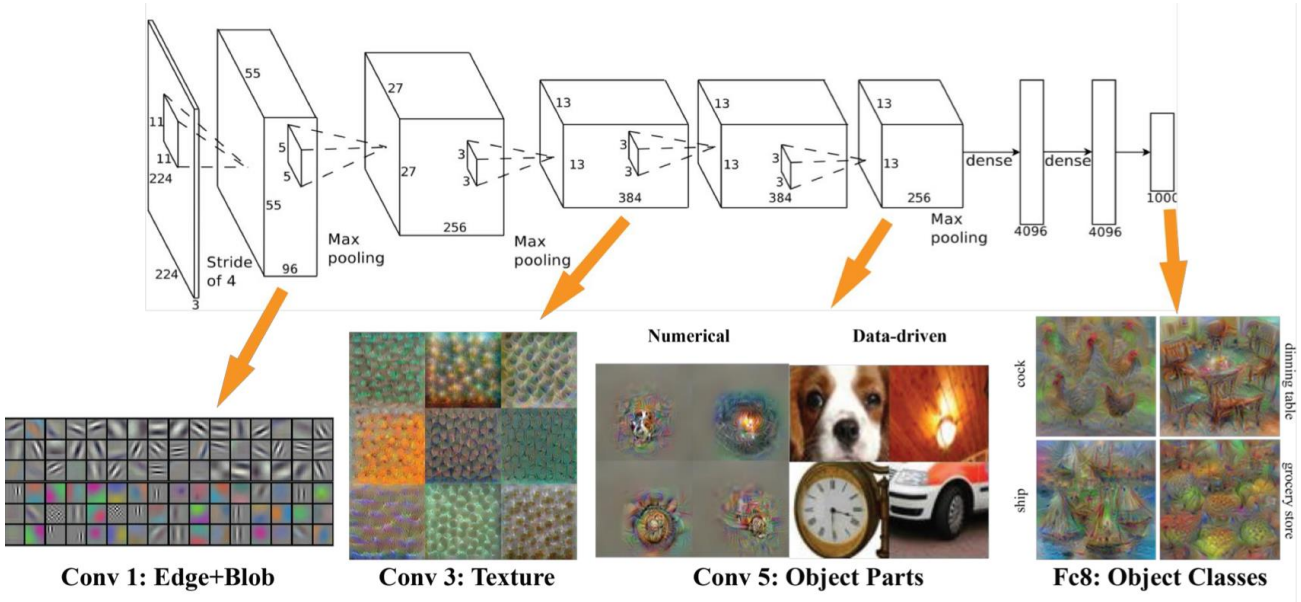
- **CNN에서 layer가 많아질수록 이미지 분류 성능이 증가하는 추세**
  - 심지어, 최근에는 효율적으로 연산(Operation수 감소), 정확한 모델 구조 등장

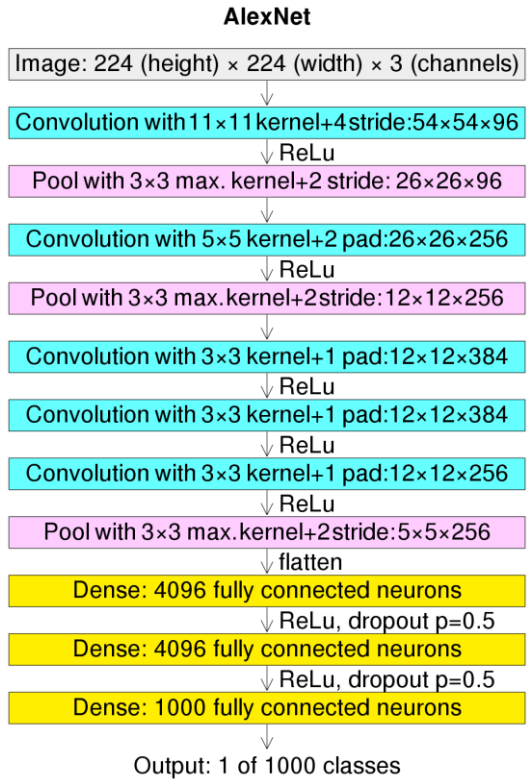



(출처: http://cs231n.stanford.edu/slides/2022/lecture_6_jiajun.pdf)

# CNN in Image Classification

- **Case 1: AlexNet**
  - 제프리 힌튼 교수팀에서 최초로 CNN을 ImageNet에 적용
  - 주요 구조
    - Conv, Max-Pooling, dropout layer 5개
    - Fully-connected layer 3개
    - Activation Function: ReLU

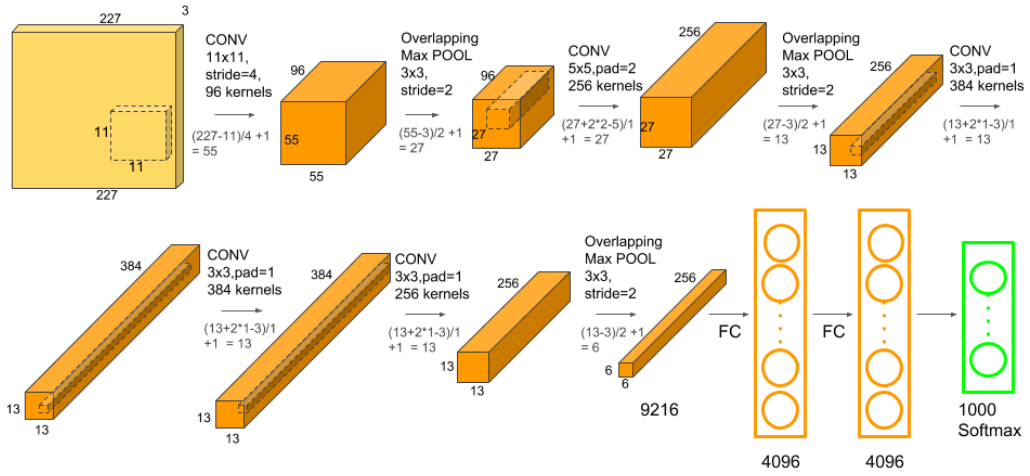


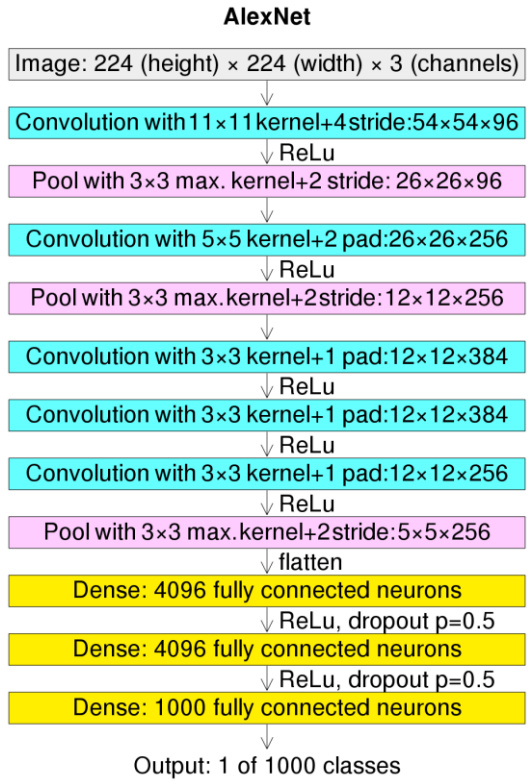




(출처: https://imgbin.com/png/XFuMf8Ce/artificial-neural-network-deep-learning-convolutional-neural-network-artificial-intelligence-machine-learning-png, https://en.wikipedia.org/wiki/AlexNet)

# CNN in Image Classification

- **Case 1: AlexNet**
  - 제프리 힌튼 교수팀에서 최초로 CNN을 ImageNet에 적용
  - 특징
    - Input Size: 224*224
    - 초기 단계에서는 large filter size & stride
    - 상위 단계에서는 smaller size
    - # of Parameter: 60 Million

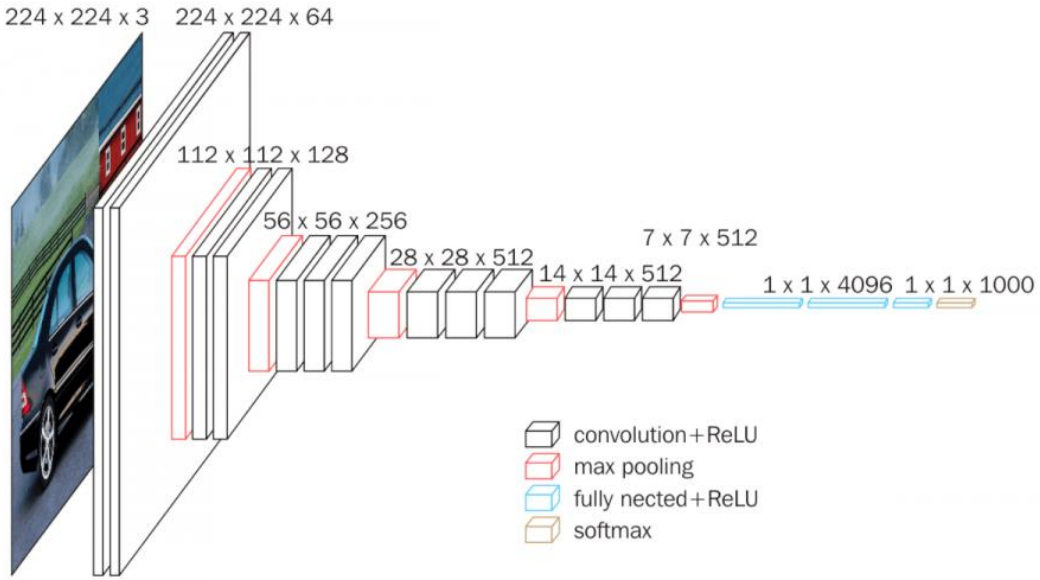


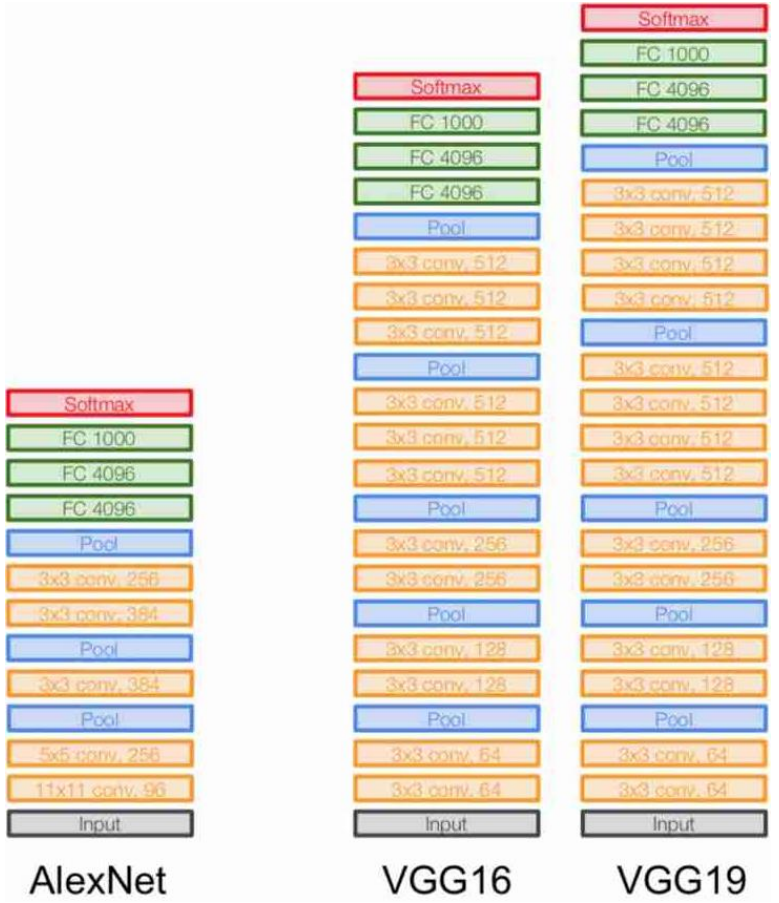




(출처: https://wikidocs.net/164787, https://en.wikipedia.org/wiki/AlexNet)

# CNN in Image Classification

- **Case 2: VGGNet**
  - A. Zisserman, K. Simonyan이 제안
  - 단순하지만 깊은 구조
    - 3*3 convolution with stride 1
    - 2*2 max pooling
  - # of Parameter: 138 Million

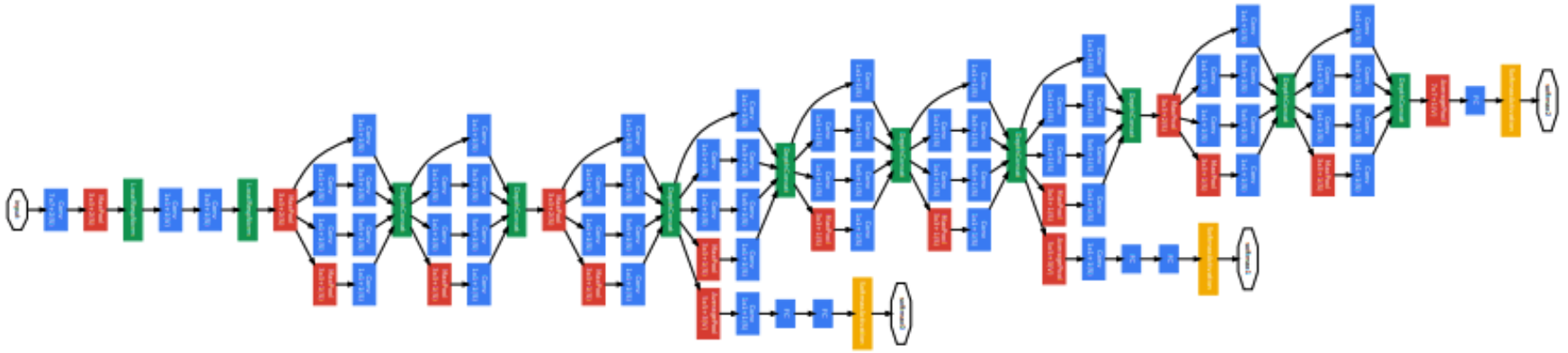



(출처: https://wikidocs.net/164796)

# CNN in Image Classification

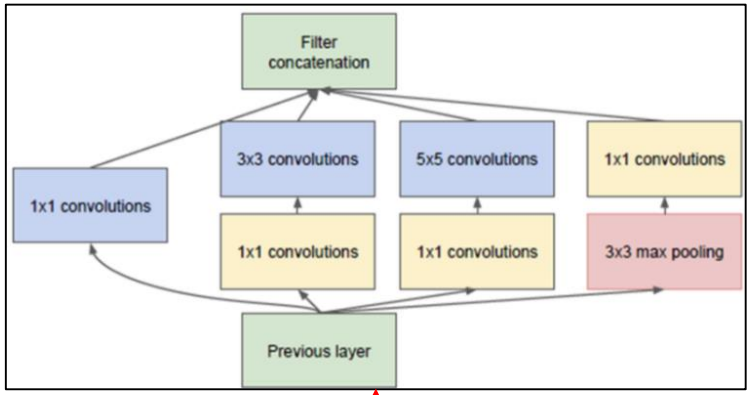
- **Case 3: GoogLeNet**
  - Szegedy et al.이 제안
  - 특징
    - 22 Layers
    - No fully connected layers
    - # of Parameters: 5 Million
      - 12x less than AlexNet(60M)
      - 27 times less than VGG-16(138M)

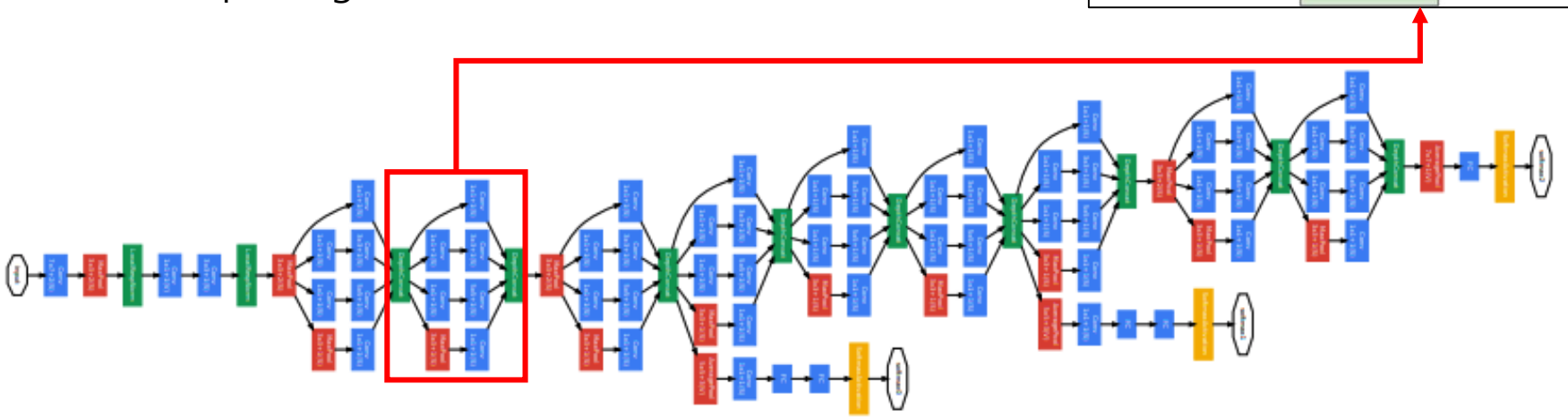


(출처: Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).)

# CNN in Image Classification

- **Case 3: GoogLeNet**
  - 핵심은 Inception Module
    - 한 입력 image에 한종류의 filter만 적용해야 하나?
    - 뭘 좋아할지 몰라서 다 준비해봤어
      - 1*1 convolution (이 녀석은 뭘까요?)
      - 3*3 convolution
      - 5*5 convolution
      - 3*3 max pooling

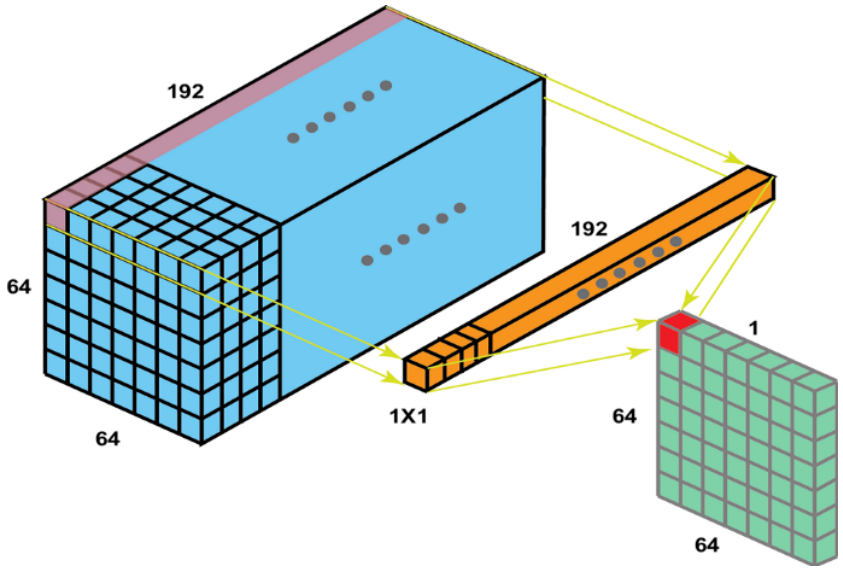


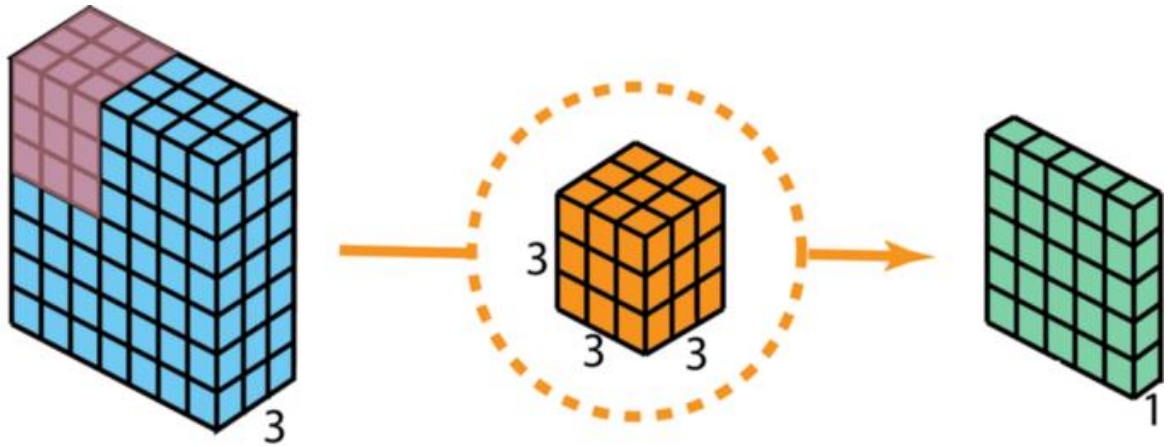
(출처: Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).)

# CNN in Image Classification

- **Case 3: GoogLeNet**
  - 1*1 Convolution
    - Feature map의 depth를 감소시켜 연산량 줄이는 효과




(1*1 Convolution)

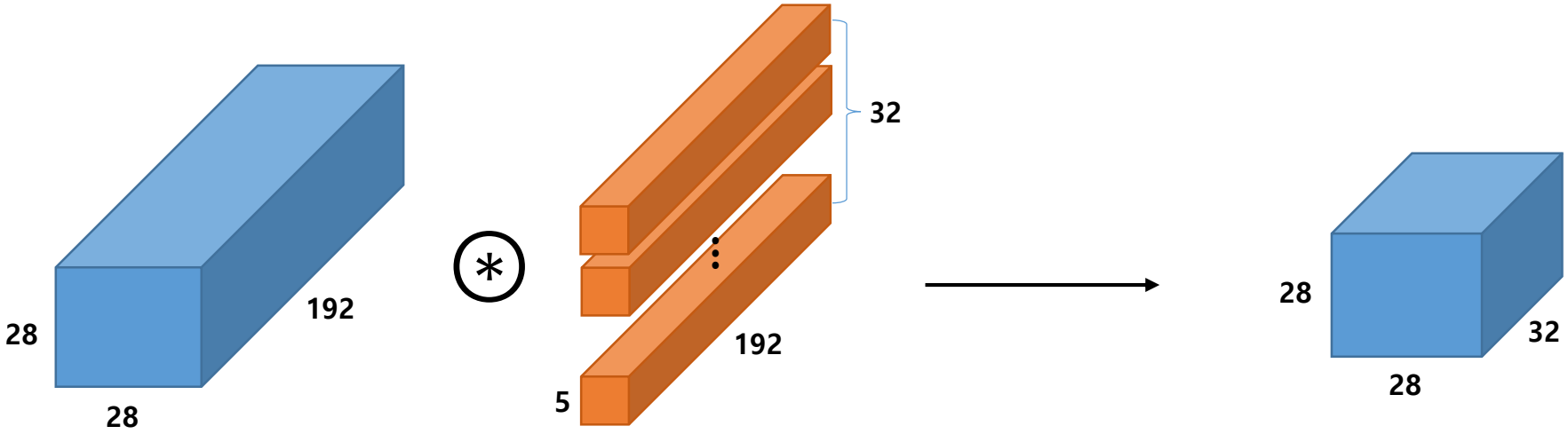



(Typical Convolution)

(출처: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215)

# CNN in Image Classification

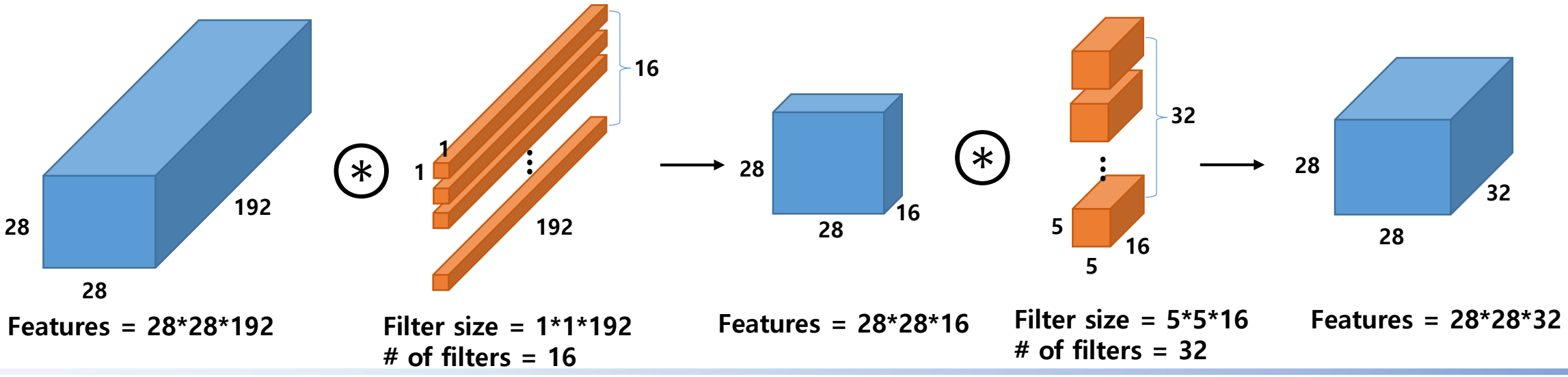
- **Case 3: GoogLeNet**
  - Without 1*1 Convolution
    - 예시: (height, width, channel) = (28,28,192)에서 5*5 convolution수행 (filter수는 32개)
    - 연산량
      - (28*28*32) * (5*5*192) = 120M
      - (한 channel의 feature마다 필요한 conv횟수) * (한 conv마다 필요한 연산 수)




**Features = 28*28*192** **Filters = 5*5*32** **Features = 28*28*32**

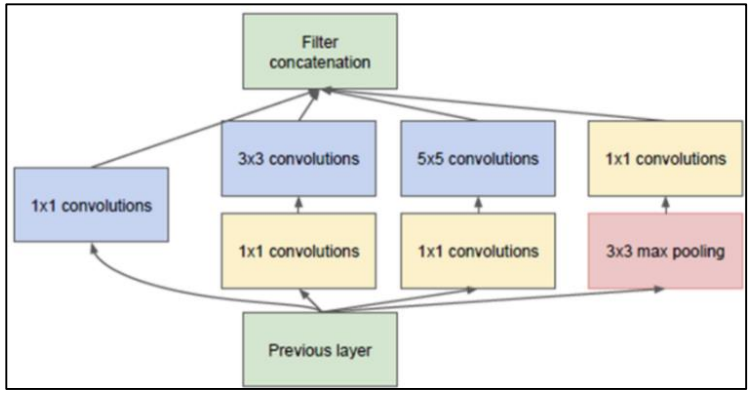
# CNN in Image Classification

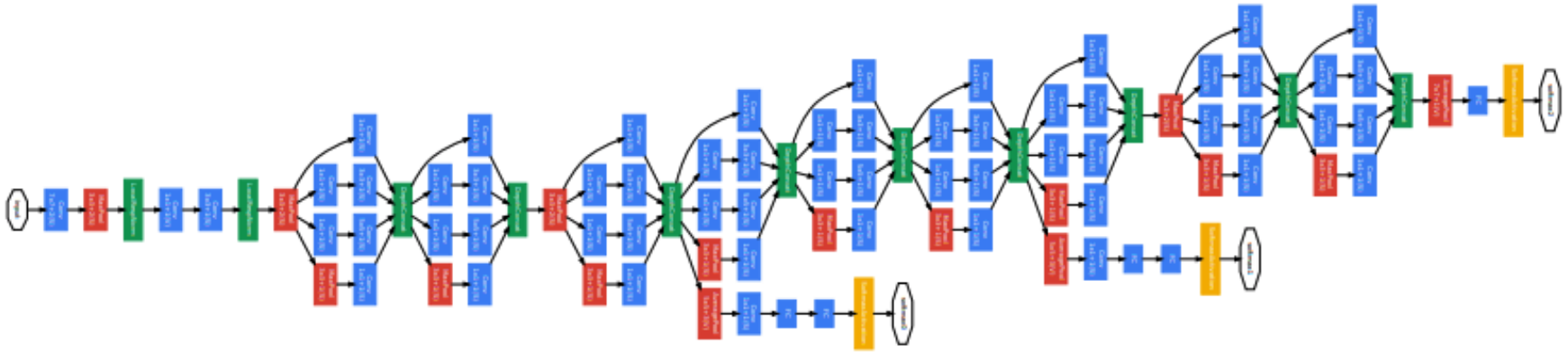
- **Case 3: GoogLeNet**
  - With 1*1 Convolution
    - 예시
      - 1. (height, width, channel) = (28,28,192)에서 1*1 convolution수행 (filter수는 16개)
      - 2. (height, width, channel) = (28,28,,16)에서 5*5 convolution수행 (filter수는 32개)
    - 연산량
      - (28*28*16) * (1*1*192) + (28*28*32) * (5*5*16) = 2.4M+10M = 12.4M <<120M



# CNN in Image Classification

- **Case 3: GoogLeNet**
  - Inception 모듈
    - 1*1 convolution
    - 3*3 convolution
    - 5*5 convolution
    - 3*3 max pooling



→ 뭐 하러 모듈을 이렇게 복잡하게 만들었을까요? 그리고 얘는 왜 성능이 좋을까요?

(출처: Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).)

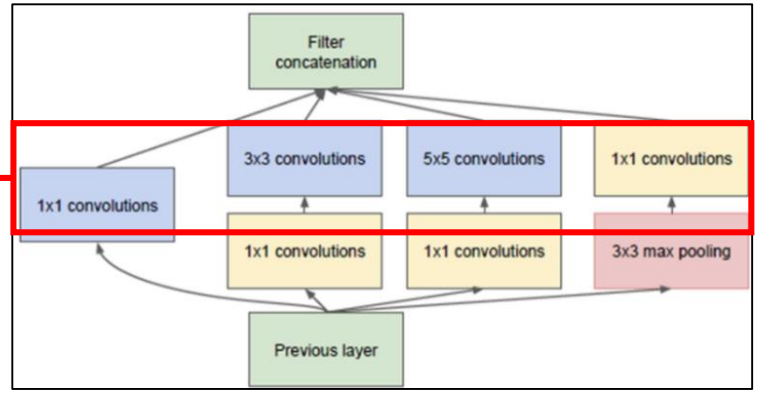
# CNN in Image Classification

- **Case 3: GoogLeNet**
  - 이미지에서 물체의 크기는 상당히 큰 편차가 있습니다
  - 정보의 위치가 다양하기 때문에 conv연산에 적합한 필터 크기 선정이 어려움
    - Filter size가 크면 전역적으로 배포되는 정보에 선호됨
    - Fitler size가 작으면 지역적으로 배포되는 정보에 선호됨
  - 또한, 깊을수록 gradient vanishing문제 발생 → 깊이가 다가 아니다!!

(출처:https://wikidocs.net/164794)

# CNN in Image Classification

- **Case 3: GoogLeNet**
  - Inception 모듈
    - 1*1 convolution
    - 3*3 convolution
    - 5*5 convolution
    - 3*3 max pooling

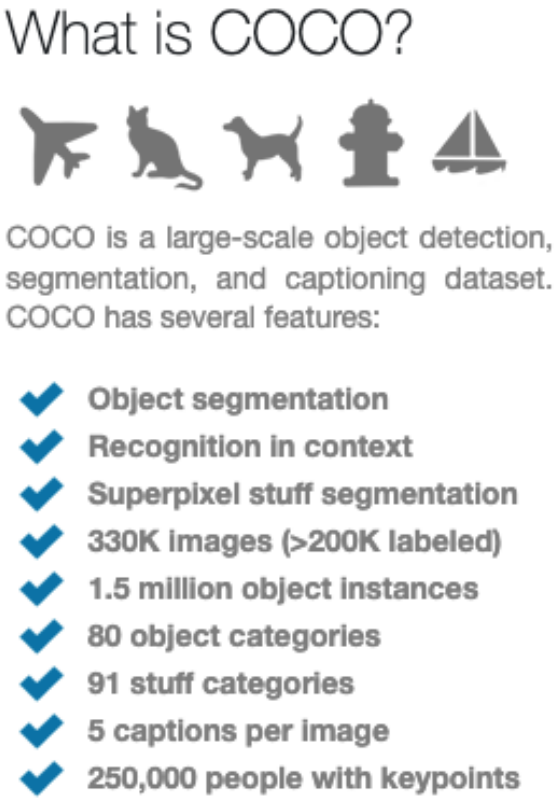



- Filter Concatenation은 filter 수 방향으로 이뤄집니다.
  - 예시: 각 Conv의 결과가 (28*28*50) → Filter concatenation 결과는 (28*28*200)

(출처: Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).)

# CNN in Image Classification

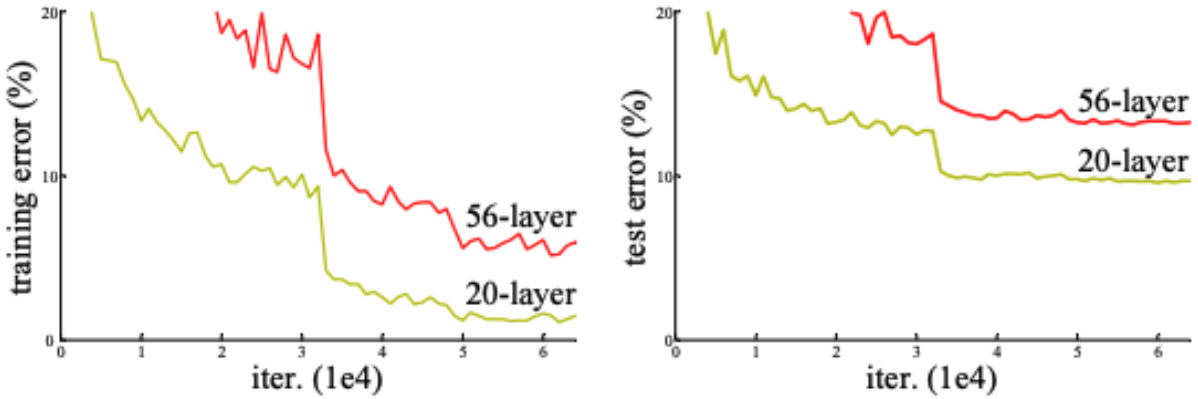
- **Case 4: ResNet**
  - Kaiming He et al.이 제안
  - 특징
    - 152 Layers
    - Residual Learning 개념 도입
    - Image Classification을 사람보다 더 높은 성능으로 수행
    - 다양한 대회 1위
      - ILSVRC 2015 Classification
      - ImageNet Detection
      - ImageNet Localization
      - COCO detection
      - COCO segmentation



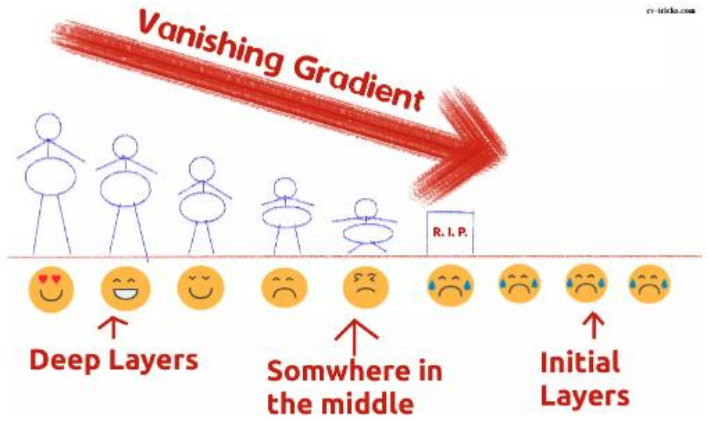

(https://cocodataset.org/#home)

# CNN in Image Classification

- **Case 4: ResNet**
  - Vanishing Gradient
    - 152 Layers (엄청나게 깊은 layer)
    - 기존에는 이렇게 쌓을 생각을 못했을까?
      - 오히려 층을 쌓을수록 어느 순간부터 성능이 감소
      - Why???




**(Layer를 더 쌓았는데 training error가 더 높음)**

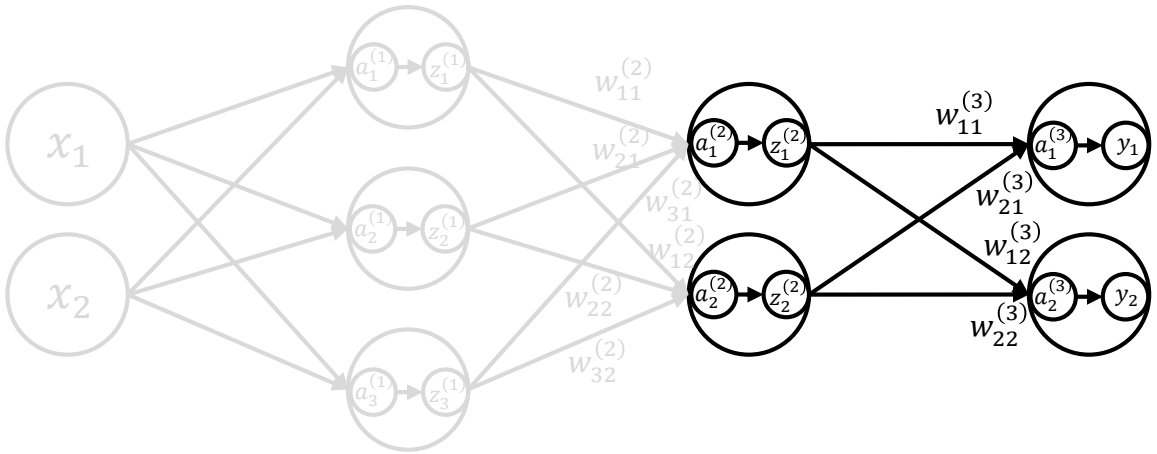


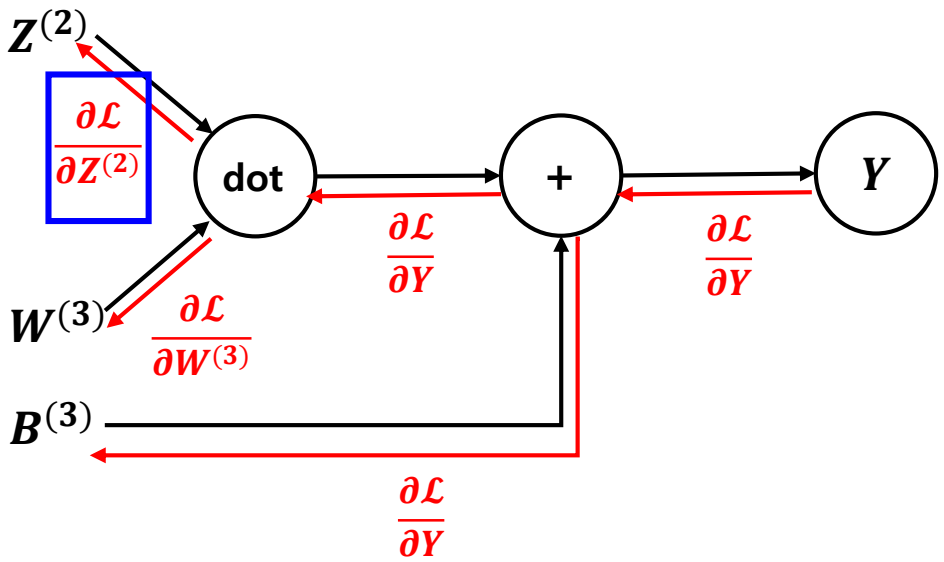

**(Vanishing Gradient)**

(출처: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778)., https://wikidocs.net/164800)

# CNN in Image Classification: Revisit Backpropagation

- (1) Backpropagation: Output → 2nd hidden layer

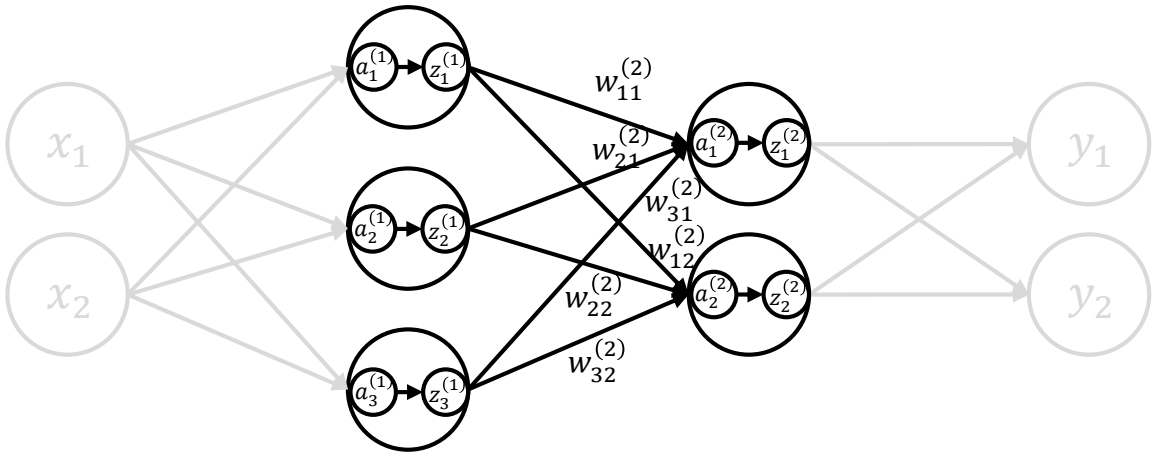


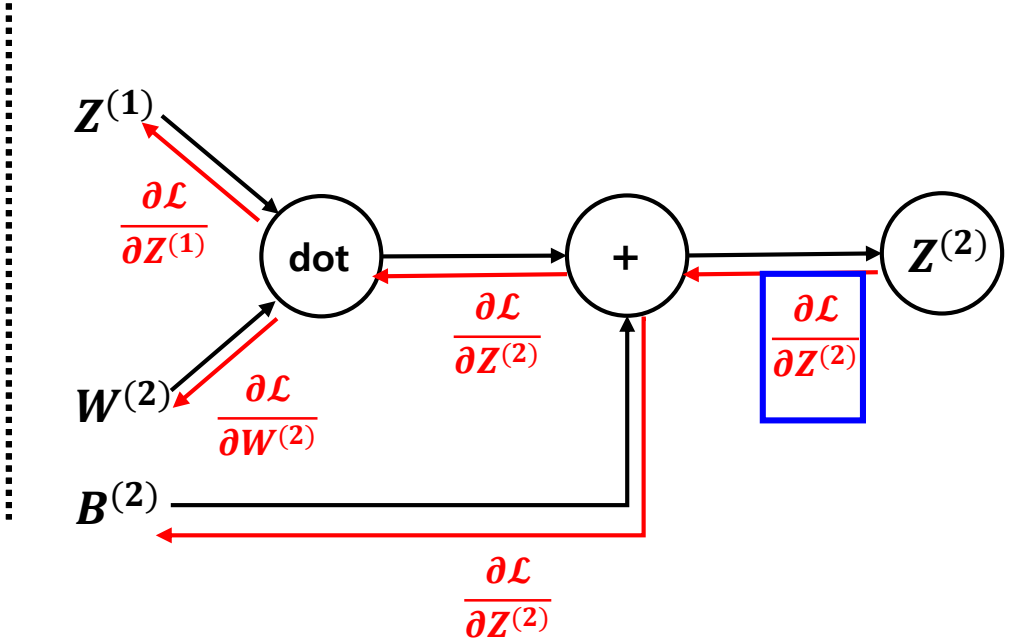




$$A^{(3)} = Z^{(2)}W^{(3)} + B^{(3)},\ Y = h\left(A^{(3)}\right)$$

# CNN in Image Classification: Revisit Backpropagation

- (2) 2nd hidden layer → 1st hidden layer



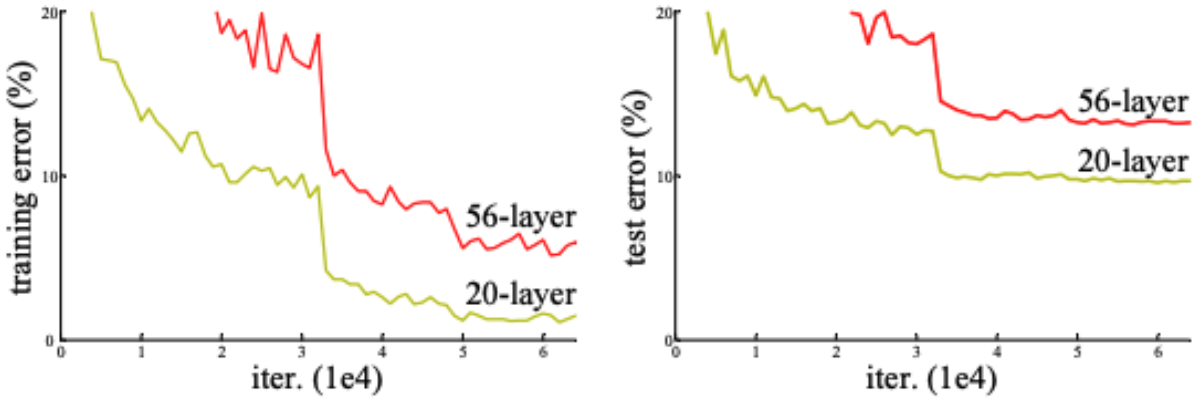

$$A^{(2)} = Z^{(1)}W^{(2)} + B^{(2)}, Z^{(2)} = h\left(A^{(2)}\right)$$

→ 나중 layer에서 계산한 gradient가 이전 layer에서 계산한 gradient에 영향

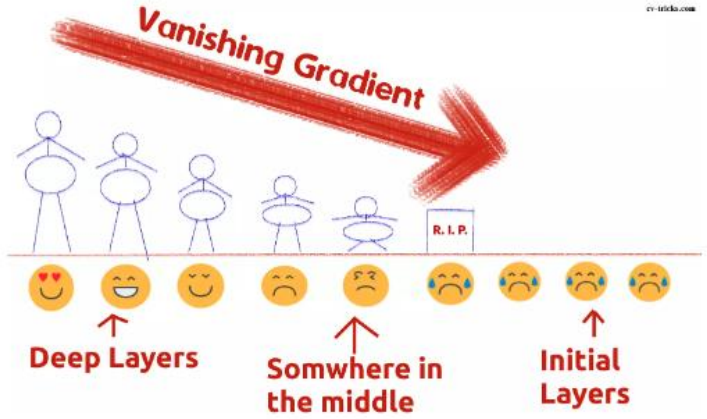
→ 여기서, (나중 layer, 이전 layer)는 두 layer중에서 각각 (output에 가까운 layer, input에 가까운 layer)를 의미

# CNN in Image Classification

- **Case 4: ResNet**
  - Vanishing Gradient
    - 152 Layers (엄청나게 깊은 layer)
    - 기존에는 이렇게 쌓을 생각을 못했을까?
      - 오히려 층을 쌓을수록 어느 순간부터 성능이 감소
      - Vanishing Gradient: 네트워크의 깊이가 깊어질수록 초기층에 gradient가 전달되지 않아 학습 X




**(Layer를 더 쌓았는데 training error가 더 높음)**

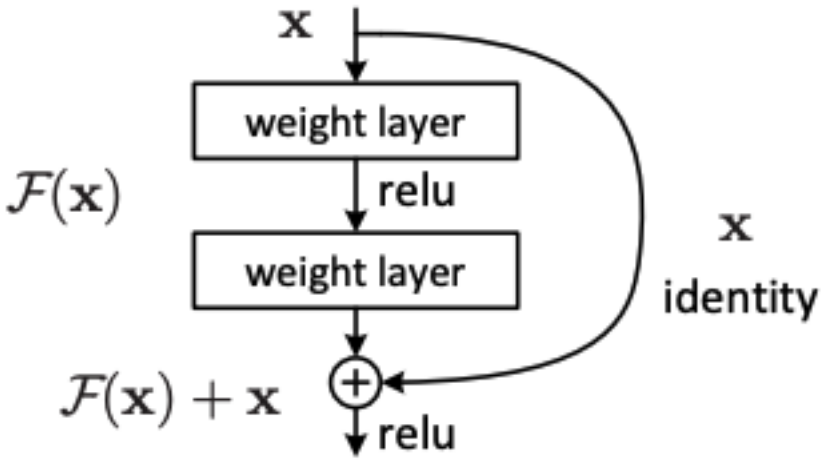



**(Vanishing Gradient)**

(출처: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778)., https://wikidocs.net/164800)

# CNN in Image Classification

- **Case 4: ResNet**
  - Residual Block
    - 이런 vanishing gradient를 해결하기 위해 도입한 방법
    - 일정 시점마다 input x자체를 skip connection을 통해 연결
      - Gradient Flow가 원활하게 이루어짐 (모델 깊이에 대한 부담 해소)
      - Fully connected layer제거 (parameter수 감소)

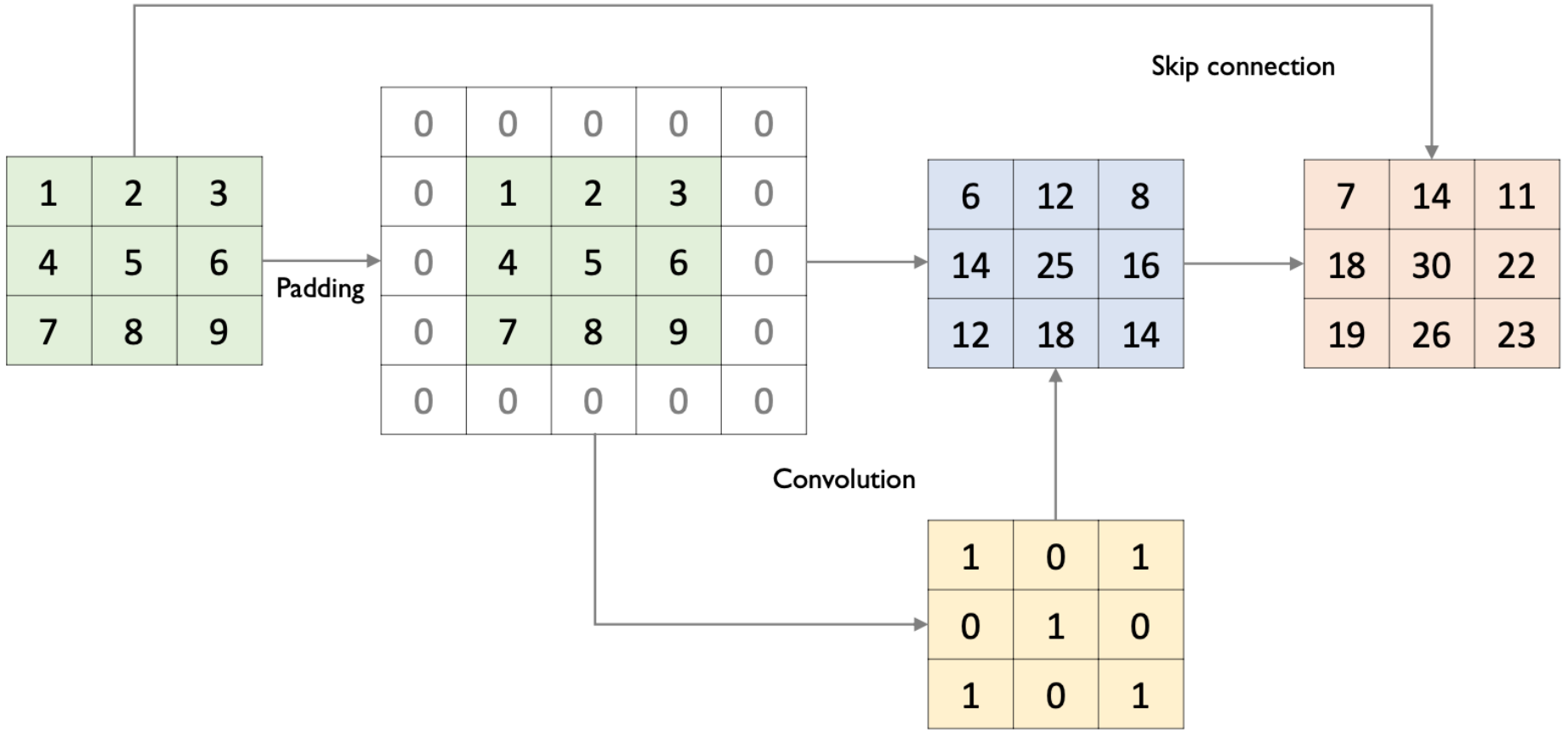



**(Residual Block)**

(출처: He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778)., https://wikidocs.net/164800)

# CNN in Image Classification

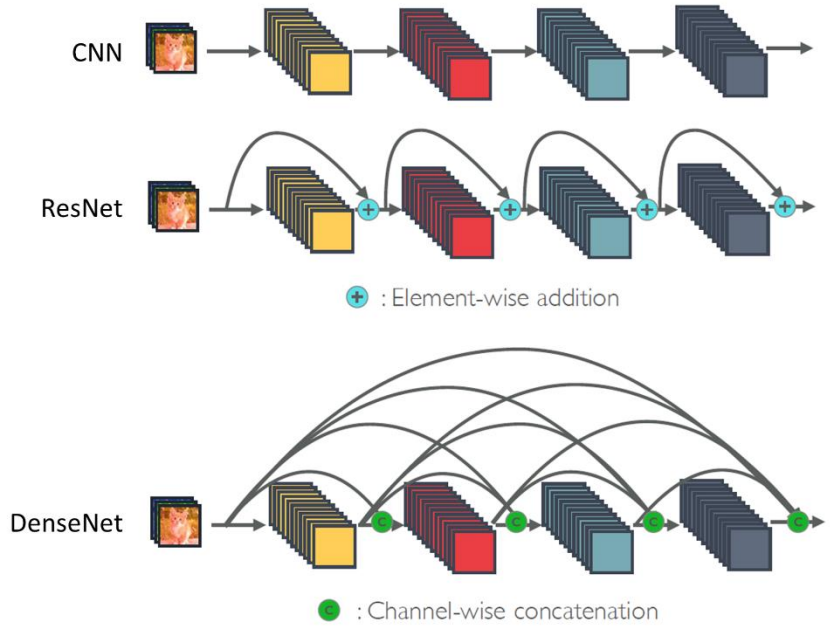
- **Case 4: ResNet**
  - Skip Connection 예시



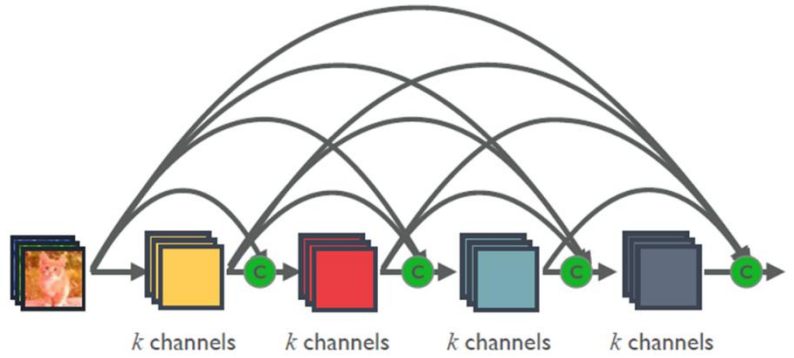

(출처: https://github.com/pilsung-kang/Machine-Learning-Basics-Bflysoft/blob/master/Lecture%207_Deep%20Neural%20Network_CNN.pdf)

# CNN in Image Classification

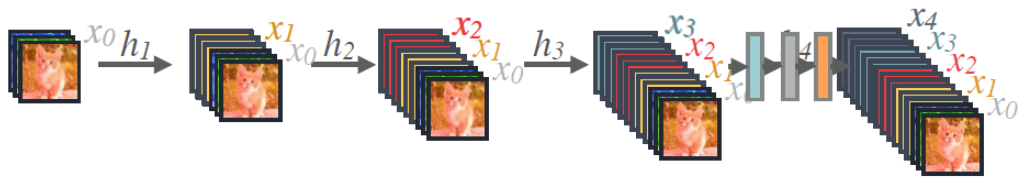
- **Case 5: DenseNet**
  - Skip Connection을 더 늘려보자! → 대신, 정보를 add 하지 말고, concatenate
  - 모든 layer를 다 연결하면 param수 급증 → 각 layer의 channel수를 줄임




**(CNN vs ResNet vs DenseNet)**




**(각 layer의 channel수를 줄임)**

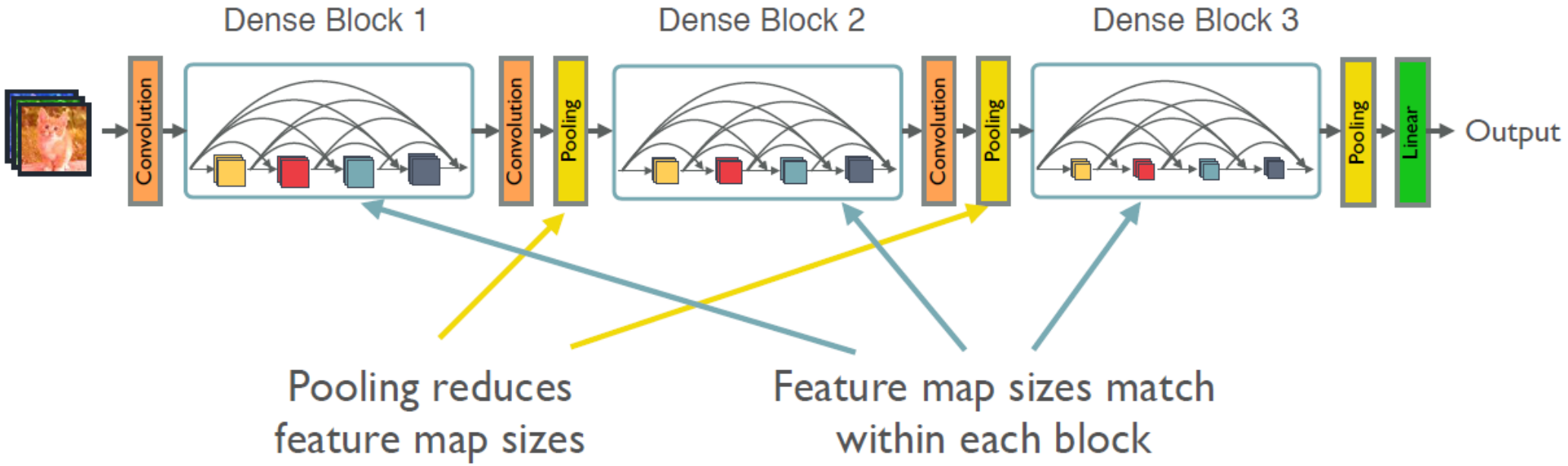


**(Forward Propagation)**

(출처: https://oi.readthedocs.io/en/latest/computer_vision/cnn/densenet.html)

# CNN in Image Classification

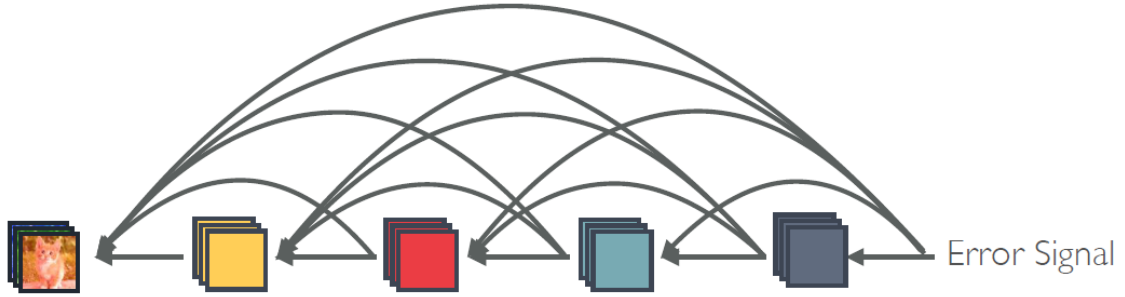
- **Case 5: DenseNet**
  - 아무리 그래도, 모든 layer를 다 연결하면 param수가...
  - Dense Block내에서 channel concatenate



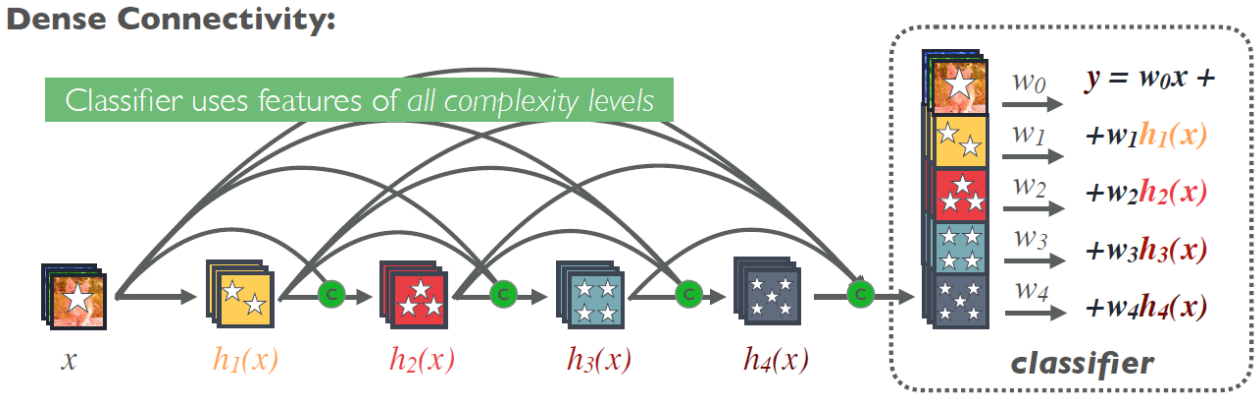

(출처: https://oi.readthedocs.io/en/latest/computer_vision/cnn/densenet.html)

# CNN in Image Classification

- **Case 5: DenseNet**
  - DenseNet의 장점(1): Gradient 전달이 잘 이뤄짐




  - DenseNet의 장점(2): Low Feature도 반영 가능

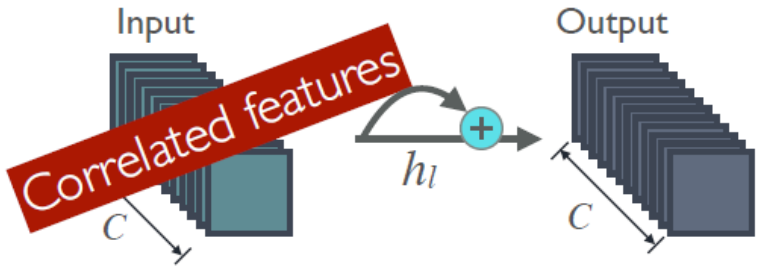



(출처: https://oi.readthedocs.io/en/latest/computer_vision/cnn/densenet.html)

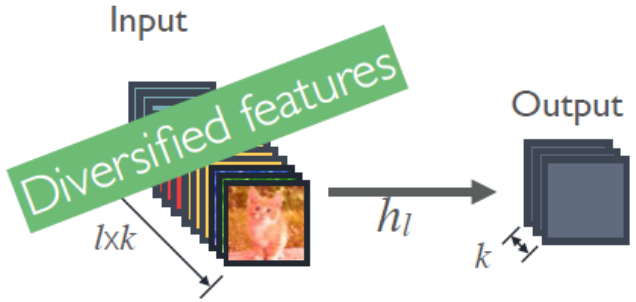
# CNN in Image Classification

- **Case 5: DenseNet**
  - DenseNet의 장점(3): Parameter수가 적음

**ResNet connectivity**:




**DenseNet connectivity:**




#parameters:

$O(C \times C)$

$k << C$

$O(l \times k \times k)$

$k$: Growth rate

(출처: https://oi.readthedocs.io/en/latest/computer_vision/cnn/densenet.html)

# 3. Class Activation Map (CAM)

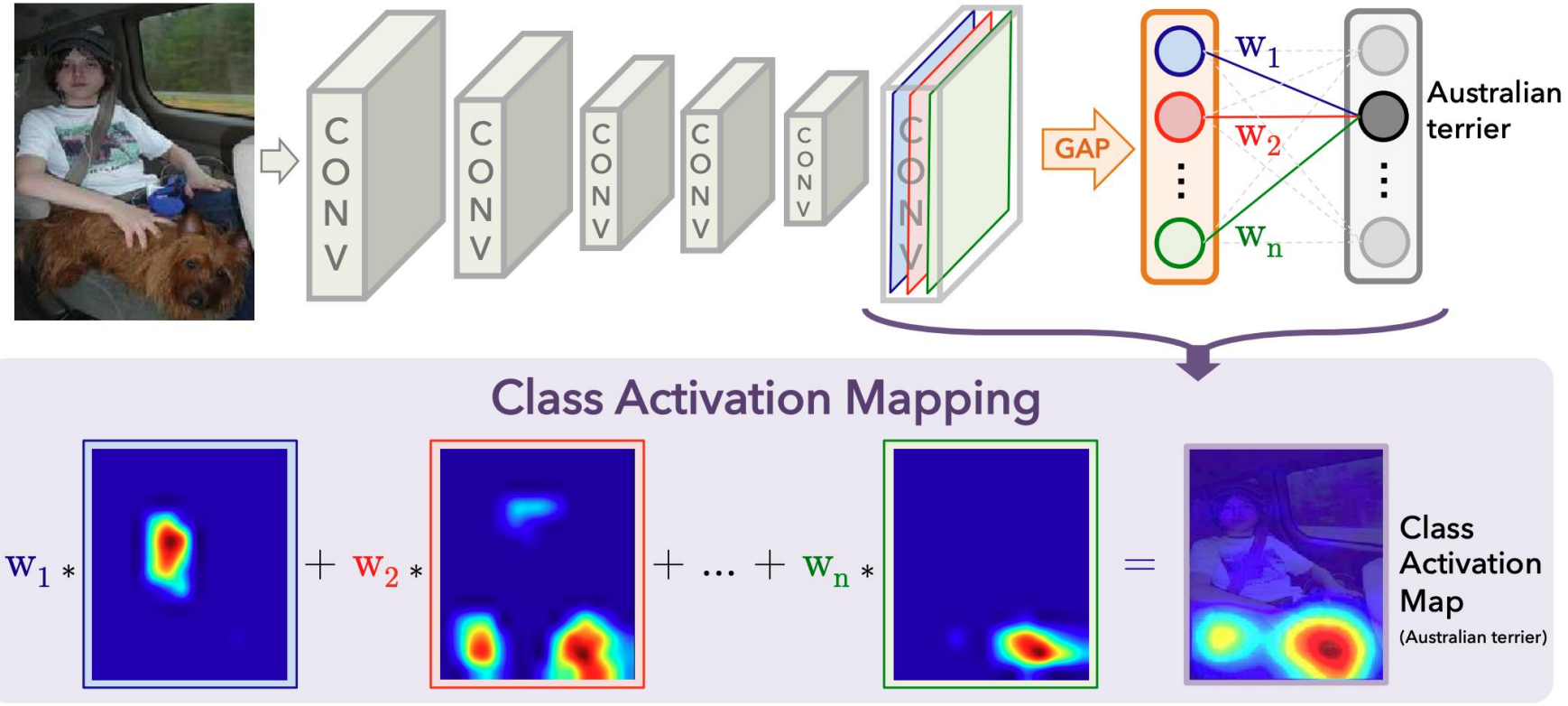
# CNN Localization

- **Image Localization**
  - Class Activation Map (CAM)
    - Convolution layer는 객체의 위치를 알고 있음
      - Fully connected layer를 쓰면 이게 망가짐
    - Classifier로 localization수행
      - Convolution layer만으로 모델을 구성
      - 마지막 feature map을 class수와 동일하게 생성
      - FC layer를 제거하고 global average pooling으로 출력과 mapping
      - Classifier CNN은 이미 localize 정보를 갖고 있음

(출처: Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2921-2929).)

# CNN Localization

- **Image Localization**
  - Class Activation Map (CAM)
    - Localize Significant areas based only on class label information

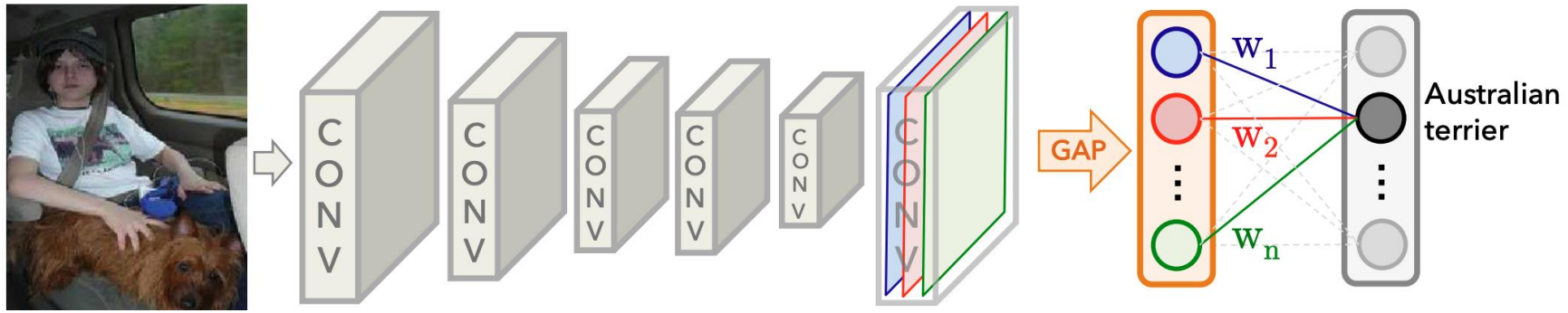



(출처: Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2921-2929).)

# CNN Localization

- **Image Localization**
  - Class Activation Map (CAM)
    - $f_k(x, y)$: 마지막 convolution layer unit k의 $(x, y)$ 위치에서의 activation값
    - $F_k = \sum_{x,y} f_k(x, y)$: global average pooling 결과 값
    - $S_c = \sum_k w_k^c F_k$, where $w_k^c$: 클래스 $c$에 대해 $F_k$의 중요도 → $P_c = \frac{exp(S_c)}{\sum_c exp(S_c)}$
    - $S_c = \sum_k w_k^c F_k = \sum_k w_k^c \sum_{x,y} f_k(x, y) = \sum_{x,y} \sum_k w_k^c f_k(x, y) = \sum_{x,y} M_c(x, y)$

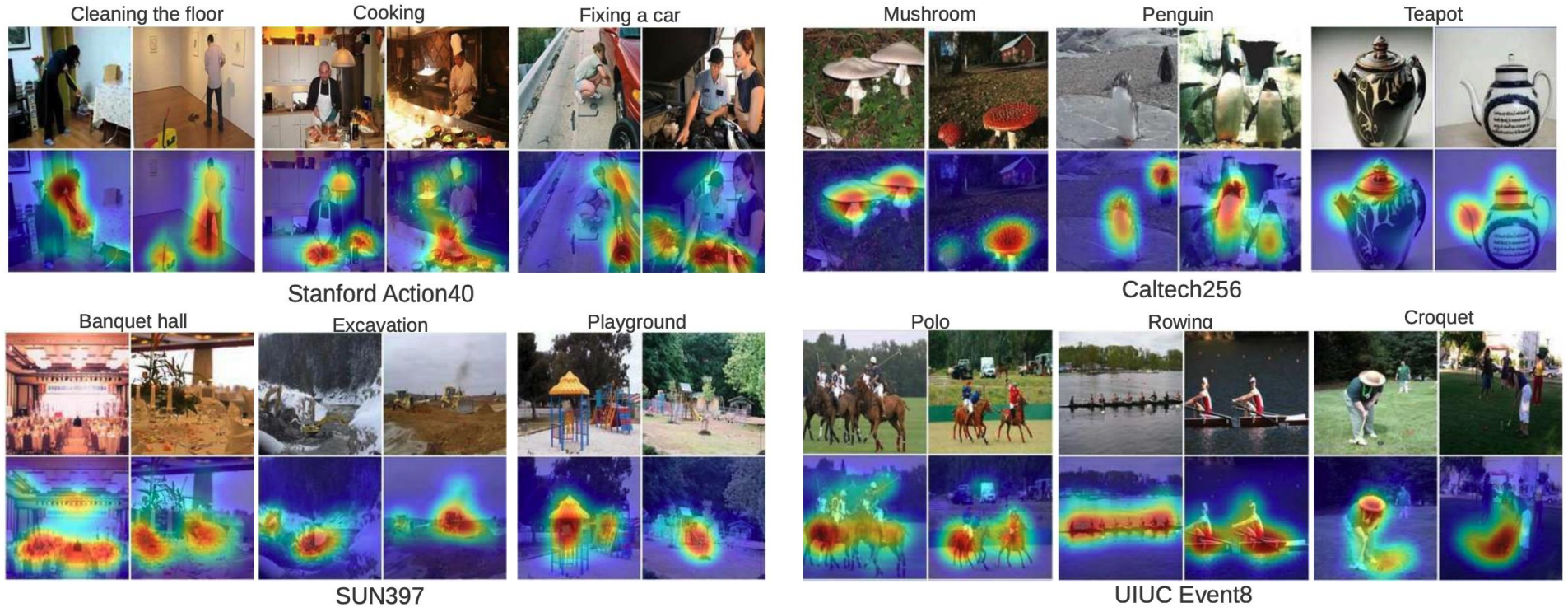
→ **$M_c(x, y)$: $(x, y)$ 위치에서 이미지를 클래스 $c$로 분류하도록 하는 중요도**




(출처: Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2921-2929).)

# CNN Localization

- **Image Localization**
  - Class Activation Map (CAM)
    - 특정 class로 분류했을 때, 가장 큰 영향을 끼치는 이미지 영역을 추정




(출처: Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2921-2929).)

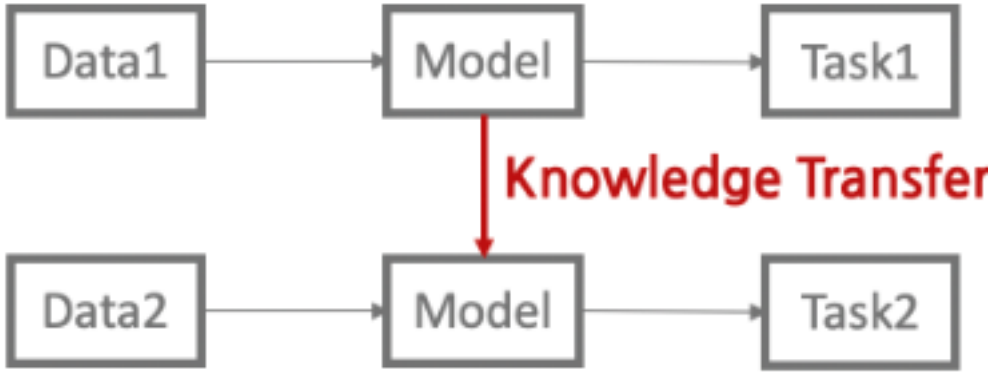
# 4. Transfer Learning

# Transfer Learning

- **Transfer Learning**
  - 특정 task를 학습한 모델을 다른 task 수행에 재사용하는 기법
    - **(1) pre-trained model 및 학습 결과(parameter)를 불러오고**
    - **(2) 우리의 task에 맞도록 모델을 변경/개선**
    - **(3) 우리의 data를 사용하여 모델을 fine-tuning**

→ 대규모 데이터에서 학습한 결과를 기반으로 빠르게 나의 모델을 구축

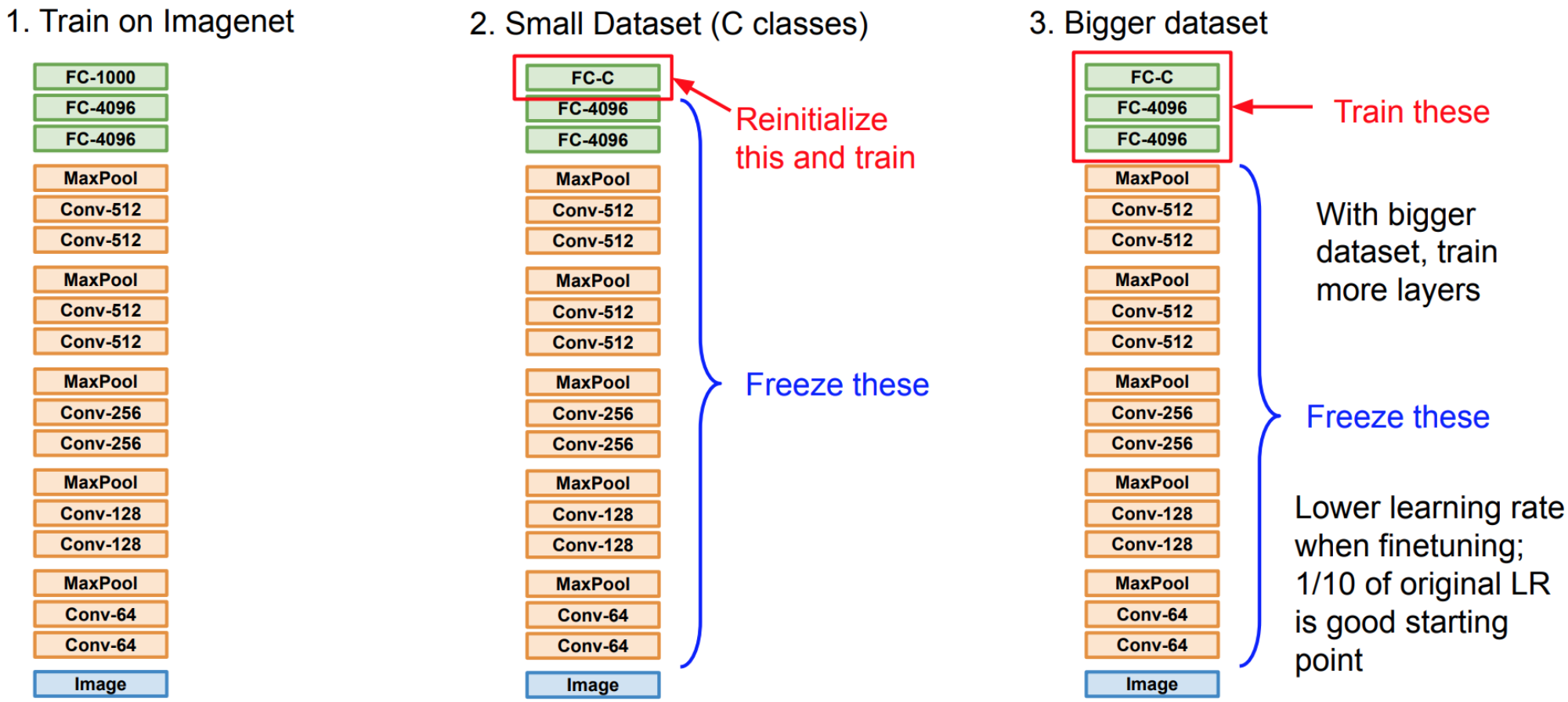
→ 비교적 적은 데이터로 좋은 결과를 얻을 수 있음




(출처: https://ratsgo.github.io/nlpbook/docs/introduction/transfer/)

# Transfer Learning

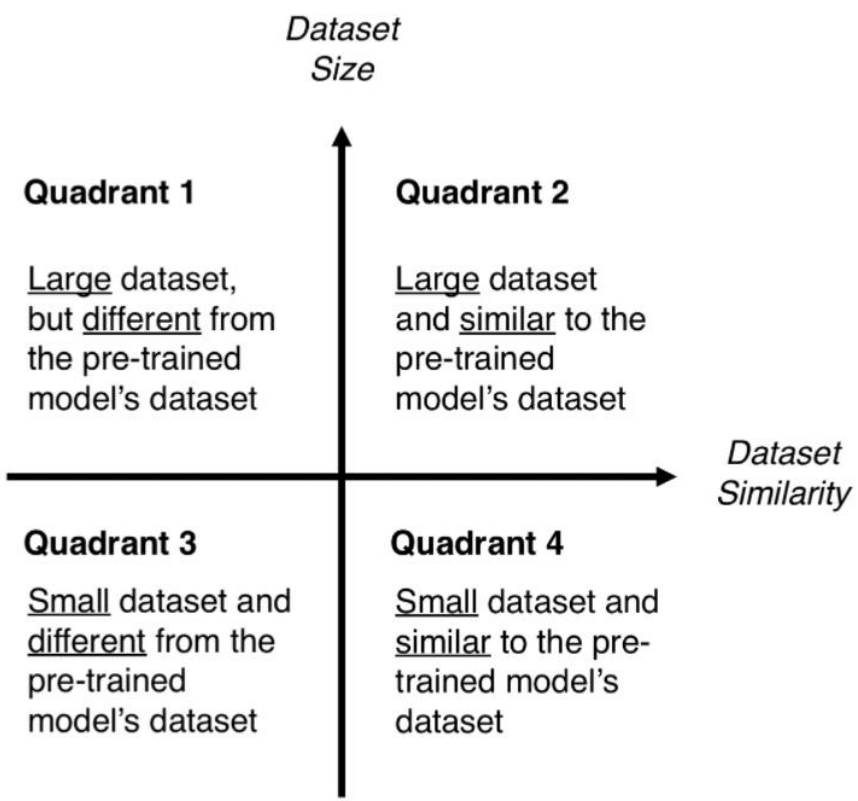
- **Transfer Learning**
  - 특정 task를 학습한 모델을 다른 task 수행에 재사용하는 기법

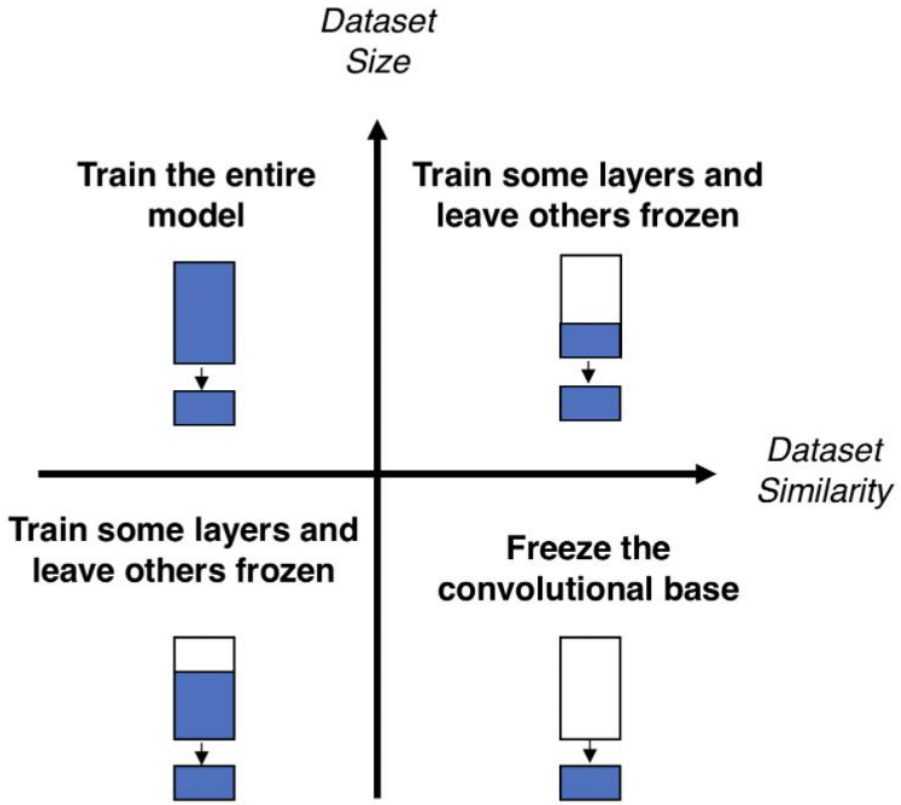



(출처: Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014 Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014)

# Transfer Learning

- **Transfer Learning**
  - 특정 task를 학습한 모델을 다른 task 수행에 재사용하는 기법

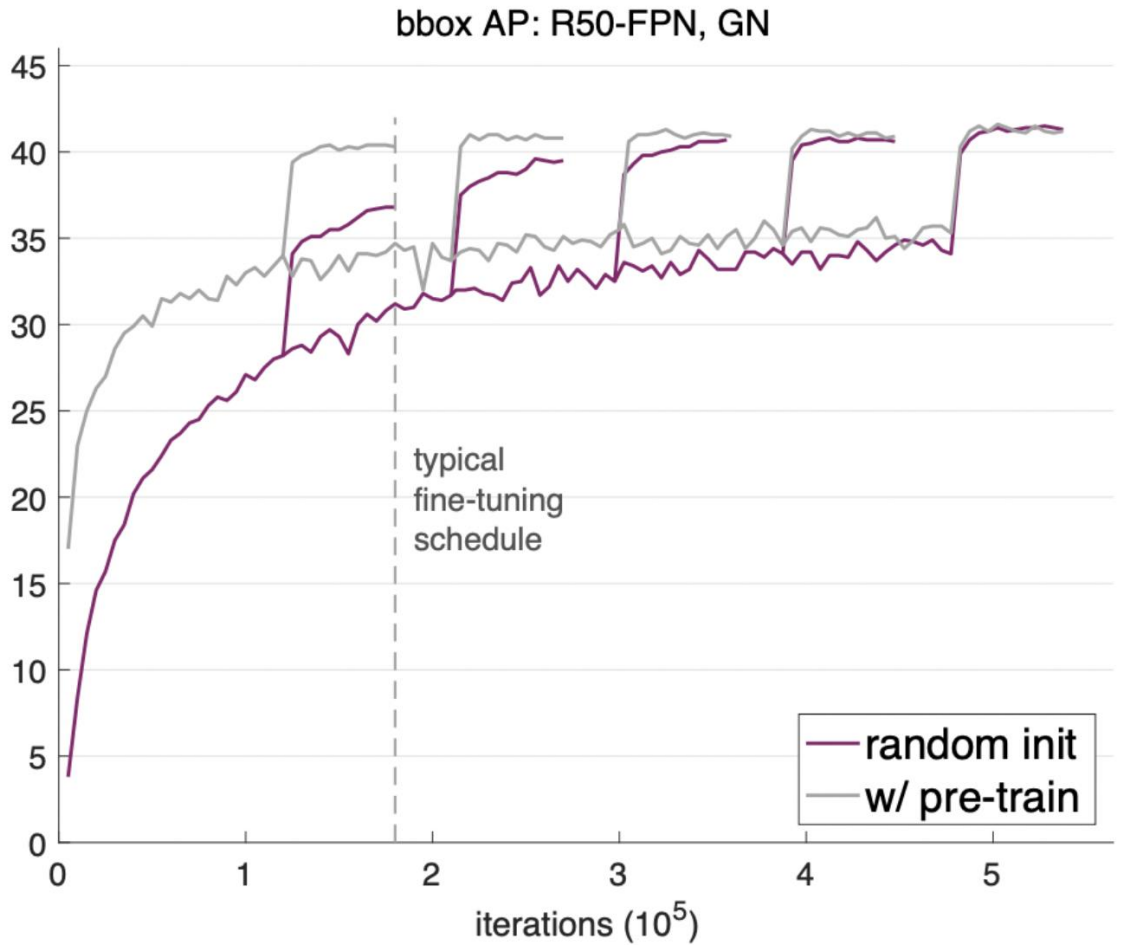



(출처: https://dacon.io/en/forum/405988)

# Transfer Learning

- **Transfer Learning**
  - 학습 속도에서 효과가 있음
    - '밑바닥부터' 학습하는 것보다 2~3배 빠름
    - (성능에는 큰 차이가 없다)




(출처: He, K., Girshick, R., & Dollár, P. (2019). Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 4918-4927).)

# Summary

# Summary

- **Advanced CNN Model**
  - AlexNet: 최초로 conv를 사용. 성능이 괜찮음
  - VGG: AlexNet보다 깊은 구조
  - GoogleNet: Inception Module을 사용
  - ResNet: Skip-connection을 활용하여 gradient vanishing 현상 극복 → 더 깊이!
  - DenseNet: 더 많이 connection 생성. Add대신 channel concatenate

- **Class Activation Map**
  - Class 정보만으로 중요한 영역을 선정

- **Transfer Learning**
  - Pretrained model을 다른 task에 맞도록 모델의 일부만 학습

# End of the documents