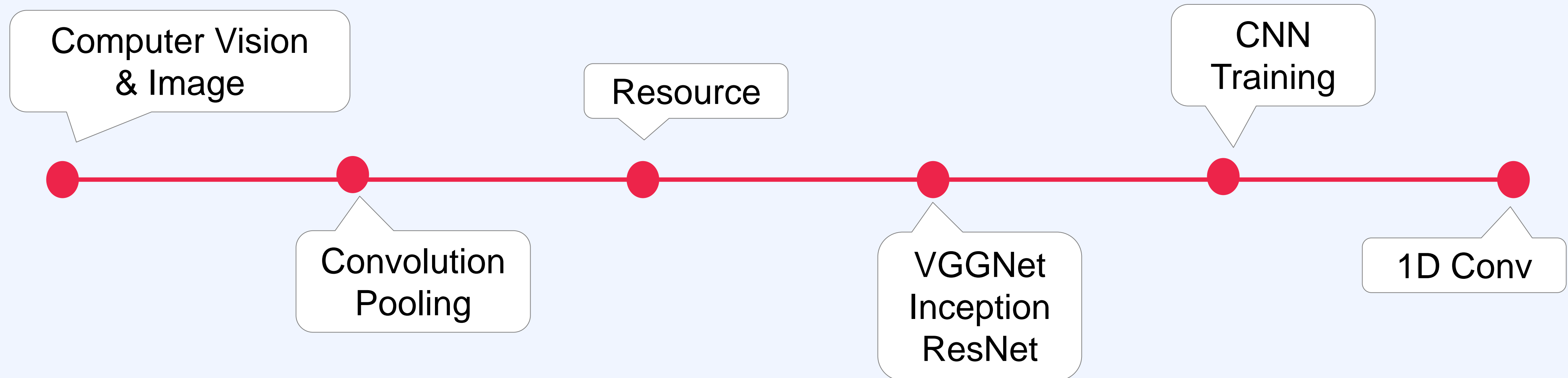


CNN

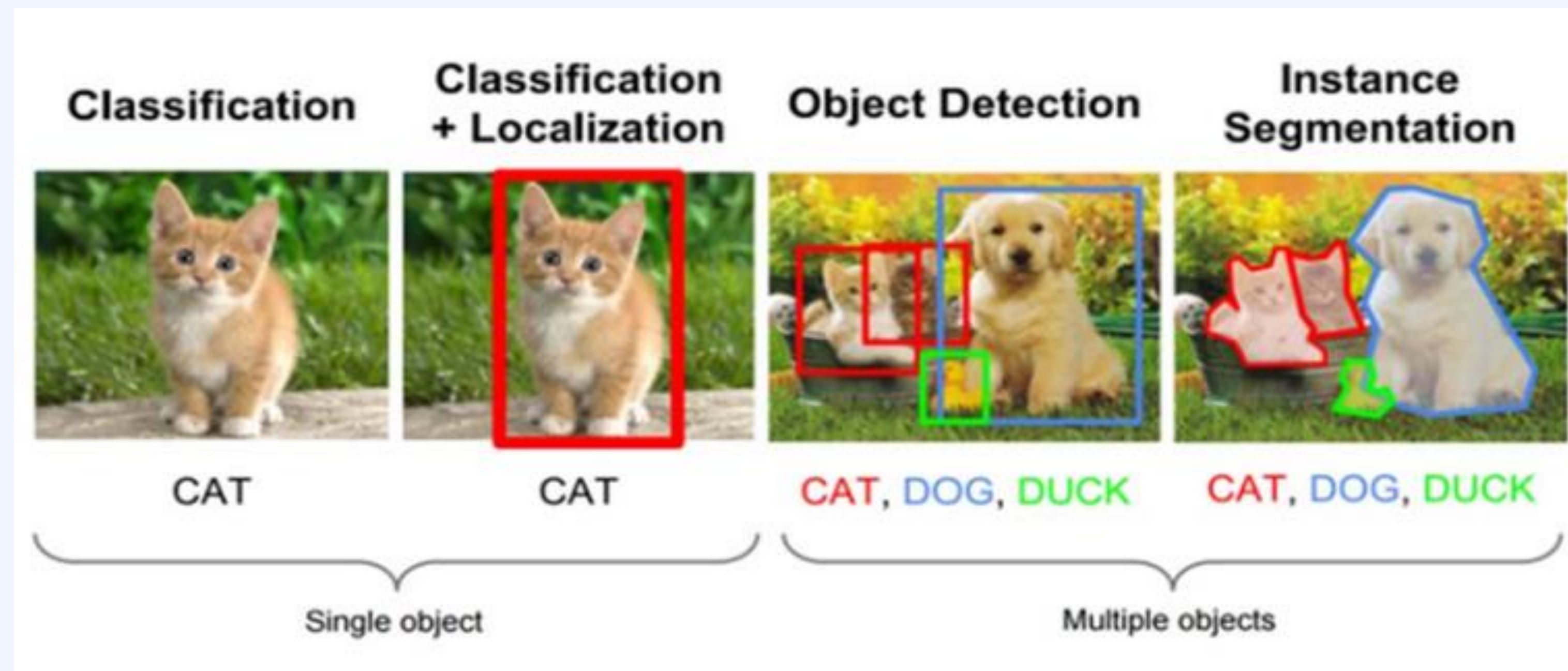
Computer Vision & Image

Contents



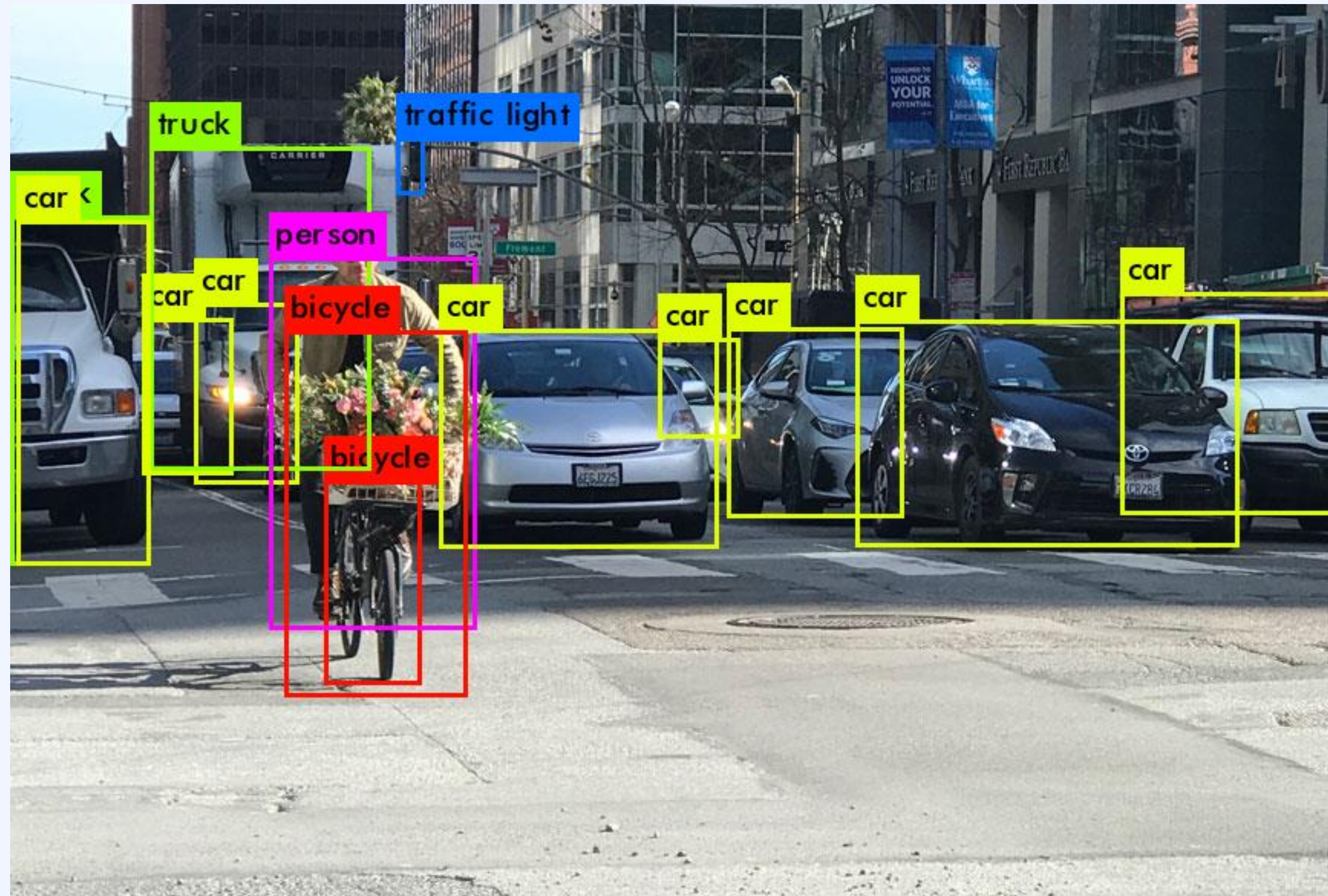
Computer Vision Task

- Image Classification(Single object)
- Localization(Single object)
- Object Detection(Multiple object)



Computer Vision Task

➤ Object Detection



Computer Vision Task

➤ Automatic Image captioning



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



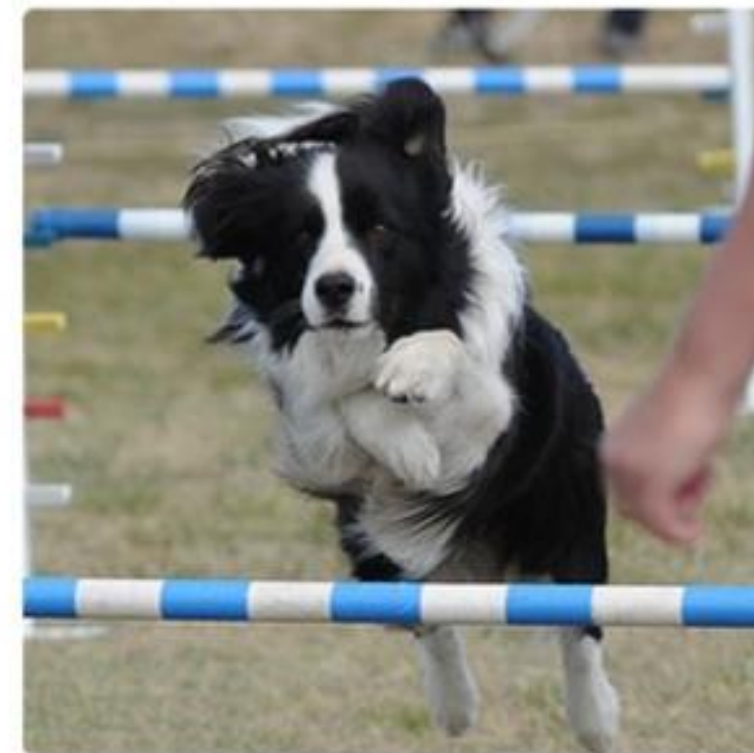
"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."

Computer Vision Task

➤ Translation



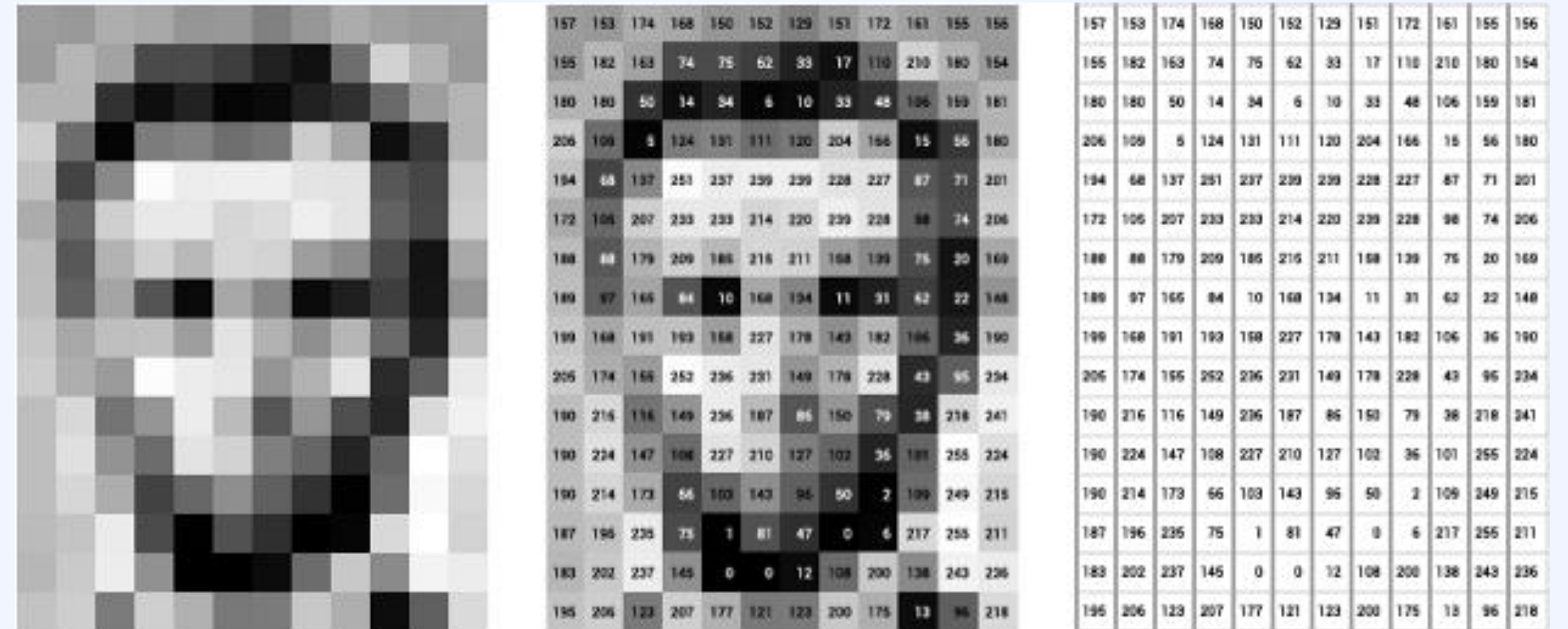
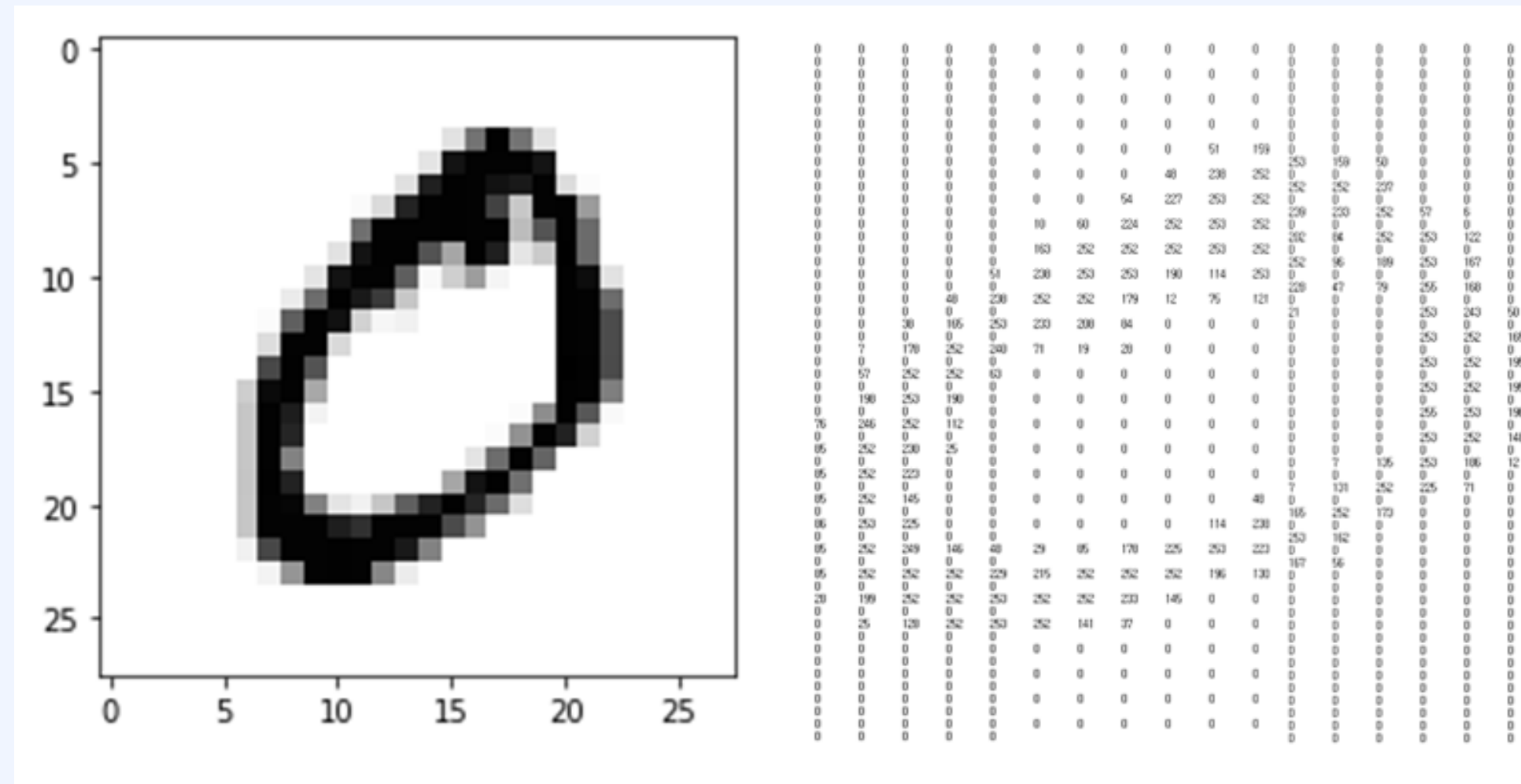
<https://www.wsj.com/articles/BL-268B-1420>



<https://mashable.com/article/word-lens-japanese>

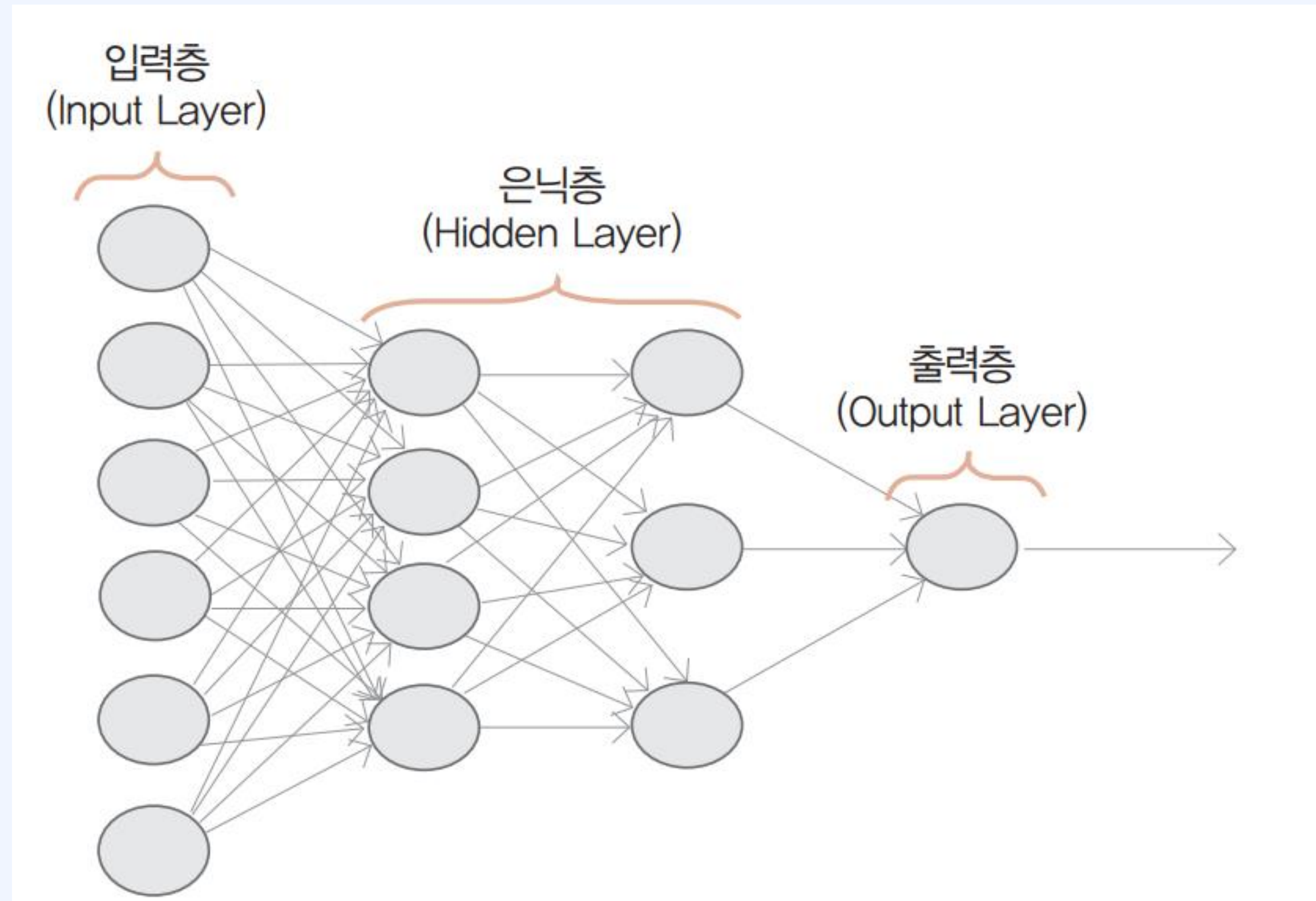
Image?

➤ 이미지도 숫자다



Review

➤ Basic Neural Network

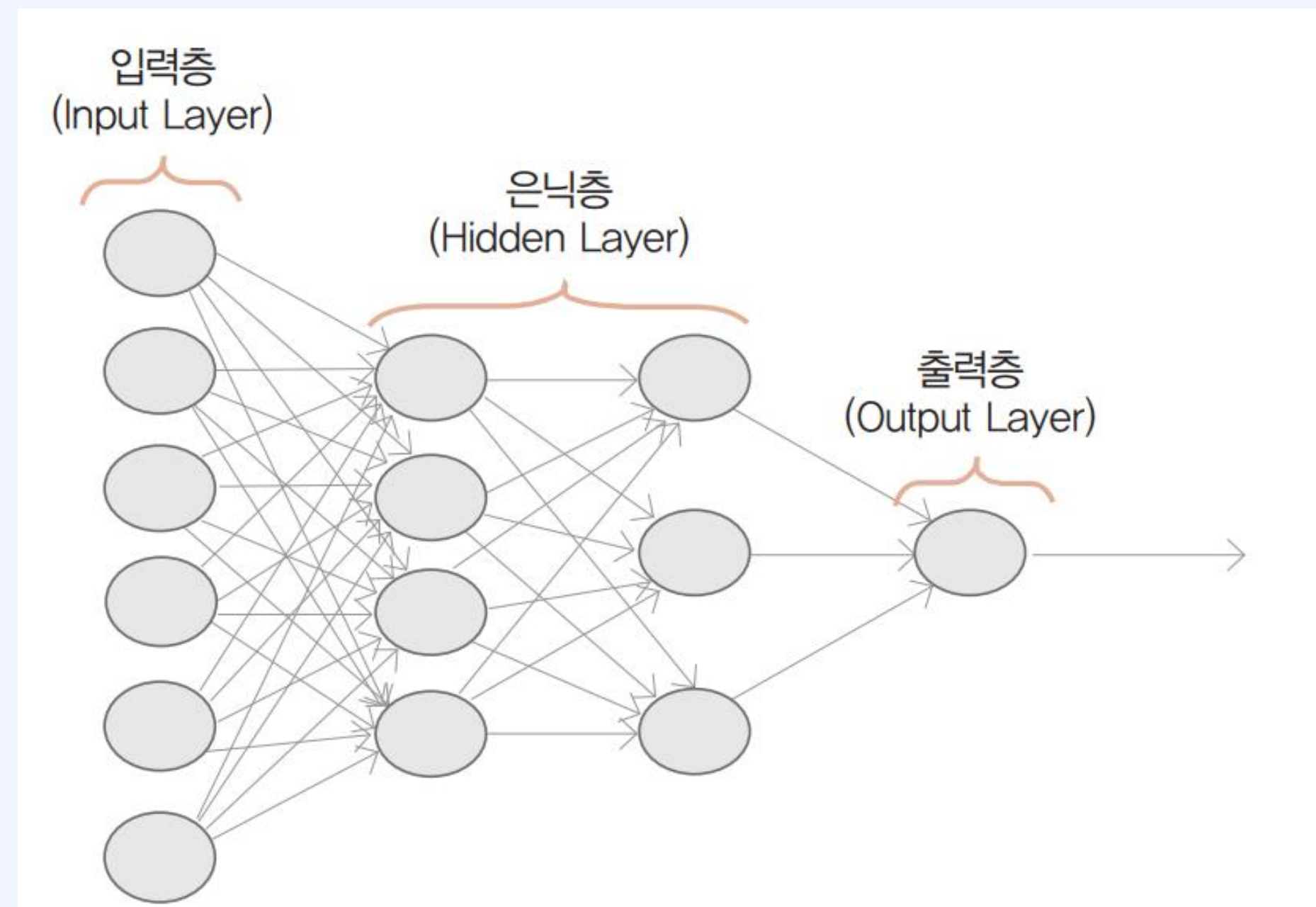
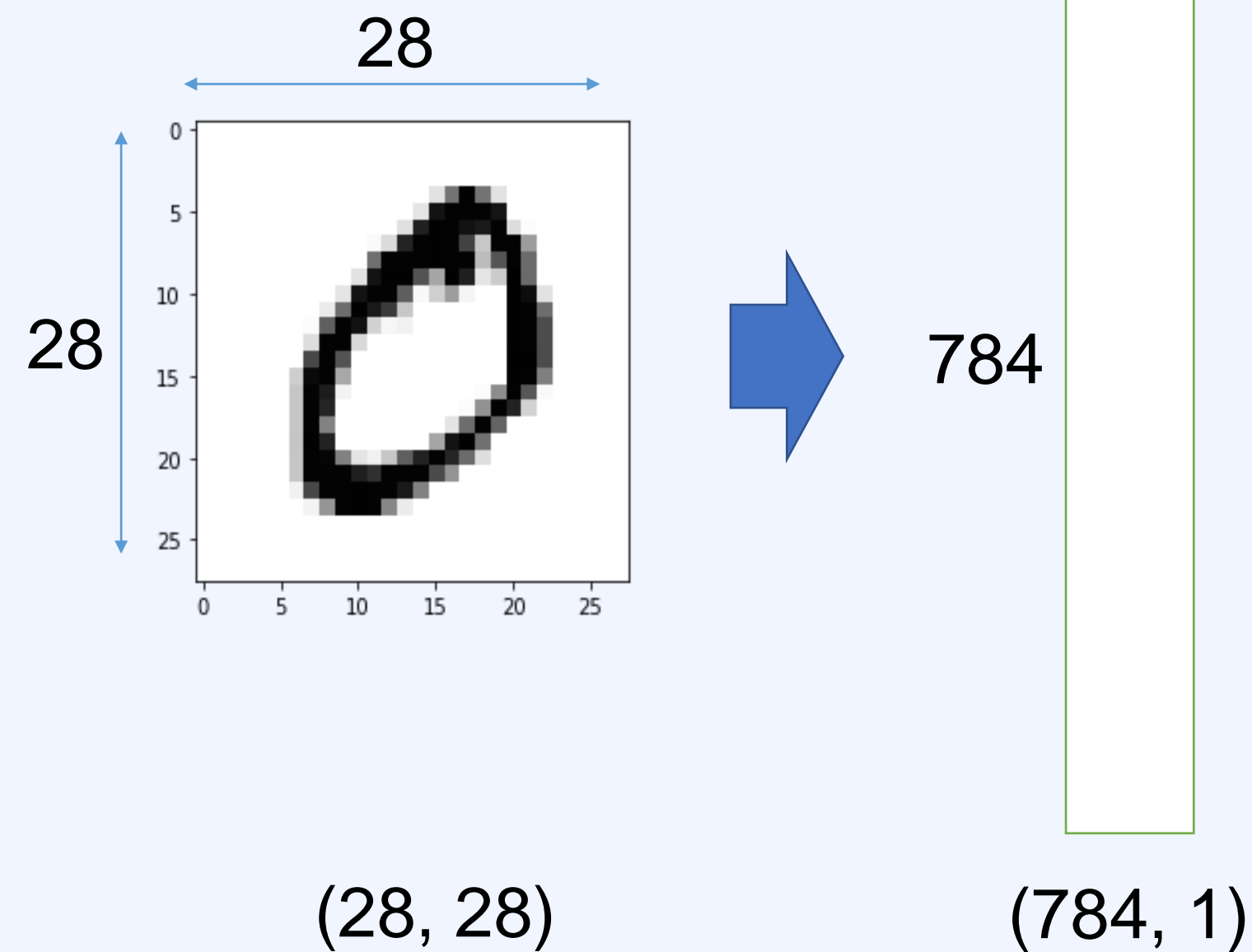


출처: 파이썬으로 시작하는 머신러닝+딥러닝(아이리포)

Review

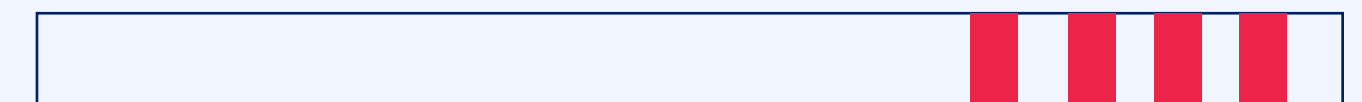
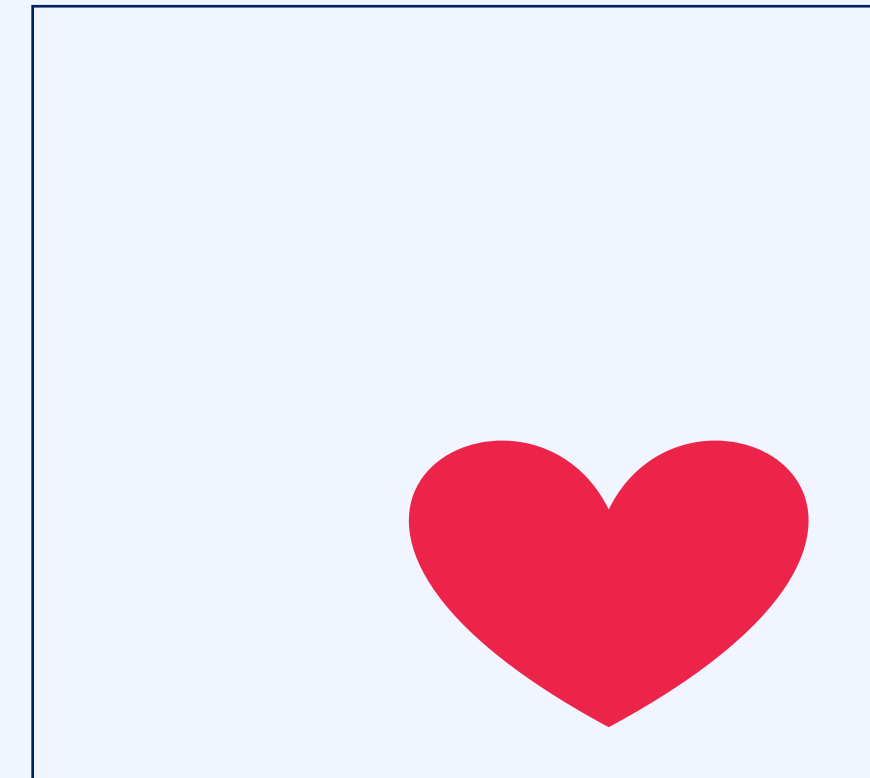
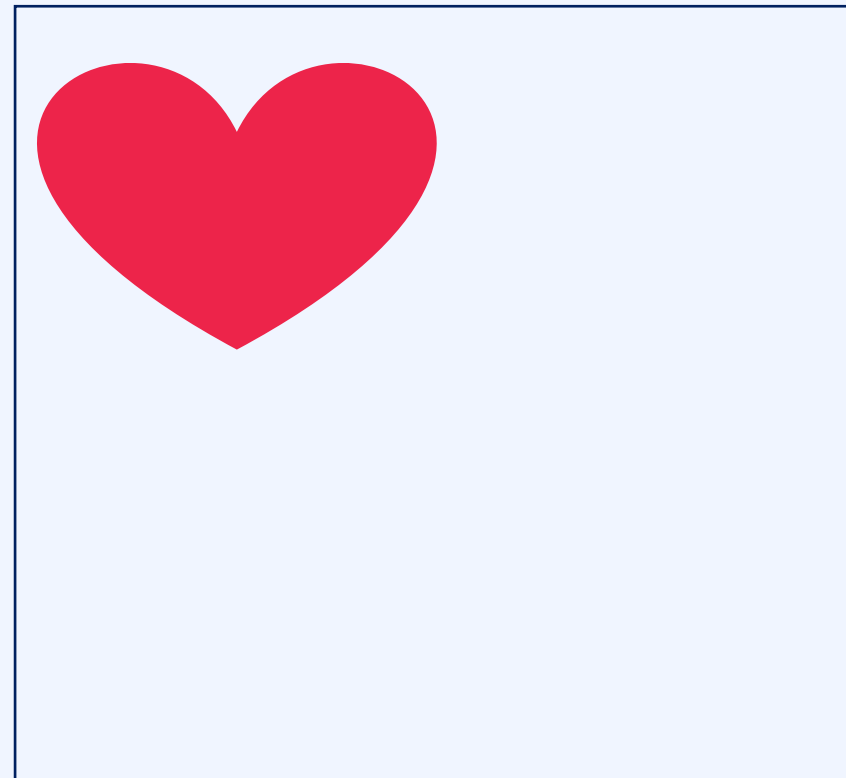
➤ Basic Neural Network

- 입력: 784(28*28)
 - 2차원 데이터를 1차원으로 변형하여 사용: $28 * 28 = 784$ 차원의 입력벡터
- 은닉층: ??
- 출력층: 10 (0~9까지의 숫자)

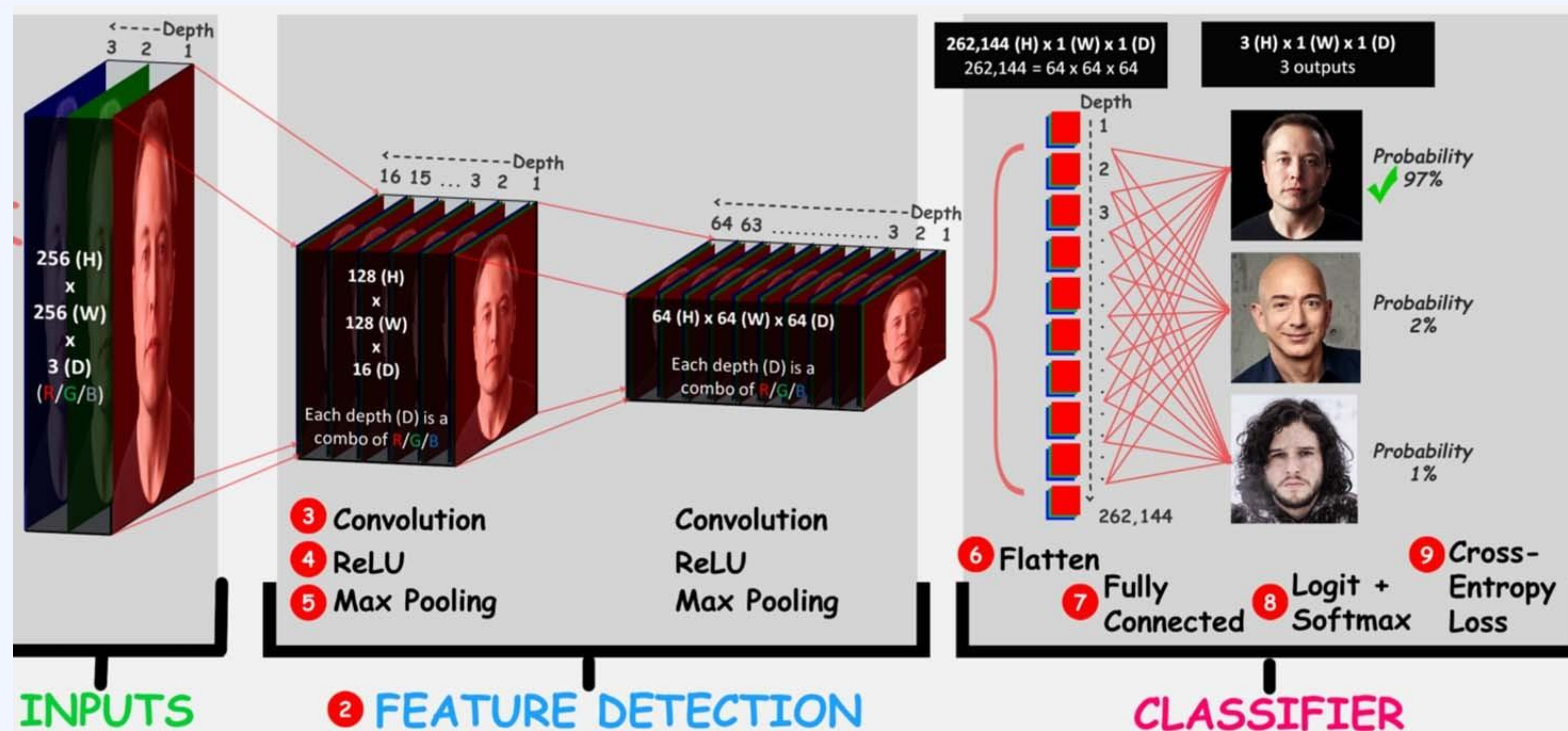


Why?

➤ 2D를 1D로 변환하면 이미지 고유의 특성을 잃어버린다....



What?

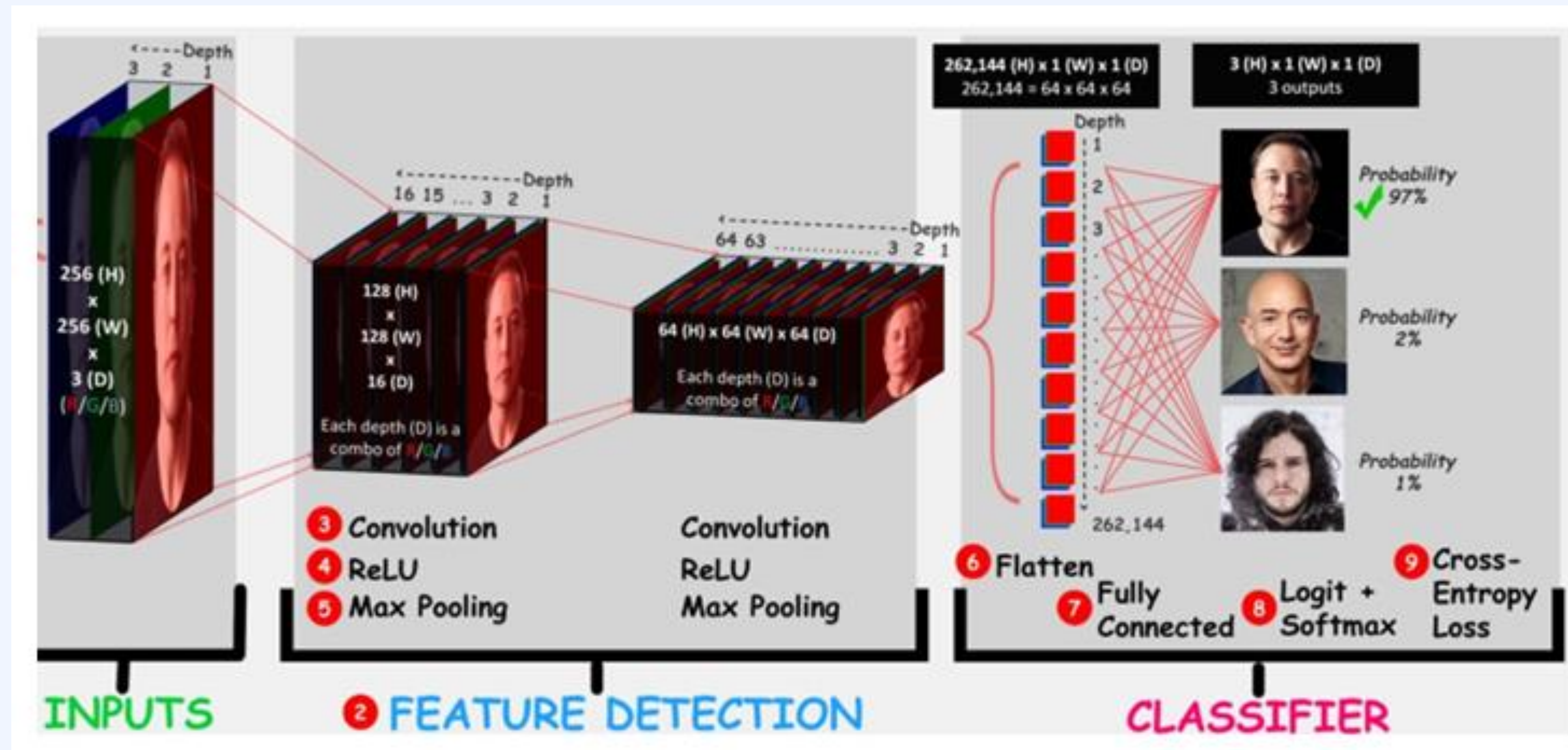


CNN

Convolution

Convolution

➤ Image classifier의 구성



Convolutional Neural Network

➤ CNN이란?

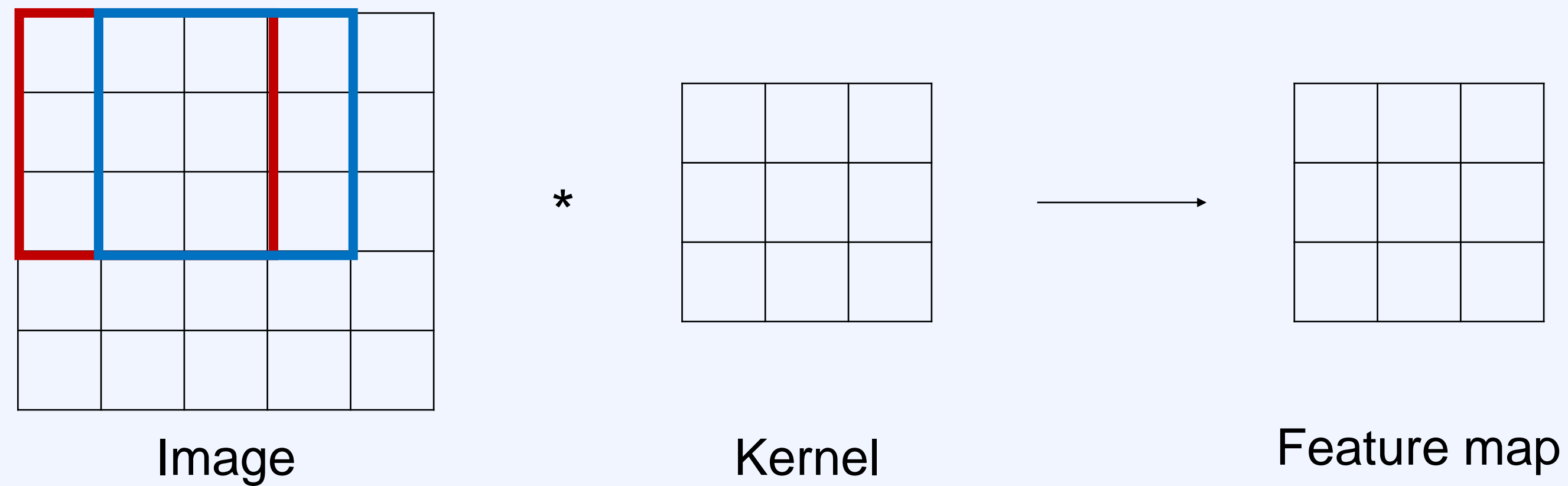
- Convolution Layer로 구성된 이미지 처리에 좋은 성능을 가지는 인공 신경망
- 이미지 처리에만 사용되는 것은 아니지만, 이미지처럼 고차원 데이터에서 탁월한 성능을 보이기때문에 이미지 데이터를 활용한 태스크에 많이 활용됨
- 구성
 - Convolutional layer → pooling layer(subsampling) → Convolutional layer → pooling layer... → fully connected layer로 구성
 - 각 layer를 단순히 교차해서 조합하는 것은 아니고, 목적에 따라 조합하는 방법은 무궁무진함
 - Pooling layer가 항상 존재하는 것은 아님



Convolution

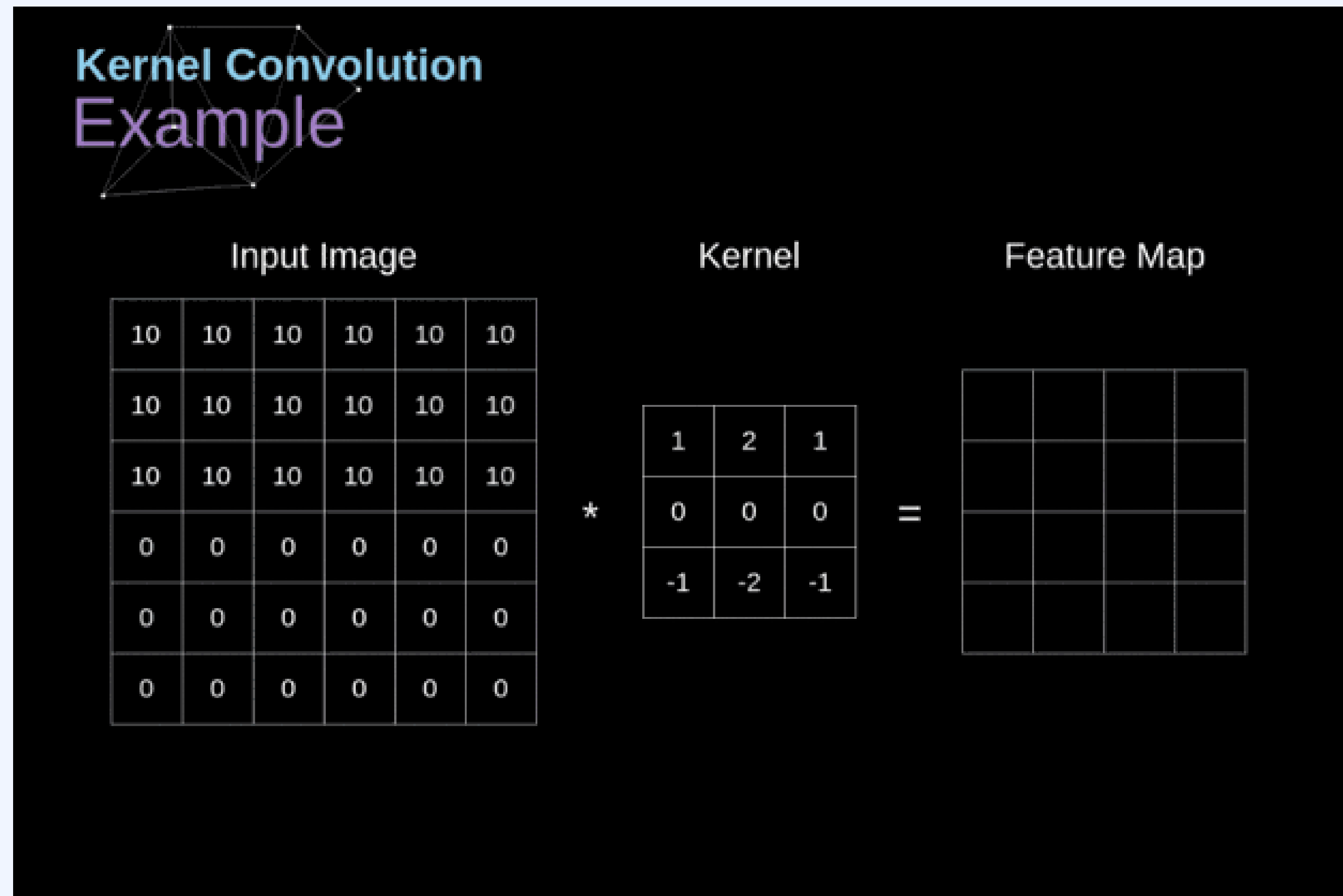
➤ Convolution 연산

- 데이터의 특징을 추출하기 위해 단계별로 다양한 필터(커널) 적용
- 필터 적용시 마다 이미지 왼쪽 위에서 오른쪽 아래까지 밀면서 곱하고 더하는 연산을 수행
- 효과
 - 이미지의 특징점을 효과적으로 찾는데 활용
 - 적은 수의 가중치로 이미지 처리를 효율적으로 할 수 있음



Convolution

➤ Convolution 연산



Convolution

➤ Kernel? Feature Map?

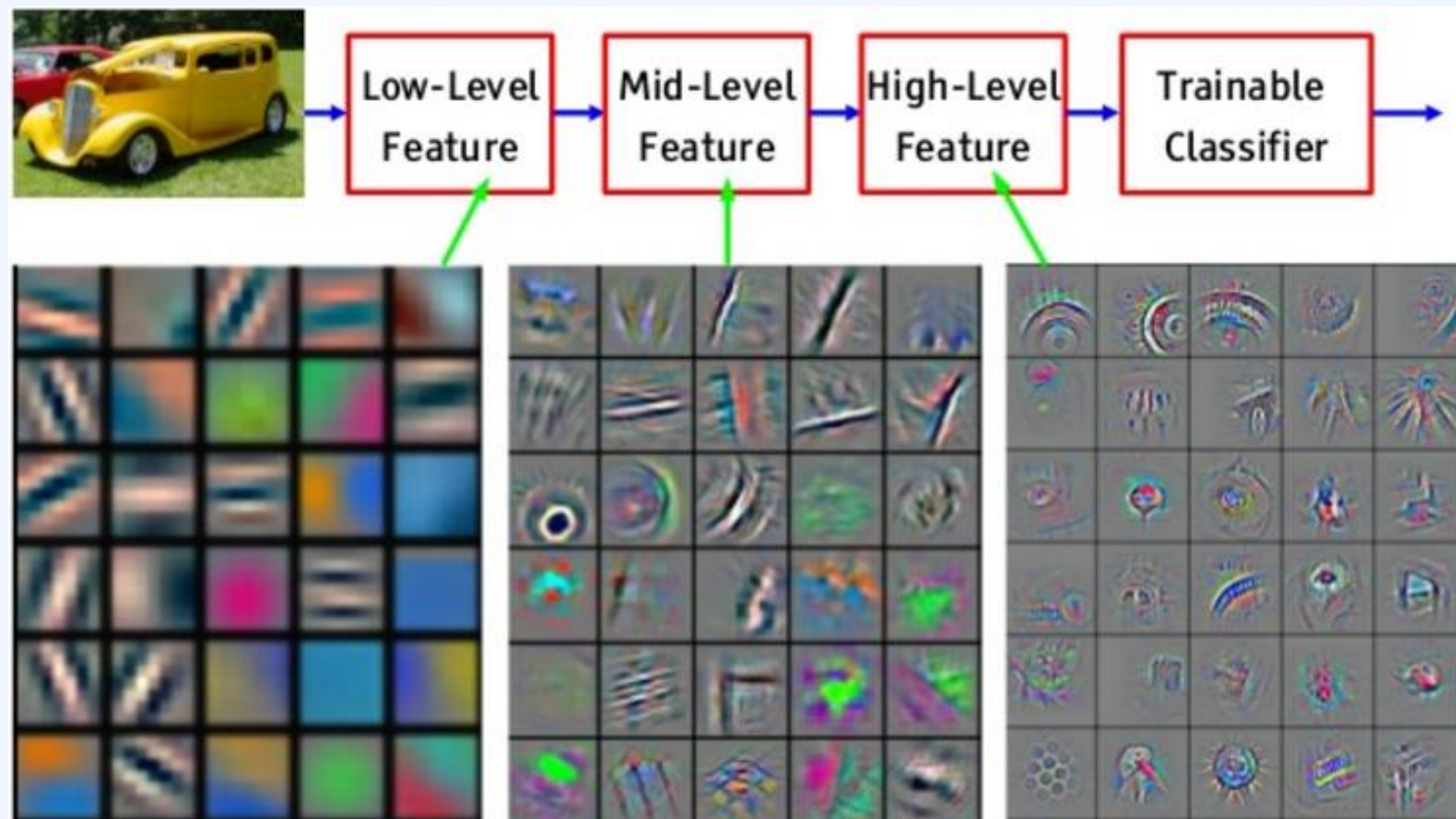


Input

Convolution

➤ Feature map

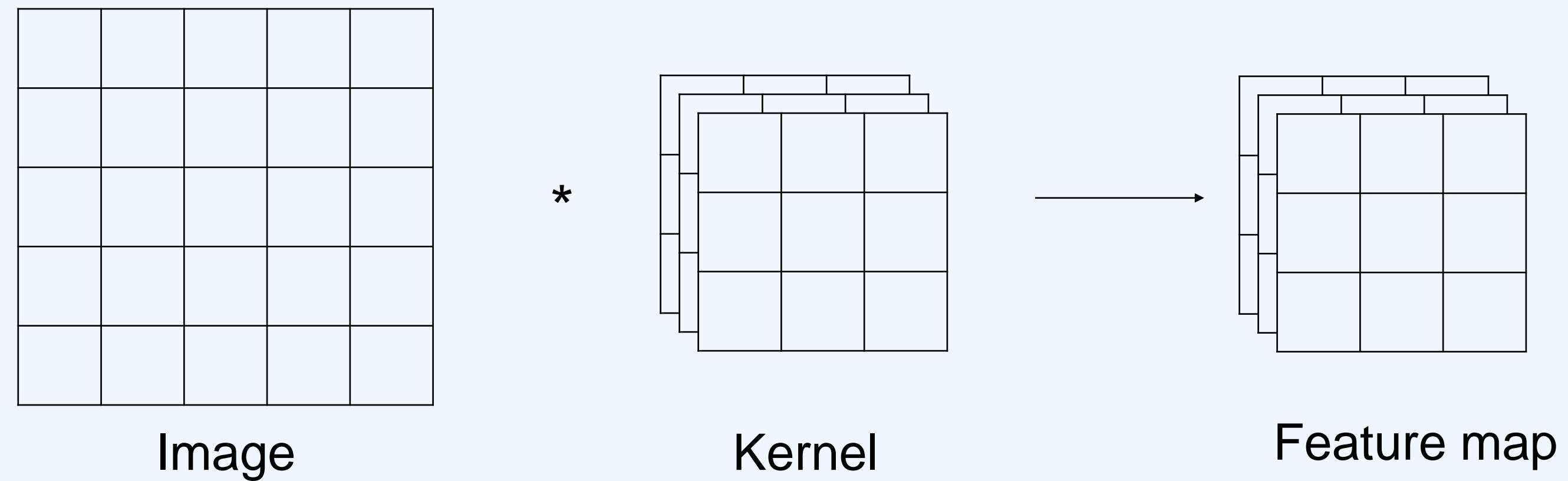
- Convolution 연산을 수행할수록 점점 복잡한 형태의 고차원 특징을 추출할 수 있음



Convolution

➤ Kernel

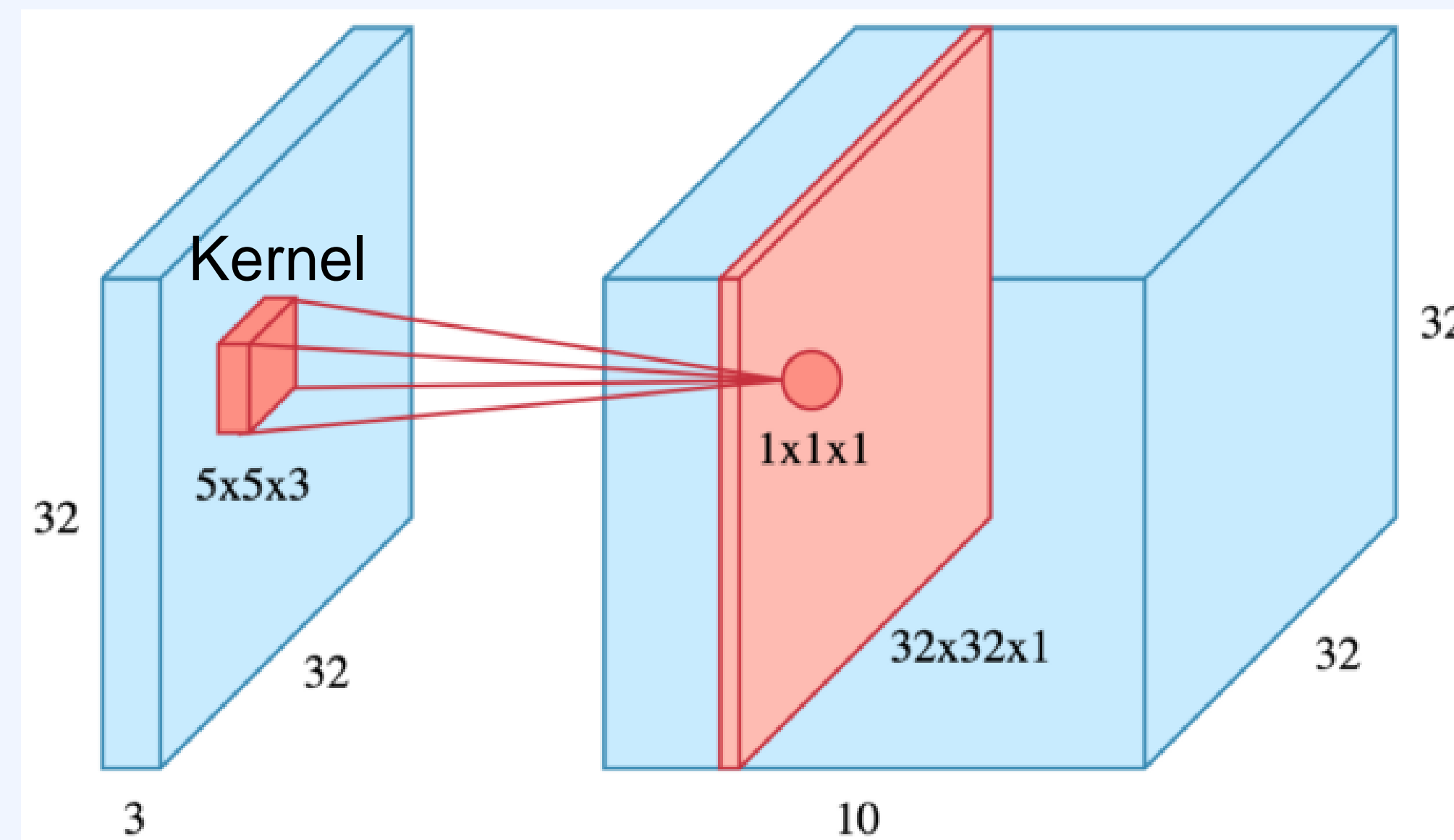
- Convolution 연산 수행 시 여러 개의 Kernel을 적용하여 다양한 형태의 Feature를 추출



Convolution

➤ Kernel 그리고 Channel

- 입력 이미지: $32 \times 32 \times 3$
- 필터 사이즈: $5 \times 5 \times 3$
- 서로 다른 필터를 10개 사용: $32 \times 32 \times 10$
 - 32×32 가 유지되는 이유는 padding



Image

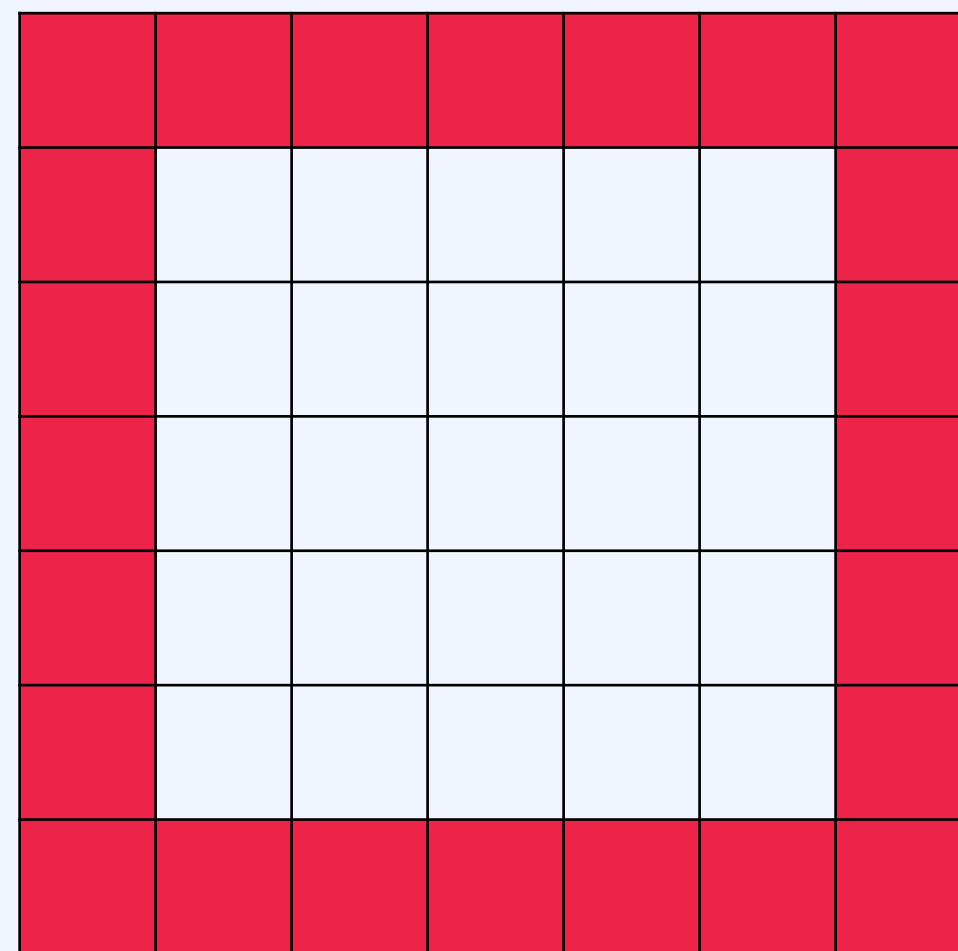
Feature map

Convolution

➤ Padding

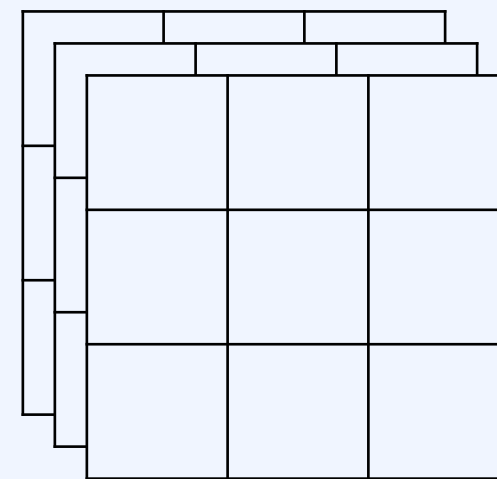
- 원본 이미지 사이즈가 줄어드는 것을 보완

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

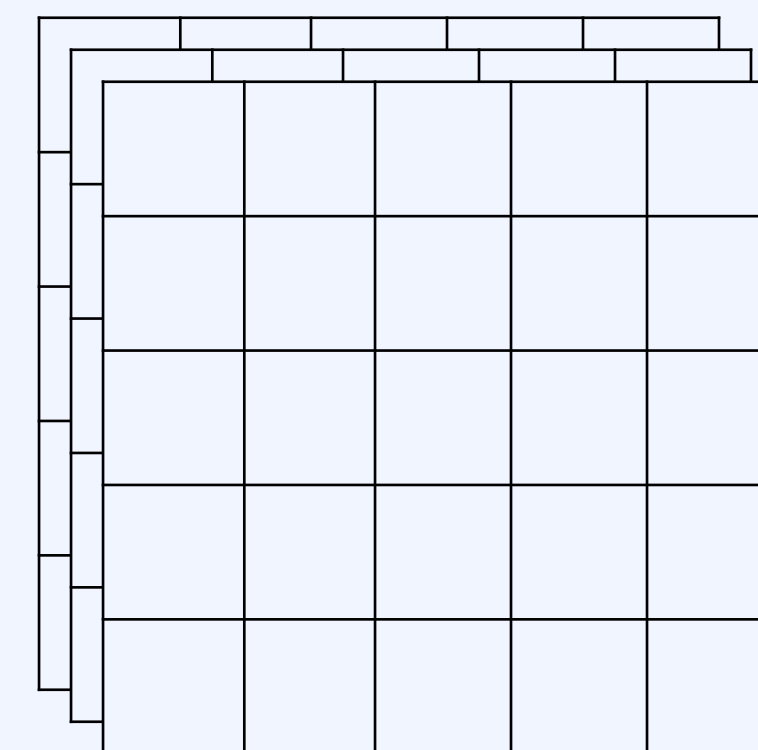
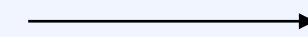


Image

*



Kernel

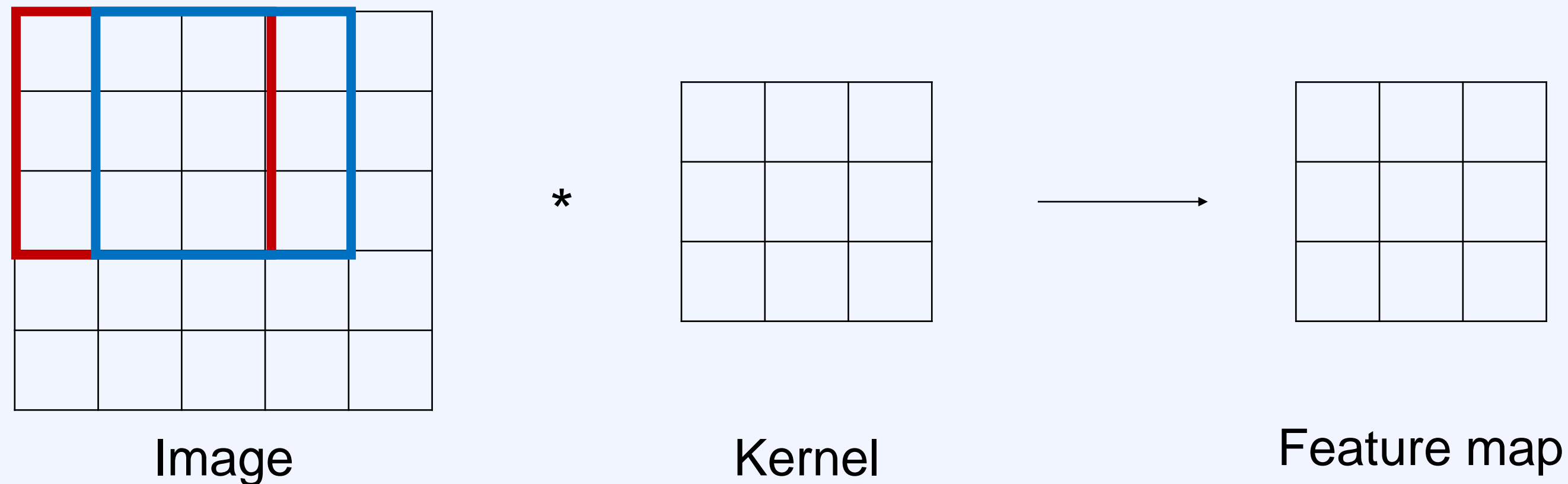


Feature map

Summary

➤ 컨볼루션(Convolution)

- 필터(Filter), 커널(Kernel)
 - 이미지에 겹치는 작은 필터 3x3 또는 5x5 사용
 - 학습을 통해 자동으로 적합한 필터 생성: 이미지의 특징을 추출하는 필터 학습
- 패딩(Padding)
 - 입력 이미지 주변을 0으로 감싸서, 이미지 크기가 줄어드는 것을 보완
- 스트라이드(Stride)
 - 필터를 몇 칸 씩 건너뛰며 적용할 지 조절하는 값
- 특징 맵(feature map)
 - Convolution 연산을 거쳐 나온 새로운 이미지유형



Quiz

Q

다음 중 Convolution 연산의 특징으로 옳지 않은 것은?

- ① Convolution은 합성곱 연산으로 연산 결과 Feature map을 생성한다.
- ② padding을 적용하면 출력 Feature map의 사이즈가 줄어들지 않는다.
- ③ Convolution 연산 수행 시 다양한 특징을 추출할 수 있도록 kernel 값을 설정해야 한다.
- ④ Convolution 연산을 수행할수록 점점 복잡한 형태의 고차원 특징을 추출할 수 있다.

해설

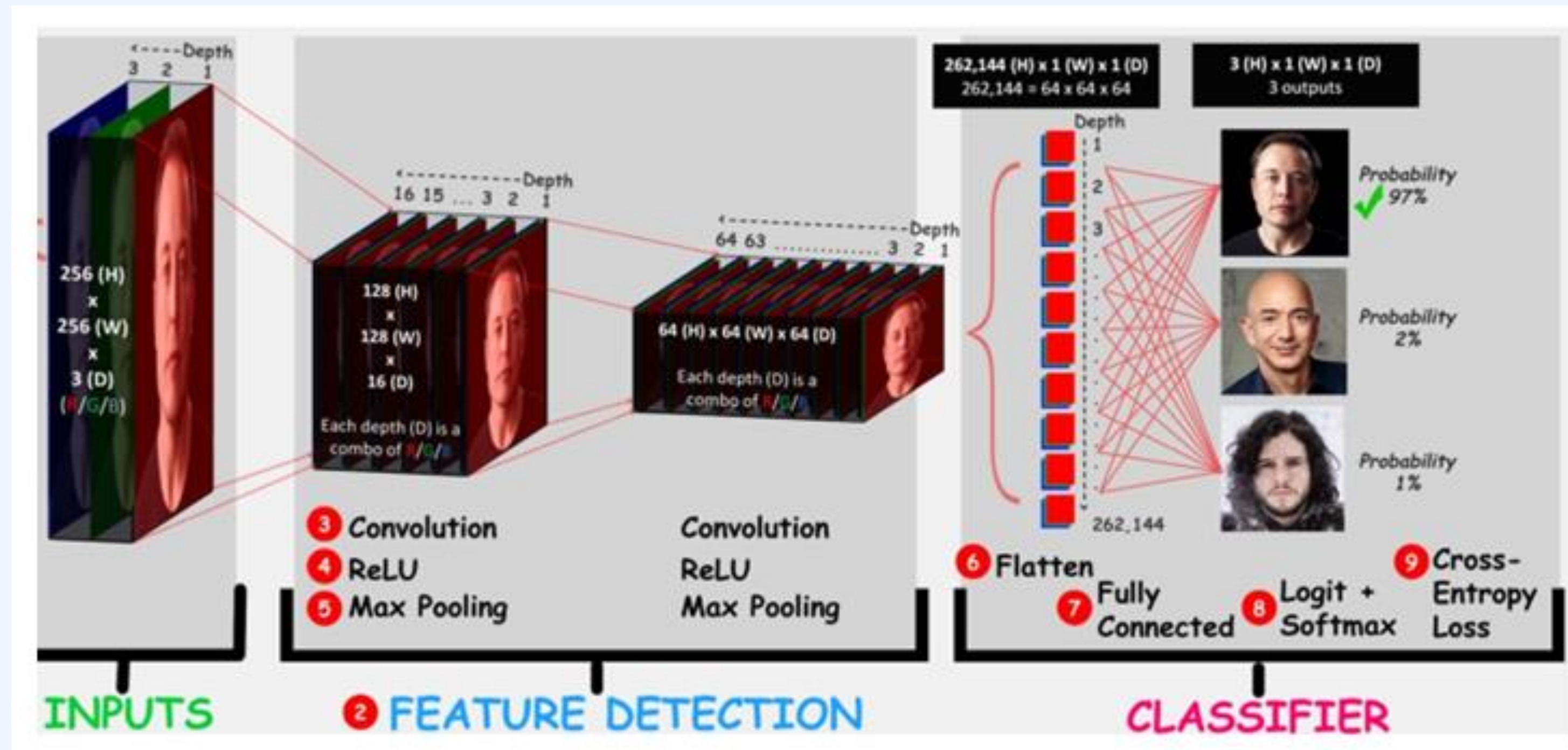
- ▶ kernel의 값은 학습을 통해 업데이트되는 값이다.

CNN

Pooling & Fully Connected

Pooling & Fully connected

➤ Image classifier의 구성

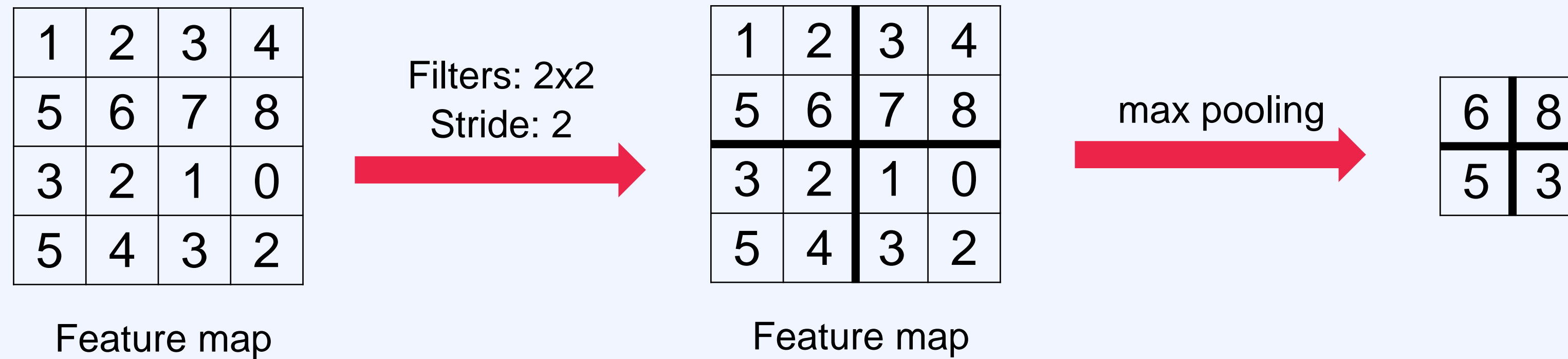


Pooling layer

➤ Pooling

- 필터를 거친 여러 특징 중 가장 중요한 특징 하나를 선택하여 feature 수를 줄임 (Sub-sampling)
- 평균값(average-pooling)이나 최대값(max-pooling) 선택
- 학습해야 할 매개변수가 없고, 채널 수가 변하지 않음

➤ Max pooling



Pooling layer

➤ Pooling

- 필터를 거친 여러 특징 중 가장 중요한 특징 하나를 선택하여 차원을 축소하는 작업 (Sub-sampling)
- 평균값(average pooling)이나 최대값(max-pooling) 선택
- 학습해야 할 매개변수가 없고, 채널 수가 변하지 않음

1	2	3	4
5	6	7	8
3	2	1	0
5	4	3	2

Feature map

Filters: 2x2
Stride: 2

1	2	3	4
5	6	7	8
3	2	1	0
5	4	3	2

Feature map

max pooling

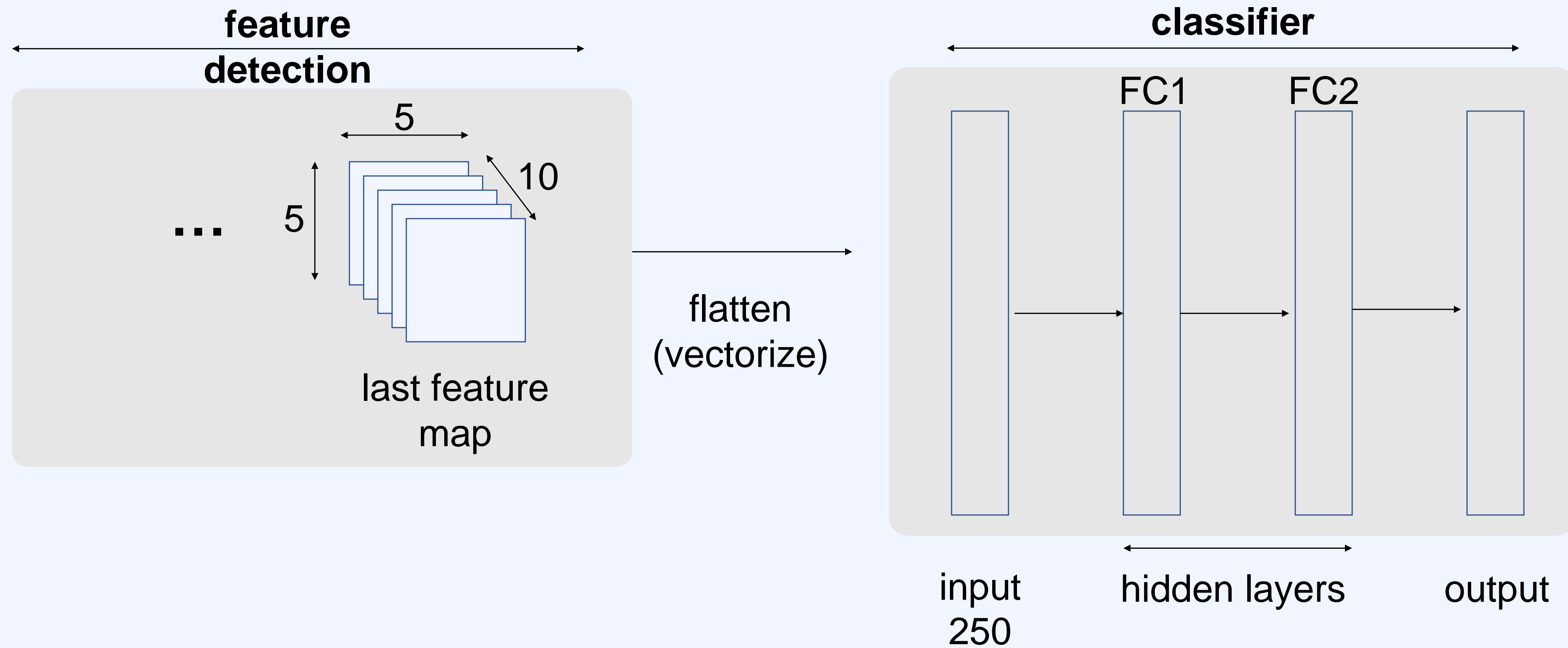
6	8
5	3

$$o = \left\lfloor \frac{i - k}{s} \right\rfloor + 1.$$

Fully Connected layer

➤ Fully Connected layer

- 이전 레이어의 모든 처리 결과를 하나로 연결
- Feature extraction 결과를 바탕으로 이미지를 분류(Classification)
 - output layer는 분류 범주의 개수만큼 노드 설정



CNN의 하이퍼파라미터

➤ Convolution Layers

- Conv layers 개수
- Filter size : 일반적으로 홀수값을 설정
- Filters의 개수: Channel 수, 일반적으로 2의 배수로 설정
- Stride, padding...

➤ Pooling Layers

- Pooling layers 개수
- Pooling windows size
- Stride

➤ FC Layers

- FC layers 개수
- FC layer를 구성하는 노드의 개수

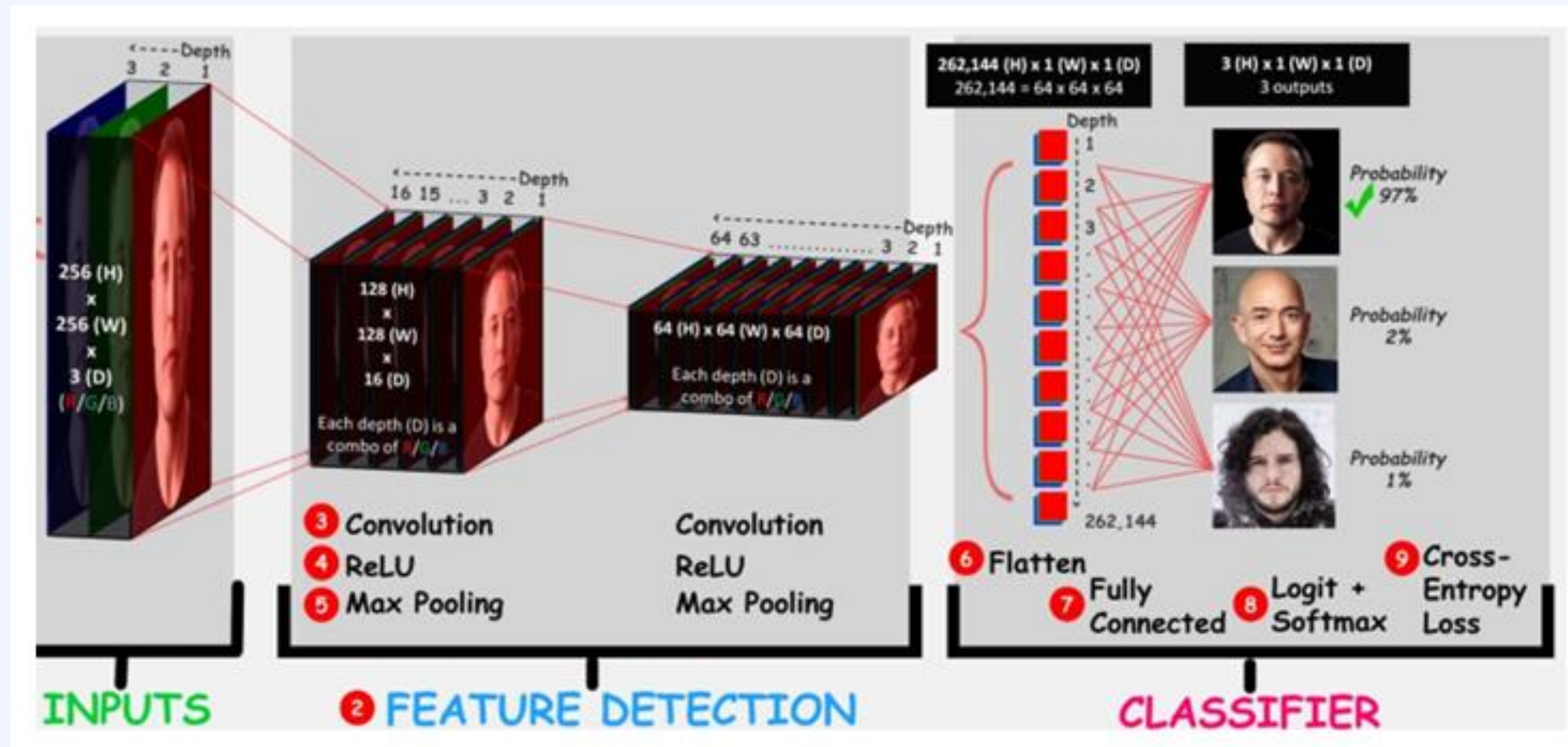
➤ 학습을 위한 하이퍼파라미터

- batch size, epcohs, learning rate, momentum coefficient ...



Summary

➤ Image classifier



Quiz

Q

다음 CNN에서 수행하는 연산에 대한 설명 중 올바르게 작성된 것은?

- ① Conv layer 다음에는 반드시 Pooling layer 를 연결해야한다.
- ② Pooling layer는 학습에 의해서 업데이트 되는 값이 존재하지 않는다.
- ③ Pooling layer를 통과하고 나면 채널의 수가 감소한다.
- ④ Feature extraction이 끝나면 Classifier의 역할을 하는 FC layer는 shape 변경없이 연결할 수 있다.

해설

- ▶ ① Conv layer뒤에 반드시 Pooling layer를 연결할 필요는 없다. ③ Pooling layer를 통과해도 채널의 수는 유지된다. ④ FC layer를 연결하기 위해서 Flatten을 통해 shape을 1차원으로 변경해야한다.

CNN

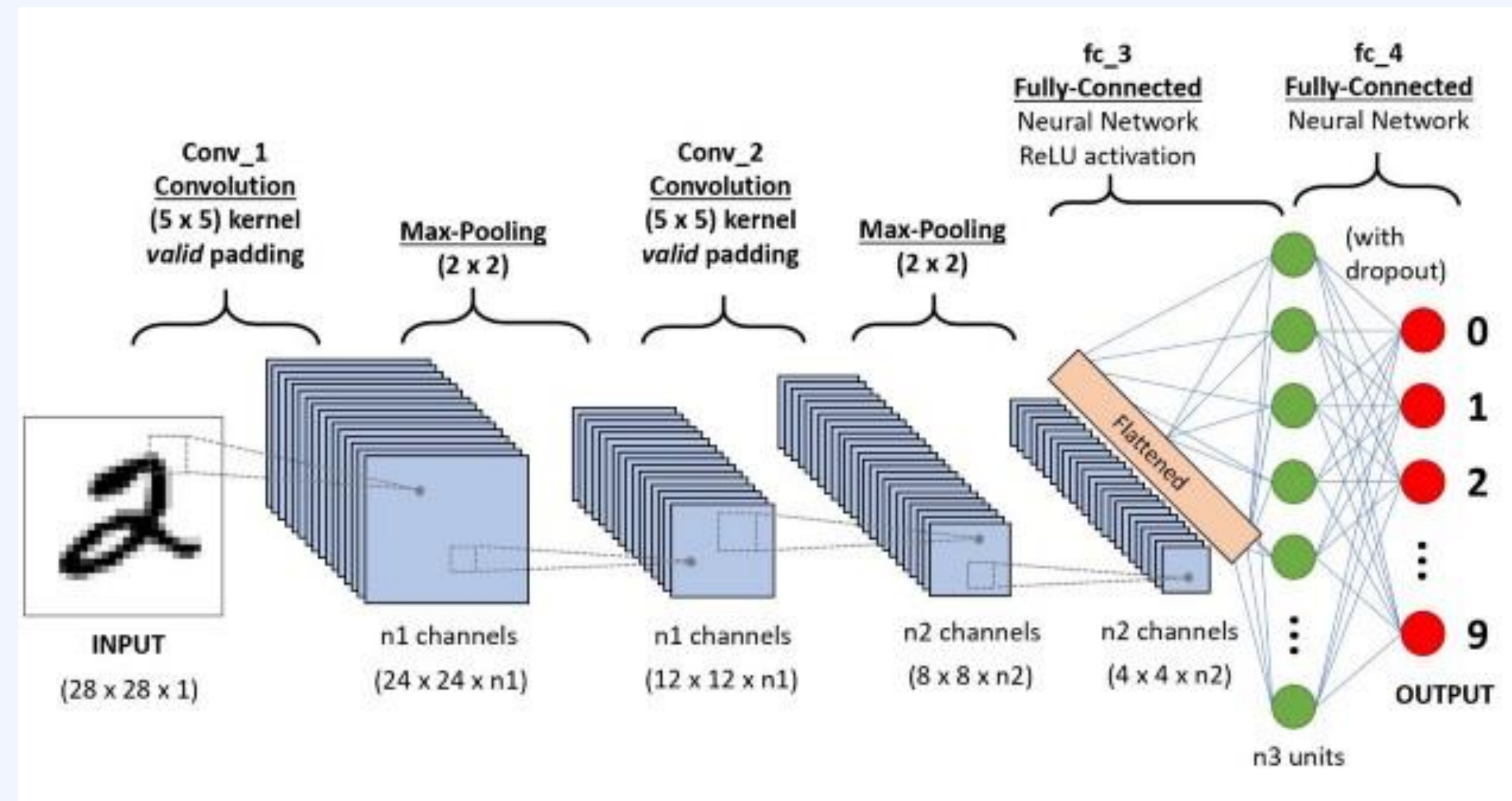
입출력 크기 및
메모리 사용량 계산

Feature Map Size

➤ feature map size

- i : 입력 feature의 수
- o : 출력 feature의 수
- k : convolution kernel 사이즈
- p : convolution padding 사이즈
- s : convolution stride 사이즈

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

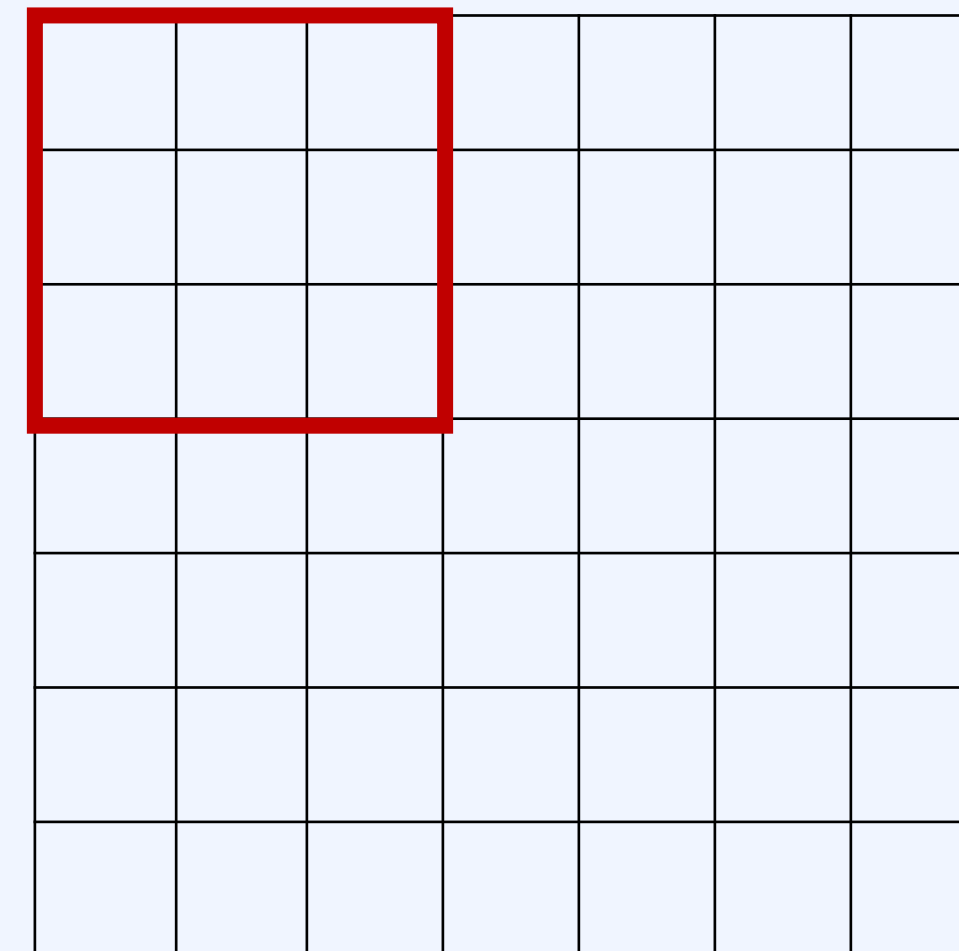


Feature Map Size

➤ feature map size

- i : 입력 feature의 수 $\rightarrow 7$
- o : 출력 feature의 수
- k : convolution kernel 사이즈 $\rightarrow 3$
- p : convolution padding 사이즈 $\rightarrow 0$
- s : convolution stride 사이즈 $\rightarrow 1$

$$\begin{aligned}
 o &= \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1 \\
 &= \left\lfloor \frac{7 + 2 \times 0 - 3}{1} \right\rfloor + 1 \\
 &= 5
 \end{aligned}$$



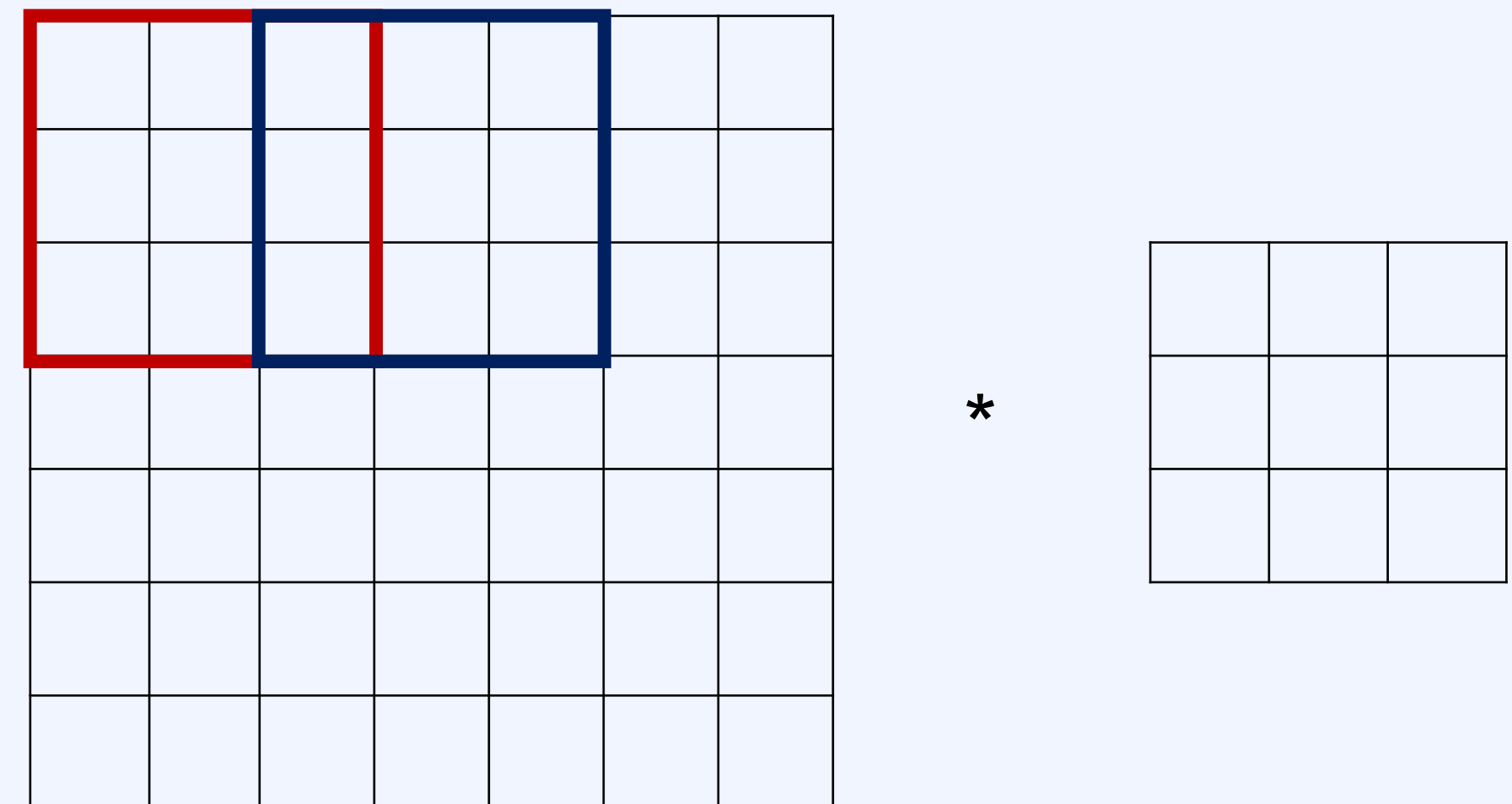
*



Feature Map Size

- stride 값을 늘리면?
 - 연산량이 줄어든다.
 - 출력 feature의 수가 줄어든다.
 - 정보의 손실이 발생한다.

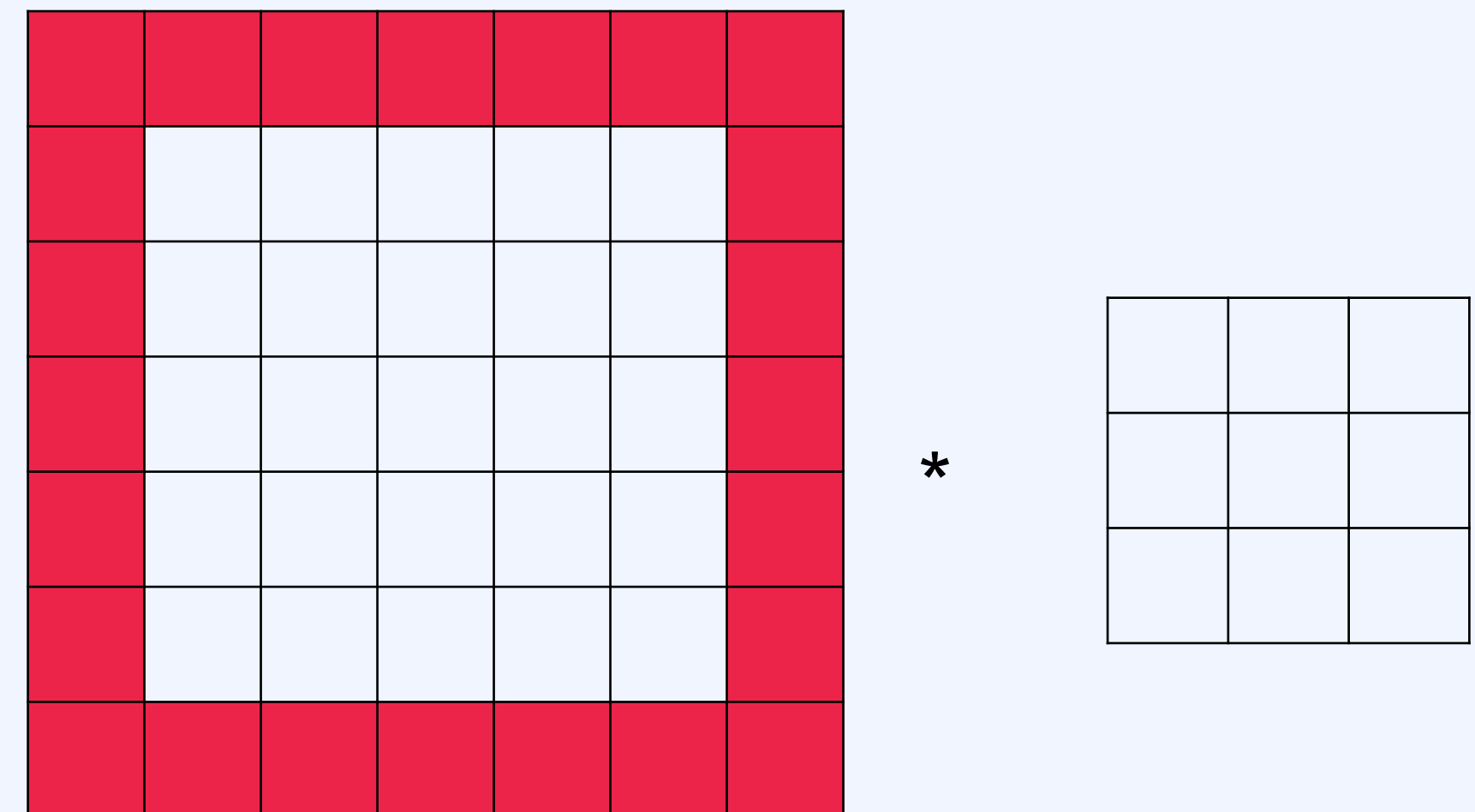
$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$



Feature Map Size

- padding을 1보다 큰 값으로 설정하면?
 - feature의 수가 줄어드는 것을 막을 수 있다.

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$



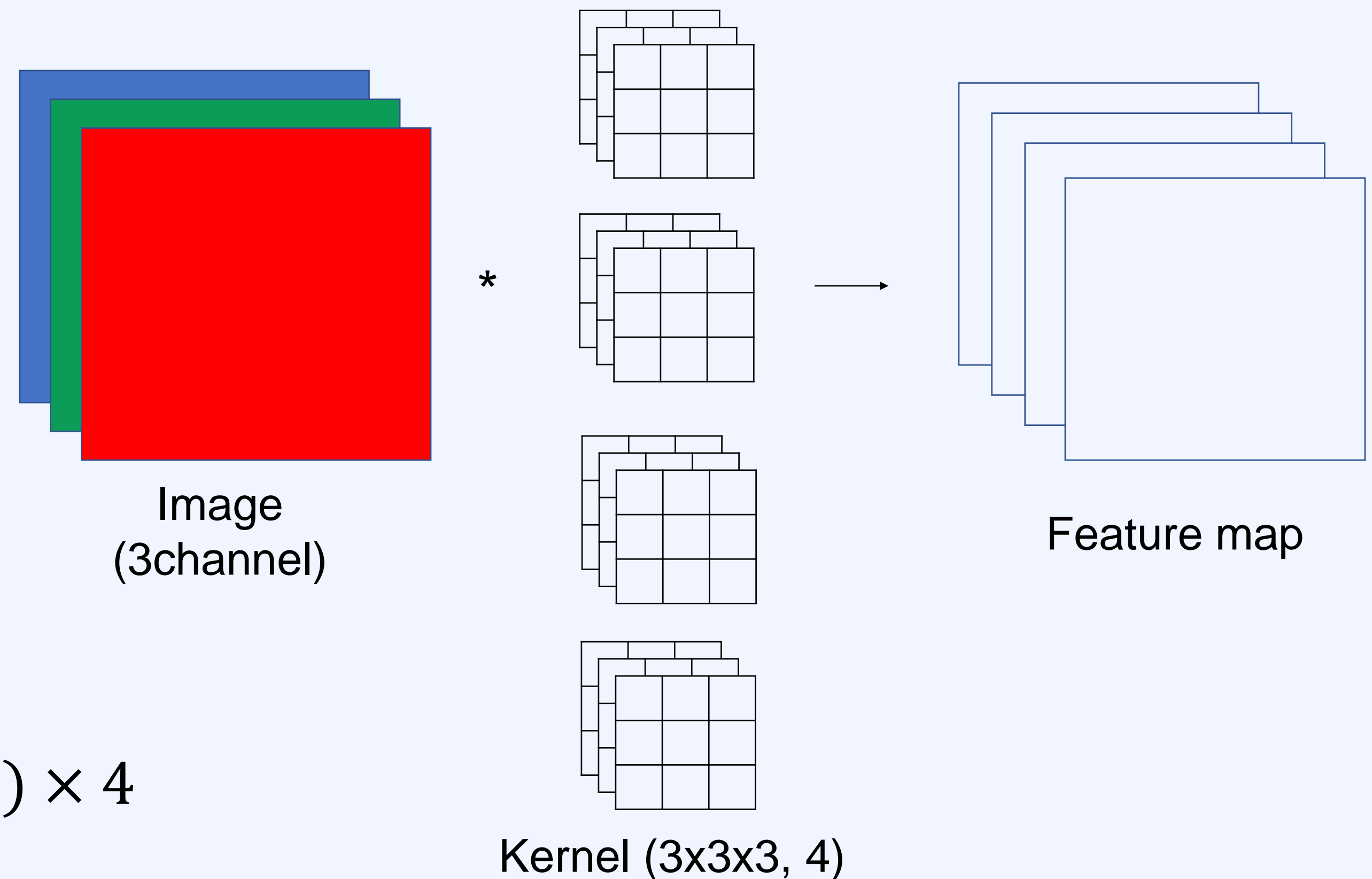
Storage Complexity - Weight 수

➤ layer i의 가중치(weights) 수 = 학습을 통해 업데이트 되는 파라미터의 수 = $(k \times k \times c_{i-1}) \times c_i$

- k : filter size
- c_{i-1} : 이전 레이어의 채널의 수
- c_i : 현재 레이어의 채널의 수

➤ 신경망 학습 이후 메모리에 저장되는 값

- storage complexity



$$(k \times k \times c_{i-1}) \times c_i = (3 \times 3 \times 3) \times 4$$

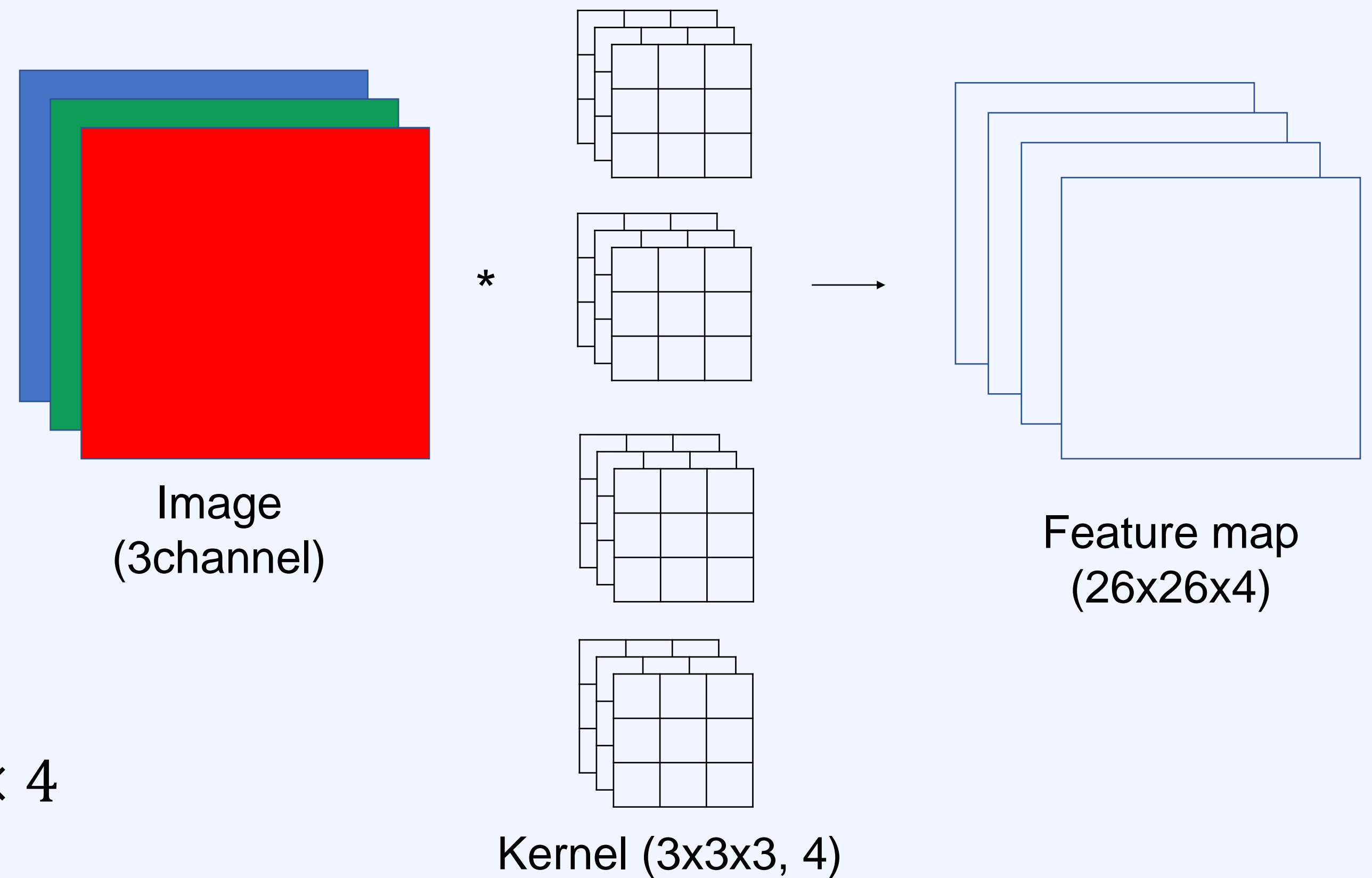
Storage Complexity – memory

➤ layer i의 메모리 = 학습 결과 = $(m_i \times m_i) \times c_i$

- m_i : 현재 레이어의 feature map 사이즈
- c_i : 현재 레이어의 채널의 수

➤ 추론 과정에서의 메모리가 소요되는 정도

- run time



$$(m_i \times m_i) \times c_i = (26 \times 26) \times 4$$

Computational Complexity

- **layer i의 계산 복잡도** = $(k \times k) \times (m_i \times m_i) \times (c_{i-1} \times c_i)$
 - k : filter size
 - m_i : 현재 레이어의 feature map 사이즈
 - c_{i-1} : 이전 레이어의 채널의 수
 - c_i : 현재 레이어의 채널의 수
- **추론 과정에서 계산 복잡도 결정**
 - 학습 과정에는 GPU 등 하드웨어 리소스를 사용할 수 있으므로 계산 복잡도는 큰 고려요소는 아님
 - # Conv Ops (FLOPs, floating point operation)

VGGNet

➤ representation size and the total number of weights:

INPUT: [224x224x3] memory: $224*224*3=150K$ weights: 0
 CONV3-64: [224x224x64] memory: $224*224*64=3.2M$ weights: $(3*3*3)*64 = 1,728$
 CONV3-64: [224x224x64] memory: $224*224*64=3.2M$ weights: $(3*3*64)*64 = 36,864$
 POOL2: [112x112x64] memory: $112*112*64=800K$ weights: 0
 CONV3-128: [112x112x128] memory: $112*112*128=1.6M$ weights: $(3*3*64)*128 = 73,728$
 CONV3-128: [112x112x128] memory: $112*112*128=1.6M$ weights: $(3*3*128)*128 = 147,456$
 POOL2: [56x56x128] memory: $56*56*128=400K$ weights: 0
 CONV3-256: [56x56x256] memory: $56*56*256=800K$ weights: $(3*3*128)*256 = 294,912$
 CONV3-256: [56x56x256] memory: $56*56*256=800K$ weights: $(3*3*256)*256 = 589,824$
 CONV3-256: [56x56x256] memory: $56*56*256=800K$ weights: $(3*3*256)*256 = 589,824$
 POOL2: [28x28x256] memory: $28*28*256=200K$ weights: 0
 CONV3-512: [28x28x512] memory: $28*28*512=400K$ weights: $(3*3*256)*512 = 1,179,648$
 CONV3-512: [28x28x512] memory: $28*28*512=400K$ weights: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [28x28x512] memory: $28*28*512=400K$ weights: $(3*3*512)*512 = 2,359,296$
 POOL2: [14x14x512] memory: $14*14*512=100K$ weights: 0
 CONV3-512: [14x14x512] memory: $14*14*512=100K$ weights: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14*14*512=100K$ weights: $(3*3*512)*512 = 2,359,296$
 CONV3-512: [14x14x512] memory: $14*14*512=100K$ weights: $(3*3*512)*512 = 2,359,296$
 POOL2: [7x7x512] memory: $7*7*512=25K$ weights: 0
 FC: [1x1x4096] memory: 4096 weights: $7*7*512*4096 = 102,760,448$
 FC: [1x1x4096] memory: 4096 weights: $4096*4096 = 16,777,216$
 FC: [1x1x1000] memory: 1000 weights: $4096*1000 = 4,096,000$

TOTAL memory: $24M * 4 \text{ bytes} \approx 93MB$ / image (only forward! ~ 2 for bwd)
 TOTAL params: 138M parameters

Summary

➤ Feature Map Size

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

➤ layer i의 가중치(weights) 수 = 학습을 통해 업데이트 되는 파라미터의 수 = $(\mathbf{k} \times \mathbf{k} \times \mathbf{c}_{i-1}) \times \mathbf{c}_i$


➤ layer i의 메모리 = 학습 결과 = $(\mathbf{m}_i \times \mathbf{m}_i) \times \mathbf{c}_i$

➤ layer i의 계산 복잡도 = $(\mathbf{k} \times \mathbf{k}) \times (\mathbf{m}_i \times \mathbf{m}_i) \times (\mathbf{c}_{i-1} \times \mathbf{c}_i)$

Quiz

Q

다음 설명 중 옳지 않은 것은?

- ① padding을 1보다 큰 값으로 설정하면 feature의 수가 줄어드는 것을 막을 수 있다.
- ② stride 값을 늘리면 연산량이 줄어들지만, 정보의 손실이 발생한다.
- ③  신경망의 Computational Complexity는 학습 과정 시 고려해야한다.
- ④ 신경망을 구성하는 가중치는 학습 이후 저장 공간에 저장되는 값이다.

해설

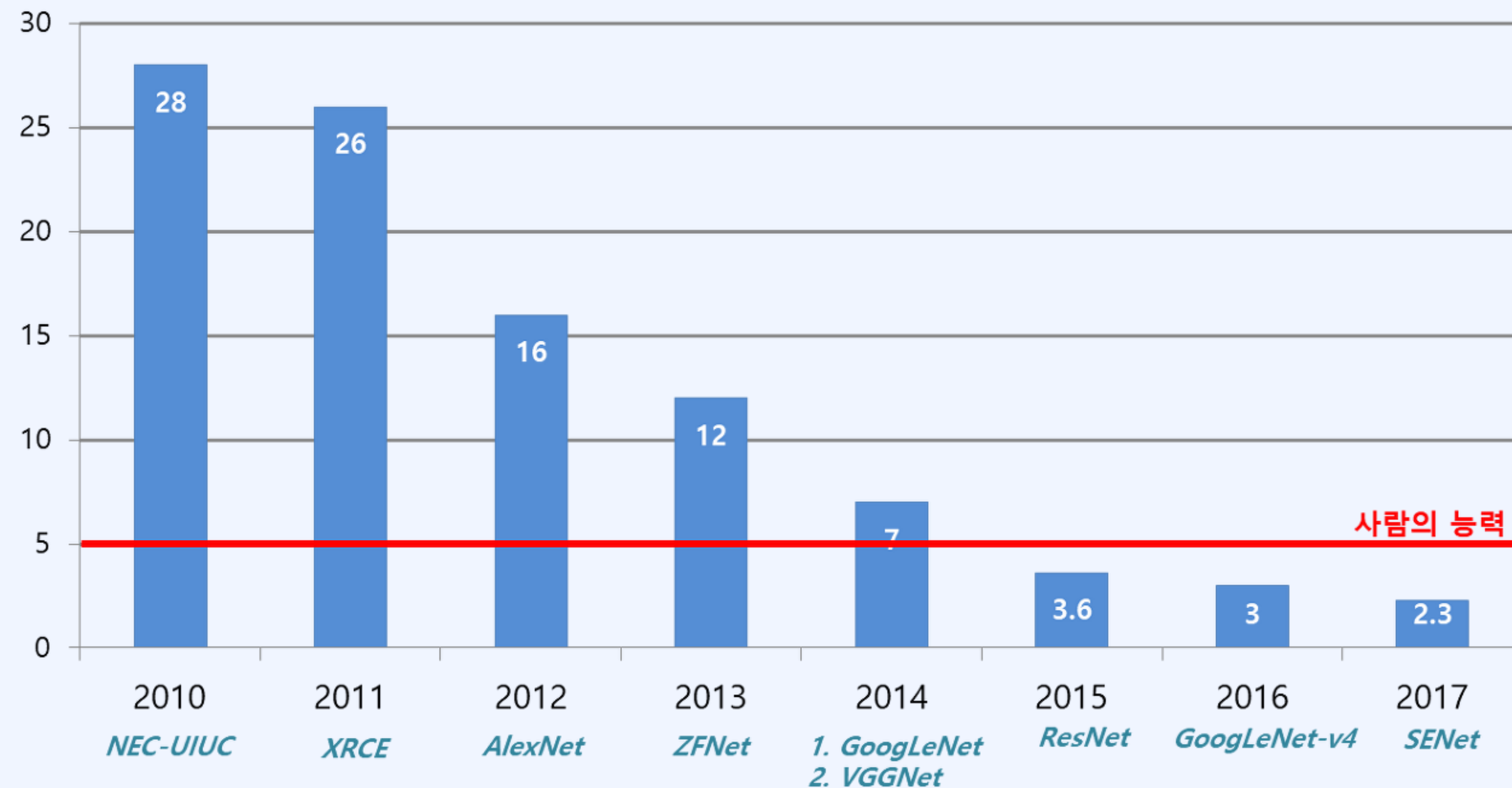
- ▶ 신경망의 Computational Complexity는 추론 과정 시 고려해야한다.

CNN
VGGNet

ILSVRC

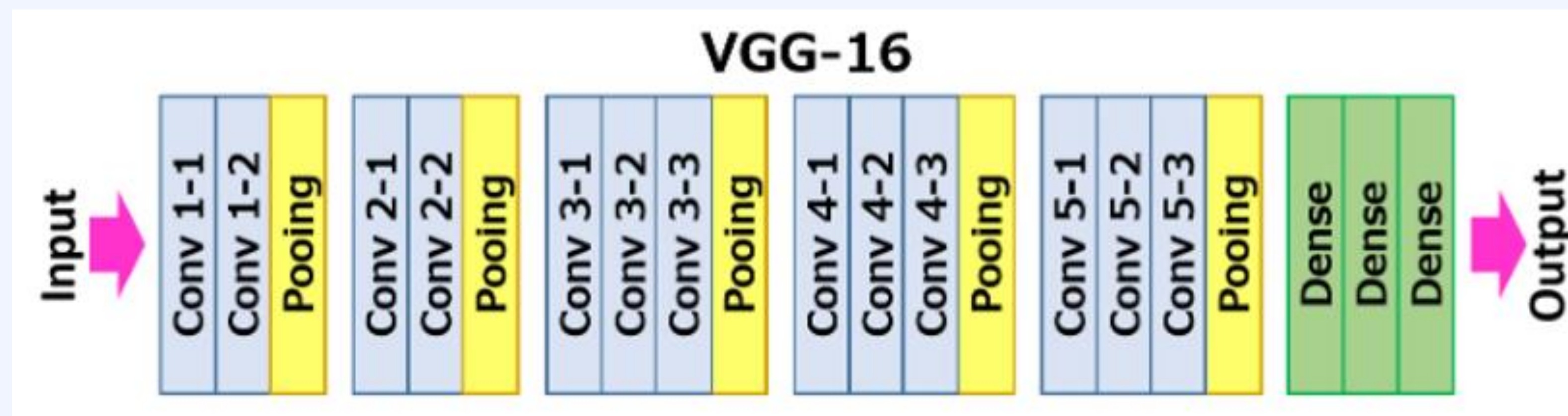
- 이미지 넷(Image Net)의 사물 인식 대회(Large Scale Visual Recognition Challenge)
- ILSVRC 대회 역대 우승 알고리즘

우승 알고리즘의 분류 에러율(%)



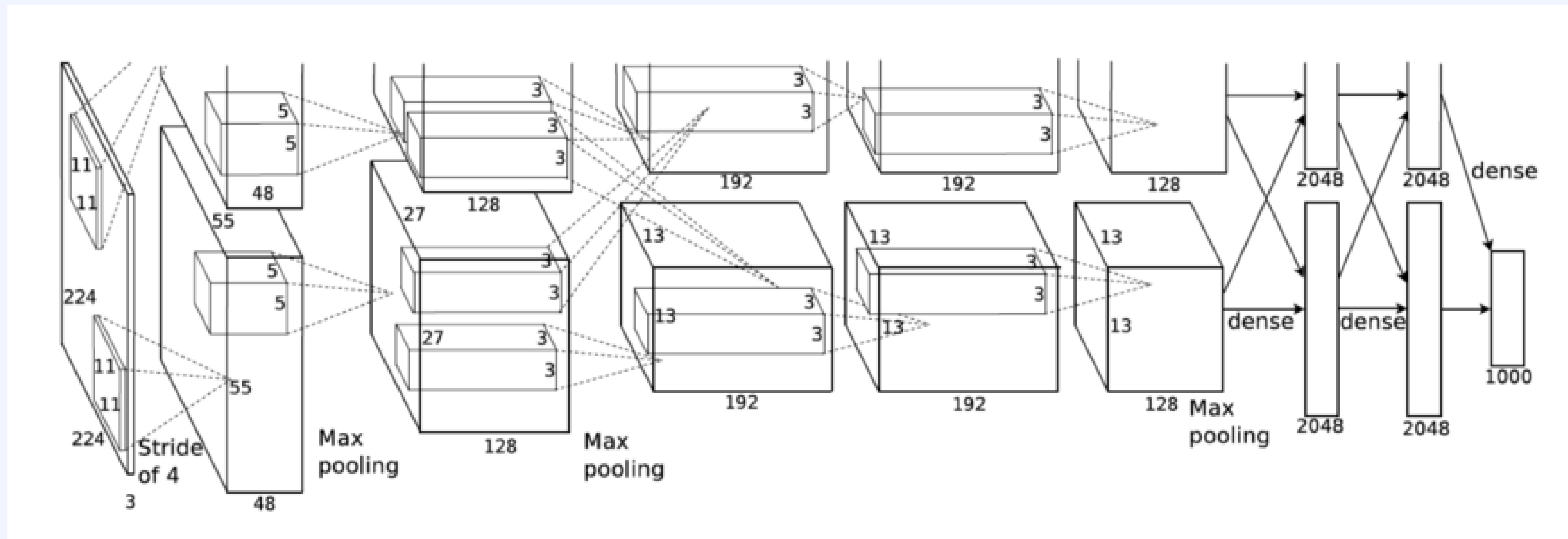
VGGNet

- 2014년 이미지 사물인식 대회에서 2위(1위는 GoogLeNet)
- Paper
 - Karen Simonyan, Andrew Zisserman. University of Oxford, UK. **Very Deep Convolutional Networks for Large-Scale Image Recognition**, arXiv preprint, 2014
- VGGNet-16
 - 13 conv layers + 3 FC
 - 138 Million weights and 500MB of storage space in total!
 - 변형: VGGNet-19



VGGNet

- VGGNet 이전에 나왔던 AlexNet의 경우에는 첫번째 Conv layer에 11x11 필터를 사용
- VGGNet의 가장 큰 특징!
 - 3x3 필터만 사용해도 충분하다는 것을 보여줌

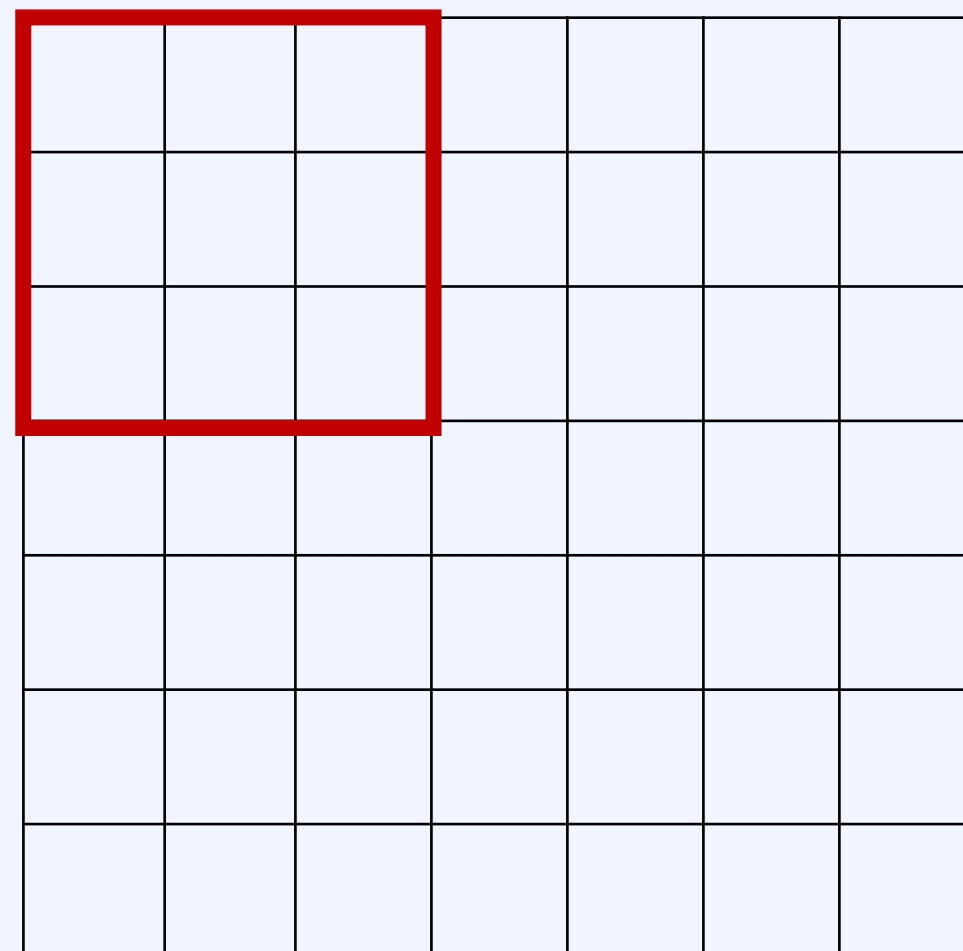


AlexNet

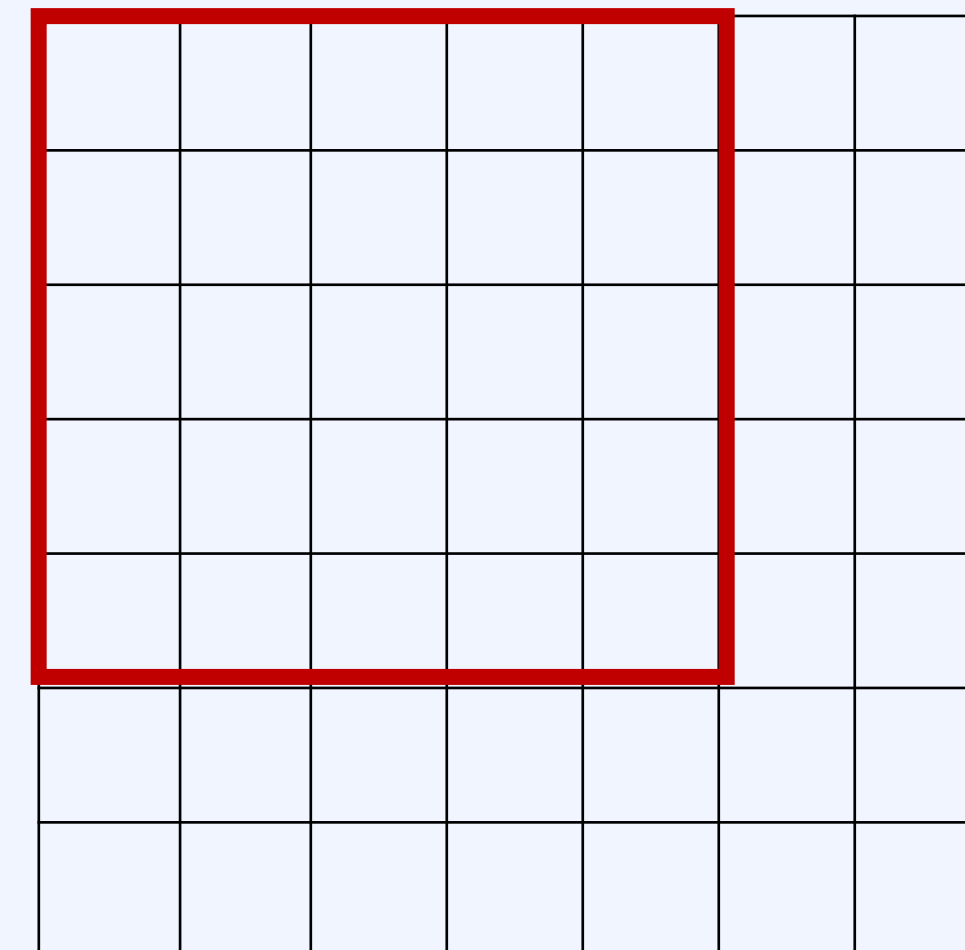
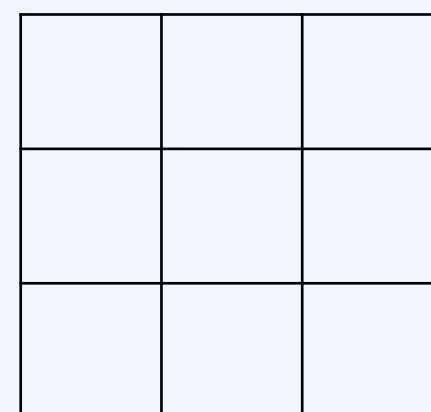
VGGNet

➤ Local Connectivity

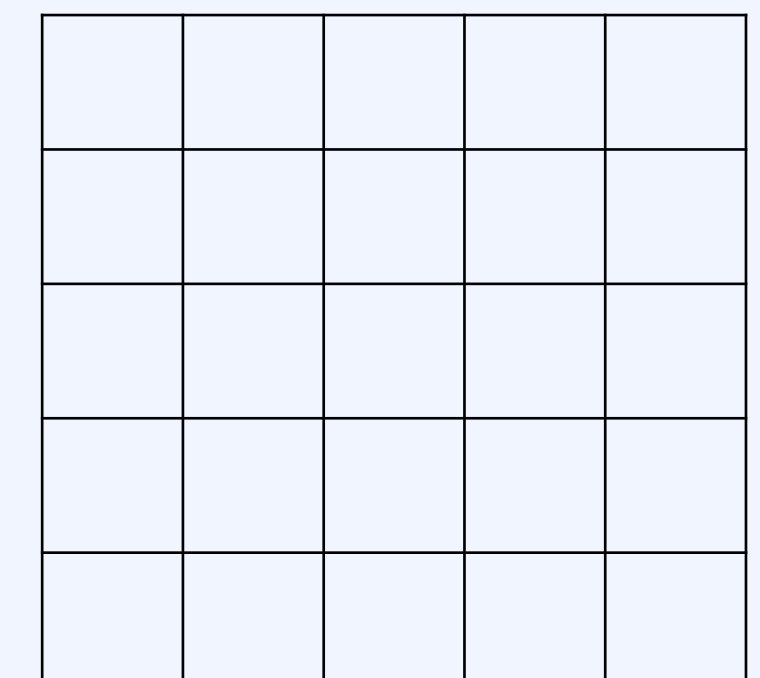
- Receptive Field(RF)
- kernel이 convolution 연산을 수행할 때 이미지의 얼마나 많은 영역을 수용할 수 있는가?



*



*

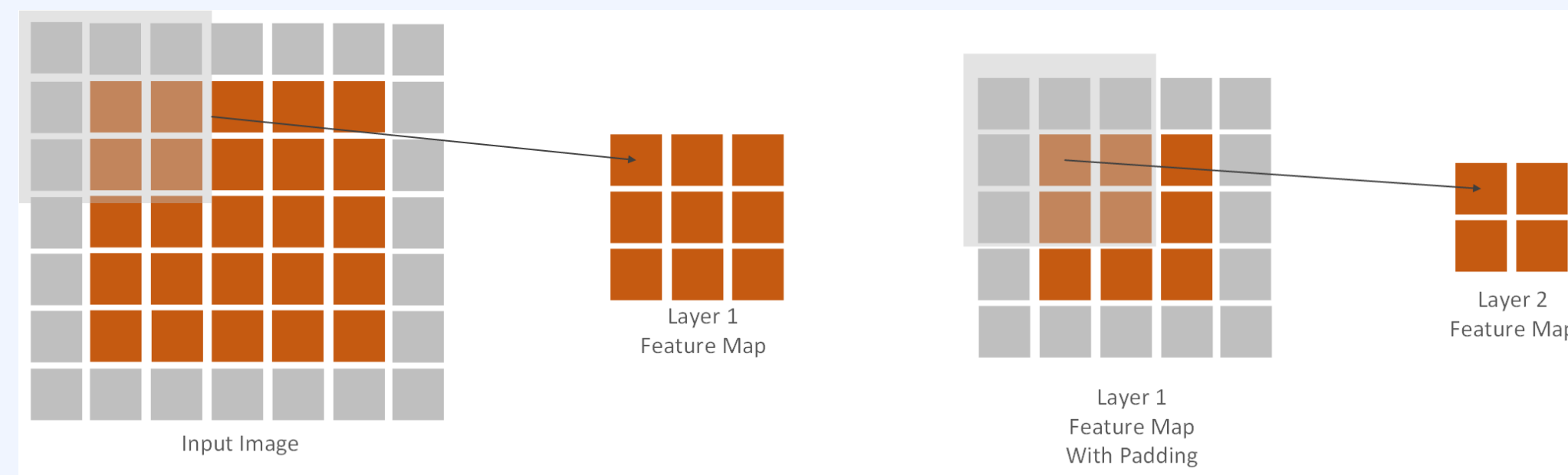


VGGNet

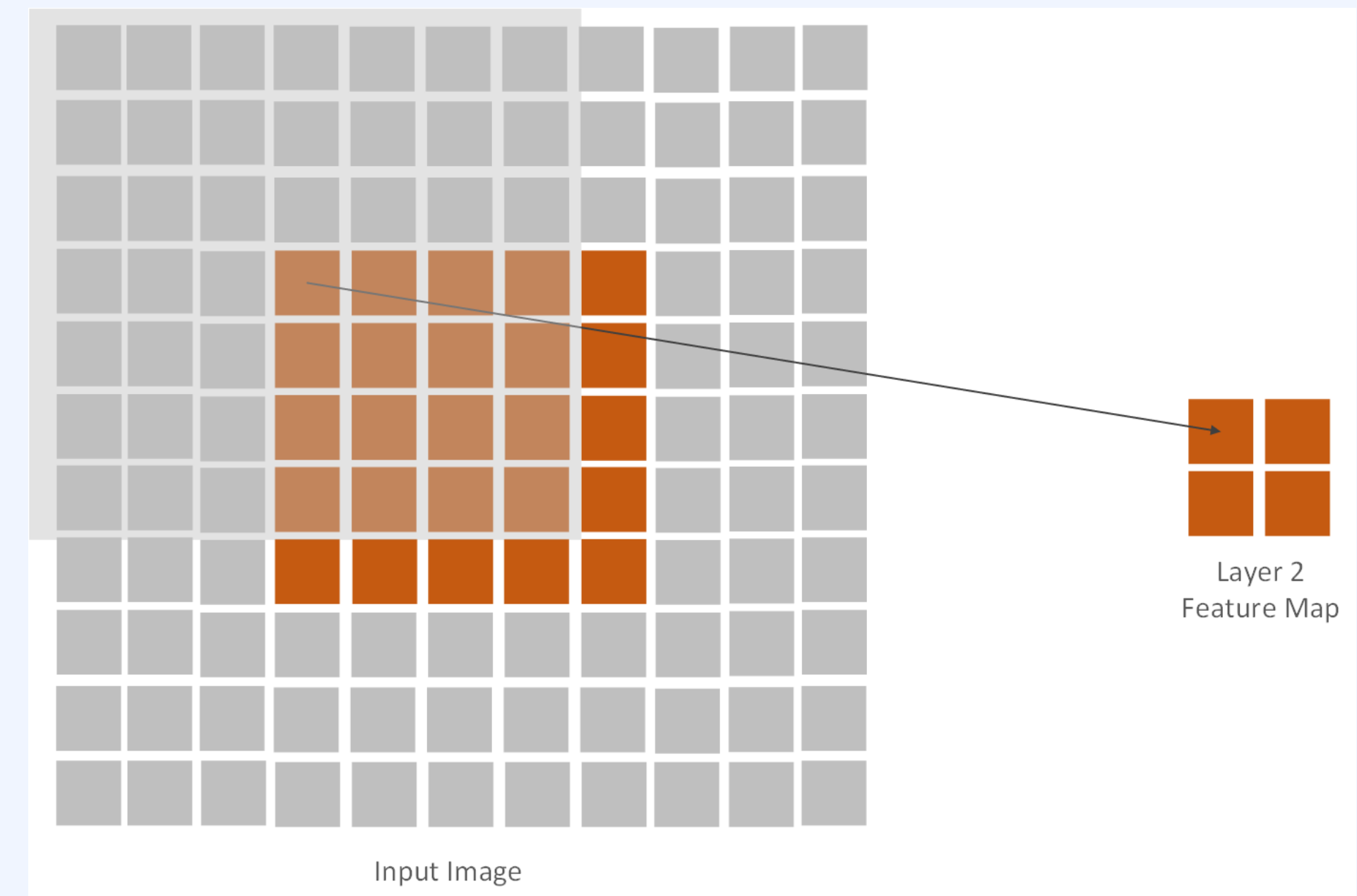
➤ Local Connectivity

▪ Receptive Field(RF)

- Convolution 연산을 수행할 수록 RF가 더 커짐



$$r_i = r_{i-1} + (k - 1) \prod_{k=1}^{i-1} s_k$$



VGGNet

- 7x7 커널을 하나만 사용한 경우 vs 3x3 커널을 3번 사용한 경우
 - 동일한 효과를 갖는다
 - Conv 연산의 횟수가 증가하는데 사용하는 메모리가 늘어나는 단점이 있지 않을까?

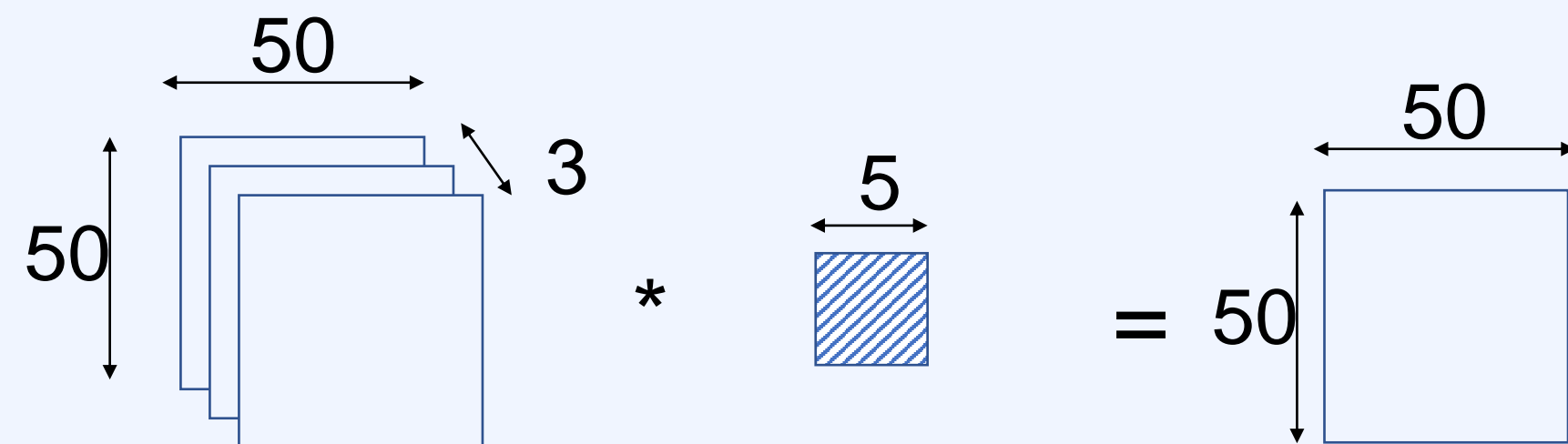
$$r_i = r_{i-1} + (k - 1) \prod_{k=1}^{i-1} s_k$$

	7x7 필터 한 번	3x3 필터 세 번
Receptive Field	$R_1 = k = 7$	$R_1 = k = 3$ $R_2 = R_1 + (3 - 1) = 3 + 2 = 5$ $R_3 = R_2 + (3 - 1) = 5 + 2 = 7$

VGGNet

➤ 5x5 필터를 사용하는 경우

- layer i의 가중치(weights) 수 = 학습을 통해 업데이트 되는 파라미터의 수 = $(k \times k \times c_{i-1}) \times c_i$
 - k : filter size
 - c_{i-1} : 이전 레이어의 채널의 수
 - c_i : 현재 레이어의 채널의 수



Input Image

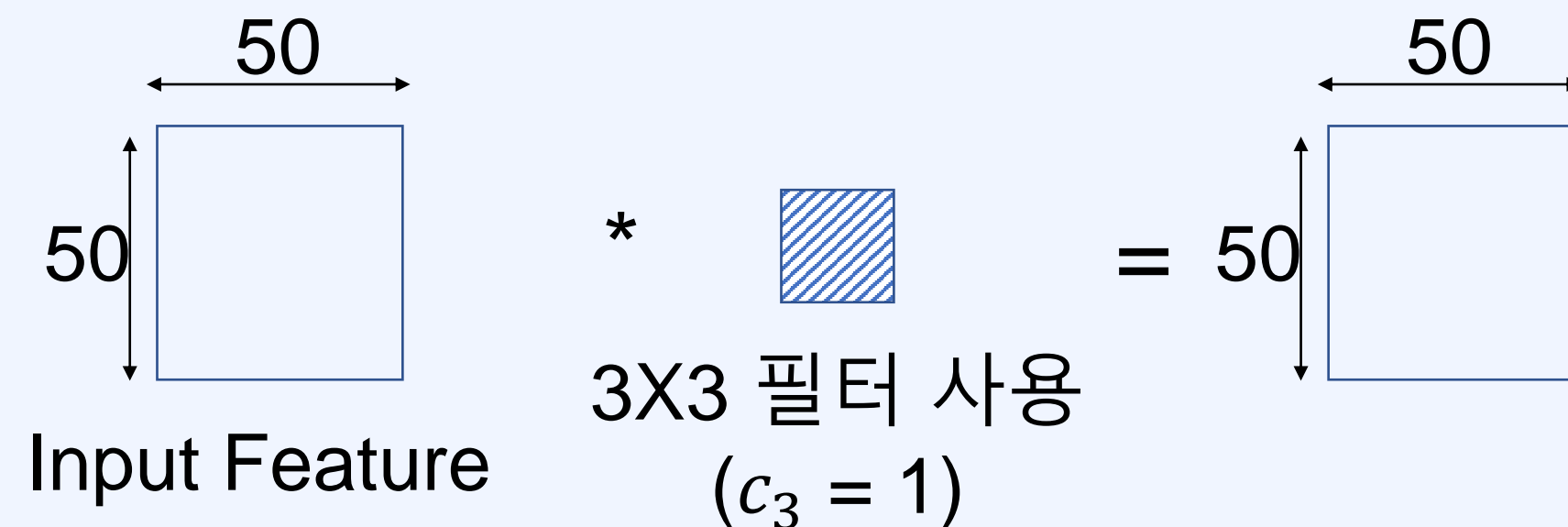
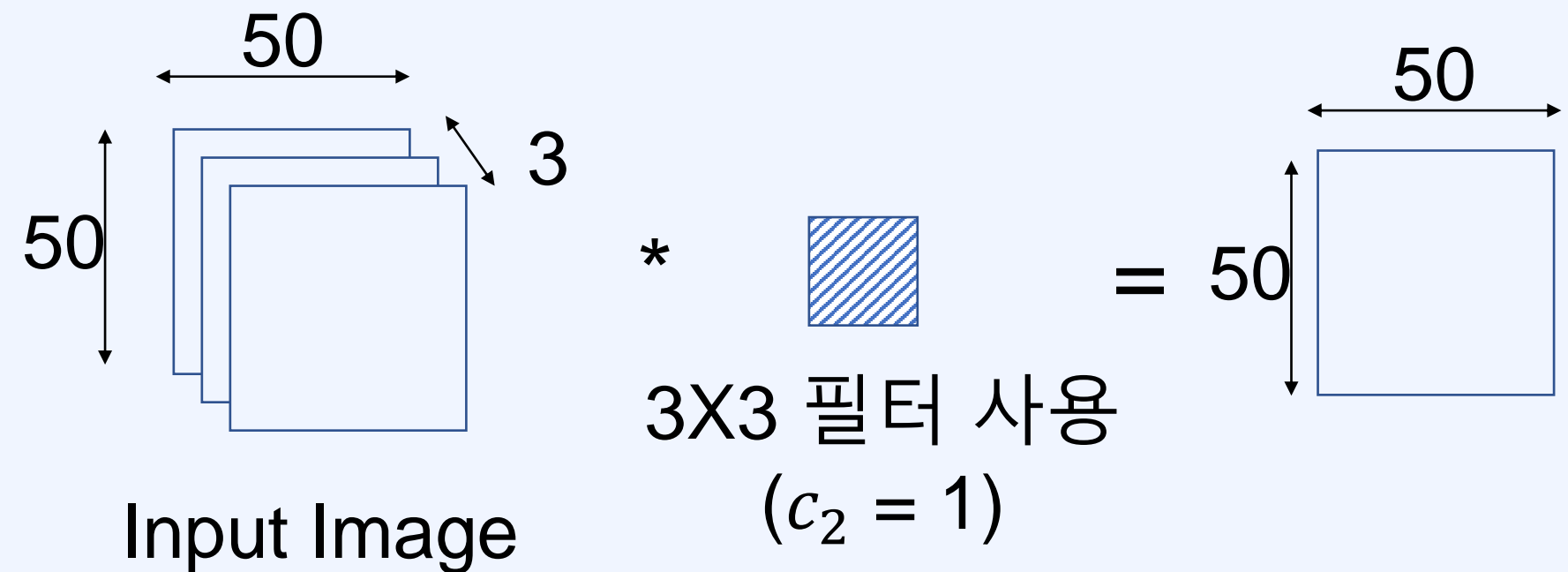
5x5 필터 사용
($c_2 = 1$)인 경우

2번째 레이어의 weight의 수
 $= (k \times k \times c_1) \times c_2 = (5 \times 5 \times 3) \times 1 = 75$

VGGNet

➤ 3x3 필터를 두 번 사용하는 경우

- layer i의 가중치(weights) 수 = 학습을 통해 업데이트 되는 파라미터의 수 = $(k \times k \times c_{i-1}) \times c_i$
 - k : filter size
 - c_{i-1} : 이전 레이어의 채널의 수
 - c_i : 현재 레이어의 채널의 수



2번째 레이어의 weight의 수
 $= (k \times k \times c_1) \times c_2 = (3 \times 3 \times 3) \times 1 = 27$

3번째 레이어의 weight의 수
 $= (k \times k \times c_2) \times c_3 = (3 \times 3 \times 1) \times 1 = 9$

총 weight의 수 = 36

VGGNet

➤ 5x5 필터를 사용하는 경우 vs 3x3 필터를 두 번 사용하는 경우

	5x5 필터 한 번	3x3 필터 두 번
총 weight의 수	75	36
Activation Function	1	2
Receptive Filed	$R_1 = k = 5$	$R_1 = k = 3$ $R_2 = R_1 + (3 - 1) = 3 + 2 = 5$

- 동일한 Receptive Filed!
- Weight의 수는 줄어든다
 - 메모리 사용량이 감소함
- 더 많은 Activation Function을 통과한다
 - 비선형성이 증가함

Summary

➤ VGGNet

- 3x3 Convolution 연산을 여러 번 사용했을 때의 효과
 - ① Receptive Field를 늘리는 효과
 - ② 큰 사이즈의 Kernel을 사용했을 때 보다 Weight의 수는 줄어든다: 메모리 사용량이 감소함
 - ③ 더 많은 Activation Function을 통과한다: 비선형성이 증가함

Quiz

Q

다음 중 VGGNet의 특징으로 옳지 않은 것은?

- ① 2014년 LSVRC 2위를 한 16층 구조의 신경망이다.
- ② 3×3 크기의 필터를 사용해서 특징을 추출하는 구조를 가지고 있다.
- ③ 3×3 크기의 필터를 여러 번 사용할 경우, Receptive Field를 확장하는 장점이 존재하지만 필요한 메모리 사이즈가 증가한다.
- ④ Convolution 연산을 여러 번 수행하기 때문에 비선형성을 늘려서 분류의 정확도를 높일 수 있다.

해설

- ▶ Receptive Field를 확장할 뿐만 아니라 메모리량도 감소하는 장점이 있다.

CNN

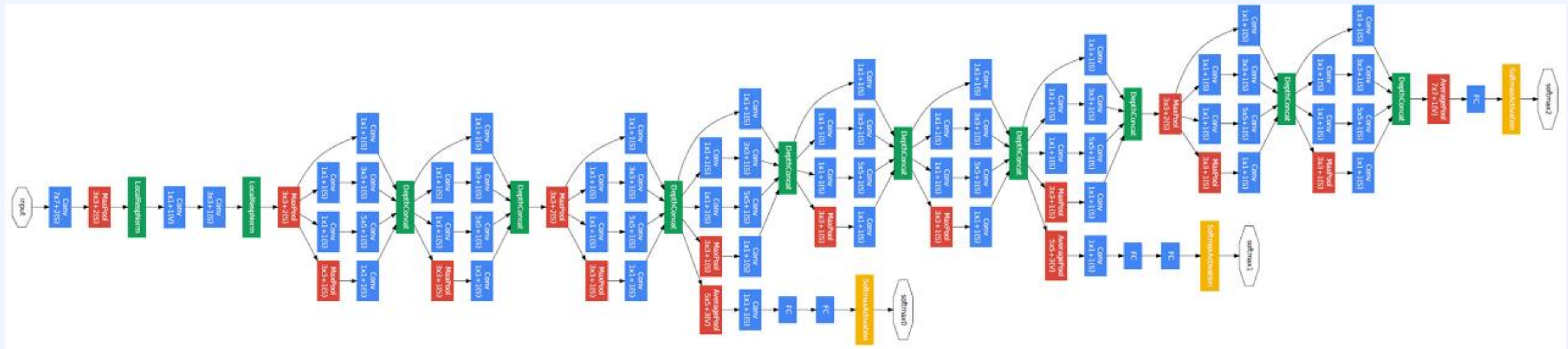
GoogLeNet(Inception)

GoogLeNet(Inception V1)

➤ 2014년 LSVRC 1위

➤ Paper

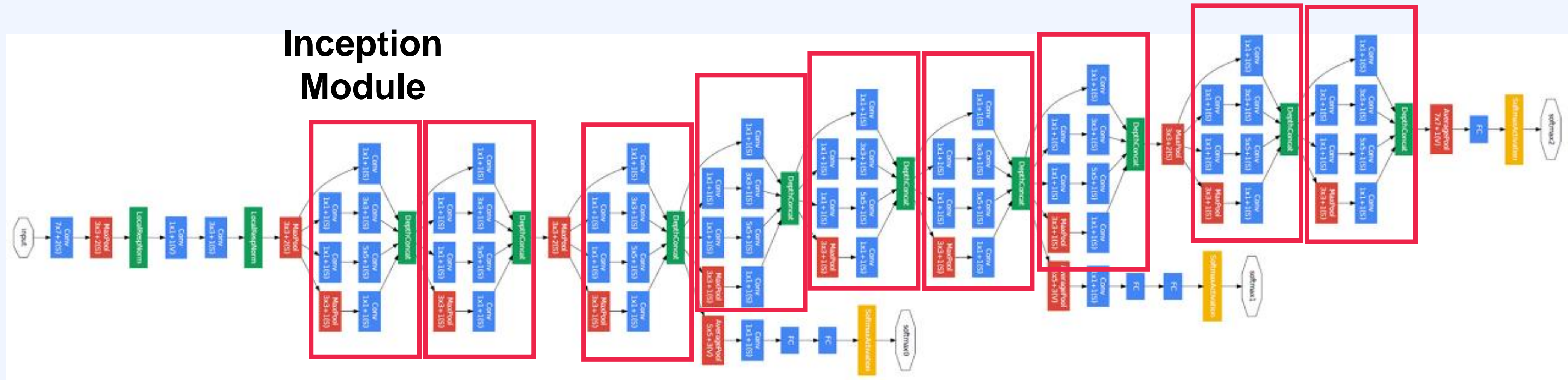
- Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Google, University of Michigan, University of North Carolina, **Going Deeper with Convolutions**, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
- 22층짜리 신경망, 인셉션 모듈



GoogLeNet(Inception V1)

- 22 layers
 - 9개의 Inception 모듈(개별적인 레이어는 100 layers 이상)
 - 5 Million weights in total

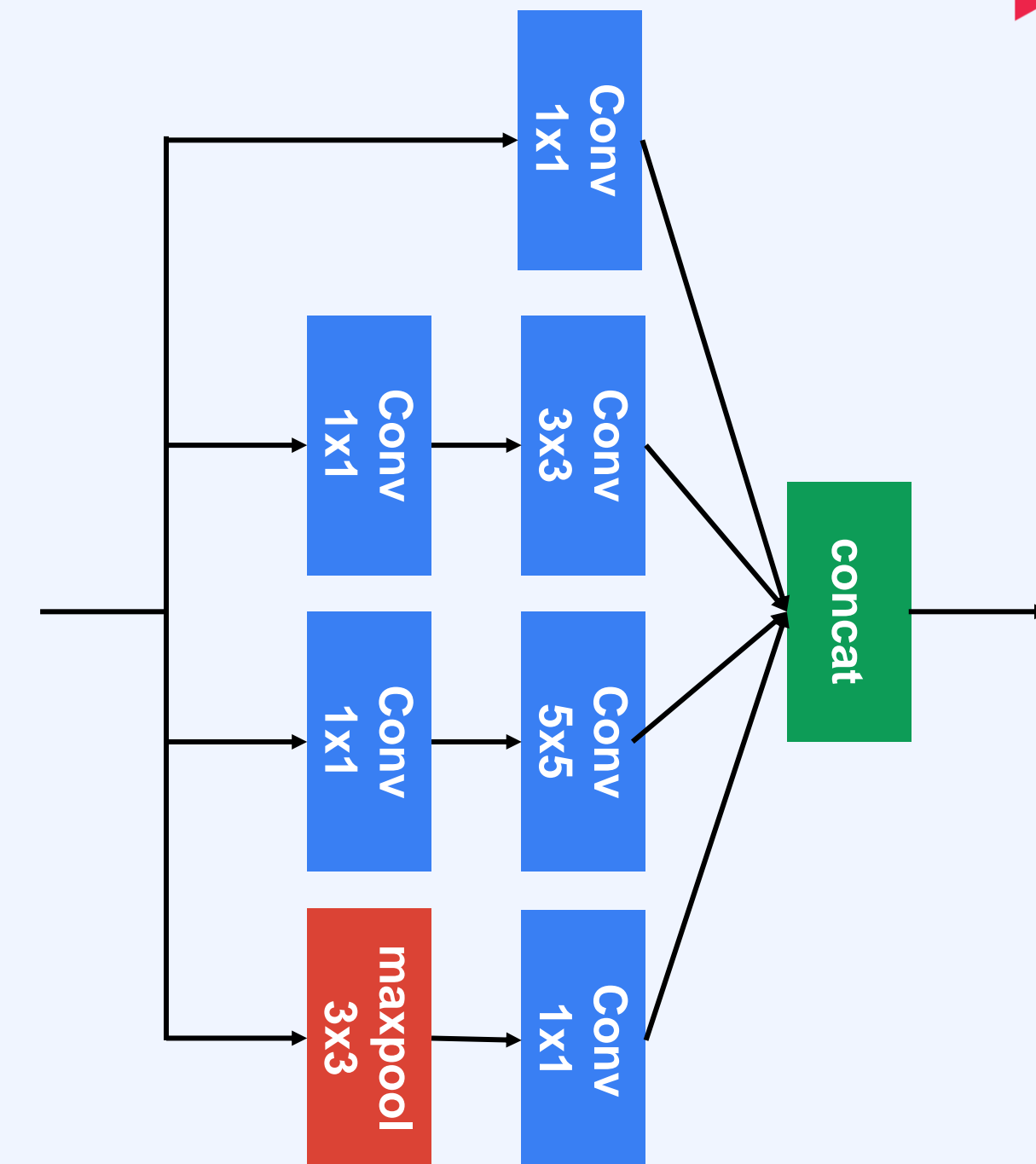
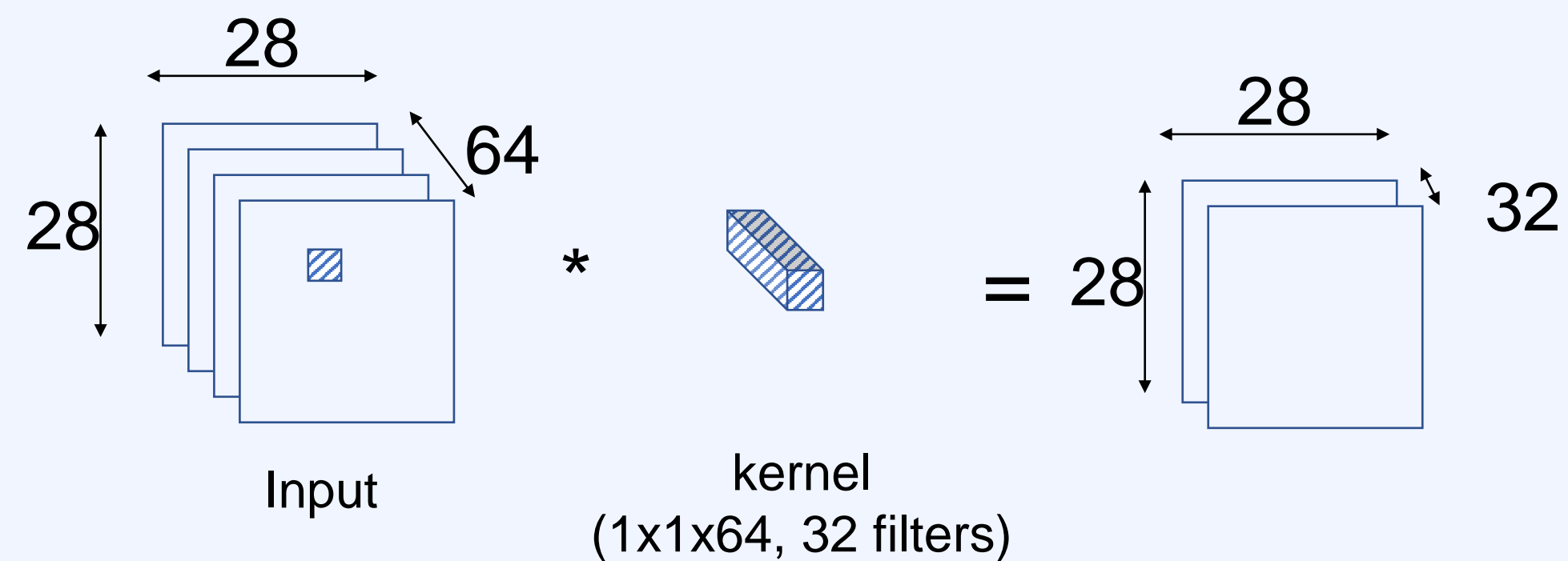
Inception Module



GoogLeNet(Inception V1)

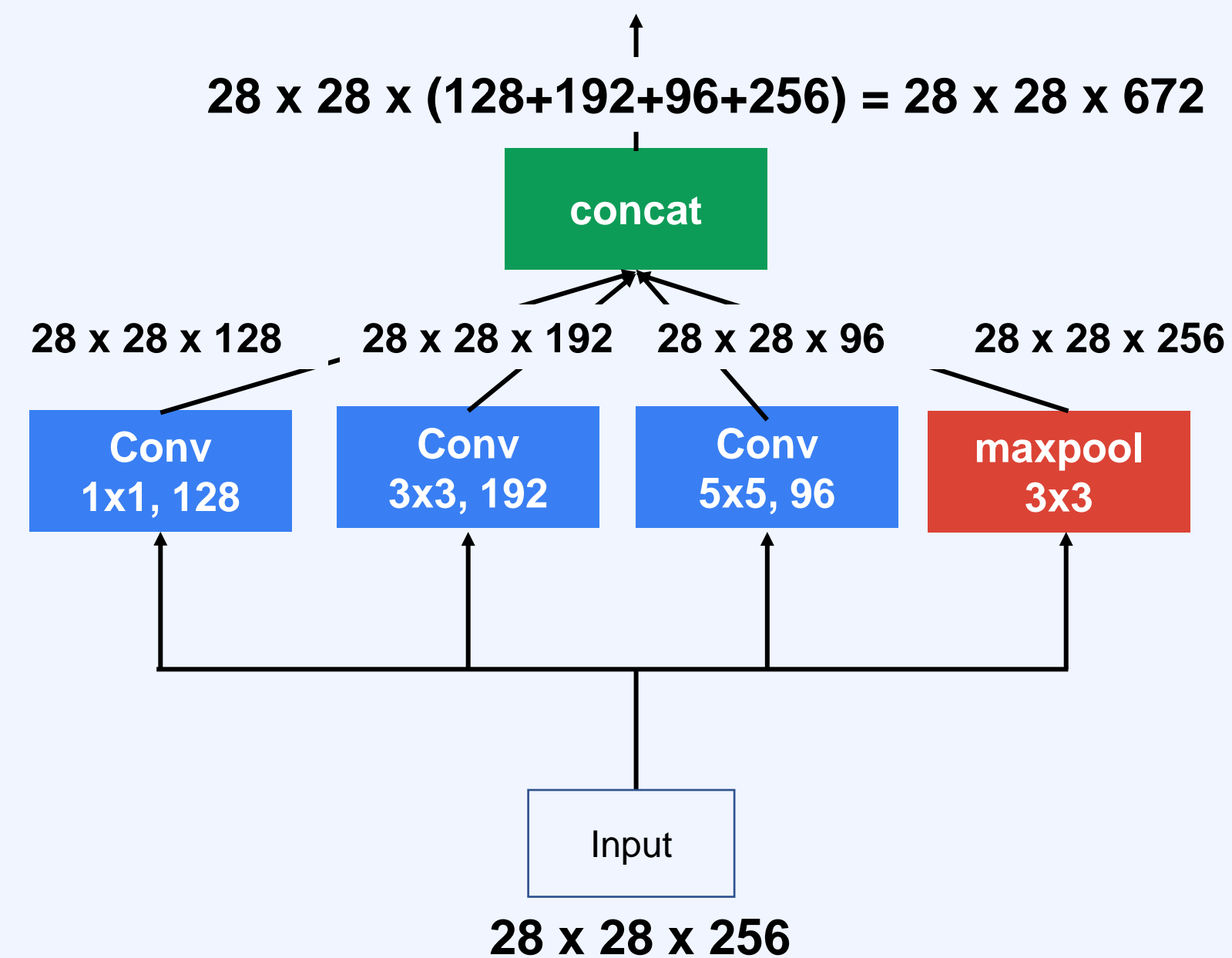
➤ Inception module

- split-transform-merge strategy
 - 다양한 크기의 필터를 사용해서 다양한 특징을 추출
 - 1x1, 3x3, 5x5 kernels
- Bottleneck structure
 - 1x1 Conv: 필터의 크기가 1
 - Dimensionality Reduction 효과
 - 입력의 필터 크기 보다 Conv의 필터 수가 작을 때
 - 연산량을 줄이고, 신경망을 더욱 깊게 쌓을 수 있음
 - 비선형성(nonlinearity) 증가

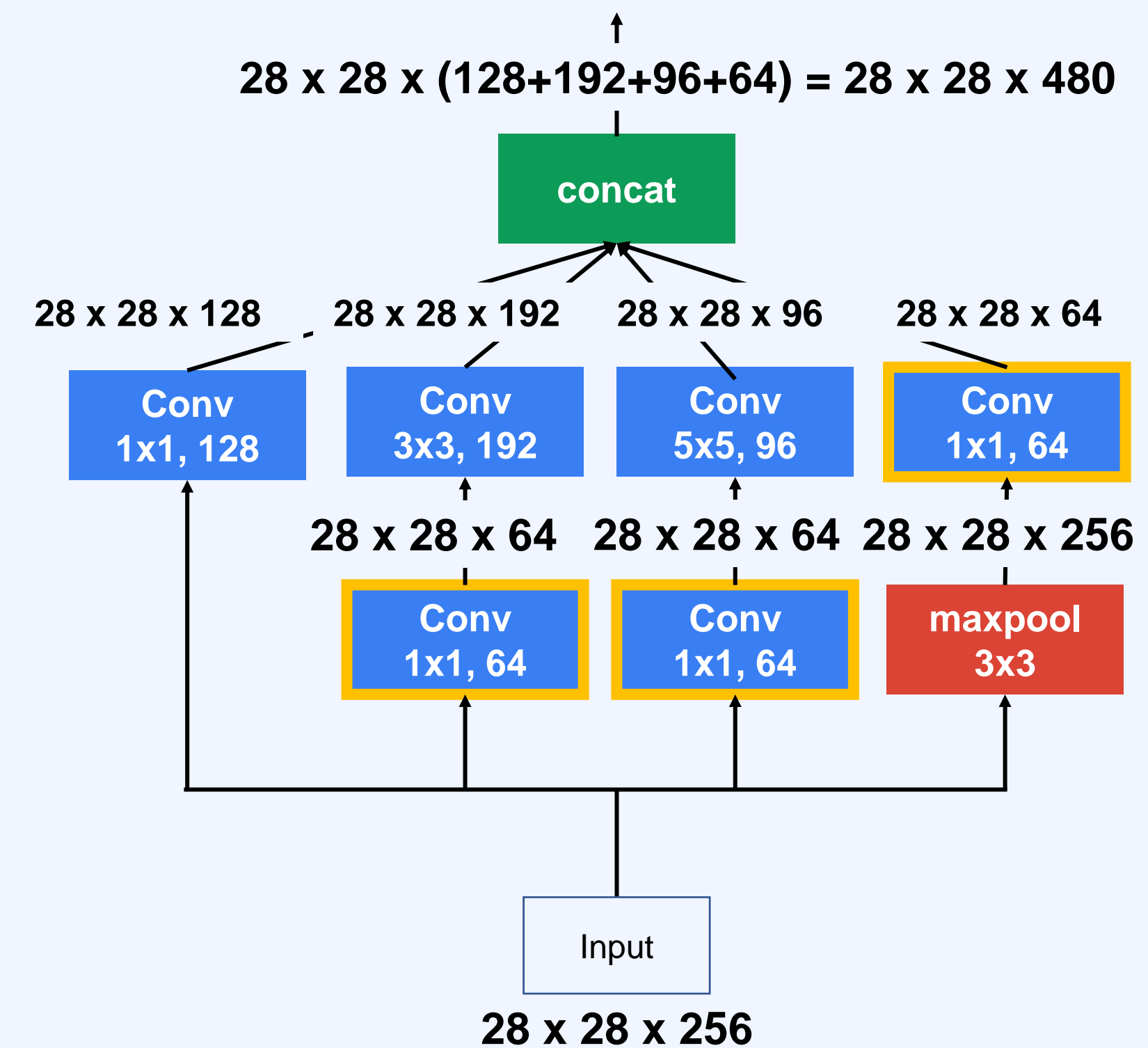


GoogLeNet(Inception V1)

➤ Memory(# of neurons)



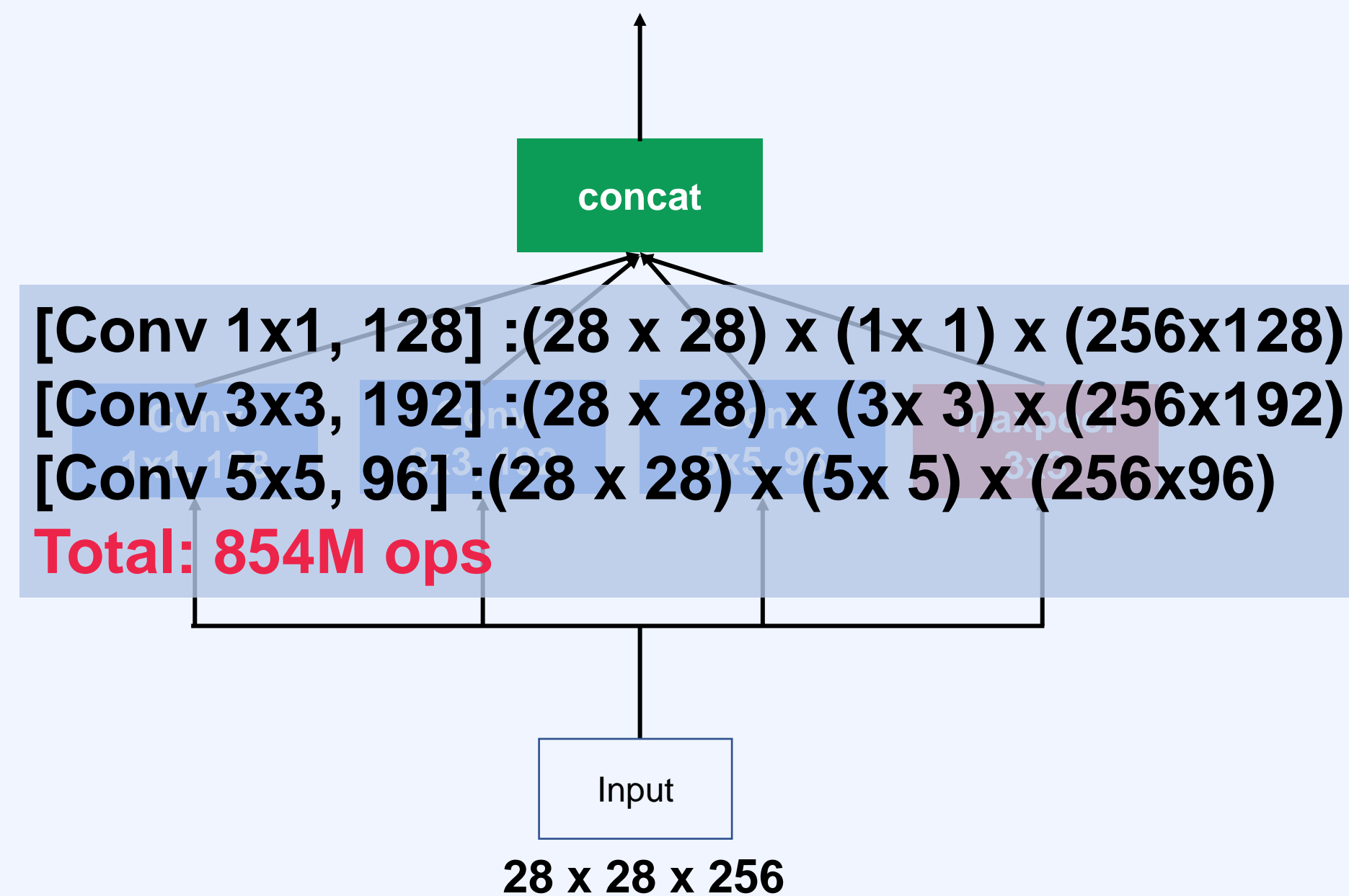
Naïve Inception



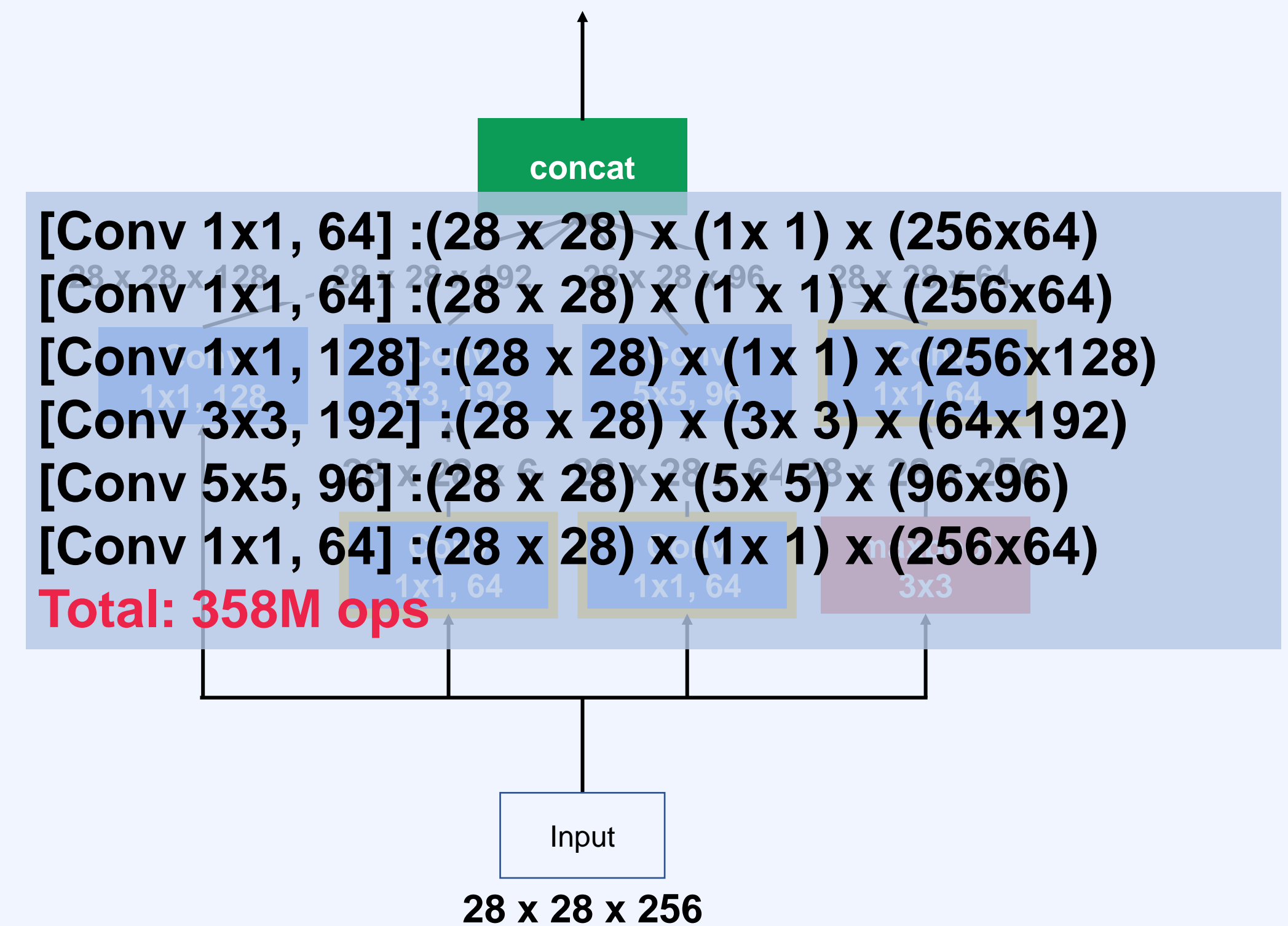
Inception with bottleneck structure

GoogLeNet(Inception V1)

➤ Conv Ops = $(k \times k) \times (m_i \times m_i) \times (c_{i-1} \times c_i)$



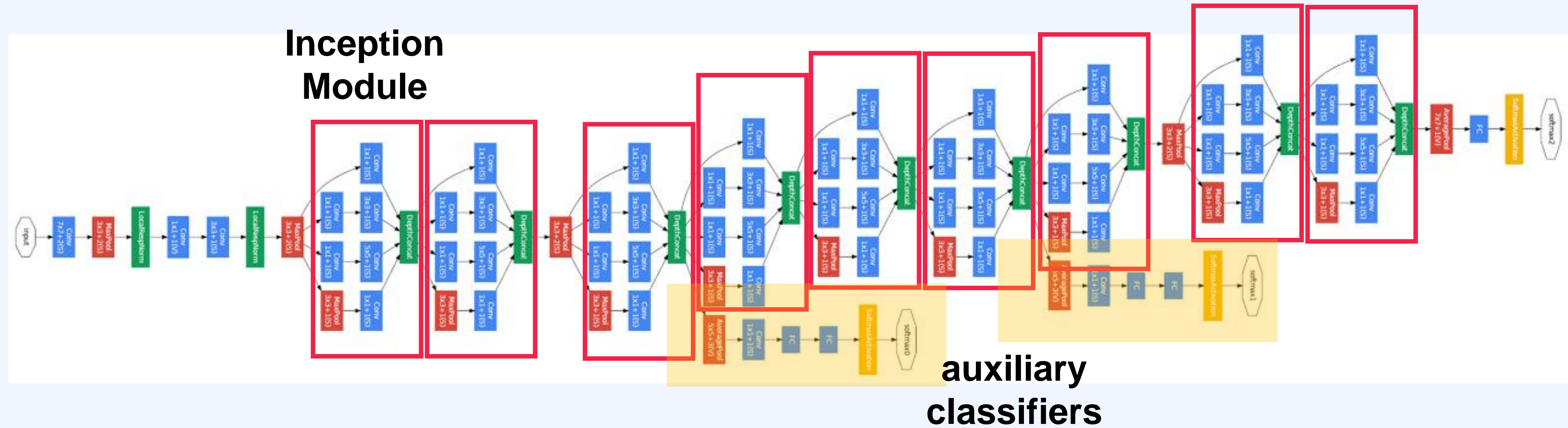
Naïve Inception



Inception with bottleneck structure

GoogLeNet(Inception V1)

- 학습 보조기(auxiliary classifiers)
 - 역전파 신호를 잘 전달하기 위해 학습 시 사용하는 모듈
 - 학습 이후 추론 단계에서는 사용하지 않음



Summary

➤ GoogLeNet(Inception V1)

- Inception
 - Parallel filters concatenation + Bottleneck architecture
- 가중치의 수를 줄임
- 비선형성을 증가시킴
- 모델 구성의 새로운 패러다임
 - 단순히 layer를 쌓아 올리는 구조가 아닌, 모듈/블록 구조를 사용

Quiz

Q

다음 중 GoogLeNet의 특징으로 옳지 않은 것은?

- ① 2014년 LSVRC 1위를 한 22층 구조의 신경망이다.
- ② 1×1, 3×3, 5×5의 다양한 크기의 필터를 사용해서 특징을 추출하는 구조를 가지고 있다.
- ③ 인셉션 모듈의 1×1 커널은 연산량을 줄이고, 신경망을 더욱 깊게 쌓을 수 있게 한다.
- ④ GoogLeNet은 인셉션 구조를 사용하여 100층의 깊은 층을 쌓았음에도 불구하고 기울기 소멸 없이 역전파를 할 수 있다.

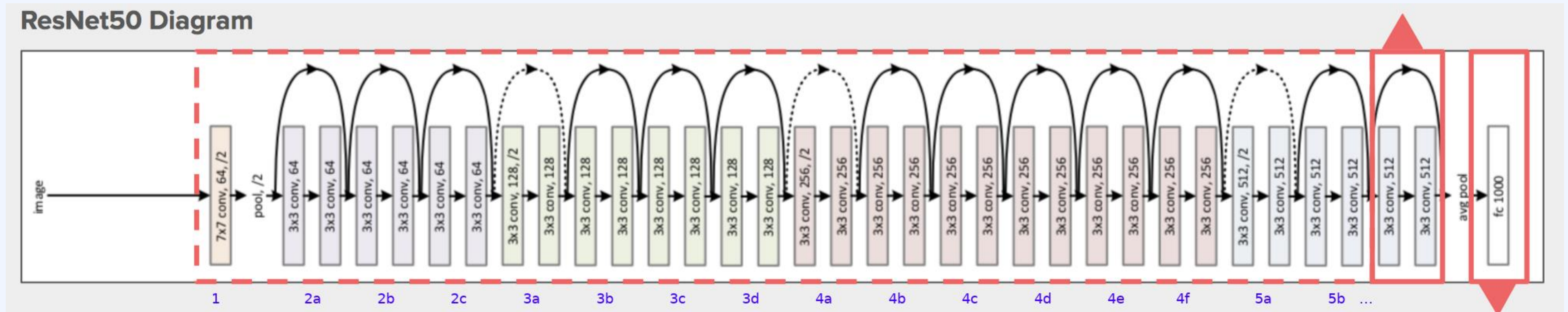
해설

- ▶ GoogLeNet은 역전파 신호를 잘 전달하기 위해 학습 보조기(auxiliary classifiers)를 사용하여 학습 시 사용했다.

CNN
ResNet

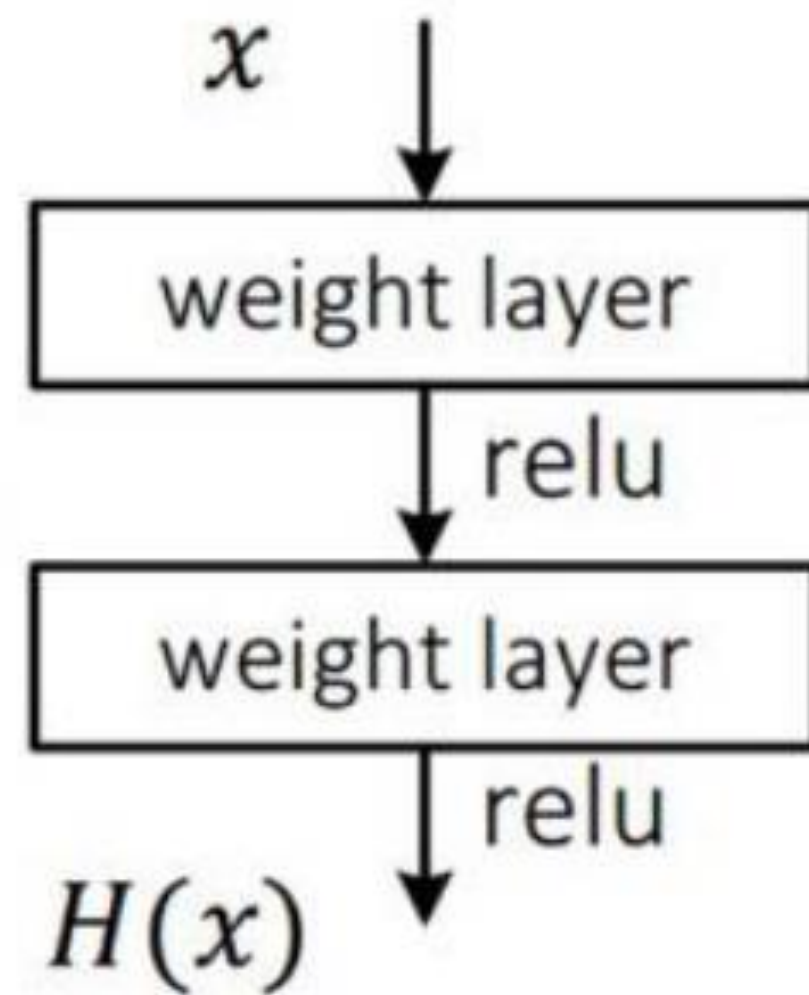
ResNet

- 2015년 LSVRC 1위
- Paper
 - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Microsoft, **Deep Residual Learning for Image Recognition**, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
- 깊은 신경망(152 layers...) 학습이 가능하게 하는 skip connections 구조
- Batch Normalization을 적용
- 분류기 학습을 위한 FC를 구현하지 않고, GAP(Global Average Pooling)을 수행

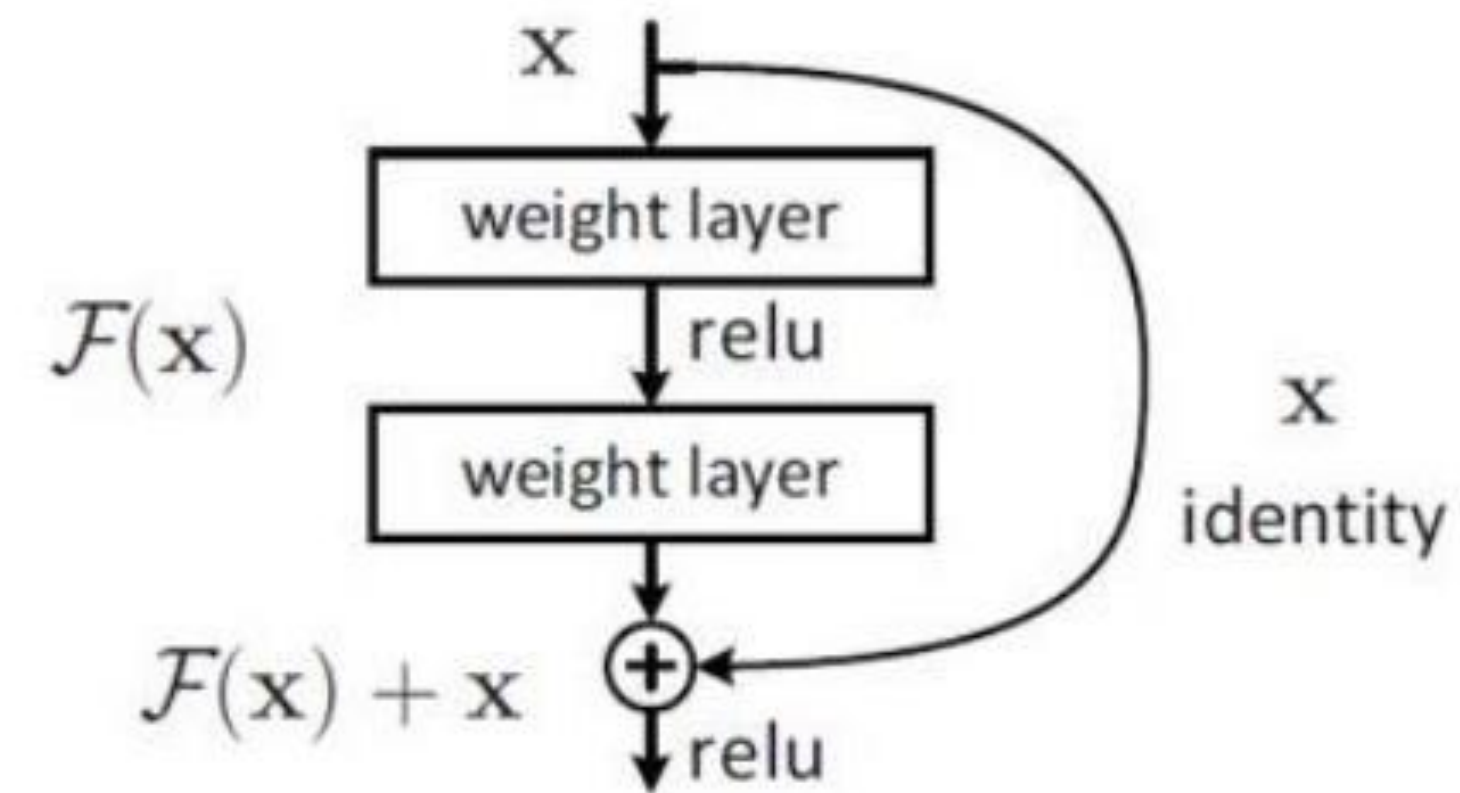


ResNet

- skip connection
 - layer의 입력을 layer의 출력에 바로 연결
 - $H(x) = F(x) + x$



일반적인 구조



Residual 구조

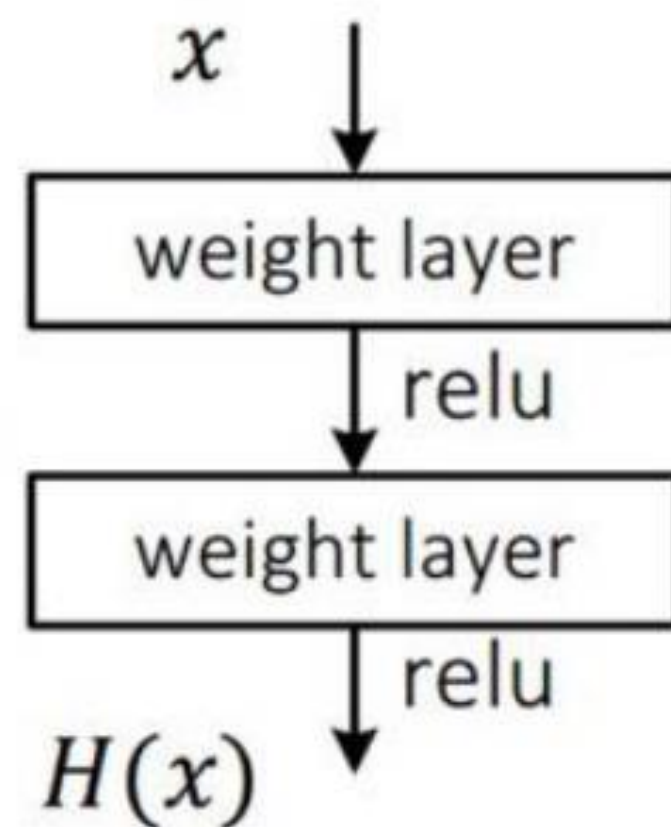
ResNet

➤ Residual 구조

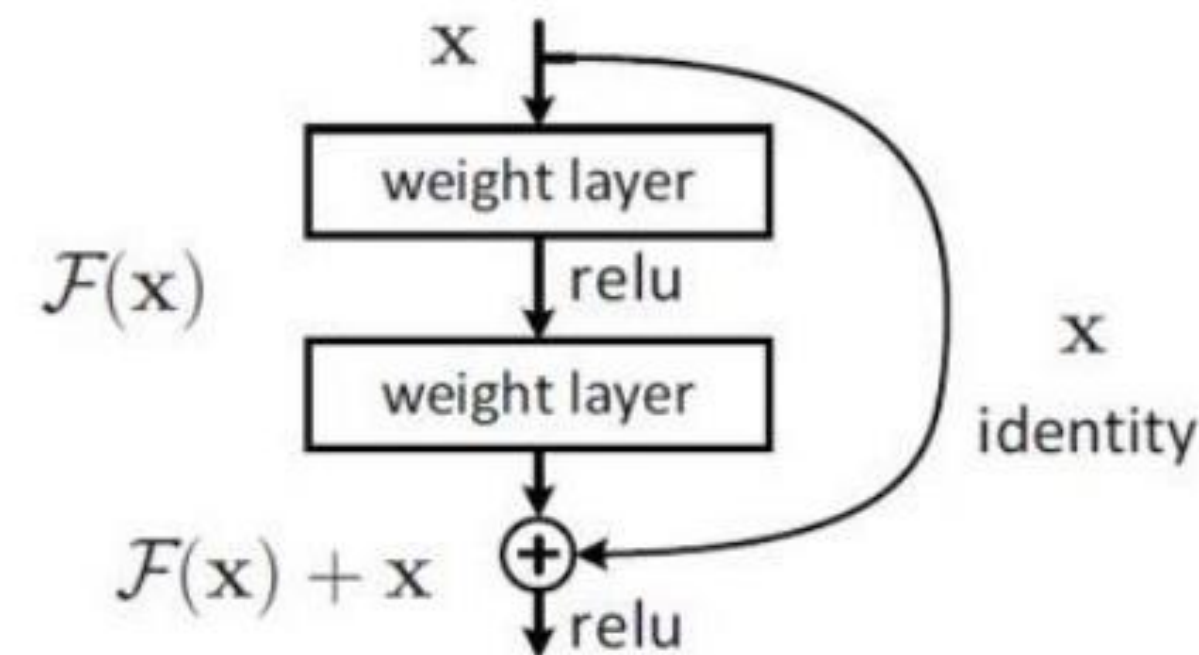
- 일반적인 신경망: $H(x)$ 를 얻기 위해 학습을 수행
- ResNet: 잔차($F(x) = H(x) - x$)를 최소화하는 방향으로 수행

➤ Residual 구조의 효과

- 깊은 층을 가지고 있지만 Gradient vanishing이 발생하지 않고 학습이 잘 됨



일반적인 구조



Residual 구조

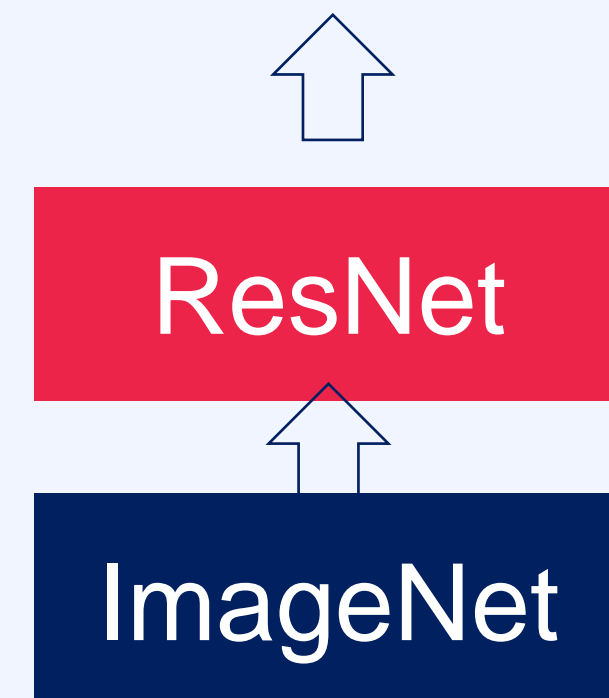
$$H(x) = F(x) + x$$

$$F(x) = H(x) - x$$

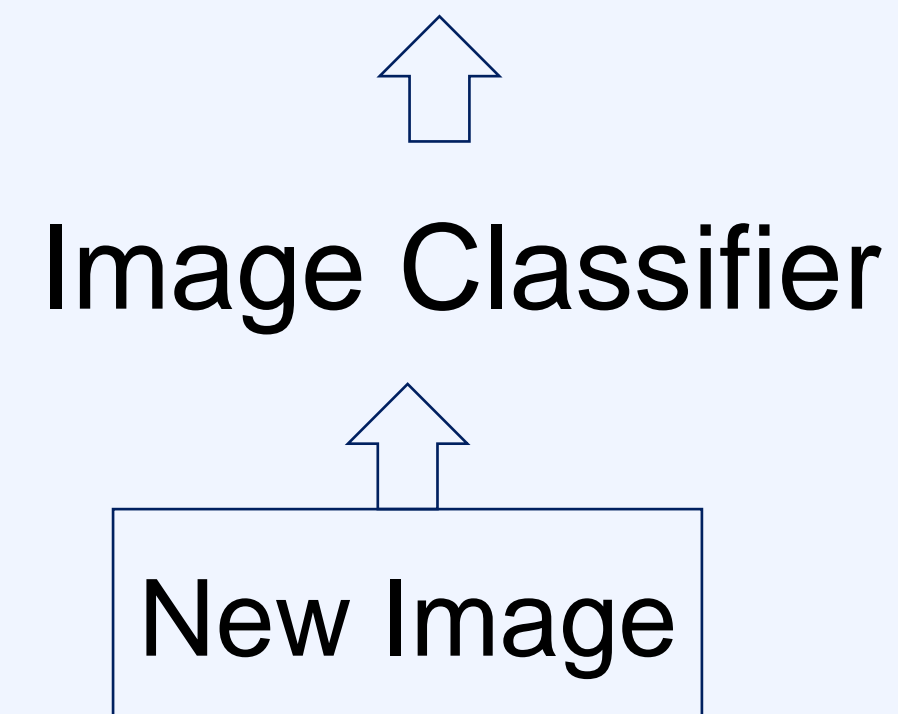
Transfer Learning

- 기존의 만들어진 모델을 사용하여 새로운 모델을 만드는 방법
- 학습을 빠르게 하며 예측 성능을 더 높임
- 이미 잘 훈련된 모델이 있고, 특히 해당 모델과 유사한 문제를 해결 시 효과적

Image Classifier



New Image Classifier



Transfer Learning

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False
```

Parameters of newly constructed modules have requires_grad=True by default

```
num_fts = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_fts, 2)
```

```
model_conv = model_conv.to(device)
```

```
criterion = nn.CrossEntropyLoss()
```

*# Observe that only parameters of final layer are being optimized as
opposed to before.*

```
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)
```

Decay LR by a factor of 0.1 every 7 epochs

```
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

!Transfer Learning

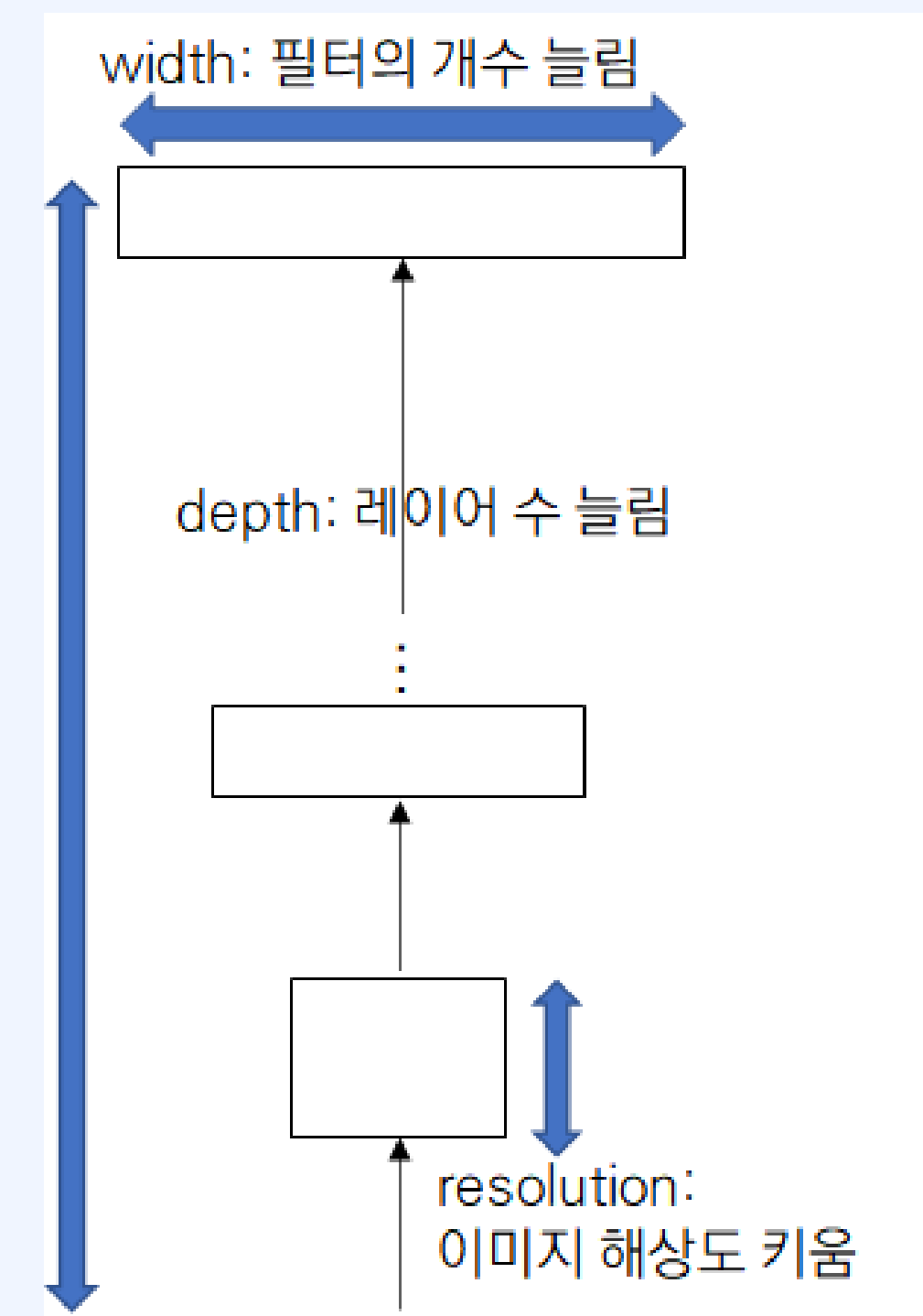
- VGGNet
 - 3x3 kernel

- Inception
 - Module
 - 1x1 kernel
 - Parallel filters

- ResNet
 - Skip-connection

EfficientNet

- depth, width, resolution을 동시에 고려한 compound scaling을 통해 ConvNet기반 이미지 분류 성능을 개선한 모델
- 목적: 이미지 분류
- 개선방향 : depth, width, resolution 동시에 키워서 성능 개선
 - width(필터의 개수 늘림), depth(레이어 수 늘림), resolution(이미지 해상도 키움)
- 성과: 연산은 줄이고, 정확도는 높임




Quiz

```
import torch.nn as nn
from torchvision import models
net = models.resnet18(pretrained=True)
for p in net.parameters():
    p.requires_grad = False
fc_input_dim = net.fc.in_features
net.fc = nn.Linear(fc_input_dim, 2)
```

Q

다음 코드에 대한 설명으로 옳지 않은 것은?

- ① resnet18 신경망을 사용하여 전이학습을 수행하는 코드이다.
- ②  위의 코드를 사용하여 신경망을 학습하면 resnet18의 가중치는 변경된다.
- ③ 2개의 카테고리를 분류할 수 있는 태스크에 적합한 신경망이다.
- ④ resnet18 신경망은 이미 학습이 완료된 파라미터가 설정되어 호출된다.

해설

- ▶ resnet18의 requires_grad = False로 설정하였기 때문에 학습이 진행되더라도 가중치는 변경되지 않는다.

Quiz

Q 사전 학습된 모델을 새로운 문제에 적용하기 위해 일부 가중치를 조절하는 학습 과정을 무엇이라고 하는가?

- ① Fine tuning
- ② Few shot learning
- ③ Fast learning
- ④ Back propagation

해설

- ▶ 미세조정(Fine tuning)은 사전학습(pre-training) 모델을 사용하여 새로운 문제에 적용할 수 있도록 조정하는 과정이다.

Quiz

Q

VGG16 등 사전 훈련된 모델을 사용하는 주된 목적은?

- ① 사용자가 만든 모델보다 CPU와 메모리 사용량이 적기 때문에
- ② 모든 변수가 자동으로 설정되어 사용자가 관여할 것이 없기 때문에
- ③ 사전 훈련된 모델을 사용할 경우, 사용자가 모델을 변경할 필요가 없기 때문에
- ④ 소규모 데이터셋에서 효과적으로 모델을 설계하고 훈련시킬 수 있기 때문에

해설

- ▶ 소규모 데이터셋에서 효과적으로 인공지능 모델을 설계하고 훈련시킬 수 있기 때문에 사전 훈련된 모델을 사용한다.

CNN

모델의 성능을 높이기 위해
사용하는 방법들

생각해보기

➤ 좋은 성능을 갖는 인공신경망이란?

- 일반화(Generalize) 측면
- 정확도(Accuracy) 측면
- 리소스(Memory and computation complexity) 측면
- 학습 속도 측면

인공신경망의 일반화를 위해서..

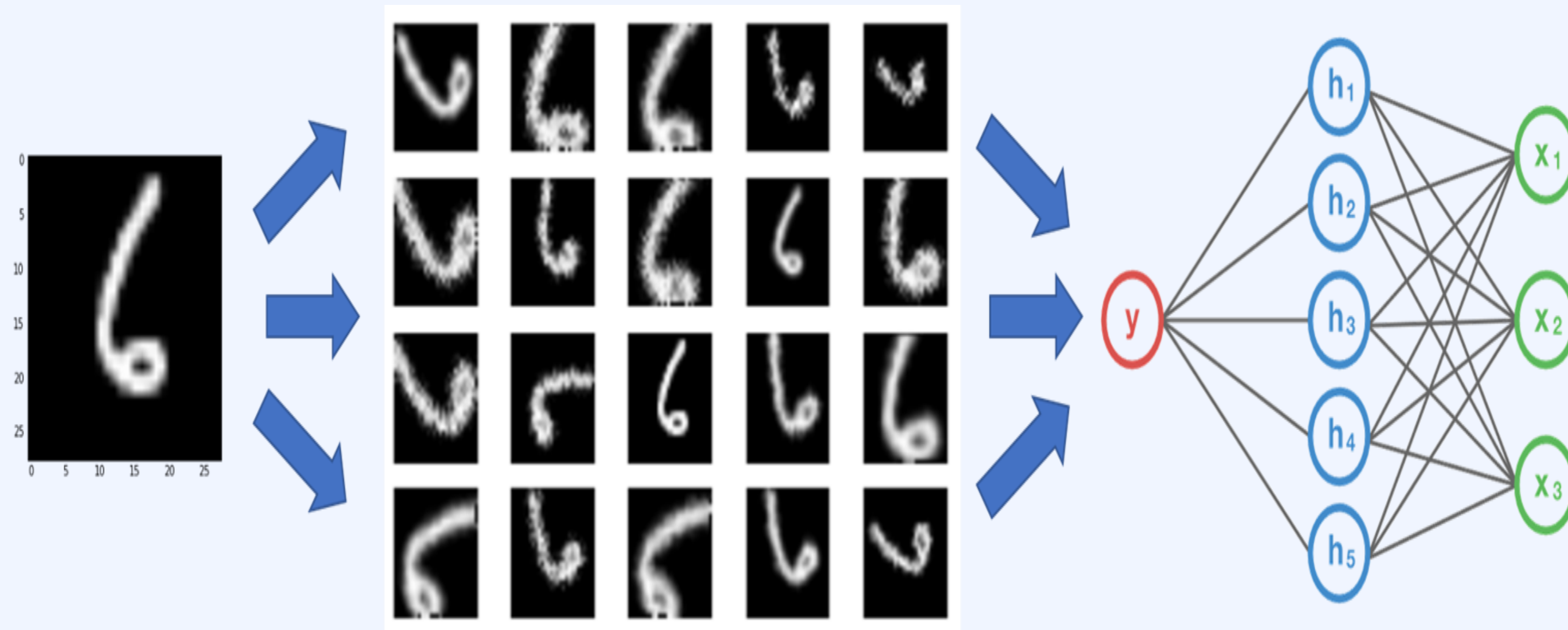
➤ Overfitting을 막기 위한 방법

- 학습용 데이터를 늘린다.
- 이른 종료(Early stopping)
- Regularization : L2, L1, Dropout 등
- 앙상블 모델

학습용 데이터를 늘린다...

➤ Image Augmentation

- 소규모 데이터 셋으로 강력한 학습 모델을 만드는 방법

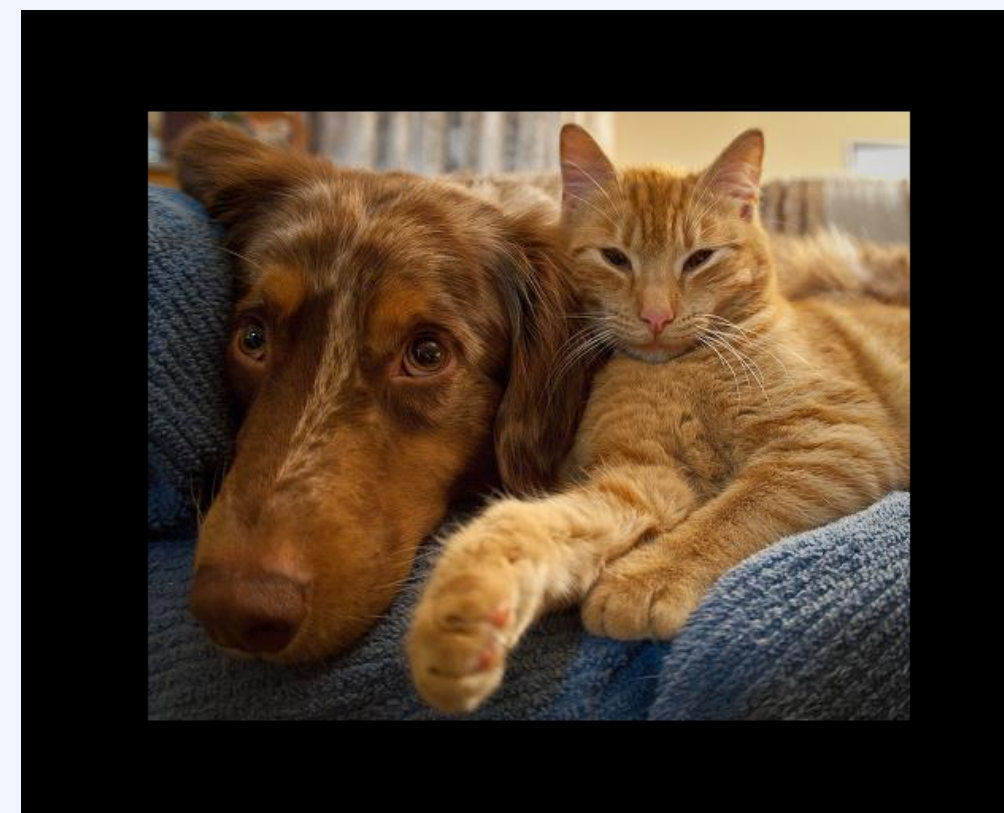


학습용 데이터를 늘린다...

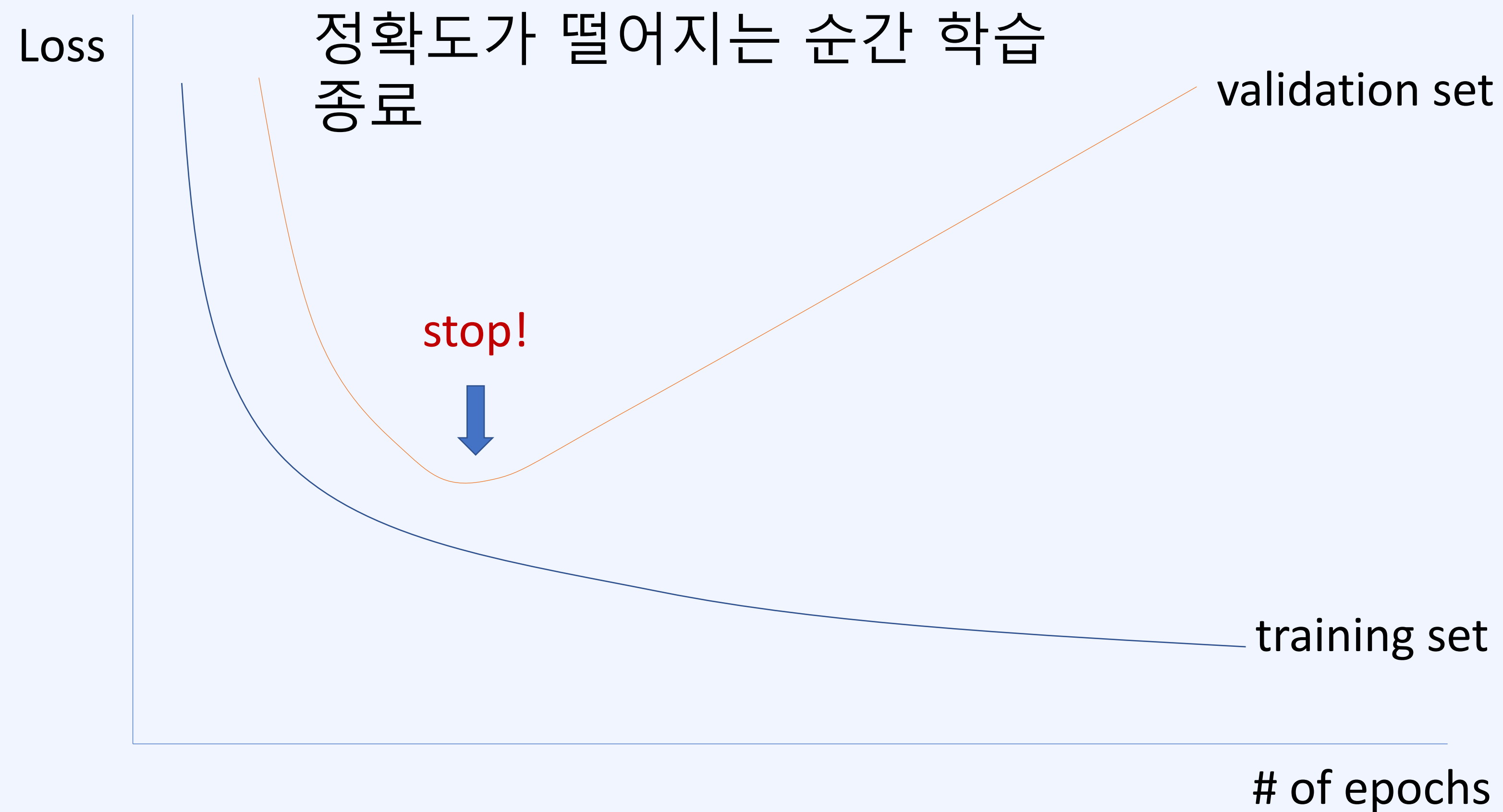
- Image Augmentation
 - rotation, stretching, crop, flip ...
 - 비슷해보이지만, 데이터는 다르다



Original

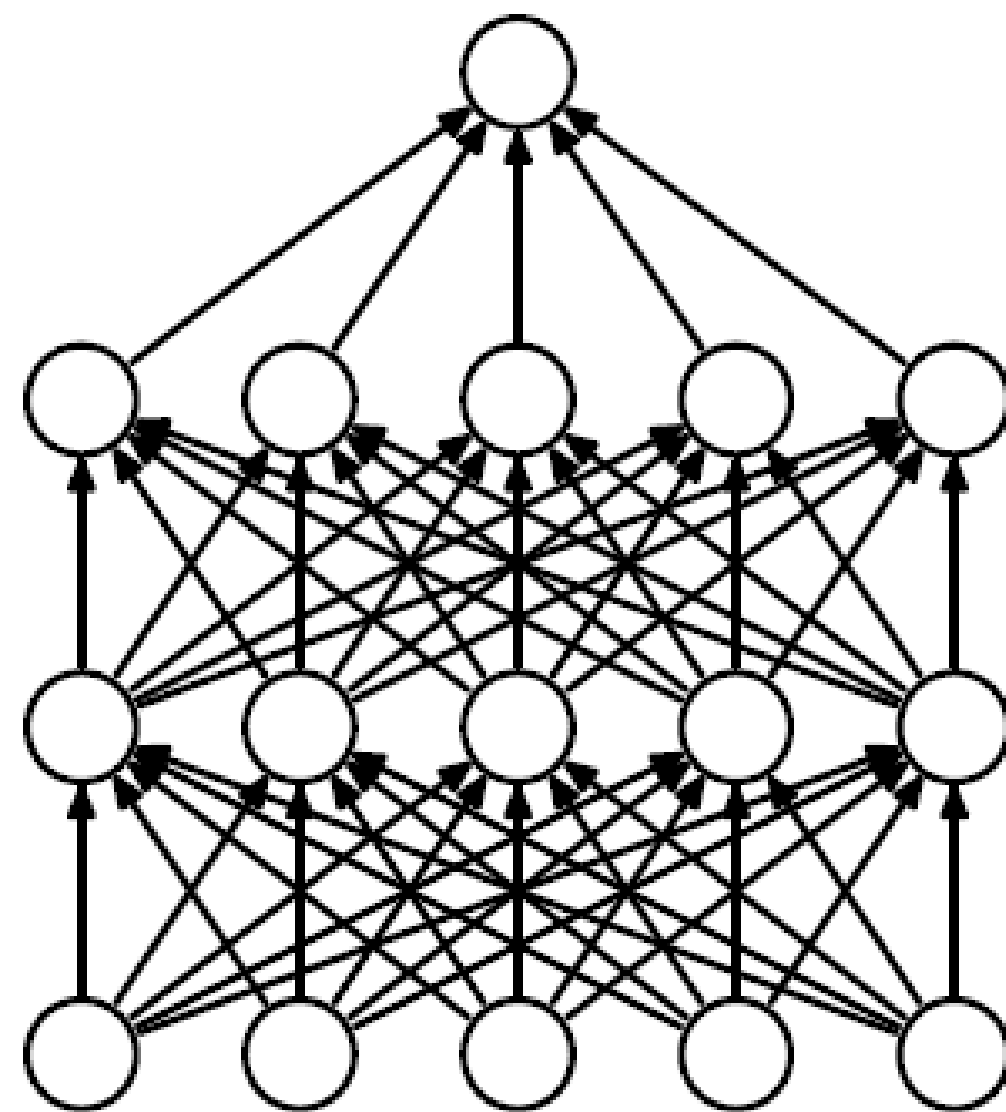


이른 종료(Early stopping)

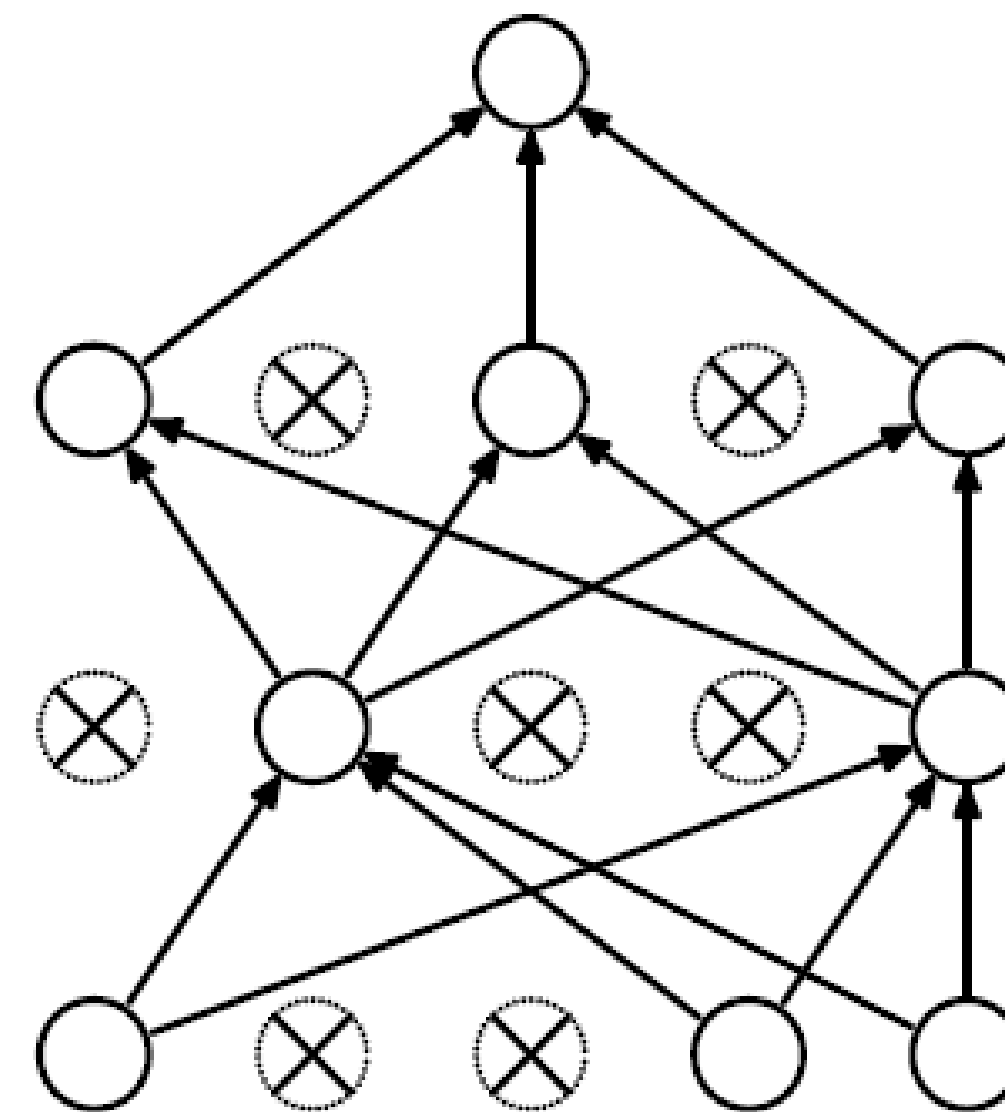


Dropout

- 학습 중에 은닉층의 뉴런을 무작위로 삭제하여 과적합을 예방하는 기법
- dropout(p)
 - p라는 확률로 출력 노드의 신호를 보내다 말다 함
 - 드롭아웃을 적용한 다음에 오는 계층은 앞 계층의 일부 노드들의 신호가 p라는 확률로 단절되기 때문에 훨씬 더 견고하게 신호에 적응



(a) Standard Neural Net



(b) After applying dropout.

Feature map의 사이즈를 줄이기 위한 방법

➤ Pooling

1	2	3	4
5	6	7	8
3	2	1	0
5	4	3	2

Feature map

Filters: 2x2
Stride: 2

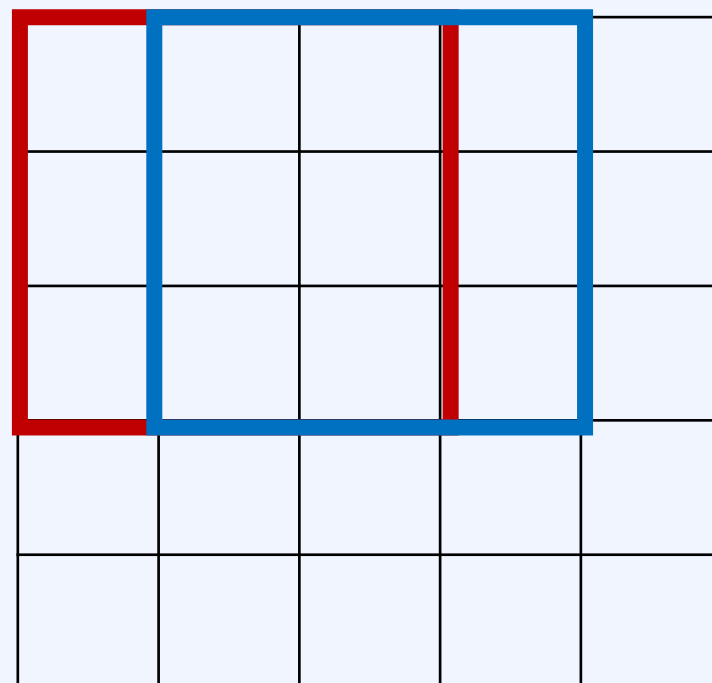
1	2	3	4
5	6	7	8
3	2	1	0
5	4	3	2

Feature map

max pooling

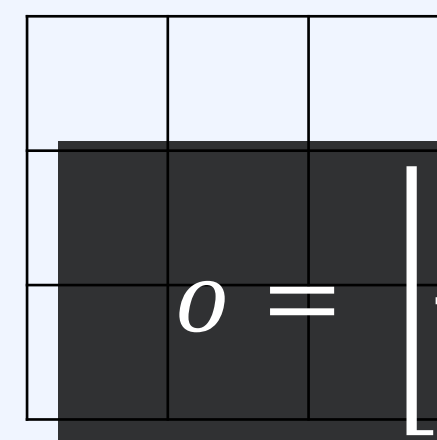
6	8
5	3

➤ Stride



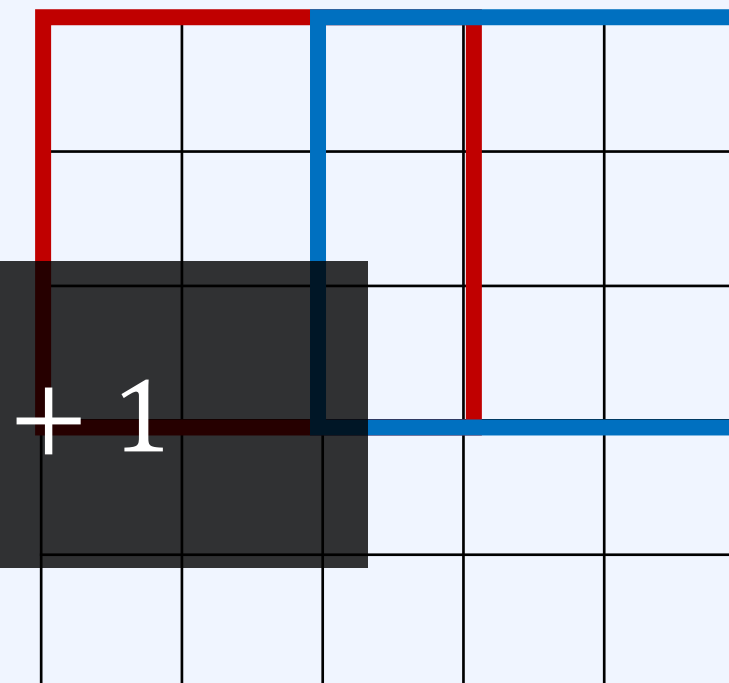
Image

*



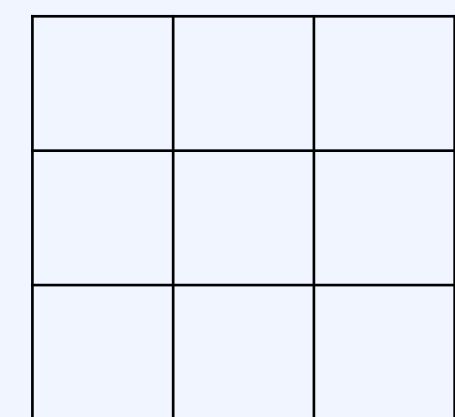
Kernel

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$



Image

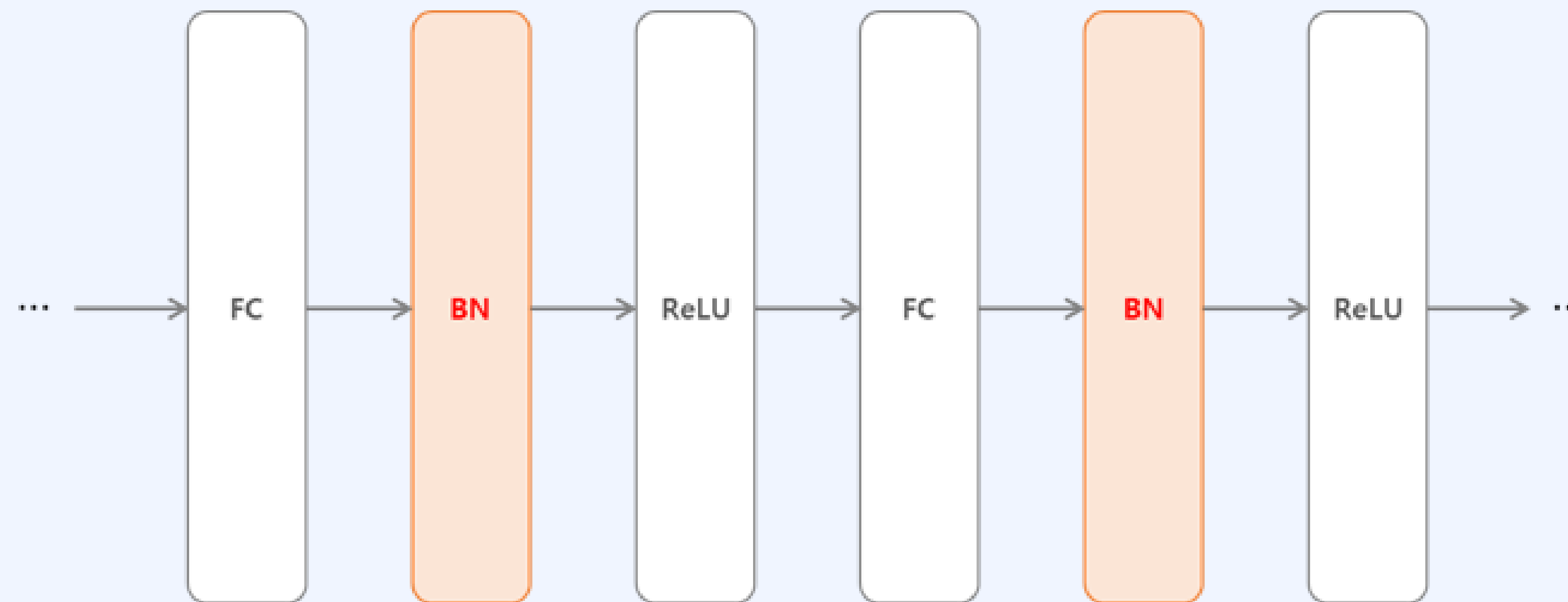
*



Kernel

Batch Normalization

- 한 번에 입력으로 들어오는 배치 단위로 데이터 분포의 평균이 0, 분산이 1이 되도록 정규화 진행
 - 빠른 학습 가능: learning rate를 높게 잡을 수 있음
 - 자체적인 regularization 효과



[BN을 사용한 신경망 예]

Weight Initialization

➤ Xavier Initialization

- **paper:** Xavier Glorot and Yoshua Bengio. "**Understanding the difficulty of training deep feedforward neural networks.**" Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010.
- **activation function :** tanh, sigmoid

`torch.nn.init.xavier_uniform_(tensor, gain=1.0)` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a uniform distribution. The resulting tensor will have values sampled from $\mathcal{U}(-a, a)$ where

$$a = \text{gain} \times \sqrt{\frac{6}{\text{fan_in} + \text{fan_out}}}$$

Also known as Glorot initialization.

`torch.nn.init.xavier_normal_(tensor, gain=1.0)` [SOURCE]

Fills the input *Tensor* with values according to the method described in *Understanding the difficulty of training deep feedforward neural networks* - Glorot, X. & Bengio, Y. (2010), using a normal distribution. The resulting tensor will have values sampled from $\mathcal{N}(0, \text{std}^2)$ where

$$\text{std} = \text{gain} \times \sqrt{\frac{2}{\text{fan_in} + \text{fan_out}}}$$

Also known as Glorot initialization.

➤ He Initialization

- **paper:** Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "**Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.**" In Proceedings of the IEEE international conference on computer vision, pp. 1026-1034. 2015.
- **activation function :** relu

Summary

➤ 좋은 성능을 갖는 인공신경망이란?

- 일반화(Generalize) 측면

- 정확도(Accuracy) 측

- 리소스(Memory and

- 학습 속도 측면

**Image Augmentation, Early stopping, Regularization(L2, L1),
Dropout, 앙상블 모델, Batch Normalization, Weight Initialization,
Pooling, Stride,
3x3 kernel, 1x1 kernel, Skip connection,
Transfer learning**

CNN

1D Conv

Remind

➤ Convolution 연산은 이미지에만 적용되는 것이 아니다!

합성곱

문서 토론

위키백과, 우리 모두의 백과사전.

합성곱(合成-), 또는 **콘벌루션**(convolution)은 하나의 함수와 또 다른 함수를 반전 이동한 값을 곱한 다음, 구간에 대해 적분하여 새로운 함수를 구하는 수학 연산자이다.

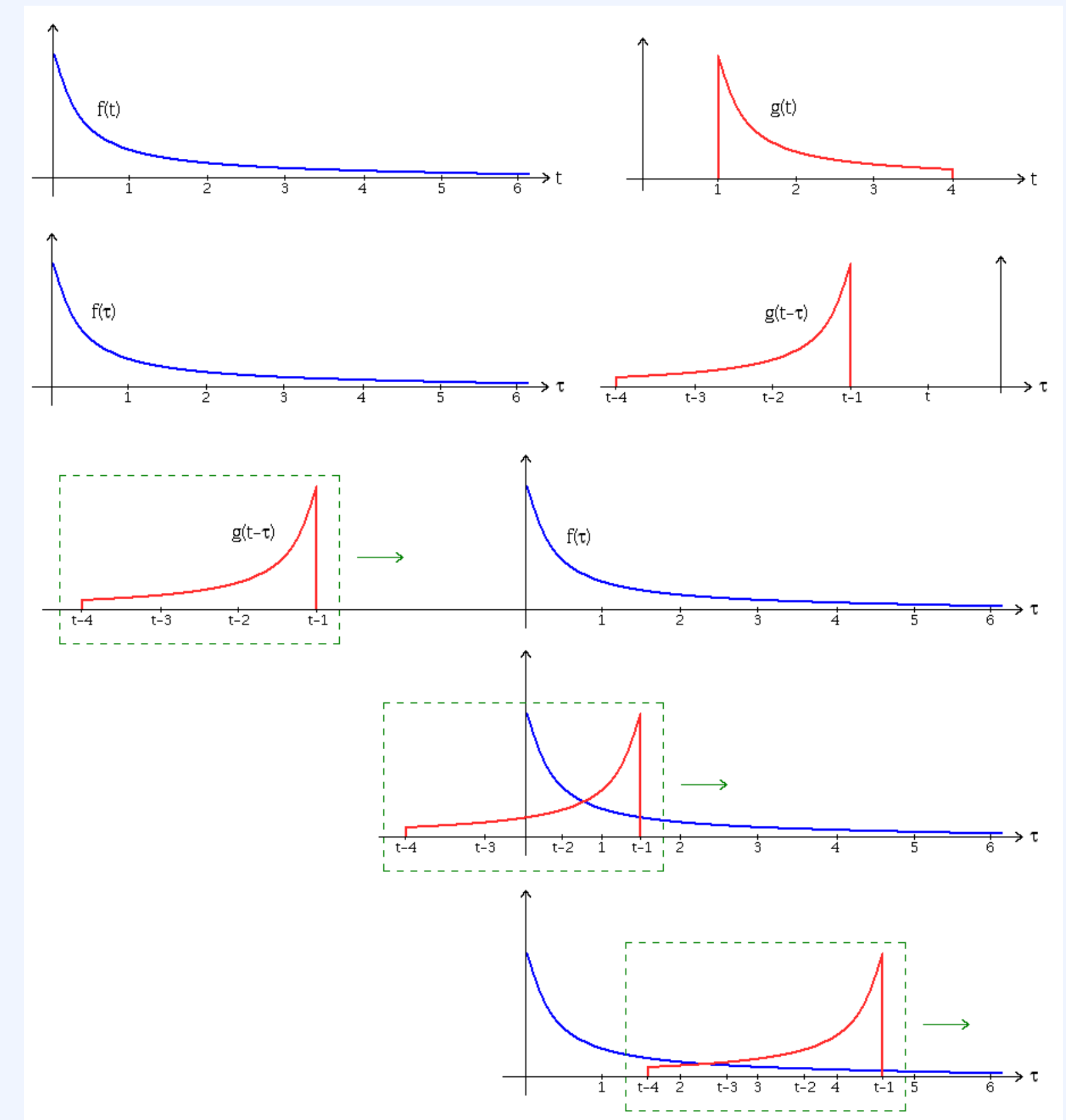
정의 [편집]

두 개의 함수 f 와 g 가 있을 때, 두 함수의 합성곱을 수학 기호로는 $f * g$ 와 같이 표시한다.

합성곱 연산은 두 함수 f, g 가운데 하나의 함수를 반전(reverse), 전이(shift)시킨 다음, 다른 하나의 함수와 곱한 결과를 적분하는 것을 의미한다. 이를 수학 기호로 표시하면 다음과 같다.

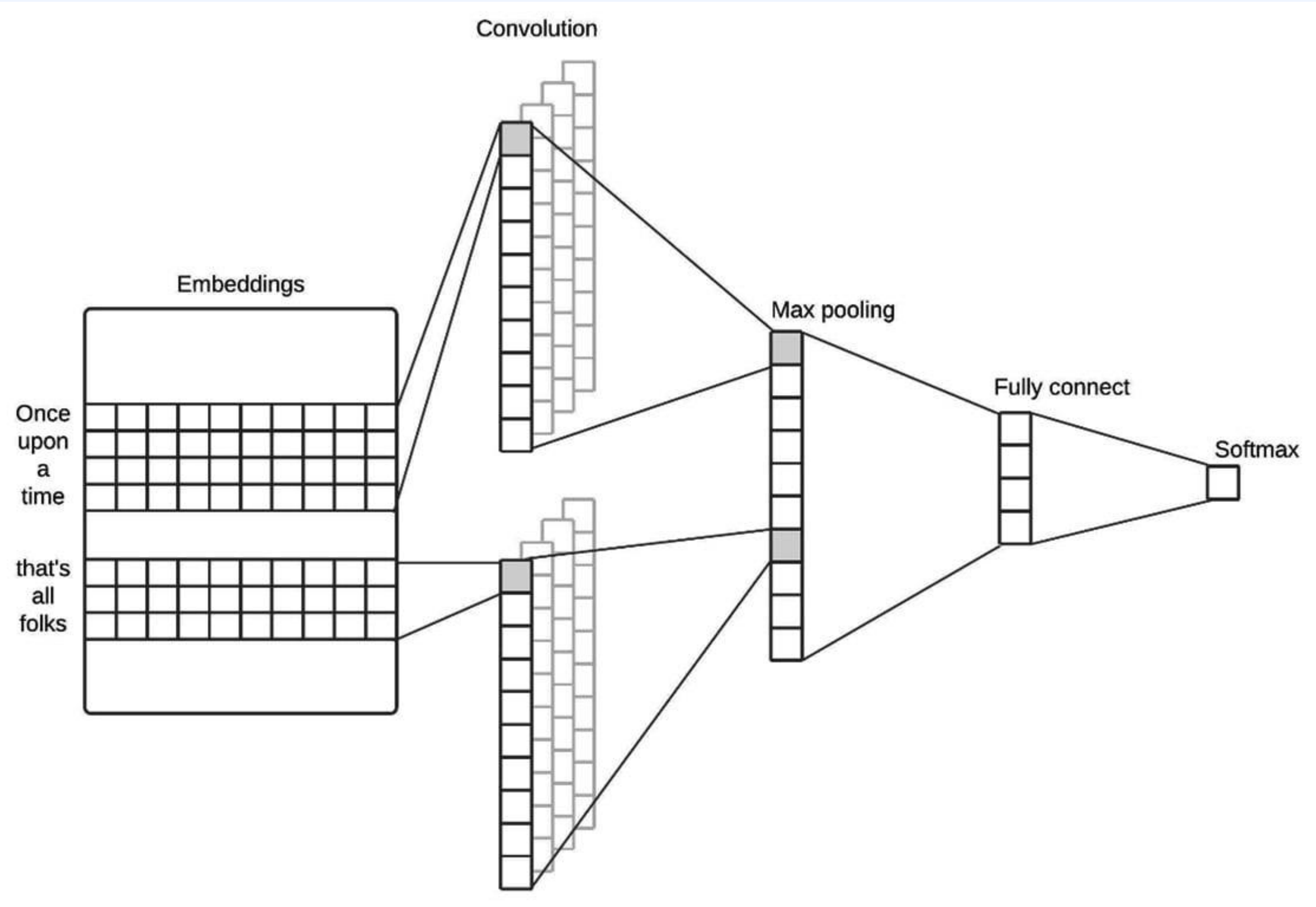
$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau$$

출처: 위키백과



Conv1D

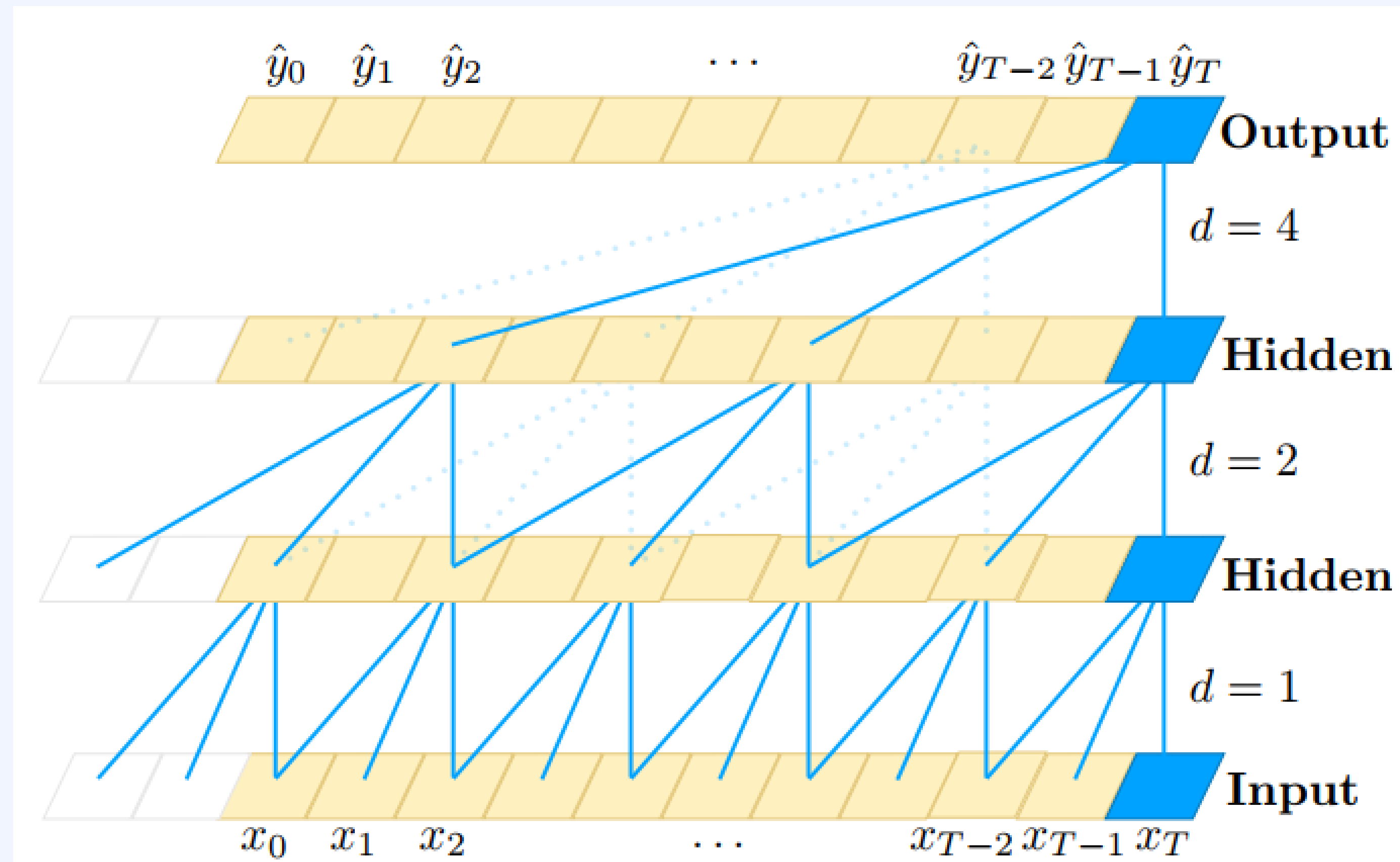
➤ Text Classification with Conv1D



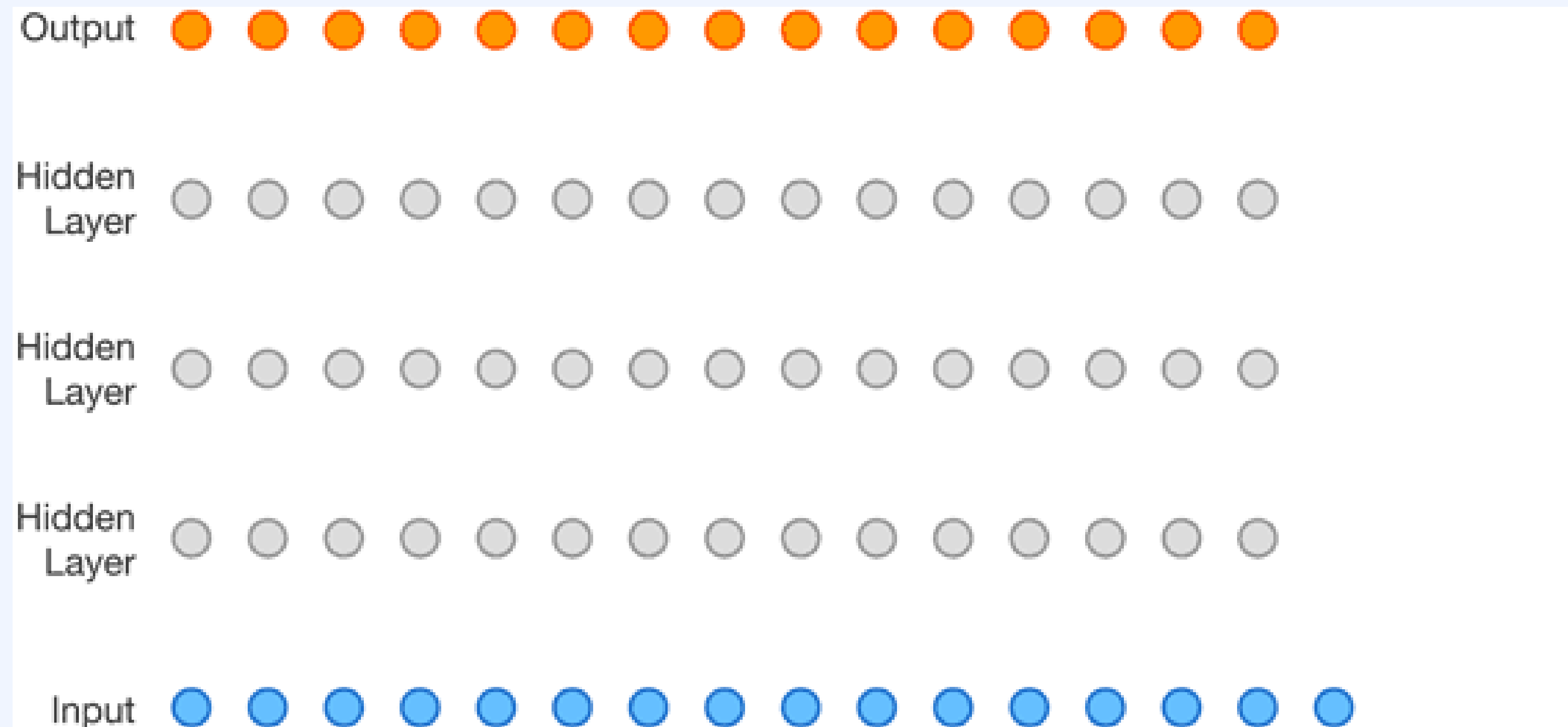
Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 1500, 50)	4429150
conv1d_1 (Conv1D)	(None, 1500, 64)	12864
max_pooling1d_1 (MaxPooling1D)	(None, 750, 64)	0
conv1d_2 (Conv1D)	(None, 750, 128)	24704
dropout_1 (Dropout)	(None, 750, 128)	0
max_pooling1d_2 (MaxPooling1D)	(None, 375, 128)	0
conv1d_3 (Conv1D)	(None, 375, 256)	65792
dropout_2 (Dropout)	(None, 375, 256)	0
max_pooling1d_3 (MaxPooling1D)	(None, 187, 256)	0
dense_1 (Dense)	(None, 187, 128)	32896
dropout_3 (Dropout)	(None, 187, 128)	0
dense_2 (Dense)	(None, 187, 64)	8256
global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 64)	0
dense_3 (Dense)	(None, 1)	65

Conv1D

➤ Time series data with Conv1D



Dilated Convolution



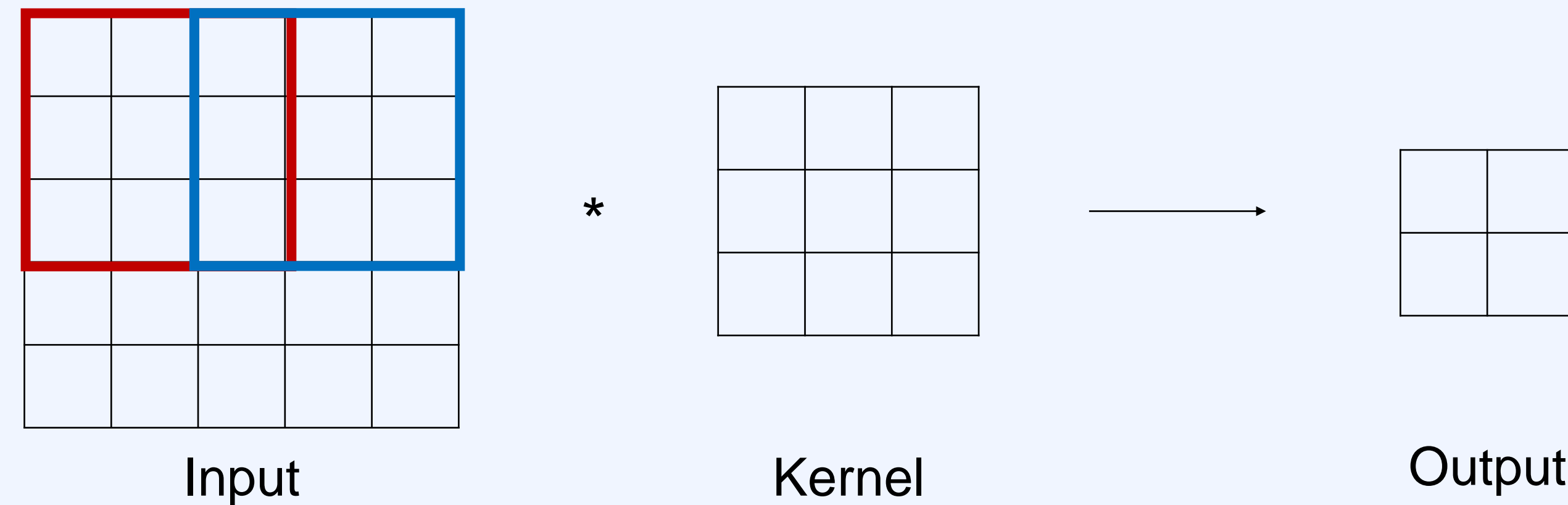
<https://www.deepmind.com/blog/wavenet-a-generative-model-for-raw-audio>

Transposed Convolutions

➤ Paper: A guide to convolution arithmetic for deep learning(Dumoulin, V., & Visin, F., 2016)

➤ Convolution

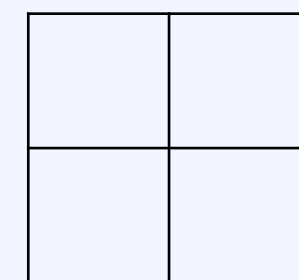
- Input: 5x5
- Kernel: 3x3
- Stride: 2
- Padding: 0
- Output: 2x2



Transposed Convolutions

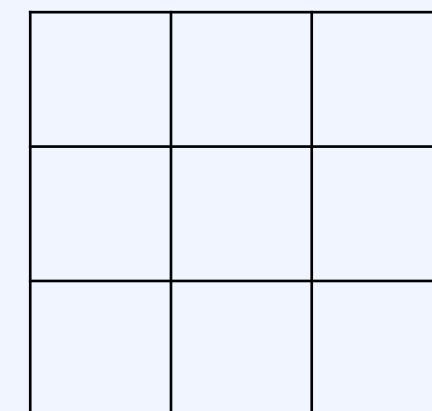
➤ Transposed Convolutions

- Input: 2x2 (1 zero inserted between inputs)
- Kernel: 3x3
- Stride: 1
- Padding: 2x2
- Output: 5x5

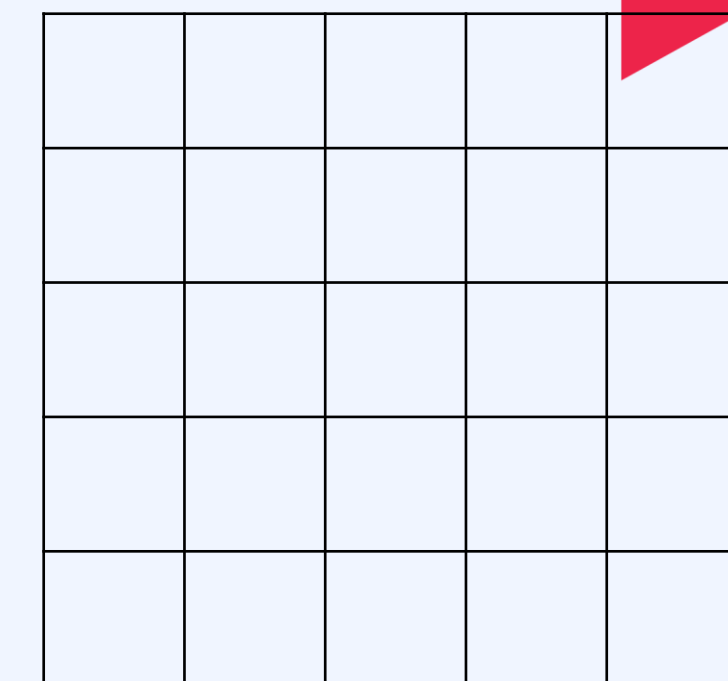
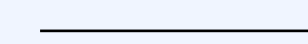


Input

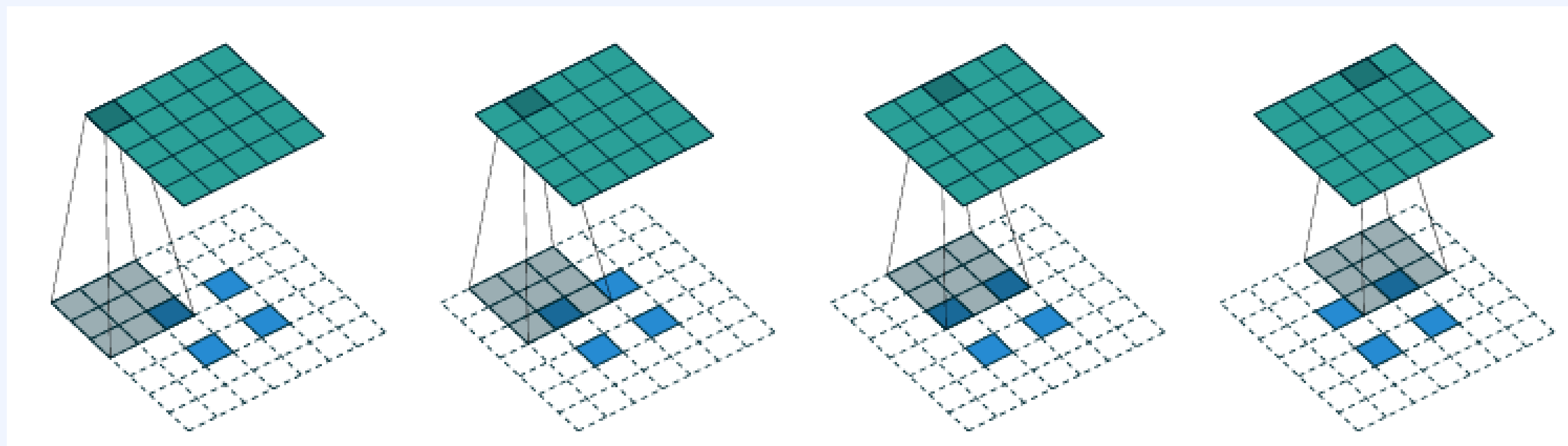
*



Kernel



Output



Closing

CNN