

Deep Learning (II)

2025. 8

Yongjin Jeong, KwangWoon University

[참고] 본 자료에는 인터넷이나 강의자료, 책 등에서 다운받아 사용한 그림이나
수식들이 있으니 다른 용도로 사용하거나 외부로 유출을 금해 주시기 바랍니다.

Convolutional Neural Network (CNN)

Convolution and Cross Correlation

- **Covariance:**
 - express the relation between two random variables (statistics)
- **Correlation:**
 - Normalized Correlation (statistics)
 - Also known as Pearson correlation coefficient
 - (c.f.) Spearman correlation: rank correlation – monotonic function
- **Convolution:**
 - Output of a LTI(linear time invariant) system (signal processing)
 - $h(t)$: impulse response of a filter
- **Cross Correlation:**
 - measure of similarity of two series (signal processing)
 - Also known as a sliding dot product or sliding inner-product
 - Commonly used for searching features
 - $h(t)$ is not flipped

$$Cov[X, Y] = \sigma_{XY} = \frac{1}{n} \sum_i (x_i - u_x)(y_i - u_y)$$

$$\begin{aligned}Corr[X, Y] &= \rho_{XY} = \frac{Cov[X, Y]}{\sqrt{Var[X]}\sqrt{Var[Y]}} \\&= \frac{1}{n} \frac{\sum_i (x_i - u_x)(y_i - u_y)}{\sigma_X \sigma_Y}\end{aligned}$$

$$y(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t - \tau) d\tau$$

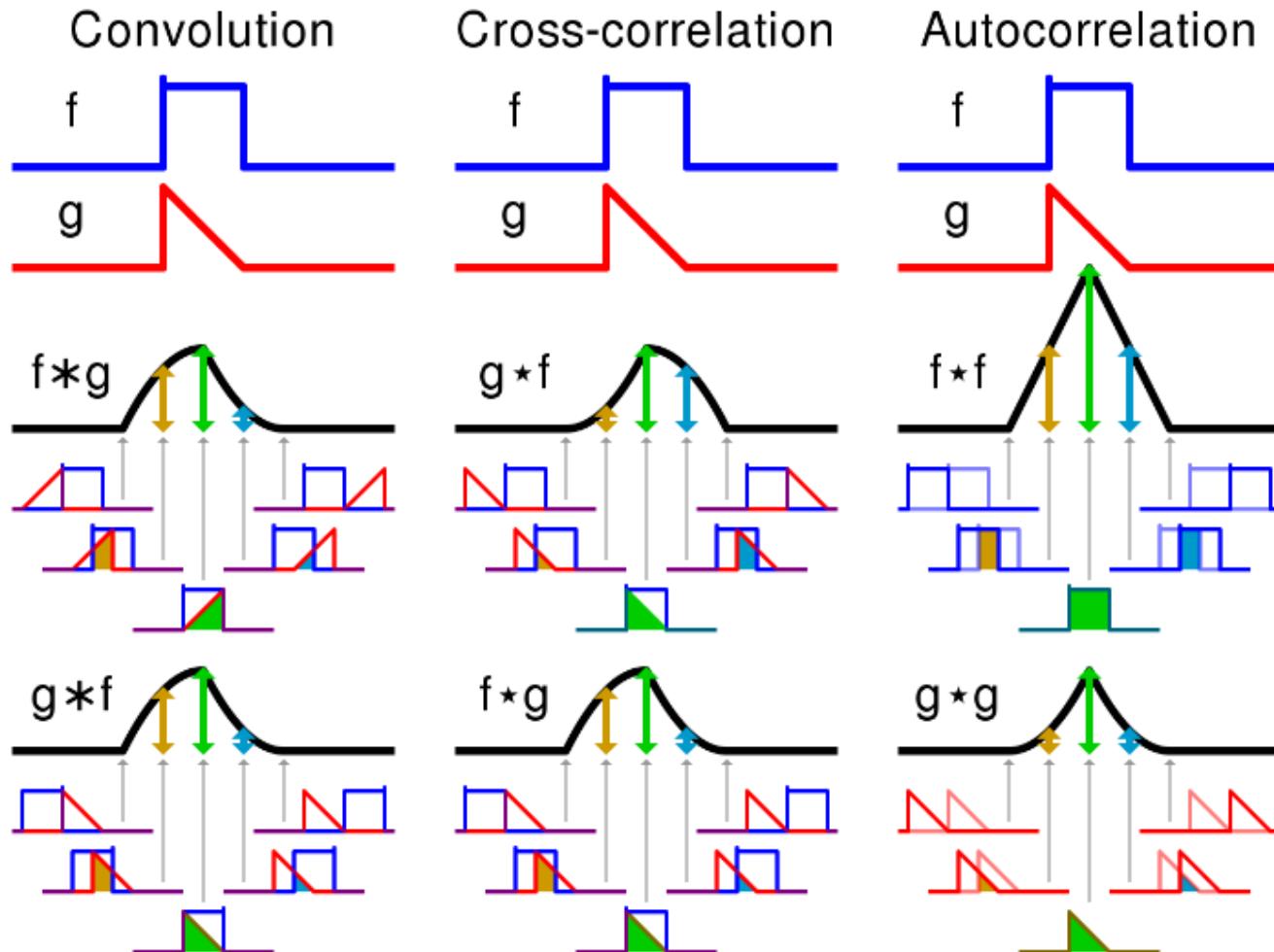
$$y(t) = \sum_{k=-\infty}^{\infty} x(k) \cdot h(t - k)$$

$$y(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t + \tau) d\tau$$

$$y(t) = \sum_{k=-\infty}^{\infty} x(k) \cdot h(t + k)$$

$$[f(t) \star g(t)](t) = [\overline{f(-t)} * g(t)](t)$$

Convolution



<https://en.wikipedia.org/wiki>

$$y(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t - \tau) d\tau$$

$$Y(s) = X(s) \cdot H(s)$$

$$y(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t + \tau) d\tau$$

$$Y(s) = \overline{X(s)} \cdot H(s)$$

In CNN (depth not shown for simplicity):

$$Z(i, j) = \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} X(i + m, j + n) \cdot W(m, n)$$

Visual comparison of convolution, cross-correlation and autocorrelation. For the operations involving function f , and assuming the height of f is 1.0, the value of the result at 5 different points is indicated by the shaded area below each point. Also, the vertical symmetry of f is the reason $f * g$ and $f * g$ are identical in this example.

CNN 동기: 기존 모델(MLP)의 한계

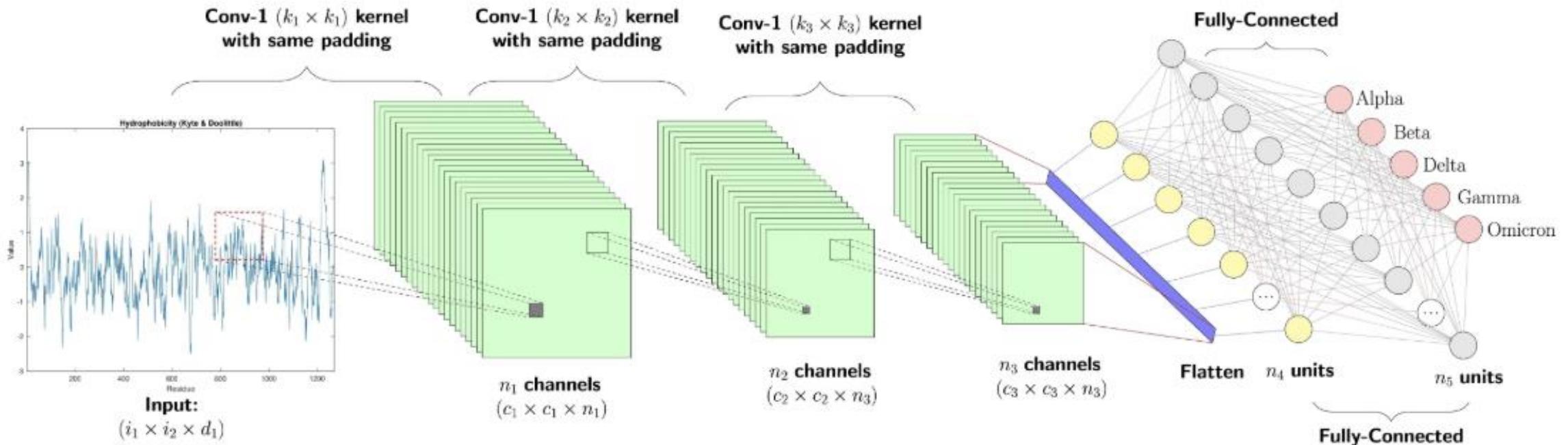
- **파라미터(가중치)의 폭증:** 100x100 픽셀의 작은 흑백 이미지(10,000 픽셀)의 경우
 - 이 이미지를 1차원으로 펼쳐(Flatten) 10,000개의 입력 노드로 만든다
 - 만약 첫 번째 은닉층에 1,000개의 노드가 있다면, 이 첫 번째 계층에만 $10,000 \times 1,000 = 10,000,000$ (천만 개)의 가중치(파라미터)가 필요.
 - 컬러 이미지(RGB)나 더 큰 이미지(1000x1000)가 되면 이 숫자는 천문학적으로 증가
 - 결과: 계산량이 너무 많아 학습이 비효율적이고, 파라미터가 너무 많아 과적합(Overfitting)이 발생하기 매우 쉽다.
- **공간적/지역적 정보의 손실**
 - 이미지는 '공간적 구조(Spatial Structure)'가 매우 중요하다.
 - 특정 픽셀은 그 주변 픽셀과 밀접하게 연관되어 있다. (예: '고양이 눈' 픽셀 옆에는 '고양이 코'나 '얼굴 털' 픽셀이 있을 가능성이 높다.)
 - MLP는 이미지를 1차원 벡터로 펼쳐버린다. 이 과정에서 픽셀 간의 2D/3D 공간 관계(누가 누구 옆에 있는지) 정보가 완전히 사라진다.
 - 또한, '고양이'가 이미지 왼쪽 위에 있든, 오른쪽 아래에 있든 같은 '고양이'로 인식해야 하는데 (Translation Invariance, 이동 불변성), MLP는 위치가 바뀌면 완전히 다른 입력으로 받아들여 학습이 어렵다.

CNN 동기: CNN의 혁신적 해결책

- **컨볼루션 (Convolution) - 지역적 정보 보존**
 - MLP처럼 전체 이미지를 한 번에 보는 것이 아니라, '필터(Filter)' 또는 '커널(Kernel)'이라는 작은 스캔 창(예: 3x3, 5x5)을 사용한다.
 - 이 필터가 이미지 전체를 훑고 지나가며(Convolution 연산) '지역적 특징(Local Feature)'(예: 모서리, 선, 질감)을 추출.
 - **효과: 이미지를 2D 구조를 그대로 유지하며 픽셀 간의 공간 정보를 보존할 수 있다.**
- **가중치 공유 (Parameter Sharing) - 파라미터 폭증 해결**
 - CNN의 가장 핵심적인 아이디어로 이미지의 서로 다른 위치에서 동일한 특징을 감지하기 위해 '하나의 필터(동일한 가중치)'를 공유.
 - 예를 들어, 수직선을 감지하는 3x3 필터 하나가 있다면, 이 필터 하나가 이미지의 왼쪽 위에서 도 수직선을 찾고, 오른쪽 아래에서도 같은 필터로 수직선을 찾는다.
 - **효과:** 10,000,000개의 가중치가 필요했던 MLP와 달리, CNN은 단 9개(3x3 필터)의 가중치만으로 이미지 전체의 수직선 특징을 추출할 수 있다. 이로 인해 파라미터 수가 극적으로 줄어들어 학습이 효율적이고 과적합 위험이 낮아진다.
- **풀링 (Pooling)**
 - 컨볼루션을 통해 얻은 특징 맵(Feature Map)의 크기를 줄여(Subsampling) 계산량을 줄이고, 특징의 사소한 위치 변화에 덜 민감하게(Robust) 만든다.

CNN 동기: CNN의 혁신적 해결책

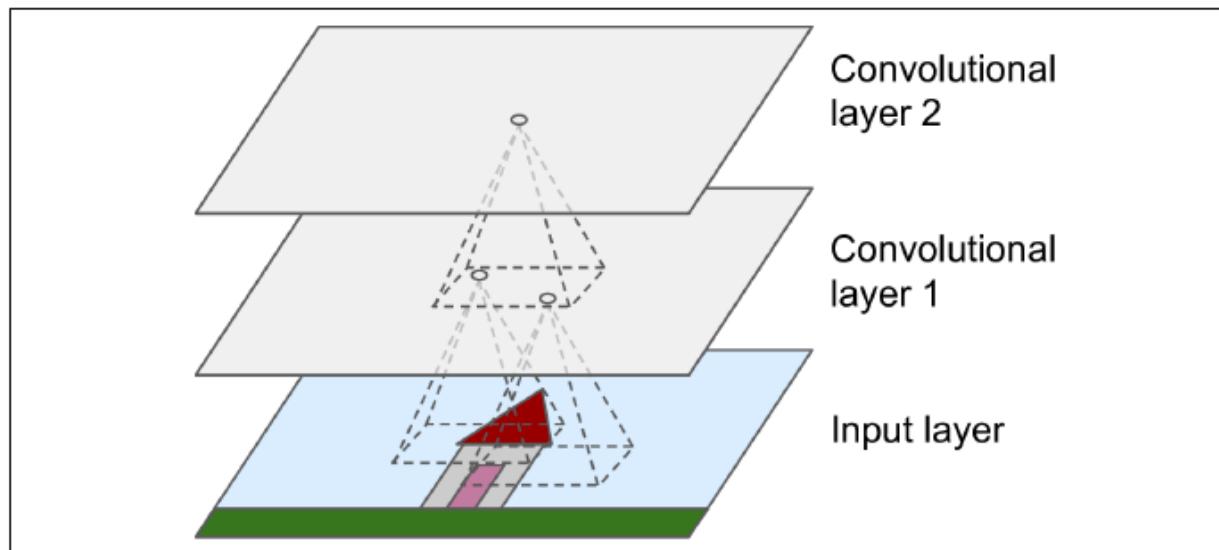
- CNN은 앞의 두 가지 문제를 해결하기 위해 인간의 시각 처리 방식(시신경)에서 영감을 받아 설계



- '영역' (Area): 입력 이미지(또는 이전 층의 특징 맵)에서 필터(Filter)가 한 번에 훑어보는 부분 (이 영역을 해당 필터(혹은 다음 층 뉴런)의 '수용 영역(Receptive Field)'이라고 부름)
- '작은 점' (Dot): 컨볼루션 연산의 결과로 다음 층에 생성되는 '특징 맵(Feature Map)'의 픽셀(뉴런) 하나를 의미. (**해당 영역에 필터가 찾도록 학습된 특정 특징(Feature)**이 얼마나 강하게 존재하는가"를 나타내는 점수 또는 활성도를 나타냄)

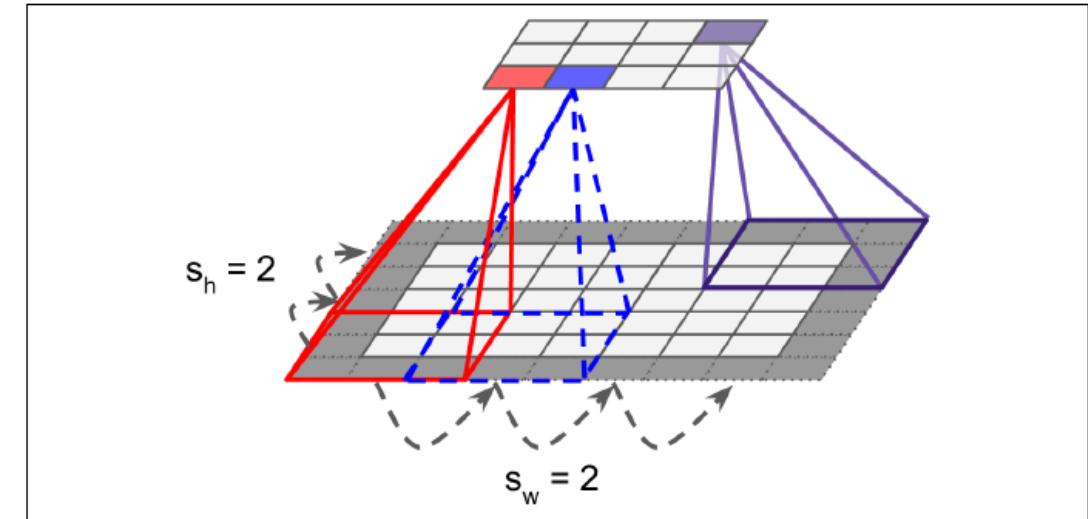
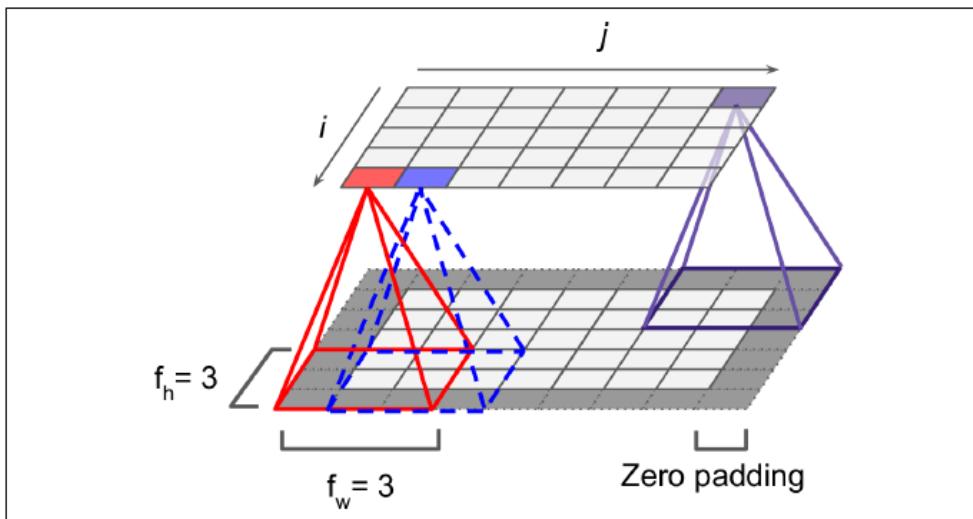
Convolutional Layer

- **The most important building block of CNN**
 - Neurons are connected only to their receptive fields of the previous layer
 - Allowing the network to concentrate on small low-level features
 - Then, assembled into larger higher-level features in the next layer
 - This hierarchical architecture is very common in real-world images.



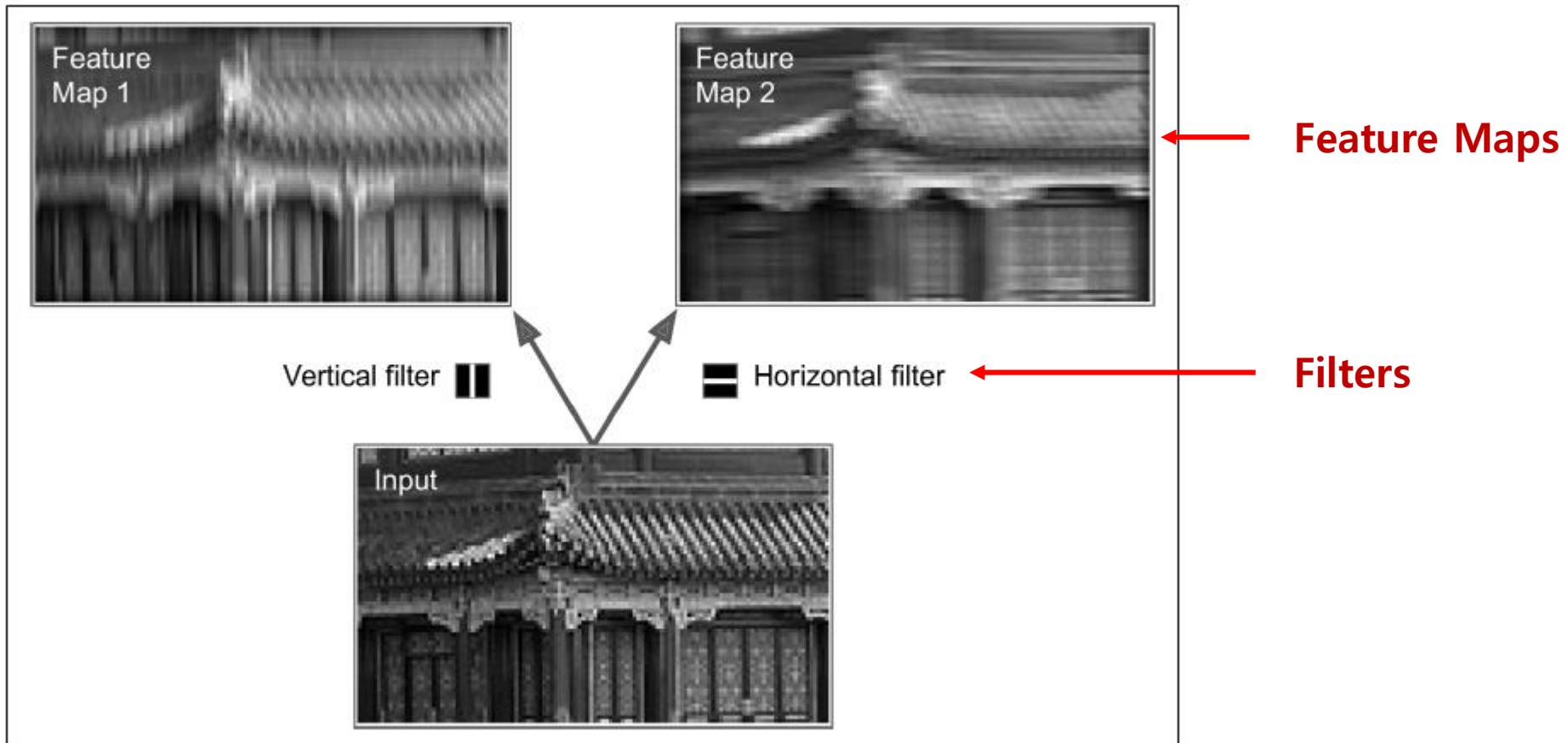
Convolutional Layer

- Connection between layers and zero padding
- Reducing dimensionality using Stride of 2



Filters

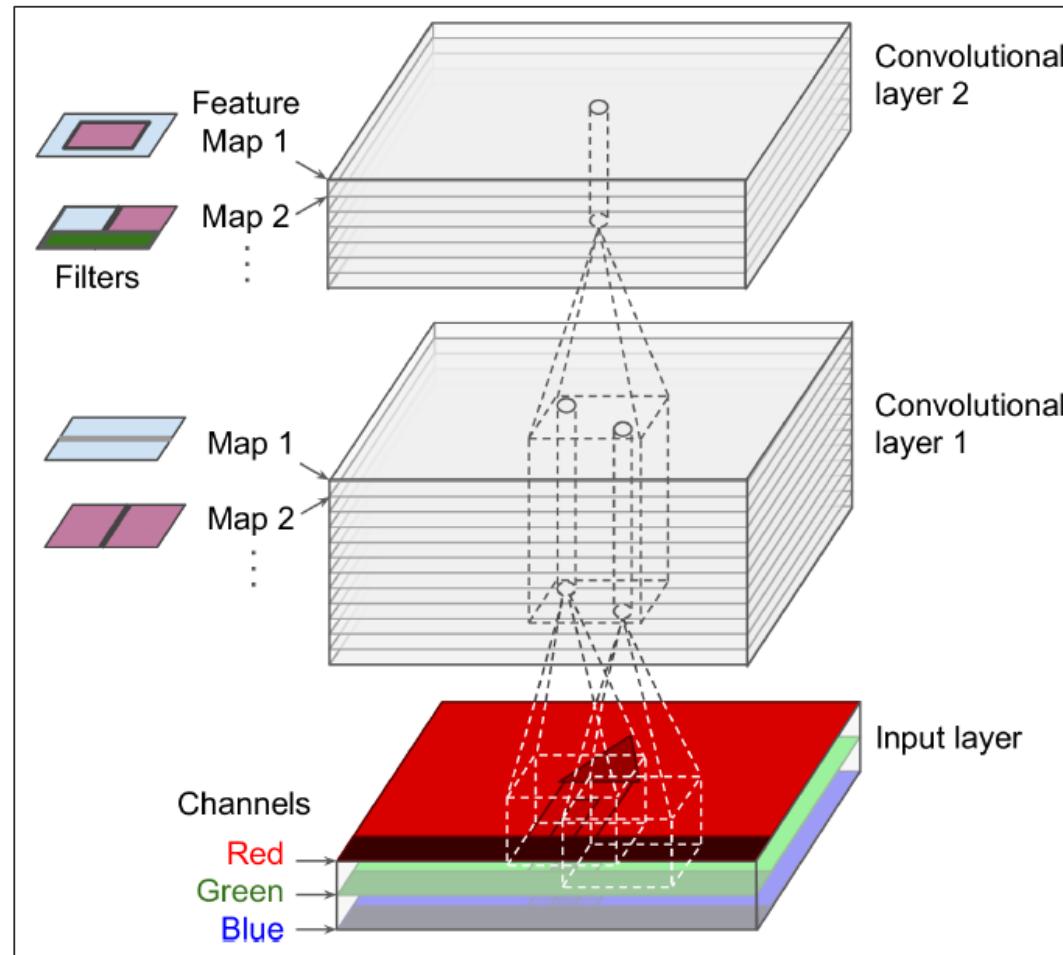
- Filters and Feature Maps



Applying two different filters to get two feature maps

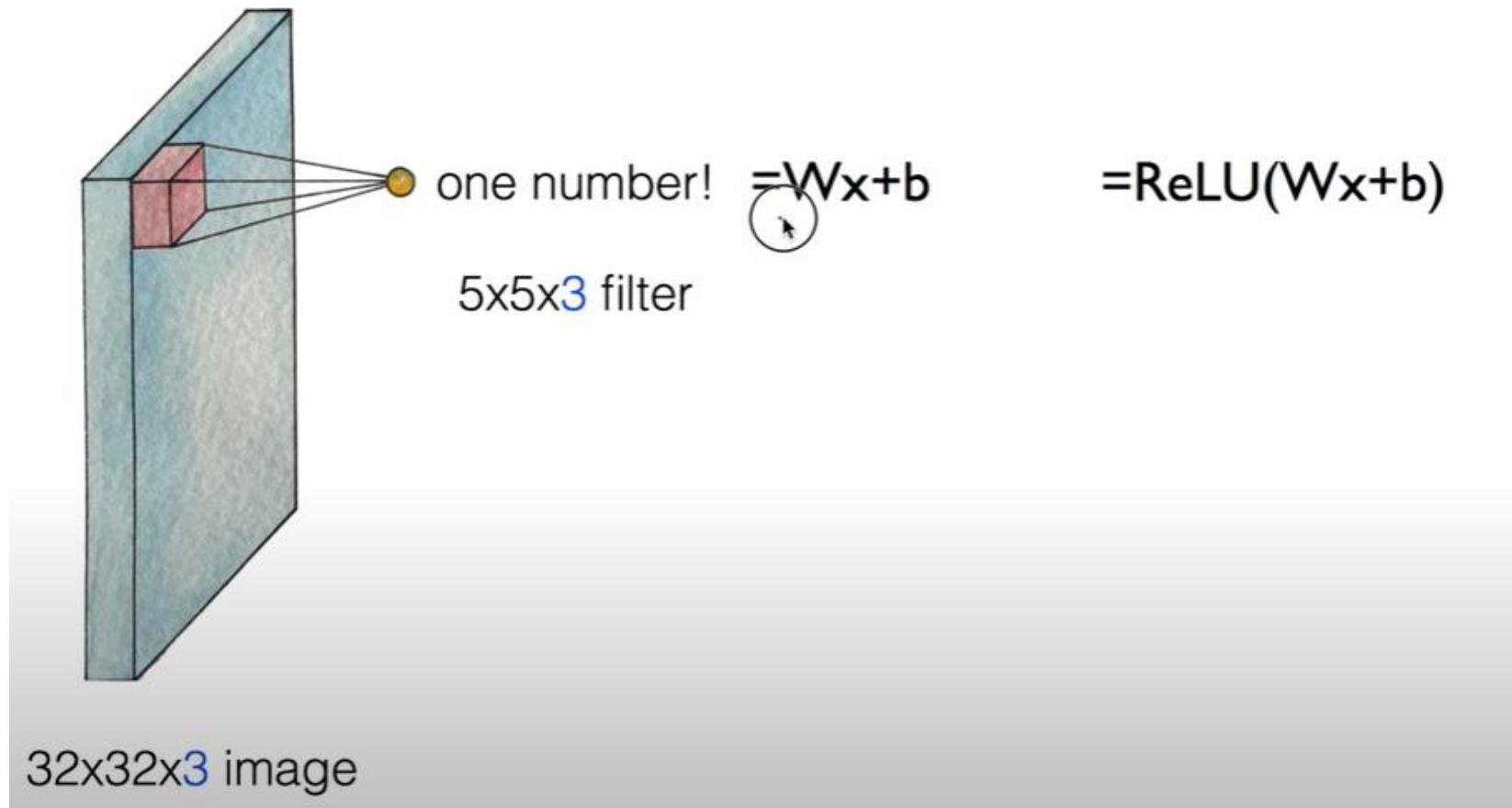
Filters

- Stacking Multiple Feature Maps



CNN operation

Get one number using the filter



Convolution

입력 데이터(height, width)에 대해 필터(filter)을
일정 간격(Stride) 만큼 이동해 가며 행렬 곱셈 연산 수행

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

입력 데이터



2	0	1
0	1	2
1	0	2

필터



15	16
6	15



1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



2	0	1
0	1	2
1	0	2



15	

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

입력 데이터



2	0	1
0	1	2
1	0	2

Filter



15	16
6	15

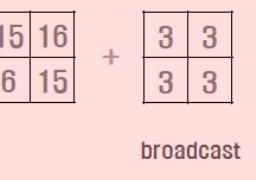


bias

18	19
9	18



출력 데이터



broadcast



1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



2	0	1
0	1	2
1	0	2



15	16
6	15



1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



2	0	1
0	1	2
1	0	2



15	16
6	15



1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



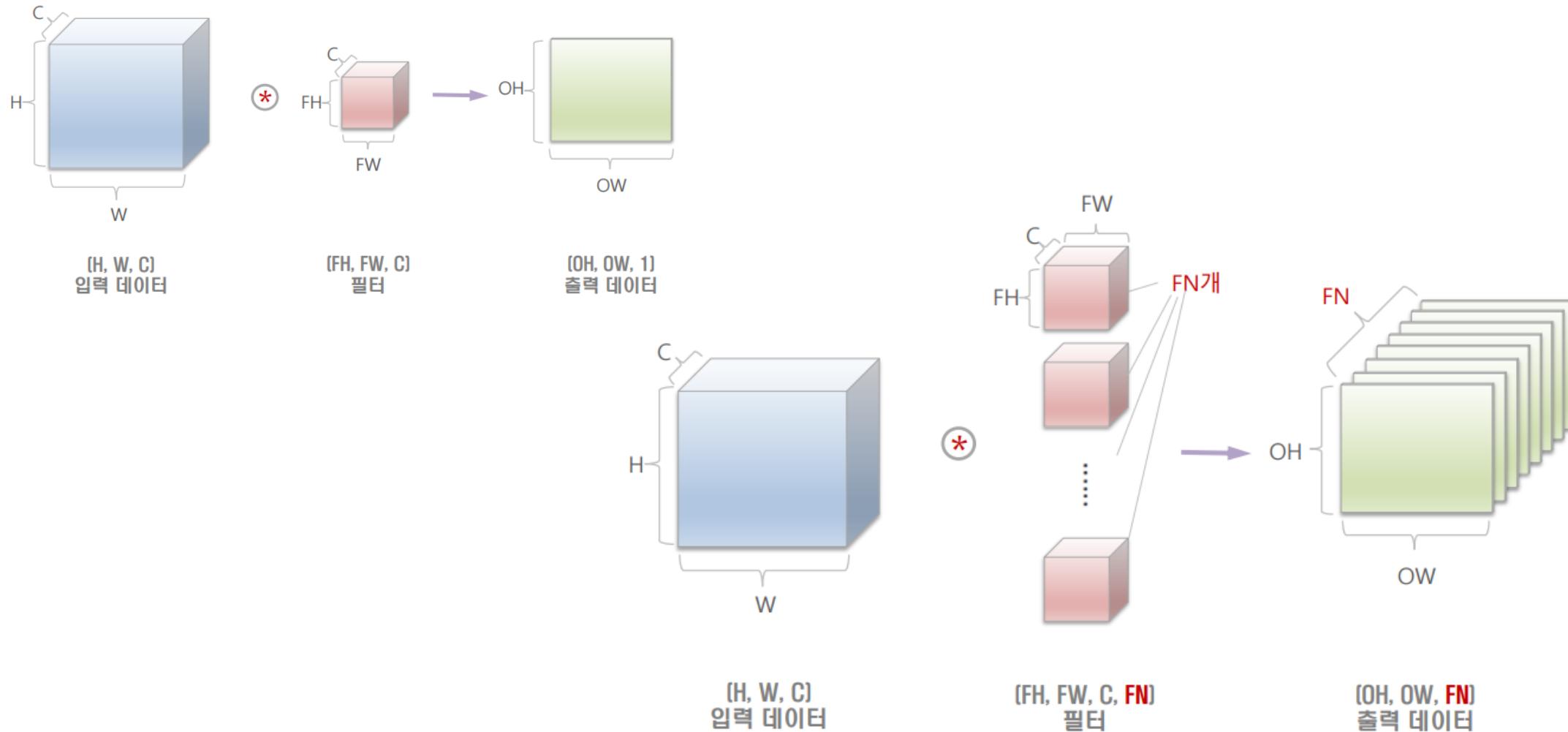
2	0	1
0	1	2
1	0	2



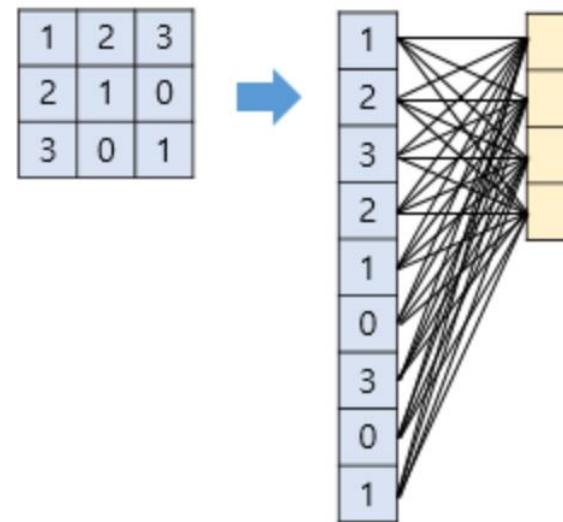
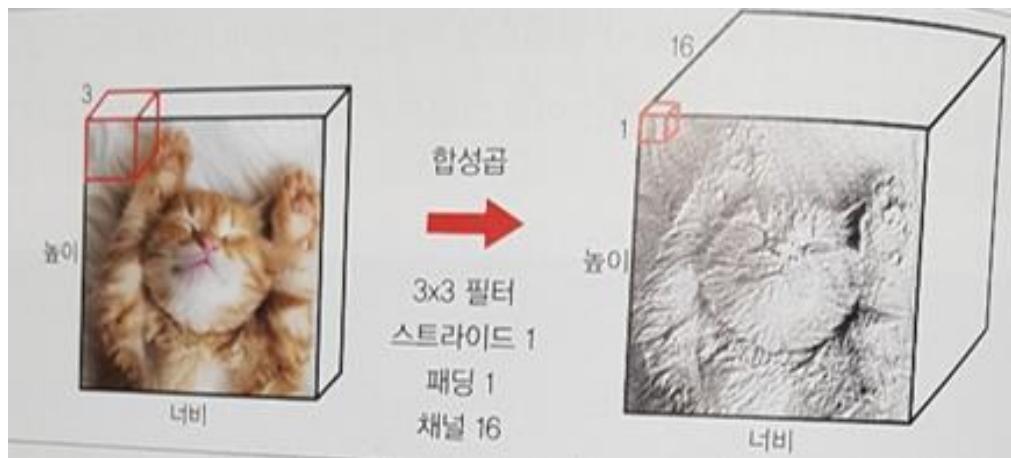
15	16
6	15



3-Dimensional Data Convolution



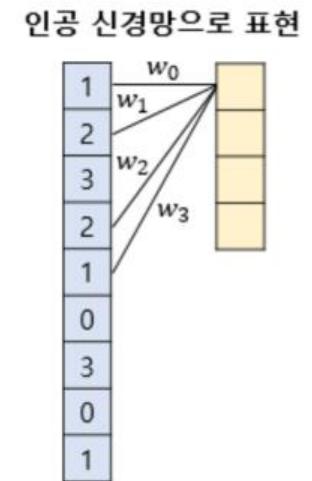
Different views of Convolution



Convolution 층 연산의 3차원적 이해



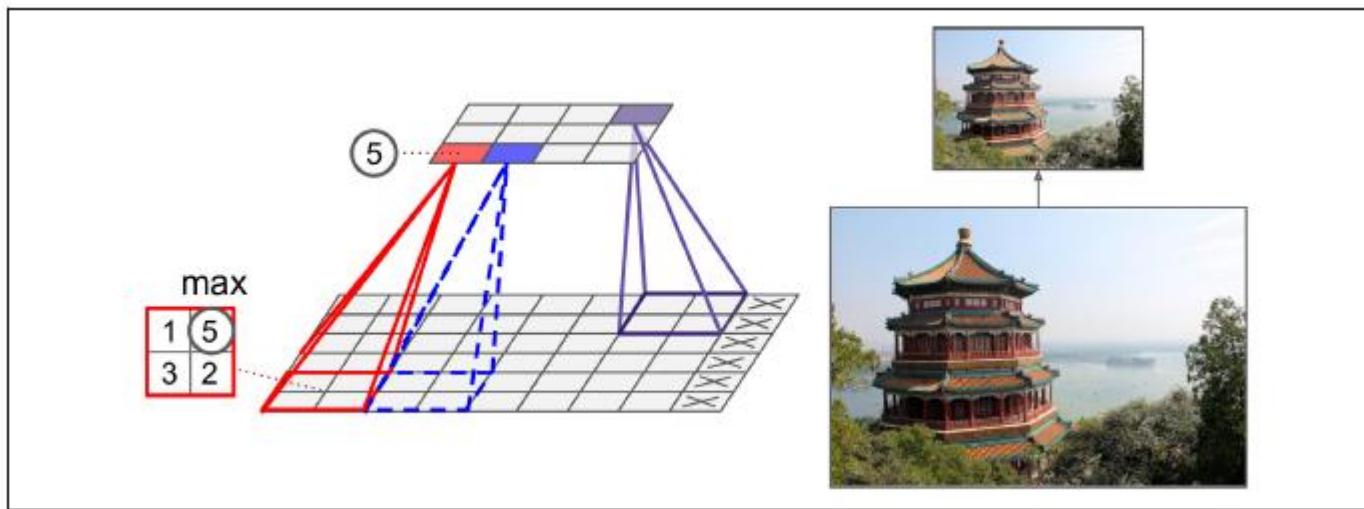
NLP 형태로 본 Convolution 층 연산



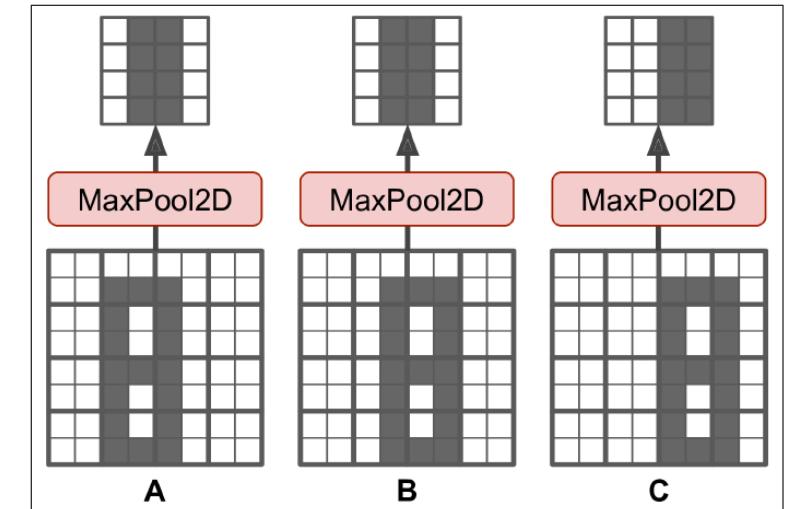
Pooling Layer

- **Max Pooling layer**

- The goal is to **subsample (i.e. shrink) the input image** (to reduce computational load, the memory usage, and the number of parameters) limiting the risk of overfitting
- Noise suppression
- Makes it invariant to translation movement (shifting or rotational) to some extent
- Helps capture essential structural features of the represented images



Max Pooling layer (2 x 2 pooling kernel, stride 2, no padding)

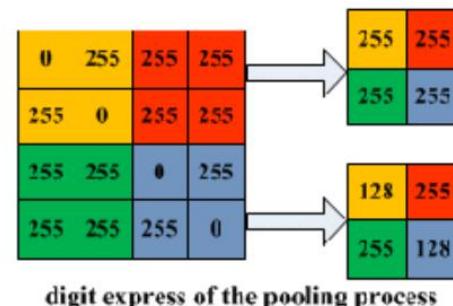
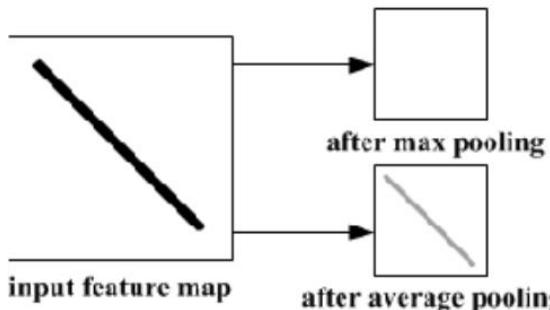


Invariance to small translations

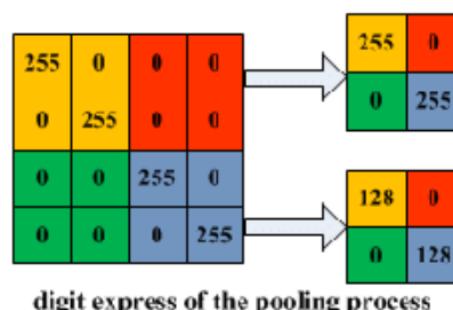
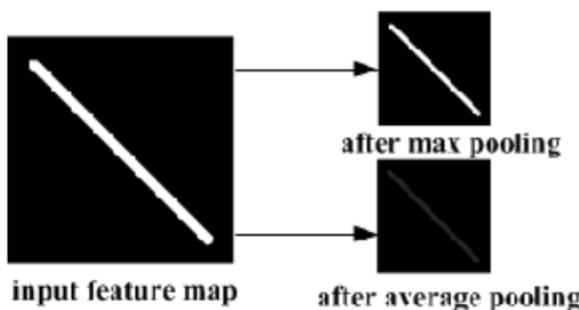
Pooling Layer

- **Maxpooling and AvgPooling**

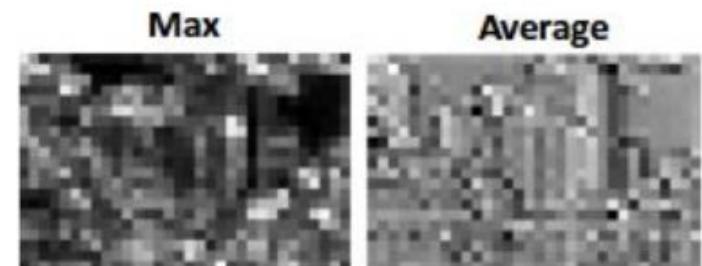
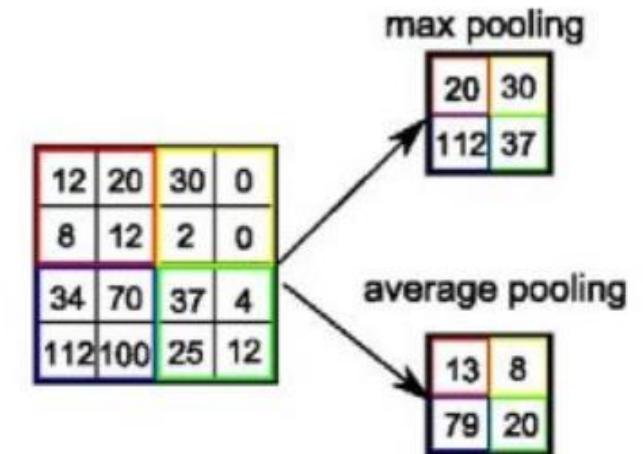
- data dependent, but in general Maxpooling is preferred



(a) Illustration of max pooling drawback



(b) Illustration of average pooling drawback



- Max Pooling extracts the most important features like edges.

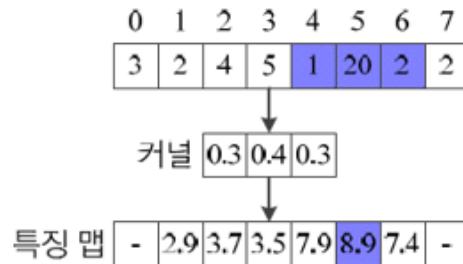
Implementing CNN

■ 컨볼루션 연산

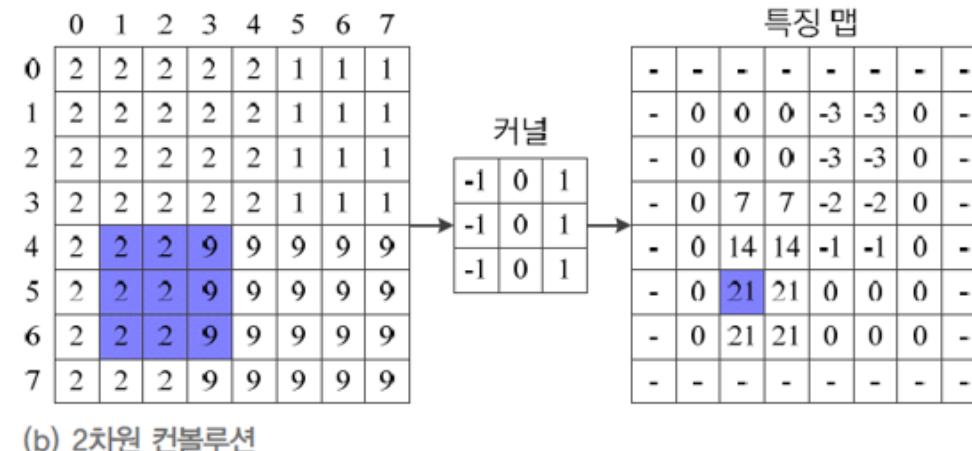
- 컨볼루션은 해당하는 요소끼리 곱하고 결과를 모두 더하는 선형 연산
- 식 (4.10)과 식 (4.11)에서 u 는 커널, z 는 입력, s 는 출력(특징 맵)

$$s(i) = z \circledast u = \sum_{x=-(h-1)/2}^{(h-1)/2} z(i+x)u(x) \quad (4.10) \quad \longleftarrow \text{1차원 입력}$$

$$s(j, i) = z \circledast u = \sum_{y=-(h-1)/2}^{(h-1)/2} \sum_{x=-(h-1)/2}^{(h-1)/2} z(j+y, i+x)u(y, x) \quad (4.11) \quad \longleftarrow \text{2차원 입력}$$



(a) 1차원 컨볼루션



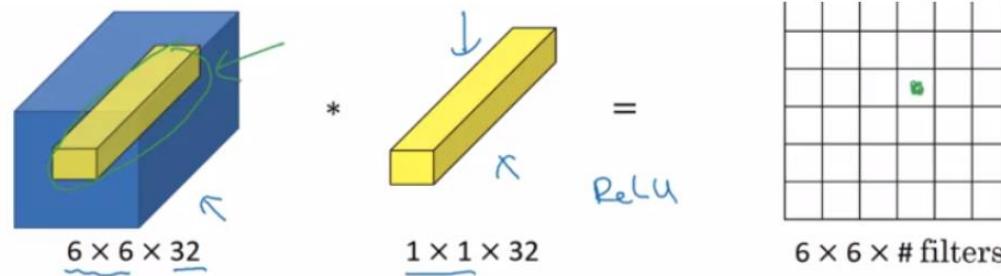
(b) 2차원 컨볼루션

ref: 오일석, 기계학습

Implementing CNN

- 1-D convolutions in Image Processing
(1×1 convolution: pointwise convolution)

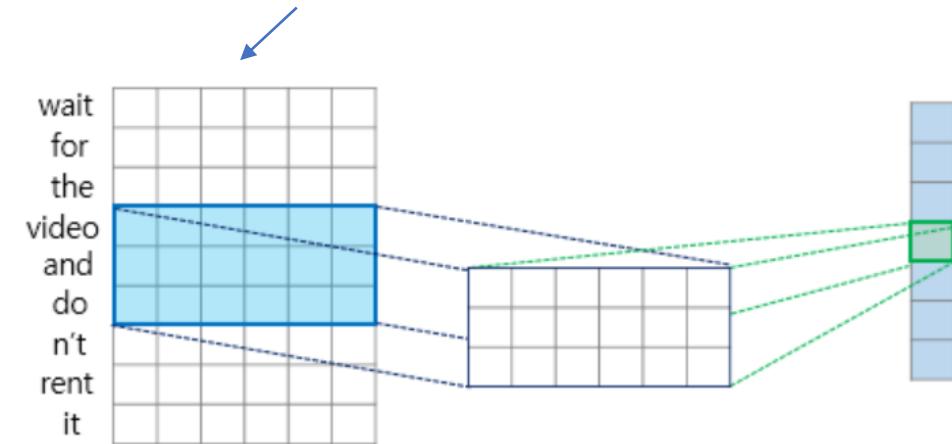
In other words,



- To reduce no of channels while keeping the same width and height. In other words, you can resize the feature map. (e.g. $(k,k,128) \rightarrow (k,k,32)$)
- Used in Inception module (GoogleNet, 2014)

- 1-D convolutions for Text Processing

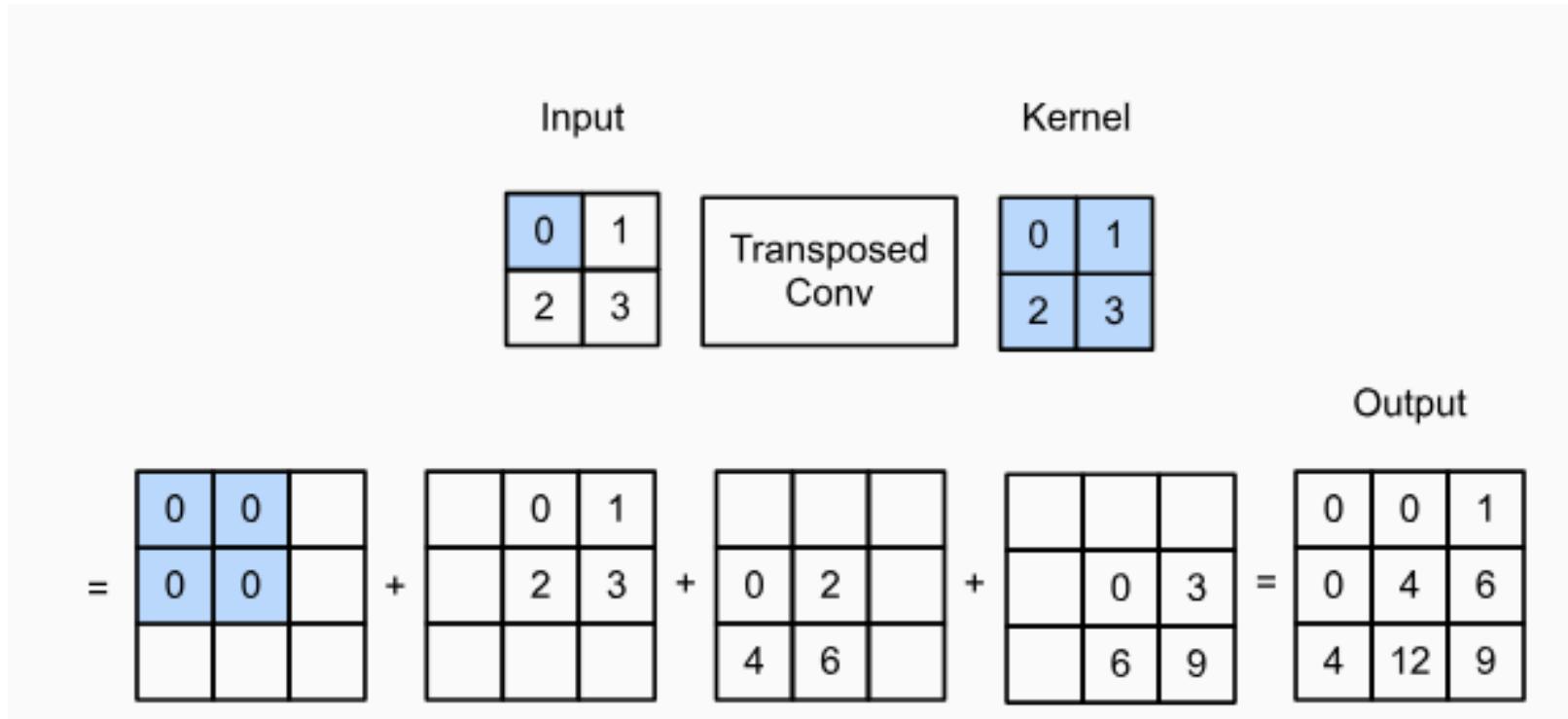
Embedding dimension of the words



- kernel size 3 -> referencing 3-gram

Up-sampling - Transposed Convolution

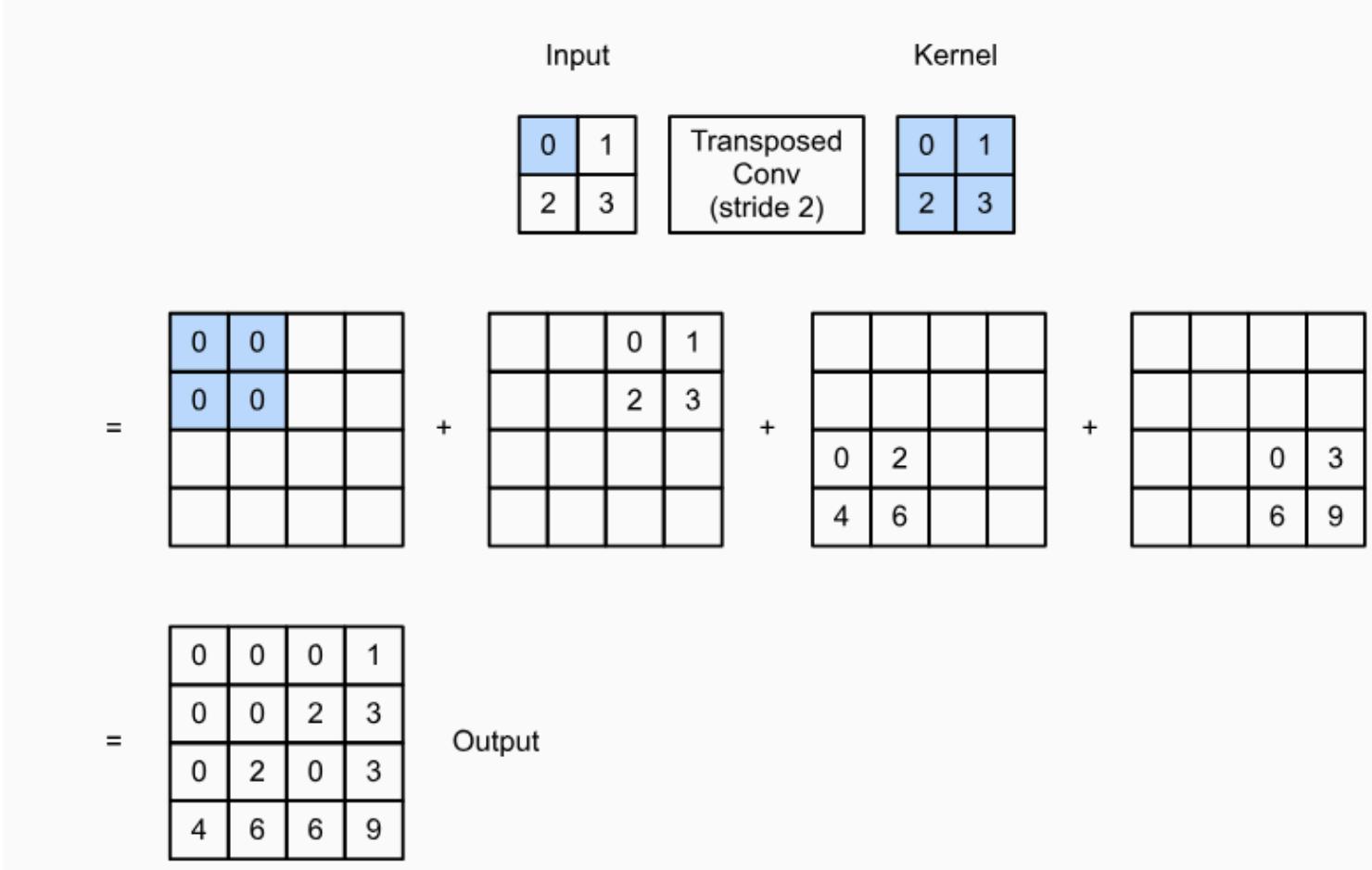
- 특징 맵(feature map)의 공간적 크기를 키워서 업샘플링(upsampling)을 수행 (일반 Convolution의 역방향 연산처럼 동작): 원본 입력을 완전히 복원하는 것은 불가능하며, 단지 공간적 차원만 확장.
- Ex: Using a 2x2 kernel with 2x2 input tensor



ref: Dive into Deep Learning, by Aston Zhang, Zachary Lipton

Up-sampling - Transposed Convolution

- Ex: Using a 2x2 kernel with 2x2 input tensor, stride 2

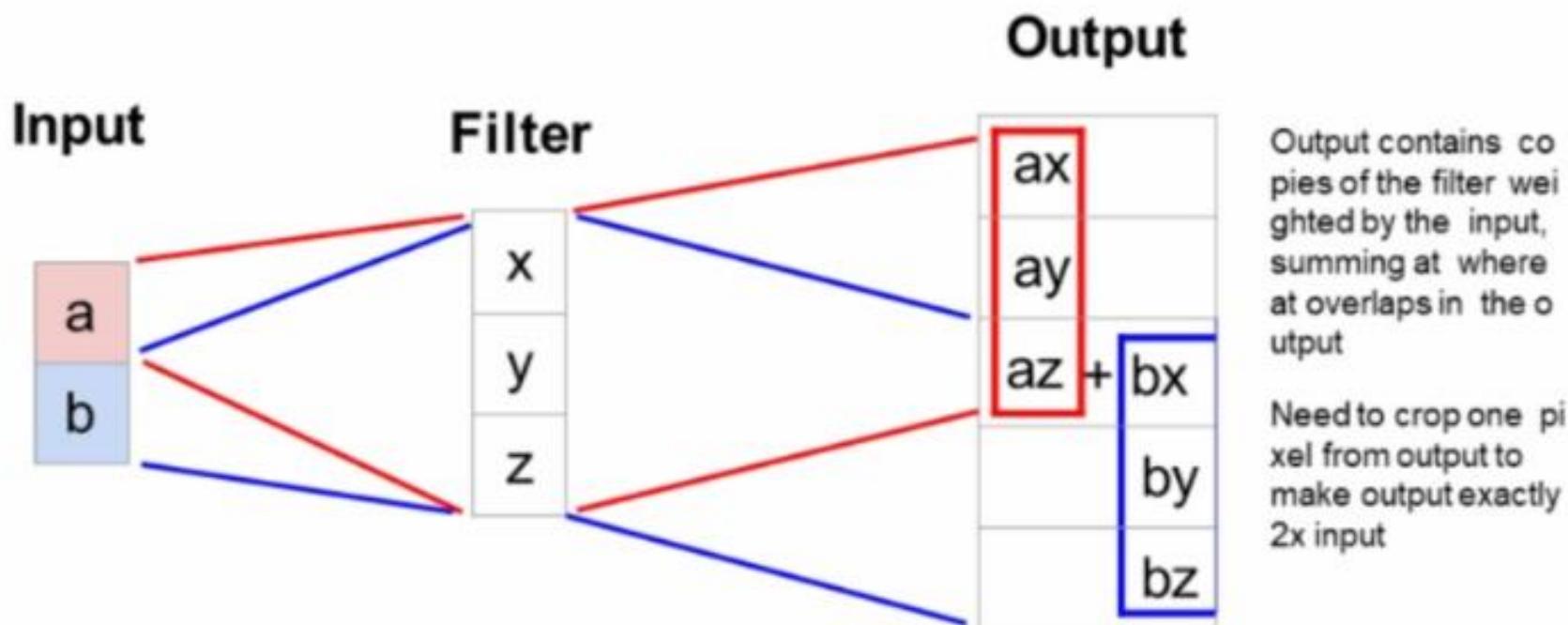


ref: Dive into Deep Learning, by Aston Zhang, Zachary Lipton

Transposed Convolution – 1D

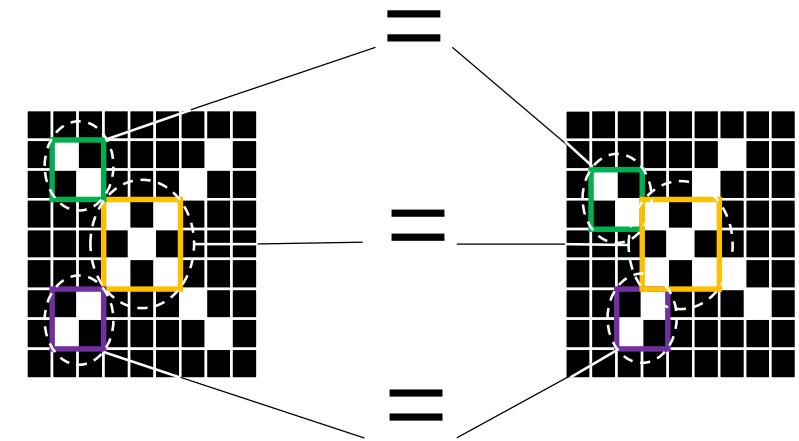
- 동작: 입력 픽셀 하나가 필터의 가중치로 곱해져 출력 맵의 영역을 채우고, 겹치는 부분은 합산된다.

Transpose Convolution: 1D Example



Detecting Patterns in CNN

- 공간상의 위치를 **픽셀 단위로 일대일 비교**하면
 - 이 두 그림에서 흰 점이 일치하는 부분은 거의 없다.
 - 따라서 이미지를 벡터로 환산(즉, 1차원 벡터로 변환)하여 비교하면 서로 다르다고 판단
- **3x3 픽셀 단위로 나누어 관찰**하면
 - 절대적인 위치는 다르지만 일치하는 패턴이 여러 곳에 나타나므로
 - 다른 위치에 있는 같은 패턴을 찾아낼 수 있다
- CNN의 필터는 바로 이러한 패턴의 활성화 성분 크기 즉, 활성 값을 찾아준다.
- 이러한 원리를 이용하여 CNN은 **어떤 패턴의 크기, 절대적 위치, 두께, 회전 각도 등이 다르더라도 이를 찾아서 다음 계층으로 전달**할 수 있다.

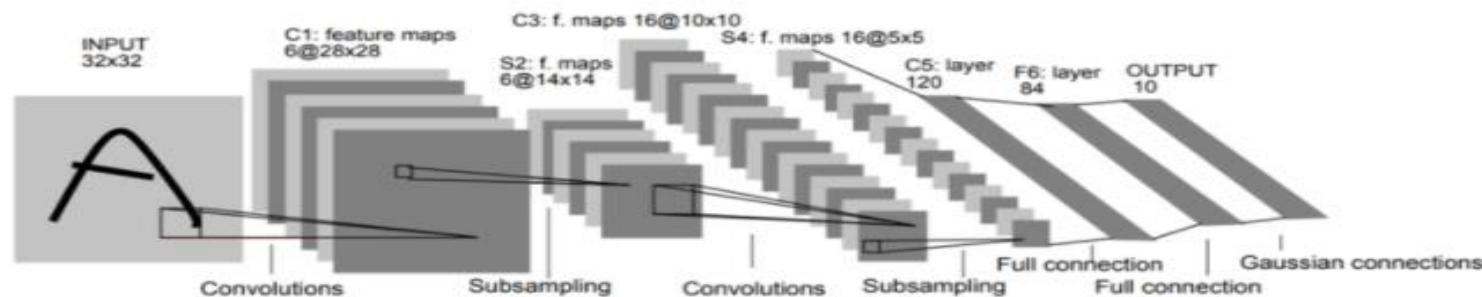
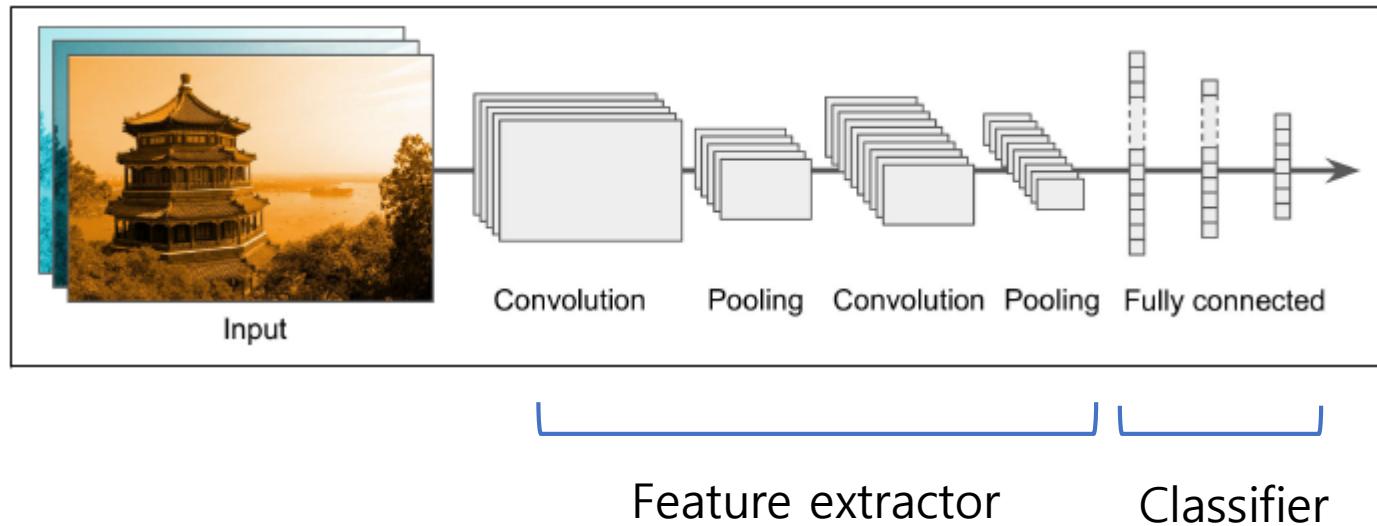


Detecting Patterns in CNN

- CNN에서는 특성맵을 한가지만 만드는 것이 아니라 수십~수백 가지를 만든다. (처음에는 3원색인 경우 3에서 출발)
- 왜냐하면 찾아내야 할 패턴이 가로 성분, 세로 성분, 대각 성분, X-형 성분, 색상 정보, 옛지 등 여러 가지가 있기 때문
- 어떤 계층의 CNN 필터의 종류 수가 32라면 이 계층에서 생산되는 특성맵은 32개가 된다.
- CNN 필터를 **커널**(kernel)이라고 부르는데 **커널 수가 많을수록 다양한 패턴**을 찾아낼 수 있다
- 그러나 **모델의 복잡도가 너무 커지면 과대적합**의 원인
 - 내부 parameter 수가 많아 학습 데이터를 너무 정교하게 모델링하기 때문

CNN Architecture

- Typical CNN architecture



(ex) LeNet-5

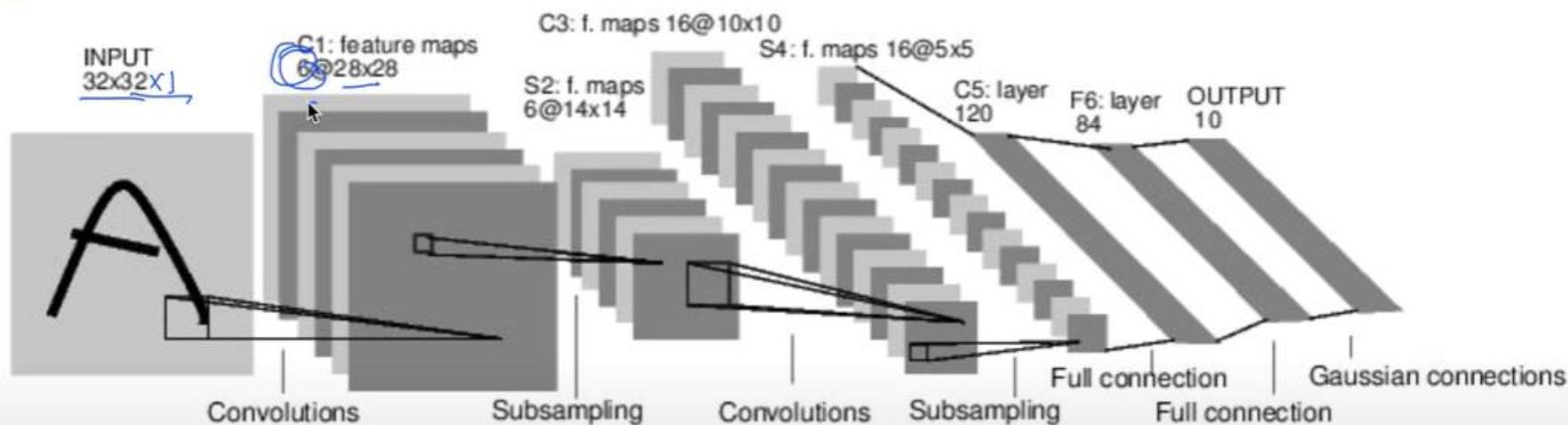
CNN Architecture

- CNN은 일반적으로 커널 필터링과 활성화 함수 그리고 최대 풀링을 묶어서 하나의 계층을 형성한다.
 - 참고) MLP에서는 전결합망과 활성화함수를 묶어서 한 계층을 형성
- CNN에서도 다양한 구조의 활성화 함수를 사용하고 필요하면 전결합망을 사용한다.
- 분류를 수행하는 경우 최종 계층에서는 전결합망을 만들고 그 결과에 소프트맥스 함수(또는 시그모이드함수)를 적용

CNN Example

Case Study: LeNet-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2×2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

CNN Example

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

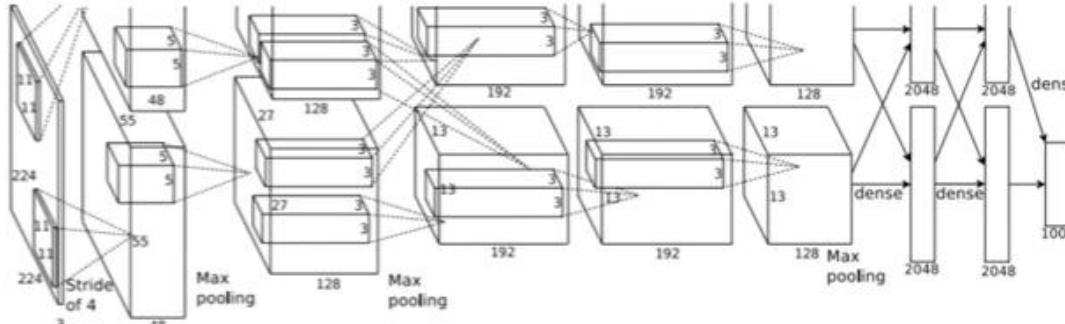
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



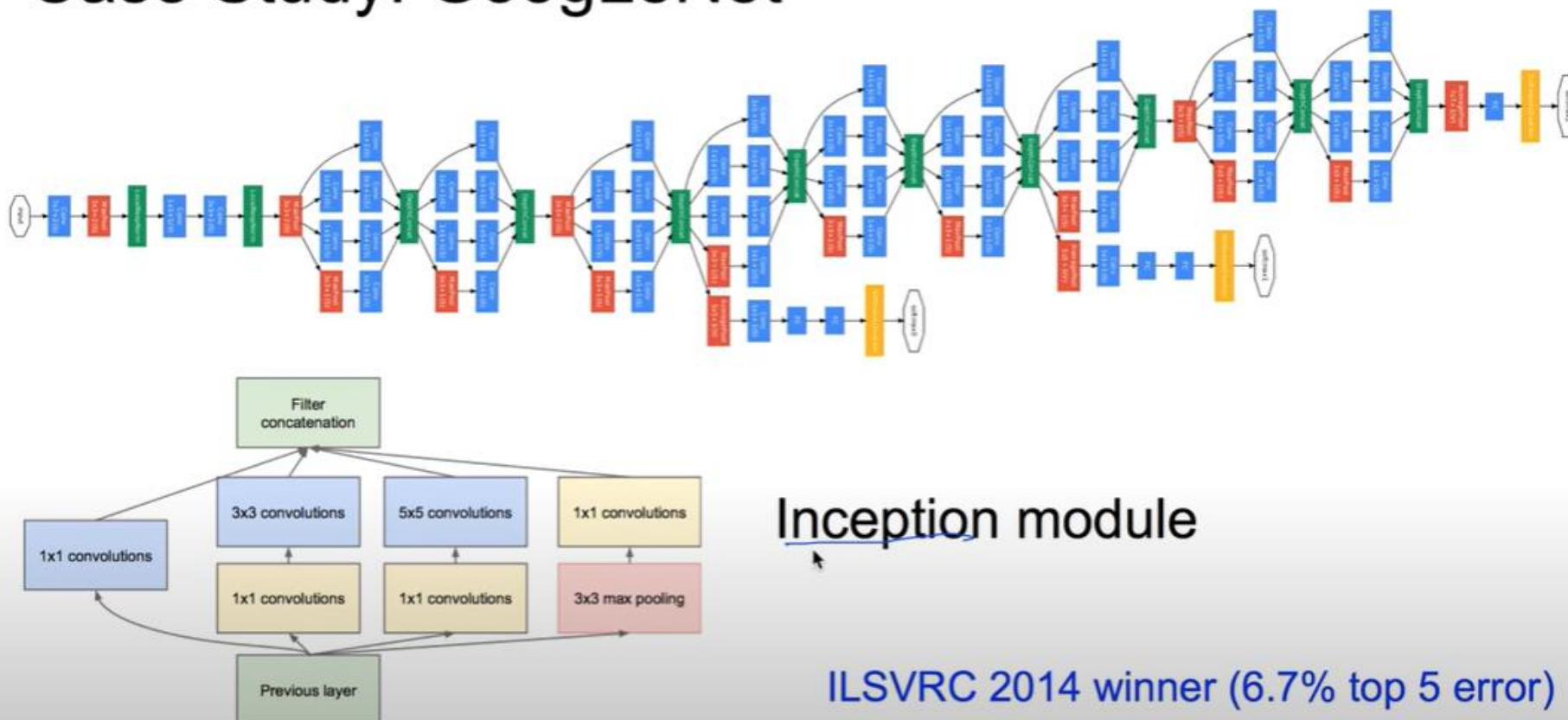
Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

CNN Example

Case Study: GoogLeNet

[Szegedy et al., 2014]

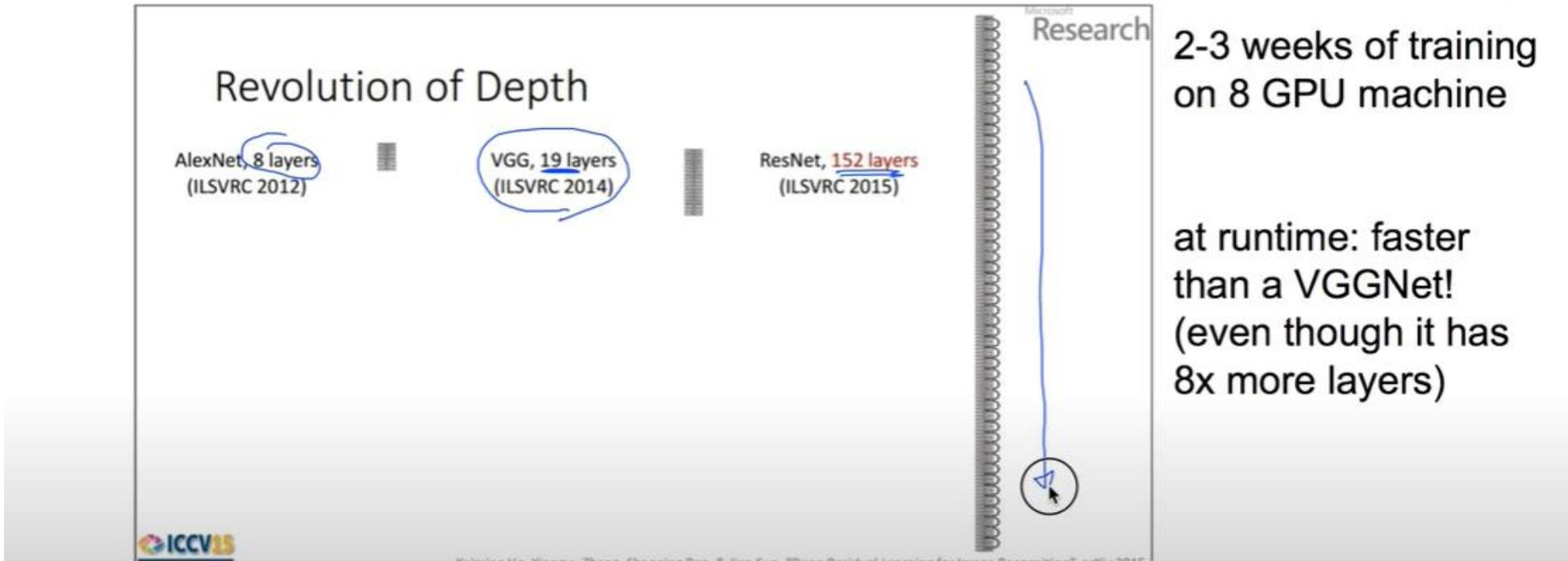


CNN Example

Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



(slide from Kaiming He's recent presentation)

CNN 성능 개선

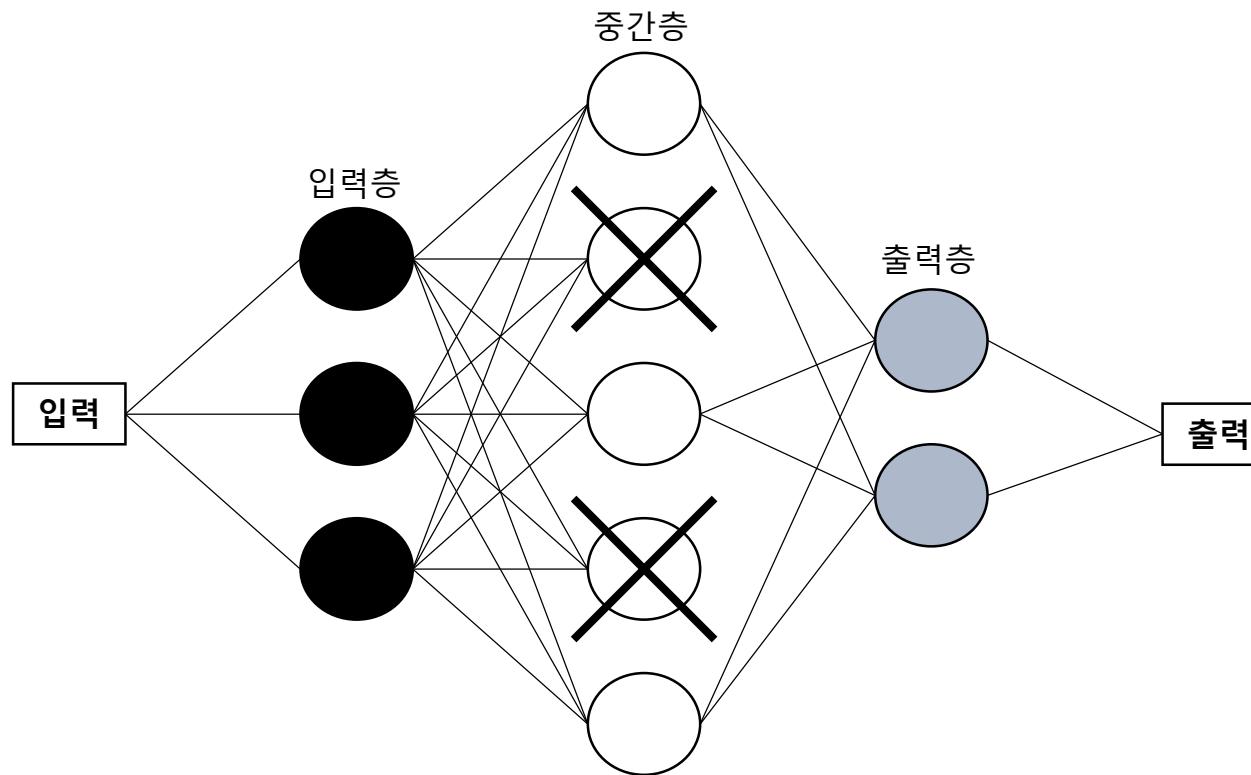
- 신경망은 파라미터 갯수가 많아서 과대적합 가능성이 항상 높다. 훈련 데이터 양이 많지 않을 때에는 특히 주의해야 한다.
 - 과대적합이 발생하면 신경망의 구조를 단순하게 만들어야 한다.
- 과대적합을 줄이는 작업을 일반화(Generalize)를 위한 규제화(Regularize)라고 부르는데,
 - 대표적인 규제화로 L2규제를 적용 - 네트워크를 구성하는 파라미터 값이 가능한 균등하게 분포하도록 제한
 - 케라스에서는 `regularizers.l2()` 함수를 사용

Drop Out

- 신경망 학습 과정에서 일부 뉴런을 무작위로 제거하여 오버피팅(overfitting)을 방지하고 일반화 성능을 높이는 방법.
- **핵심 아이디어:**
 - **무작위 뉴런 제거**: 학습 중에 일정 비율의 뉴런(및 연결)을 랜덤하게 꺼버린다.
 - **양상블 효과**: 매 학습 단계마다 다른 작은 네트워크를 학습하는 것과 같아, 여러 모델을 양상 블한 효과를 낸다.
 - **특정 뉴런 의존 방지**: 특정 뉴런이나 패턴에 과도하게 의존하지 않도록 하여, 더 견고한 표현을 학습하게 한다.
- **Dropout의 역할**
 - **오버피팅 방지**: 복잡한 신경망은 훈련 데이터에 과도하게 맞춰져 일반화 성능이 떨어질 수 있다. Dropout은 이를 완화하는 효과를 준다.
 - **일반화 성능 향상**: 새로운 데이터에 대해 더 잘 작동하도록 모델을 정규화(regularization)하는 효과가 있다.
 - **학습 안정화**: 뉴런 간의 공모(co-adaptation)를 줄여, 다양한 특징을 고르게 학습하게 만든다.

Drop Out

- 드롭아웃은 학습을 하는 동안에만 적용
- 학습이 종료된 후 예측(테스트)을 하는 단계에서는 모든 유닛을 사용하여 예측 (학습 때 제거된 비율만큼 출력 값을 스케일링하여 균형을 맞춤.)



How to find a Good CNN architecture?

- 최적의 신경망을 구성하는 계층의 수, 유닛의 수, 배치 크기, 학습률의 설정 등이 어려운 과제
- 기본 전략 : 처음에는 구조를 간단히 출발
- 일단 동작을 확인하고 성능을 개선한다.
 - 계층이 2~3만으로도 동작하는지를 확인
 - 만일 2~3개의 계층으로 모델이 동작하지 않으면 계층 수를 늘려도 동작하지 않는다고 알려져 있다.
- 입력 데이터를 간단히 만들어 보는 것도 필요
 - 예를 들어 10가지 동물 이미지를 구분하는 모델이 필요하다고 하여도 우선 몇 가지 대표적인 동물들을 구분하는 모델을 먼저 만들어보는 것

Batch Size

- 배치 크기 : 신경망이 한번에 학습하는 입력 데이터 수
- 배치 크기가 클수록 학습이 정교하고, 기울기를 정확히 구할 수 있으나 계산량이 많아진다.
 - 필요한 메모리 사용량이 많아 메모리 오류가 날 가능성이 높다(특히 GPU를 사용할 때). 처음에는 배치 크기를 작게 16~32정도로 적게 잡고 시작
- 배치 크기가 작을 때에는 기울기가 상대적으로 정확하게 계산되지 못하므로 학습률도 작게 잡아야 한다.
 - 일단 작은 값의 학습률로 동작하는 것을 확인하고 학습률을 조금씩 크게 한다.

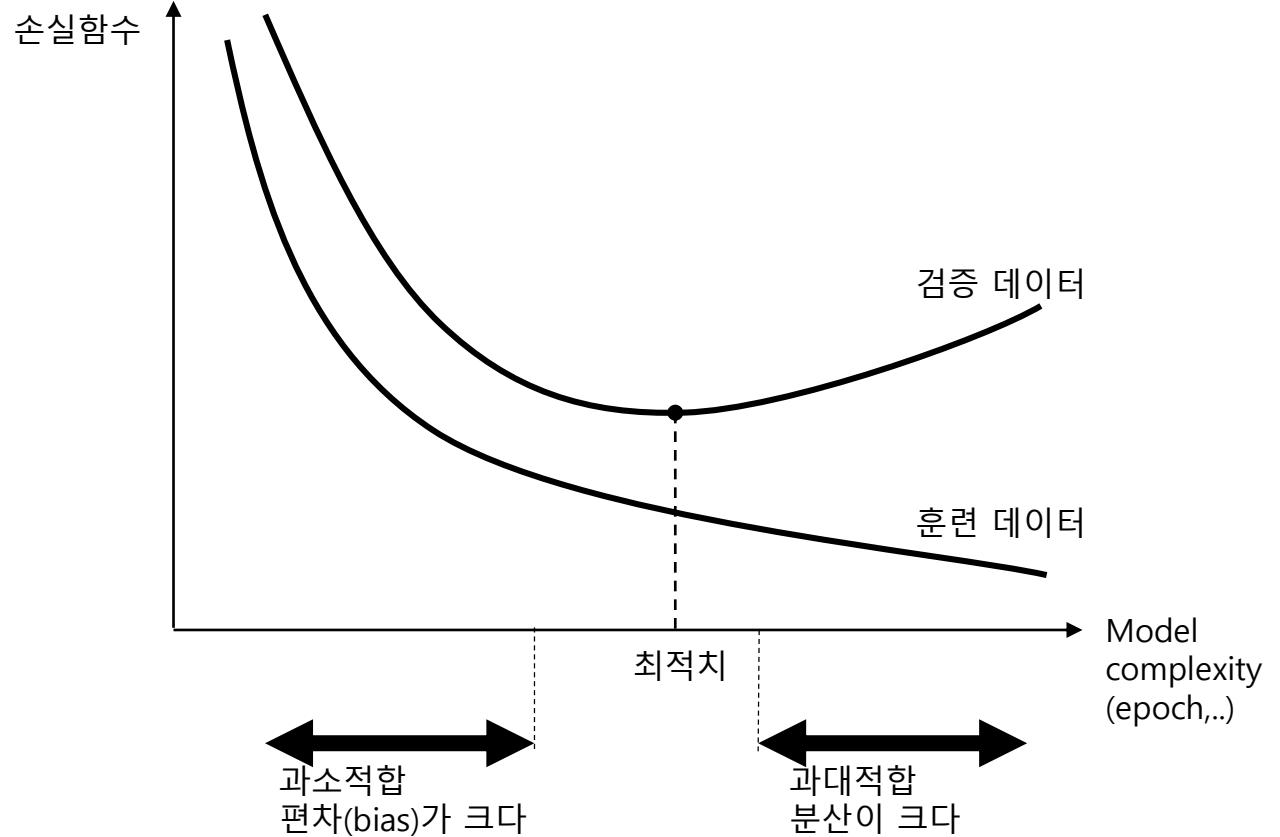
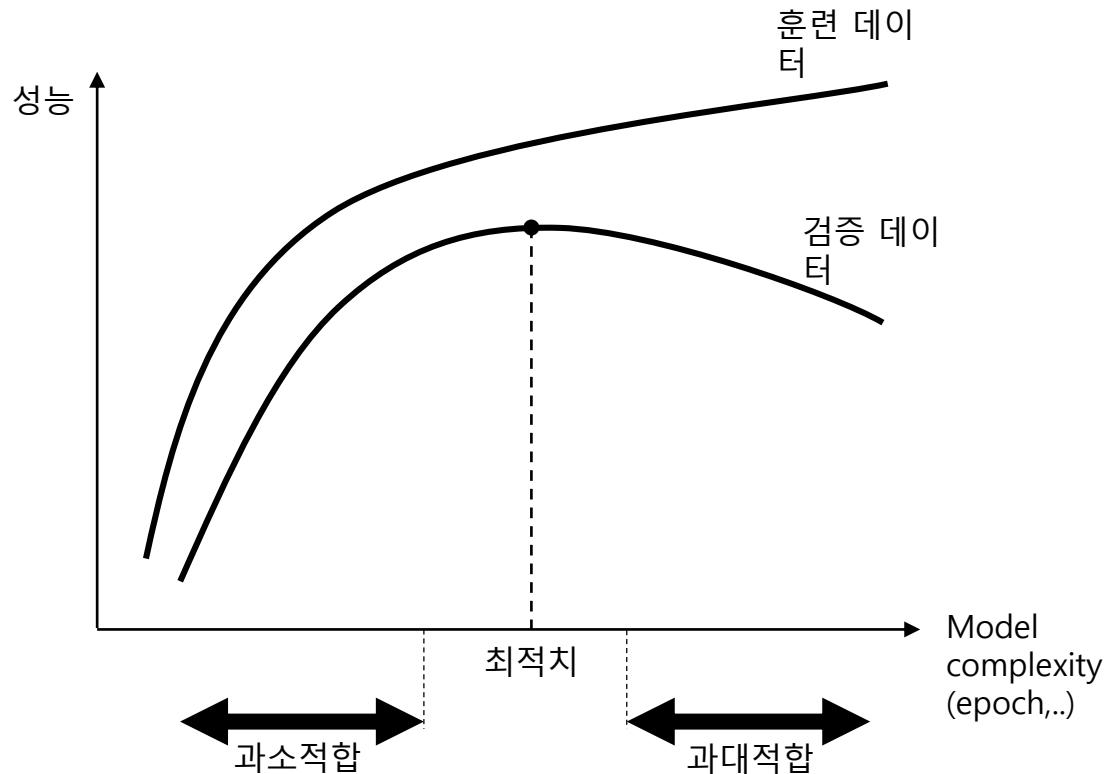
Batch Normalization (배치 정규화)

- 딥러닝 학습에서 해결해야할 문제 중 하나
 - Vanishing Gradient Problem or Gradient Exploding Problem
 - 레이어가 많아질수록 기울기가 연속적으로 곱해지는데, 작은 값이 누적되면 기울기 소실 발생. 반대로 큰 값이 누적되면 Gradient Exploding(기울기 폭주) 문제도 생길 수 있음
- 이러한 문제를 해결하기 위해 배치 정규화 제시
 - 계층별로, 주어진 배치 데이터를 대상으로 정규화를 다시 수행하여 **데이터의 분포**가 너무 작거나 너무 커지지 않게 하는 방식 (주로 Dense layer나 Convolution layer 뒤에 삽입)
 - **학습 시 미니배치 단위로 정규화 : (평균 0, 분산 1)**
- BN and LN
 - Batch Normalization (BN): 같은 레이어에 들어오는 **미니배치 전체**(특성별로 독립적으로 정규화)
 - Layer Normalization (LN): 입력 특성 전체를 함께 정규화

Overfitting (과대적합)

- 신경망은 파라미터가 많아서 과대적합되기 쉽다. 즉, 상세한 모델링이 가능하여, 훈련 데이터 수가 적으면 이 훈련 데이터의 속성을 모두 기억할 수 있어 과대적합되기 쉽다.
- 훈련데이터에 대해서는 계속 성능이 좋아지지만 검증 데이터에 대해서는 오히려 성능이 나빠지는 현상을 과대적합이라 한다.
- 성능의 개선되는지를 측정하는 방법
 - 정확도 등 최종 성능 지표를 관찰
 - 손실 함수가 줄어드는지를 관찰

Overfitting



How to Avoid or Reduce Overfitting?

- 과대적합의 원인:

- 모델이 지나치게 복잡할 때 (파라미터 수가 많음)
- 학습 데이터가 적거나 다양성이 부족할 때
- 학습을 너무 오래 시켜서 데이터의 노이즈까지 학습할 때

- 해결 방법:

- Regularization (L1/L2, Dropout 등)
- 데이터 증강(Data Augmentation)
- 모델 단순화
- Early Stopping

Data Augmentation (데이터 확장)

- 과대적합이 일어나는 이유 중 하나
 - 훈련데이터가 부족하기 때문
- 훈련 데이터가 충분히 많다면 과대적합을 줄일 수 있다.
- 데이터 확장이라 **훈련 데이터를** 다양하게 변형하여 변형된 새로운 훈련 데이터처럼 사용함으로써 마치 훈련 데이터 수가 늘어난 효과를 얻는 것이다.
- 데이터 확장을 사용하면 여러 이포크를 수행해도 똑같은 데이터를 가지고 학습하지 않게 된다.
- Affine Transform
 - Rotation, shifting, rescaling, flipping, shearing, stretching
- Synthetic Data Generation using GAN

Data Augmentation (Example)

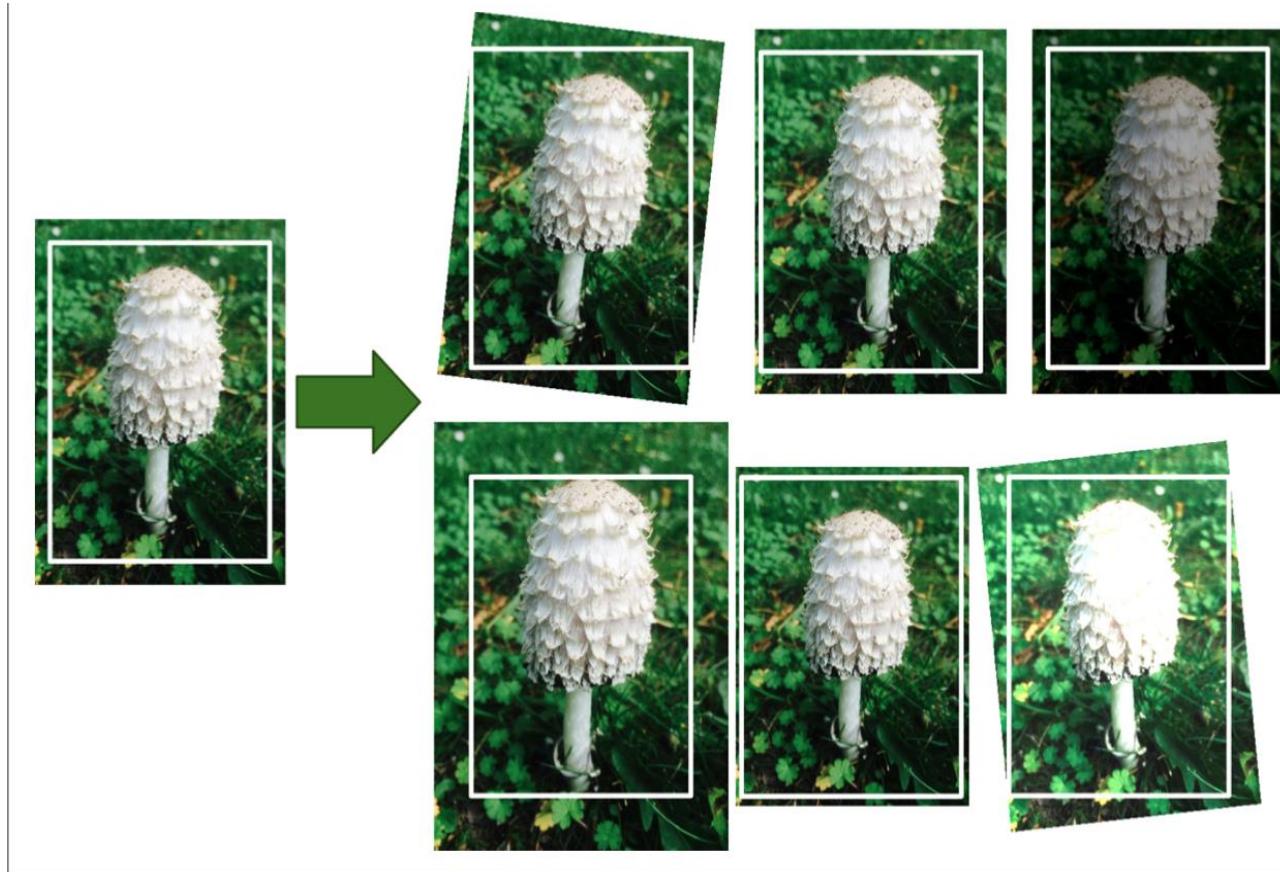
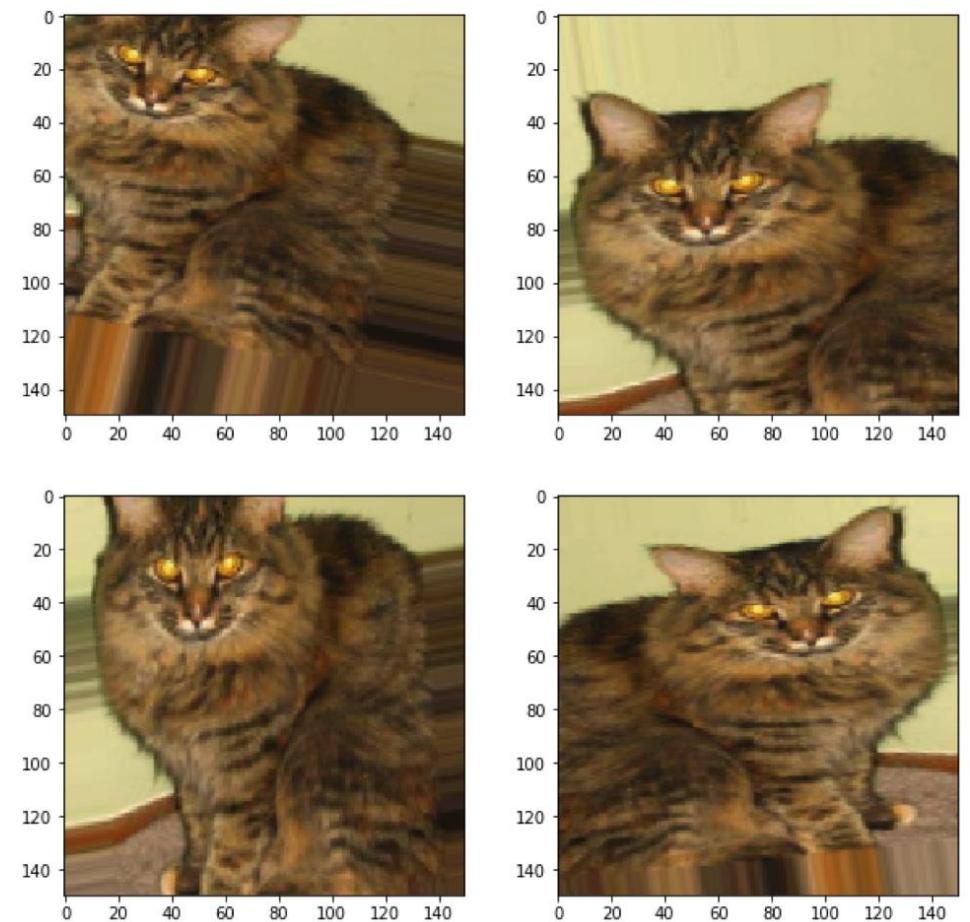
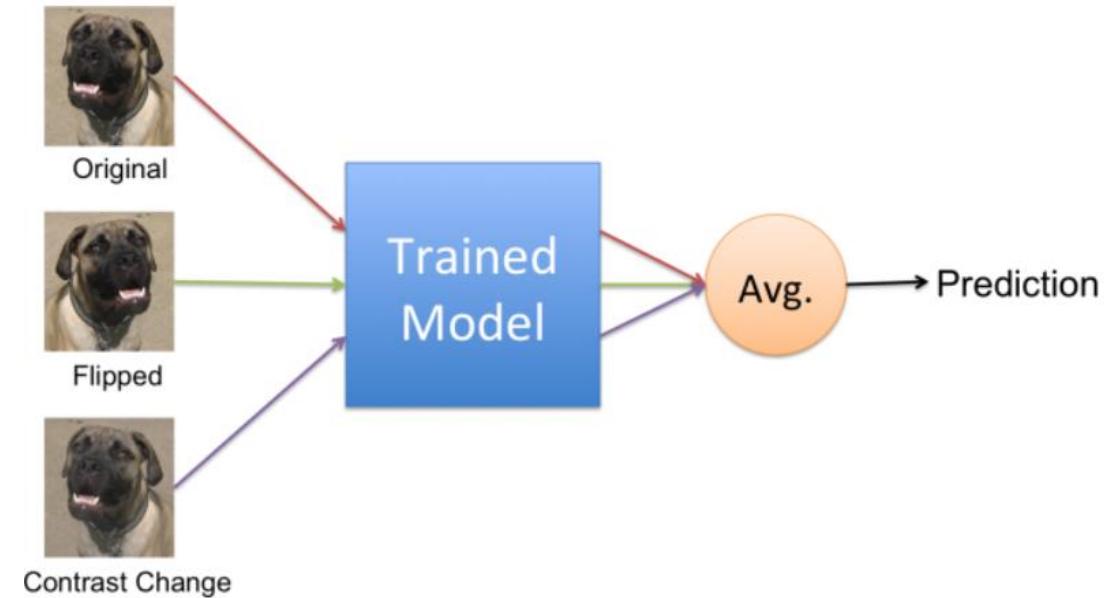


Figure 11-10. Generating new training instances from existing ones



Test Time Augmentation (TTA)

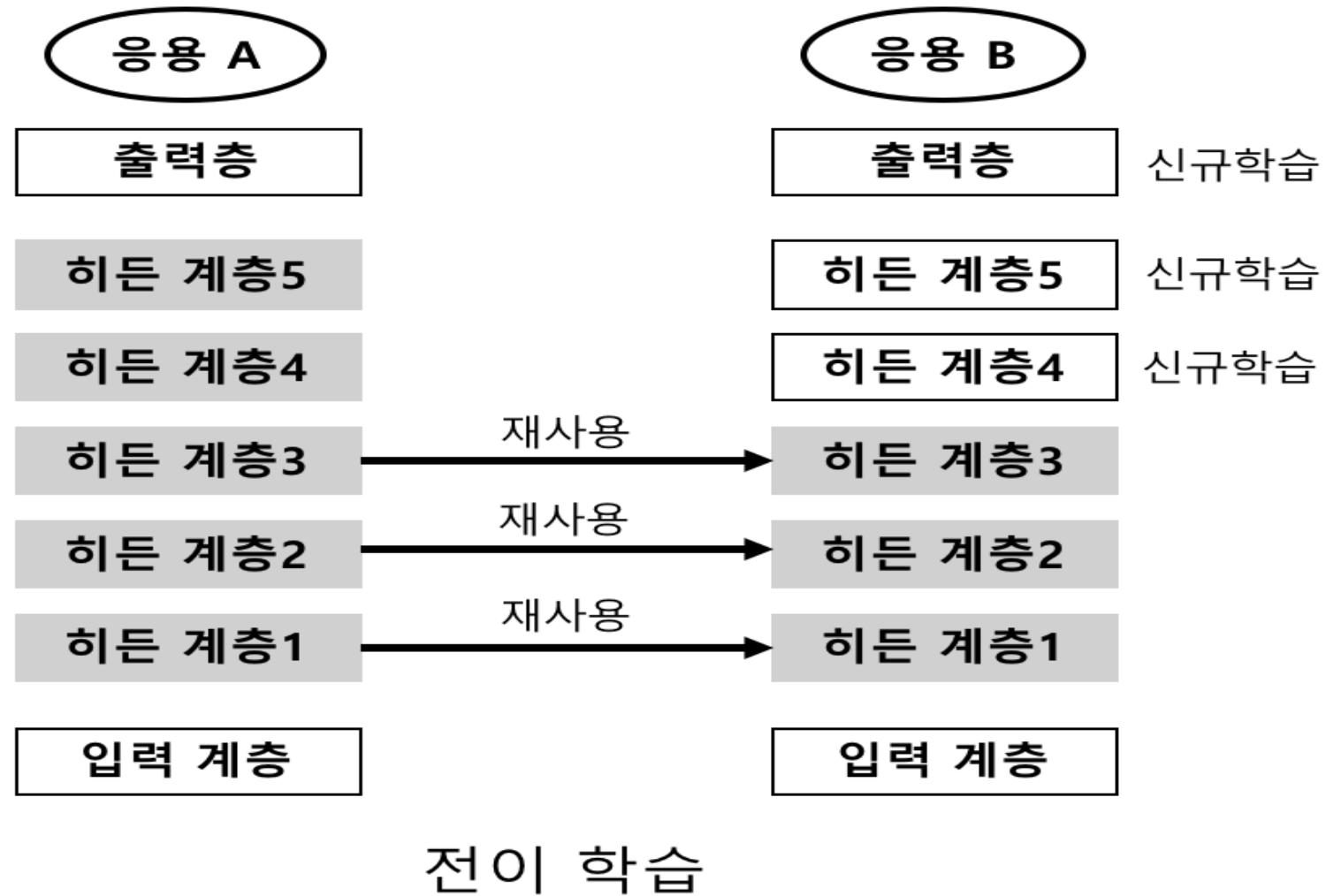
- Augmentation in Training:
 - 훈련 데이터가 부족할 때 데이터 셋을 회전/반전/뒤집기/늘이기/줄이기/노이즈 등 다양한 방법을 사용
- TTA (Test Time Augmentation):
 - 학습할 때 Augmentation하는게 아닌, 테스트 셋으로 모델을 테스트하거나, 실제 운영 (deploy)할 때 Augmentation을 수행
 - 각 확장된 데이터에 대해 Model 을 적용(Test) 한 후 평균하여 최종 결과 도출 (다양한 관점의 테스트 이미지에 대해 테스트하므로 성능이 향상, **but not always !**)



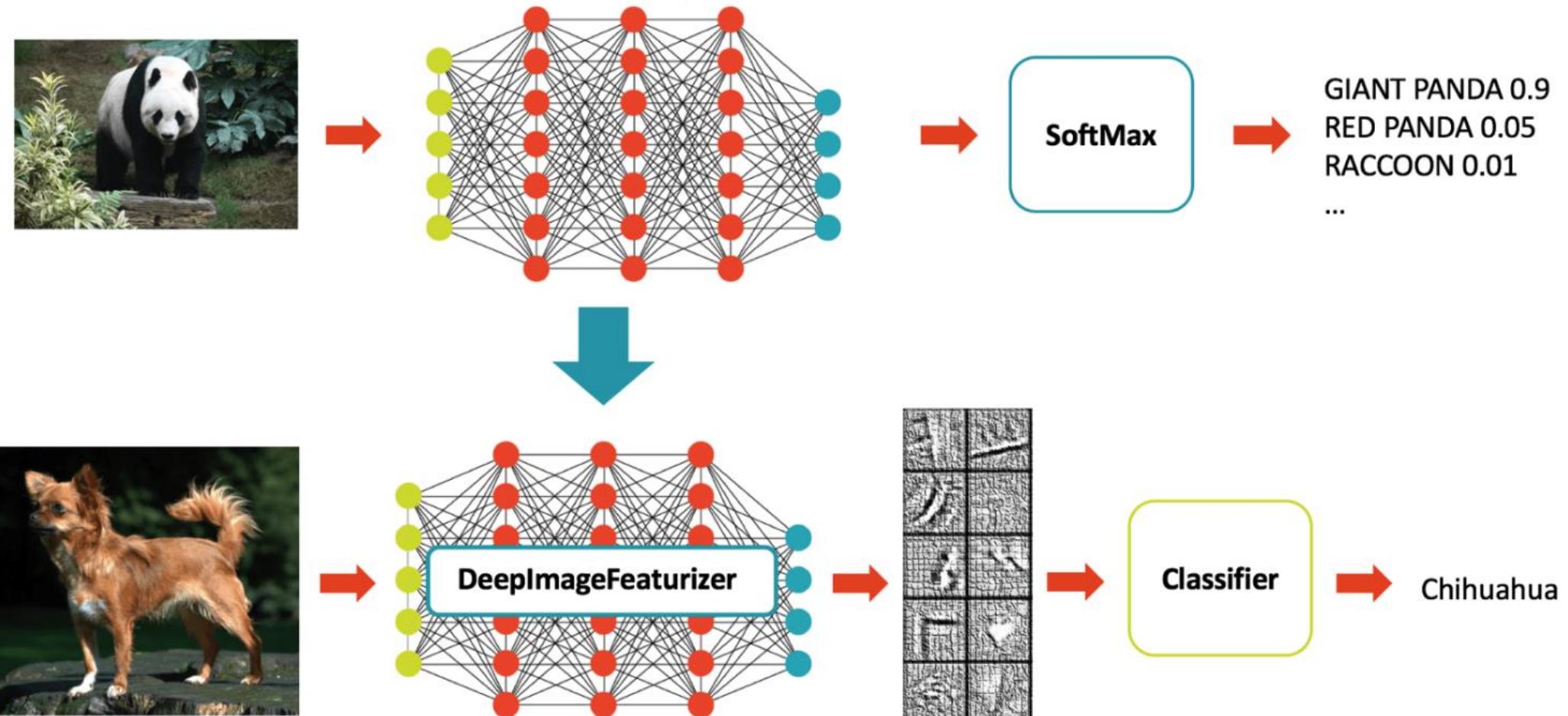
Transfer Learning (전이학습)

- 전이학습 이란 다른 데이터 셋을 사용하여 이미 학습한 모델을 유사한 다른 데이터를 인식하는데 사용하는 기법이다.
- 이 방법은 특히 새로 훈련시킬 데이터가 충분히 확보되지 못한 경우에 학습 효율을 높여준다.
- Example:
 - 이미지넷(imagenet)에는 동물이나 일상생활의 물건들을 주로 포함하여 1000종의 이미지를 갖고 있으며 140만장의 사진이 있다.
 - 이미지넷에는 고양이 강아지를 포함한 많은 동물 이미지도 들어 있으며, 이를 강아지 고양이 분류 문제에 사용할 수 있다.
 - 예제코드에서는 2014년에 소개된 VGG16 모델을 사용한다.
- 사전학습모델을 이용하는 방법은 특성 추출(feature extraction) 방식과 미세조정(fine-tuning) 방식이 있다.

Transfer Learning (전이학습)



Transfer Learning (전이학습)



Transfer Learning – 특성 추출 방식

- 특성 추출이란 이전의 네트워크로부터 배운 표현 방식을 사용하여 새로운 데이터 샘플에서 추가로 흥미로운 특성들을 추출한다.
- 이렇게 얻은 특성들을 사용하여 새로운 분류기를 작동시키고 학습을 수행한다.
- 이미지 분류기는 특성추출 부분(컨볼류션 네트워트와 풀링 계층의 조합)과 분류기 부분(전결합망)으로 크게 두 부분으로 구성된다.
- 컨볼류션과 풀링으로 구성된 부분을 **컨볼류션 베이스**라고 한다.
- 특성 추출 방식은 컨볼류션 베이스를 그대로 사용하고, 새로운 데이터를 여기에 적용하여 훈련시키는 방식이다.

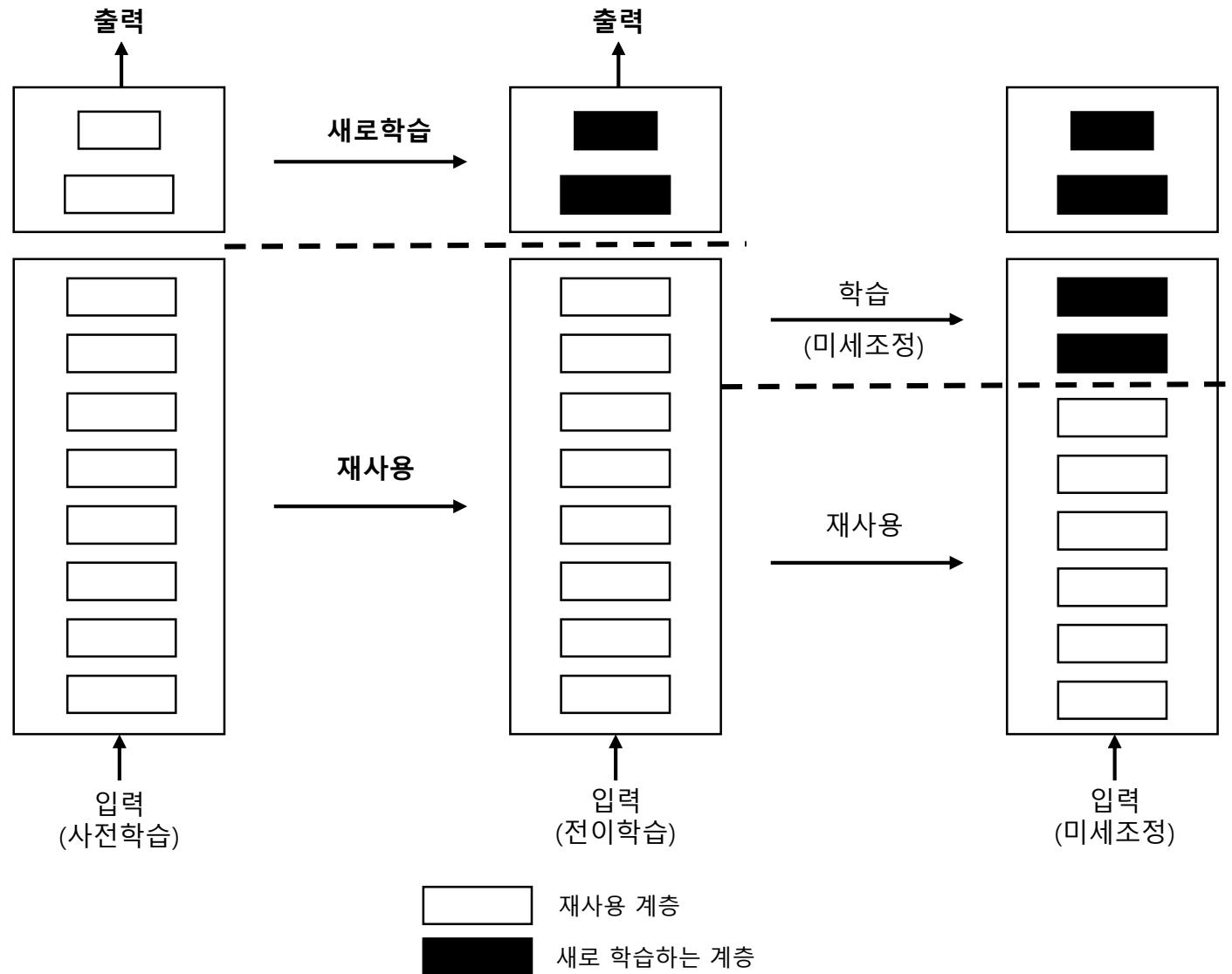
Transfer Learning – 특성 추출 방식

- 컨볼류션 베이스 부분만 재사용하는 이유는 이 부분은 상당히 일반적인 학습정보를 포함하고 있기 때문이다.
- 컨볼류션 계층에서도 재사용할 수 있는 정보의 수준은 몇 번째 계층인지에 따라 다르다. 모델의 앞단의 계층일수록 에지, 색상, 텍스처 등 일반적인 정보를 담는다.
- 반면에 뒷 부분의 깊은 계층일수록 추상적인 정보를 담는다 (예를 들어 고양이 귀, 강아지 귀 등)
- 새롭게 분류할 클래스의 종류가 사전 학습에 사용된 데이터와 특성이 매우 다르면, 컨볼류션 베이스 전체를 재사용해서는 안되고 앞 단의 일부 계층만을 재사용해야 한다.

Transfer Learning – 미세 조정 방식

- 모델 베이스 중 상위 몇개의 계층은 전결합층 분류기와 함께 새로 학습시키는 방식이다.
- 최종 분류기의 계수가 랜덤하게 초기화 되어 있으므로 이를 먼저 학습시키며 이때 VGG16 모델의 컨볼류션 베이스를 초기에는 고정해야 한다.
- 먼저 분류기를 계수를 학습시킨 다음에 (즉, 이 동안은 미세조정을 하지 않도록 상위계층의 계수를 고정시켜 두고), 그 이후에 미세조정을 해야 한다.
- 처음부터 베이스 상위계층의 계수를 같이 훈련시키면 분류기에서 발생하는 큰 에러 값으로 인해, 사전 학습된 정보가 많이 손실된다.

Transfer Learning – 미세 조정 방식



Transfer Learning – 미세 조정 절차

1. 사전 학습된 기본 네트워크 상단에 새로운 네트워크를 추가한다.
2. 기본 네트워크를 고정시킨다.
3. 새로 추가한 부분을 학습시킨다.
4. 기본 계층 중에 학습시킬 상위 부분의 고정을 푼다
5. 고정을 푼 계층과 새로 추가한 계층을 함께 훈련시킨다.
6. 미세 조정을 천천히 수행하기 위해서 느린 학습 속도를 선택한다. 갑자기 큰 변화를 주면 사전 학습된 내용이 훼손되기 때문이다.

When does Transfer Learning make sense?

- Task A and B have the same input X.
- You have a lot more data for Task A than for Task B.
- Low level features from A could be helpful for learning B.

Callback (콜백)

- 케라스, 콜백 함수
 - 모델 학습을 하는 동안 중간 결과를 저장하거나 조기 종료를 시키거나 할 수 있는 기능
- 콜백은 여러 가지를 동시에 지정할 수 있으며 이를 fit 함수의 callbacks 인자로 넘겨줄 수 있다.
- [History\(\)](#)
 - 훈련중에 발생하는 여러 이벤트를 History 객체에 저장
 - keras.callbacks.History()

Callback (콜백)

- 모델 저장: 매 이포크마다 모델을 저장
 - 저장하는 주기(period)와 베스트 모델만 저장 등을 지정
 - keras.callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=0, save_best_only=False, save_weights_only=False, mode='auto', period=1)
- 조기 종료 (Early Stopping)
 - 손실값, 성능 등 관찰 중인 지표가 어떤 조건을 만족하면 이포크 실행을 종료
 - 조기 종료 조건들을 인자로 지정
 - keras.callbacks.EarlyStopping(
monitor='val_loss',
min_delta=0, patience=0, verbose=0, mode='auto',
baseline=None, restore_best_weights=False)
- 참고) <https://keras.io/callbacks/#modelcheckpoint>

Datasets for Image Processing (CNN)

- **ImageNet**

- largest image dataset for computer vision, used in [ILSVRC](#)(ImageNet Large Scale Visual Recognition Challenge) for image classification and object detection (150 GB)
- 469*387 resolution in average (usually cropped to 256*256 or 224*224 before usage)
- More than 14 million images with more than 21,000 groups(classes)
- More than 1 million images have bounding box annotations

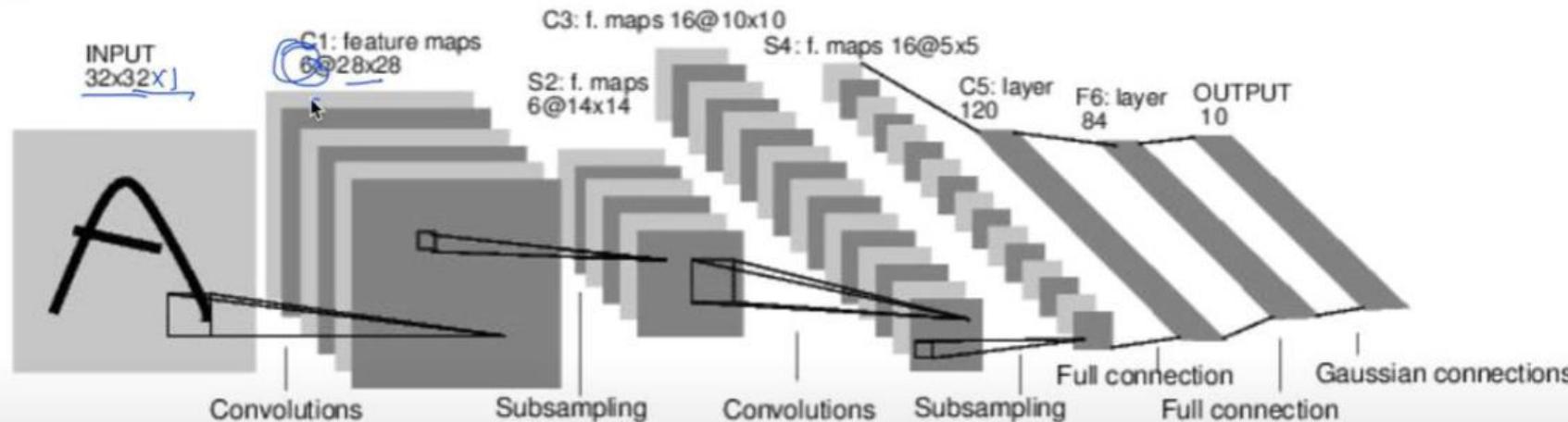
- **ILSVRC**

- To evaluate algorithms for object detection and image classification (and localization) at large scale
- To measure the progress of computer vision for large scale image indexing for retrieval and annotation
- Uses smaller portion of the ImageNet (1,000 categories with 1.3 million train images, 50,000 validation images, and 1,00,000 test images)
- Available in Kaggle
- 2010 ~ 2017



Lenet-5

[LeCun et al., 1998]



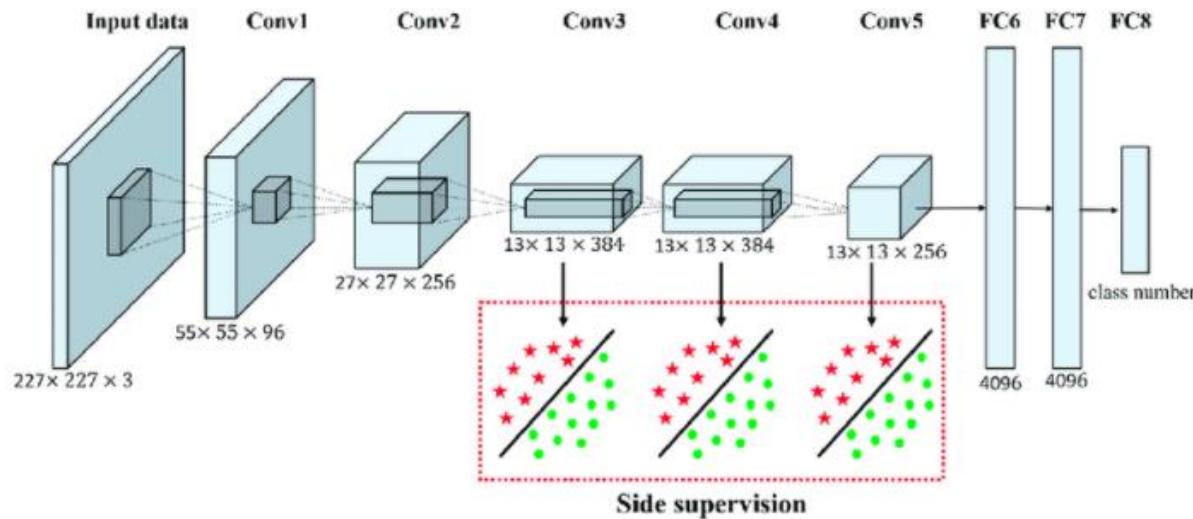
Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

- First architecture for CNN (excellent on MNIST dataset)
- Small and easy to understand
- Uses **tanh** activation function

ILSVRC Winners

- **ALexNet – 2012 Winner (top-5 error rate 15.3%)**
 - Use CNN (prior to 2012, the classification model error was 25%)
 - Regarded as a Pioneer of CNN and starting point of the Deep learning
 - 60 million parameters
 - Used **ReLU** activation function, heavy data augmentation, dropout, and overlapped pooling layers

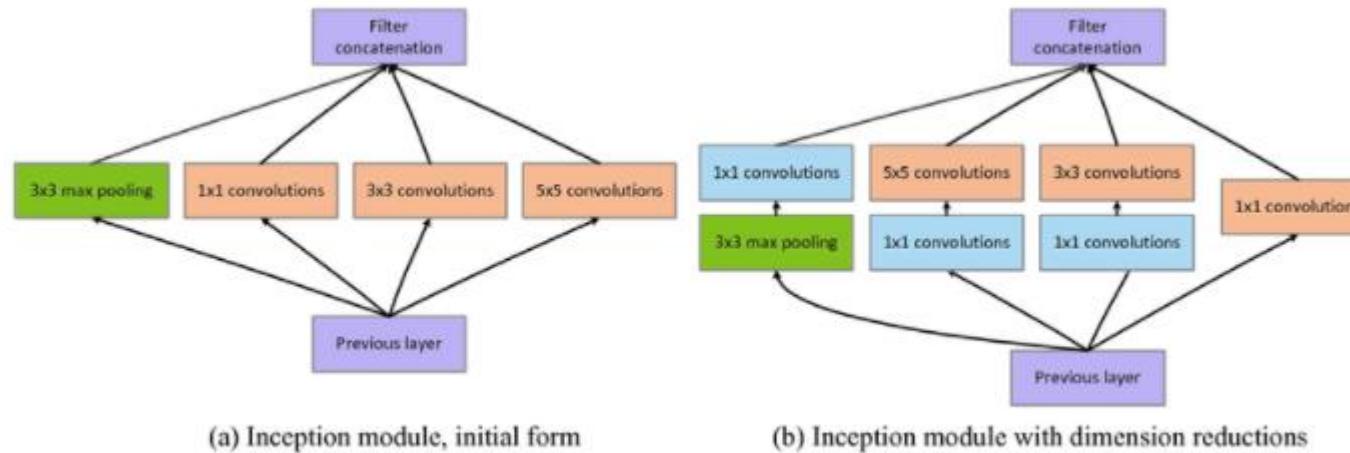


ImageNet Challenge (2012) – AlexNet ([Source](#))

(*) top-5 error: The model is considered to have classified a given image correctly if the target label is one of the model's top 5 predictions. (image classification)

ILSVRC Winners

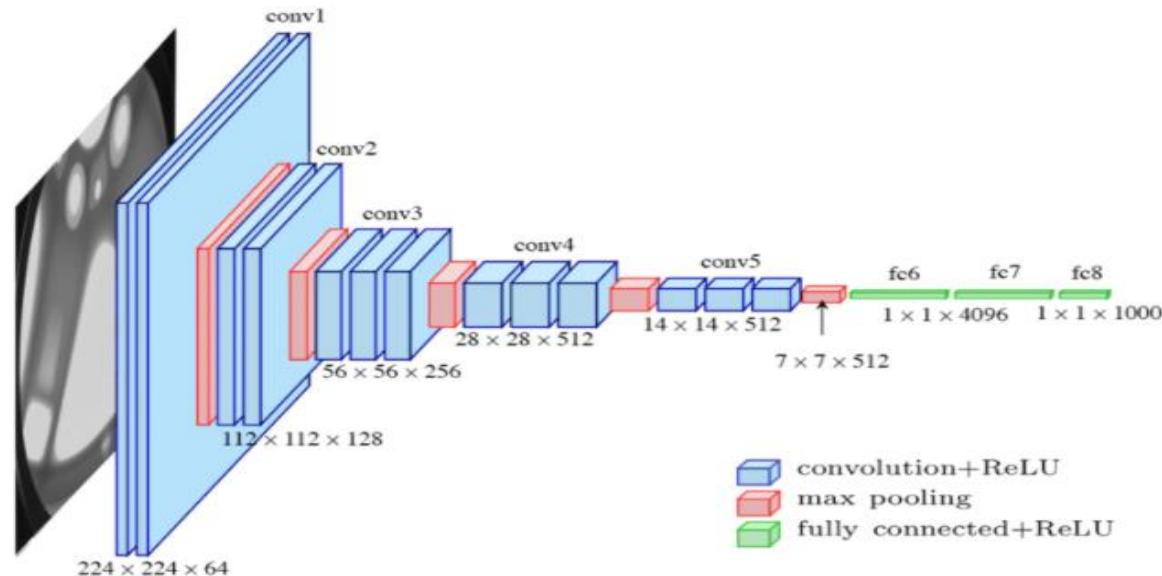
- **Inception V1 (GoogleNet) – 2014 Winner (top-5 error rate 6.67%)**
 - The v1 stands for 1st version and later there were further versions v2, v3, etc. It is also popularly known as GoogLeNet.
 - Deep with 22 layers.
 - Used multiple types of filter size, instead of being restricted to a single filter size, in a single image block, which we then concatenate and pass onto the next layer.
 - Used 1x1 convolutional with ReLU to reduce dimensions and number of operations.



ImageNet Challenge (2014)- Inception-V1 (GoogLeNet) [Source](#)

ILSVRC Winners

- **VGG-16 (University of Oxford) – 2014 Runners-Up (top-5 error rate 7.3%)**
 - Despite not winning the competition, VGG-16 architecture was appreciated and went on to become one of the most popular image classification models.
 - instead of using large-sized filters like AlexNet, it uses several 3×3 kernel-sized filters consecutively. The hidden layers of the network leverage ReLU activation functions.
 - VGG-16 is however very **slow to train** and the network weights, when saved on disk, occupy a **large** space.

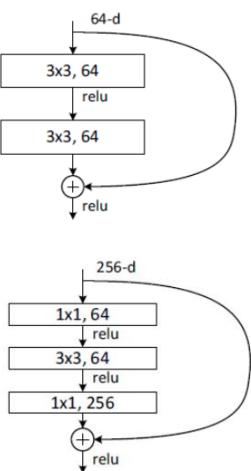


ImageNet Challenge (2014) – VGG-16 ([Source](#))

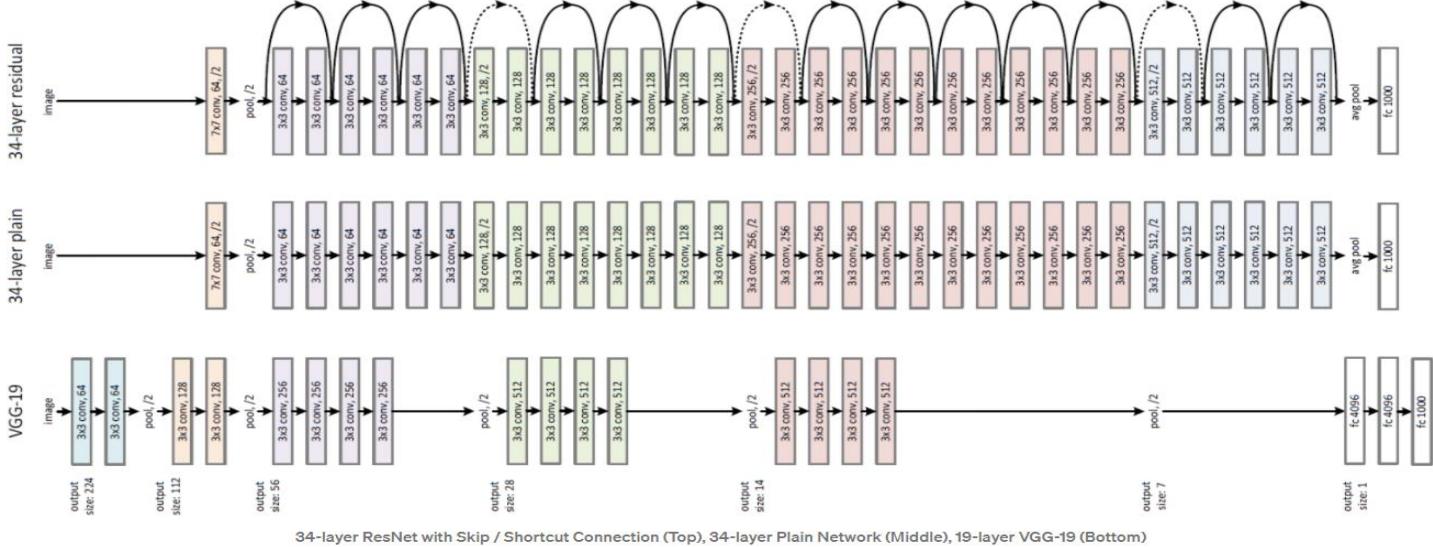
ILSVRC Winners

- **ResNet – 2015 Winner (top-5 error rate 3.57%)**

- ResNet ([Residual Network](#)) was created by the Microsoft Research team.
- To solve the problem of vanishing/exploding gradients, a [skip/shortcut connection](#) is added to add the input x to the output after few weight layers as below:
- 1×1 Conv can reduce the number of connections (parameters) while not degrading the performance of the network so much. (as in Inception-V1)
- ResNet-18/34/50/101/152 has 1.8/3.6/3.8/7.6/11.3 GFLOPs (lower than VGG-16/19 with 15.3/19.6 GFLOPS)



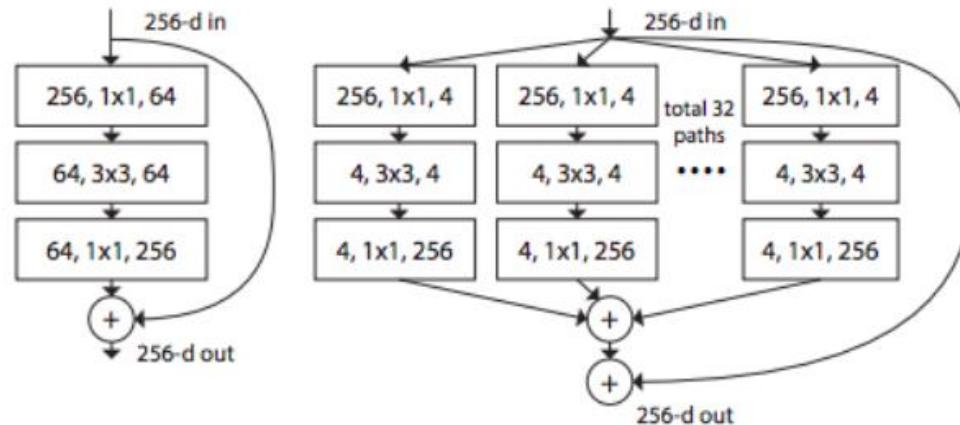
The Basic block (top) and the Proposed Bottleneck design (bottom)



(*) VGG-19 (bottom): state-of-the-art approach in 2014
middle: deeper network of VGG-19 (i.e. more Conv layers)

ILSVRC Winners

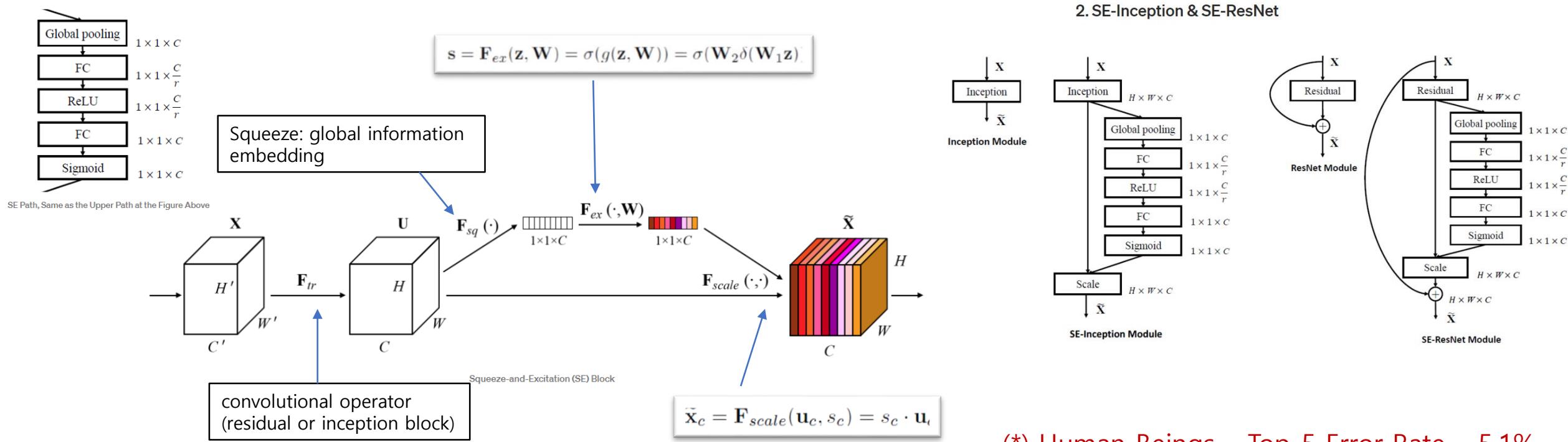
- **ResNeXt – 2016 Runners-Up (top-5 error rate 4.1%)**
 - Developed in the collaboration of the Researchers from UC San Diego and Facebook [AI](#) Research.
 - Inspired by (ResNet + VGG + Inception).
 - Still it became a popular model.
 - stacks the blocks and then use the ResNet approach of residual blocks. Here the hyper-parameters such as width and filters were also shared.



ImageNet Challenge (2016)- ResNeXt ([Source](#))

ILSVRC Winners

- **SENet(Squeeze-and-Excitation Network) – 2017 Winner (top-5 error rate 2.251%)**
 - Developed in University of Oxford.
 - With “Squeeze-and-Excitation” (SE) block that **adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels**, SENet is constructed.
 - SE block can be added to both Inception and [ResNet](#) block easily as **SE-Inception** and **SE-ResNet**. (achieved the best 2.25%).



(*) Human Beings – Top-5 Error Rate – 5.1%

Recurrent Neural network (RNN)

Motivation

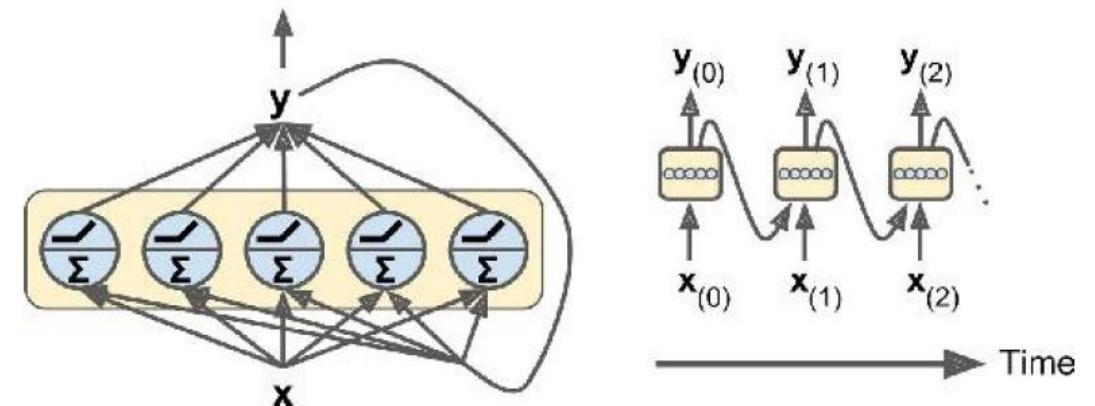
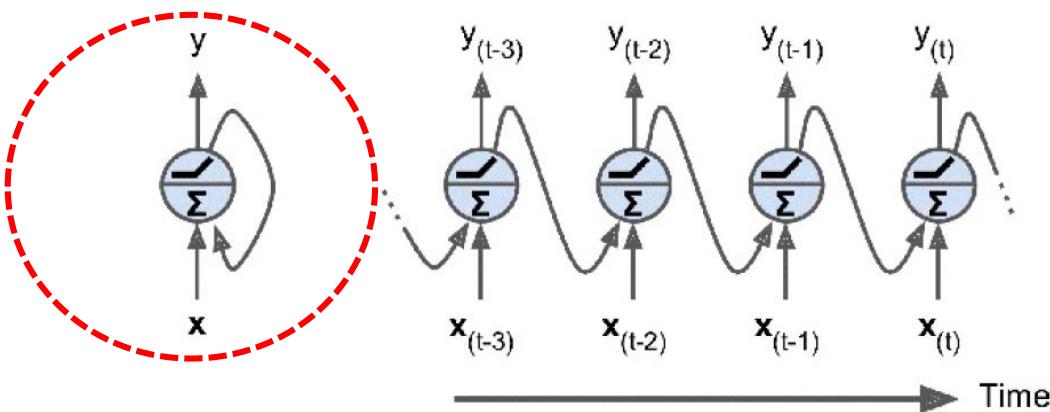
- 합성곱 신경망(CNN)
 - 2차원(공간) 필터에서 좋은 성과 입증, 이미지 학습 분야에서 널리 사용
- 그러나 CNN은 시간적인 차이를 두고 연속적으로 발생하는 데이터 분석에는 성능이 부족
 - 연속된 단어로 구성되는, 문장 분석, 자연어 처리, 자동 번역, 시계열 데이터 분석 등
- 이런 데이터에는 **순환신경망**(Recurrent Neural Network)이 잘 동작

Motivation – Text processing

- 기존 텍스트 분석에서 BoW(Bag of Words)를 이용
 - 문장 내에 어떤 단어들이 존재하는지는 파악하지만 단어의 배열 정보는 사라지고 이용하지 못함
- 단어의 순서 정보도 분석하려면
 - BoW (bag of words), 단어 벡터 (embedding)의 도입만으로는 부족하고, 단어의 발생 순서 정보를 활용할 수 있는 신경망 모델을 사용해야 한다.
 - 이를 위해서 순환신경망(RNN)이 널리 사용
- 주요 응용
 - 음성인식
 - 텍스트 분석
 - 자연어 처리
 - 챗봇
 - 감성 분석
 - 언어 모델링(language modeling)

Recurrent Neurons

- looks very much like a feedforward neural network, except it also has **backward connections**.
- Can also create a **layer** of recurrent neurons (for mini-batch input $\mathbf{X}_{(i)}$)



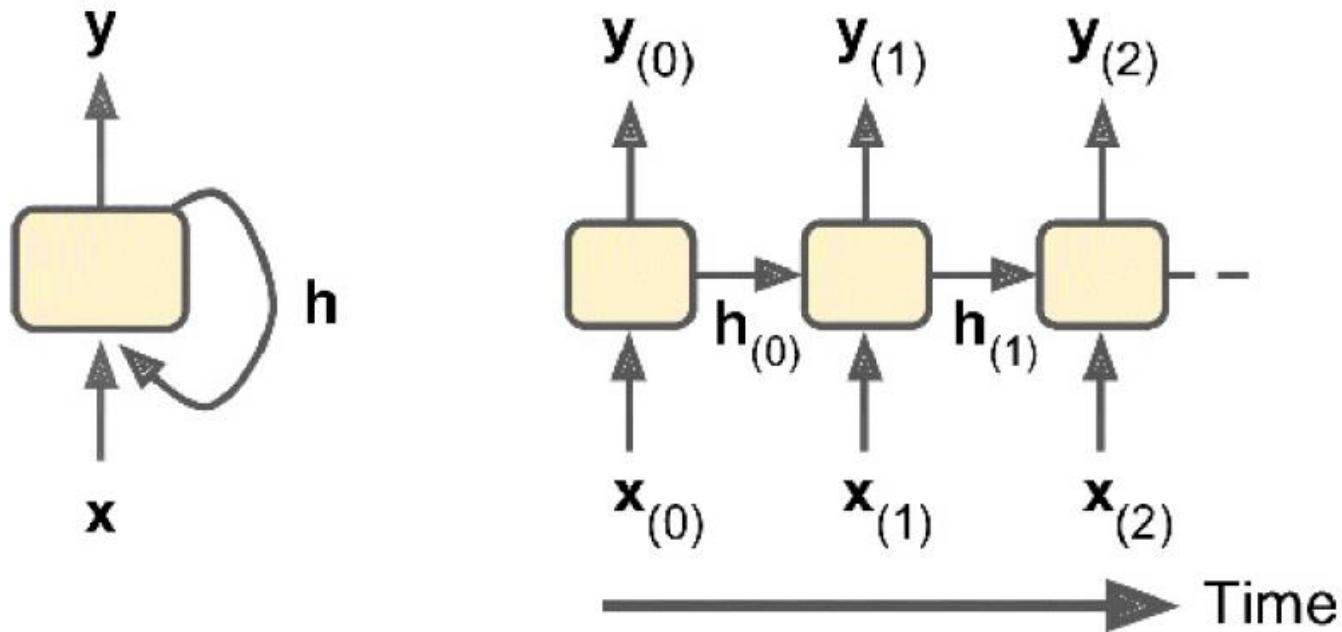
$$\mathbf{y}_{(t)} = \phi(\mathbf{x}_{(t)}^T \cdot \mathbf{w}_x + \mathbf{y}_{(t-1)}^T \cdot \mathbf{w}_y + b)$$

$$\mathbf{Y}_{(t)} = \phi(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b})$$

$$= \phi([\mathbf{X}_{(t)} \quad \mathbf{Y}_{(t-1)}] \cdot \mathbf{W} + \mathbf{b}) \text{ with } \mathbf{W} = \begin{bmatrix} \mathbf{W}_x \\ \mathbf{W}_y \end{bmatrix}$$

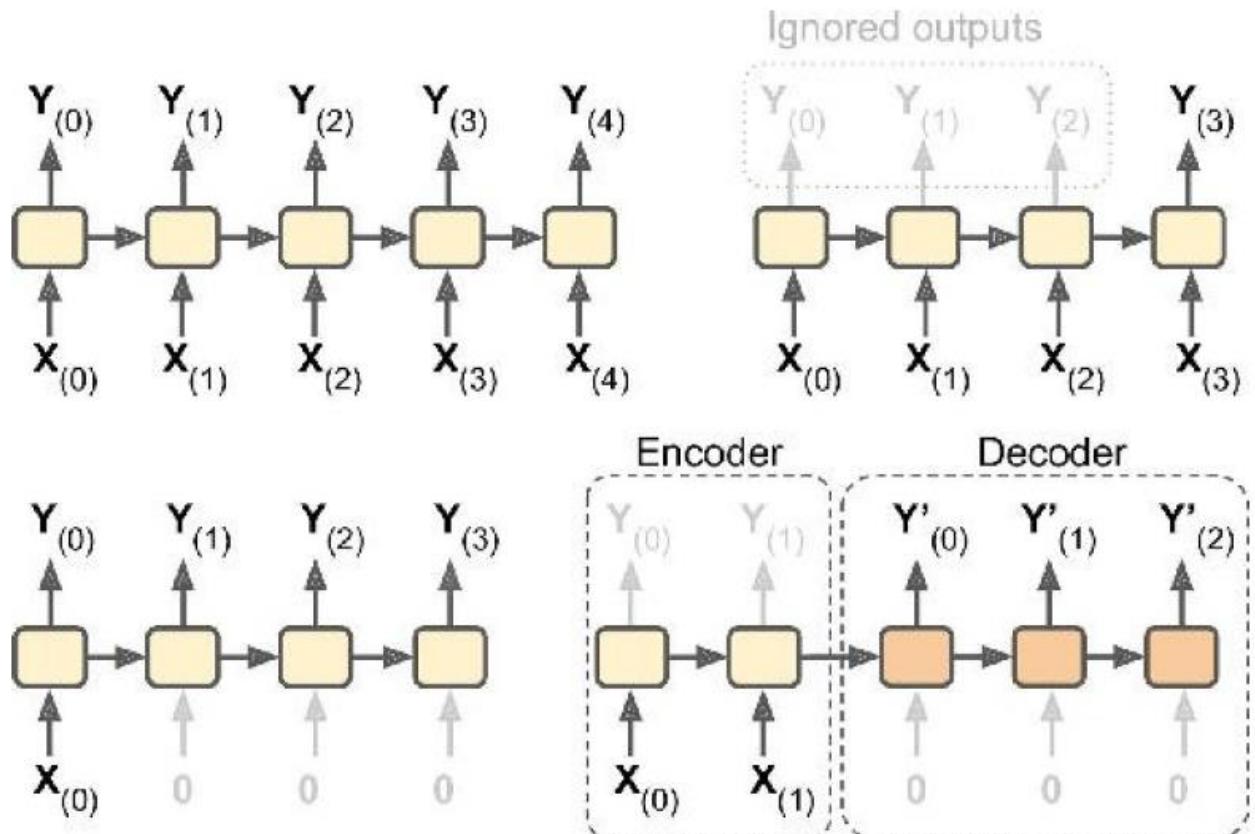
Recurrent Neurons

- In more complex cells, a cell's hidden state, $h_{(t)}$, and its output, $y_{(t)}$, may be different.



Input and Output Sequences of RNN

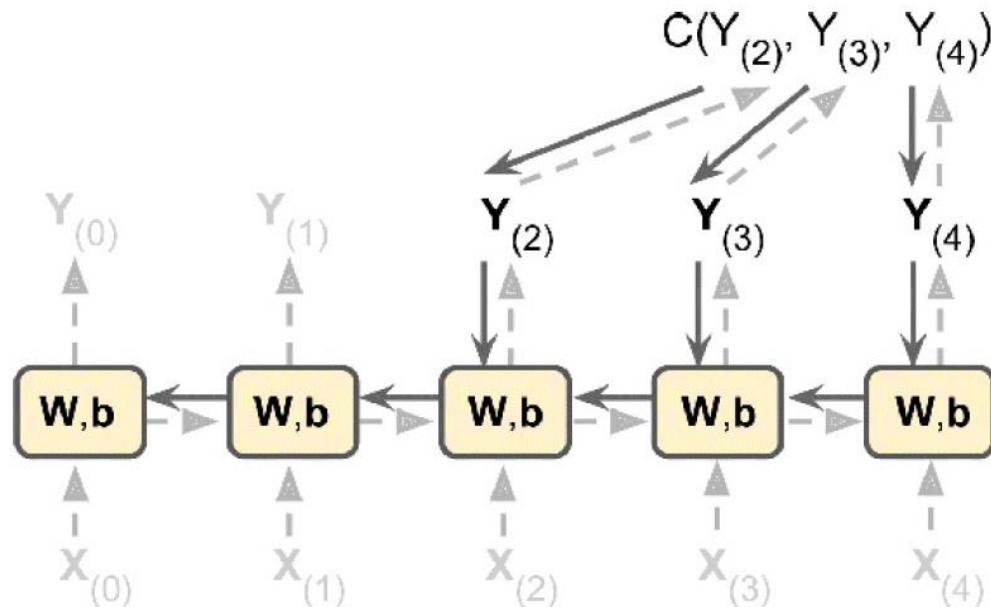
- seq-to-seq
 - (ex) **Predicting time series** such as stock prices
- seq-to-vector
 - (ex) predicting a **sentiment score** for a movie review
- vector-to-seq
 - (ex) **captioning** for an input image
- delayed seq-to-seq (encoder-decoder)
 - (ex) **translating** a sentence from one language to another



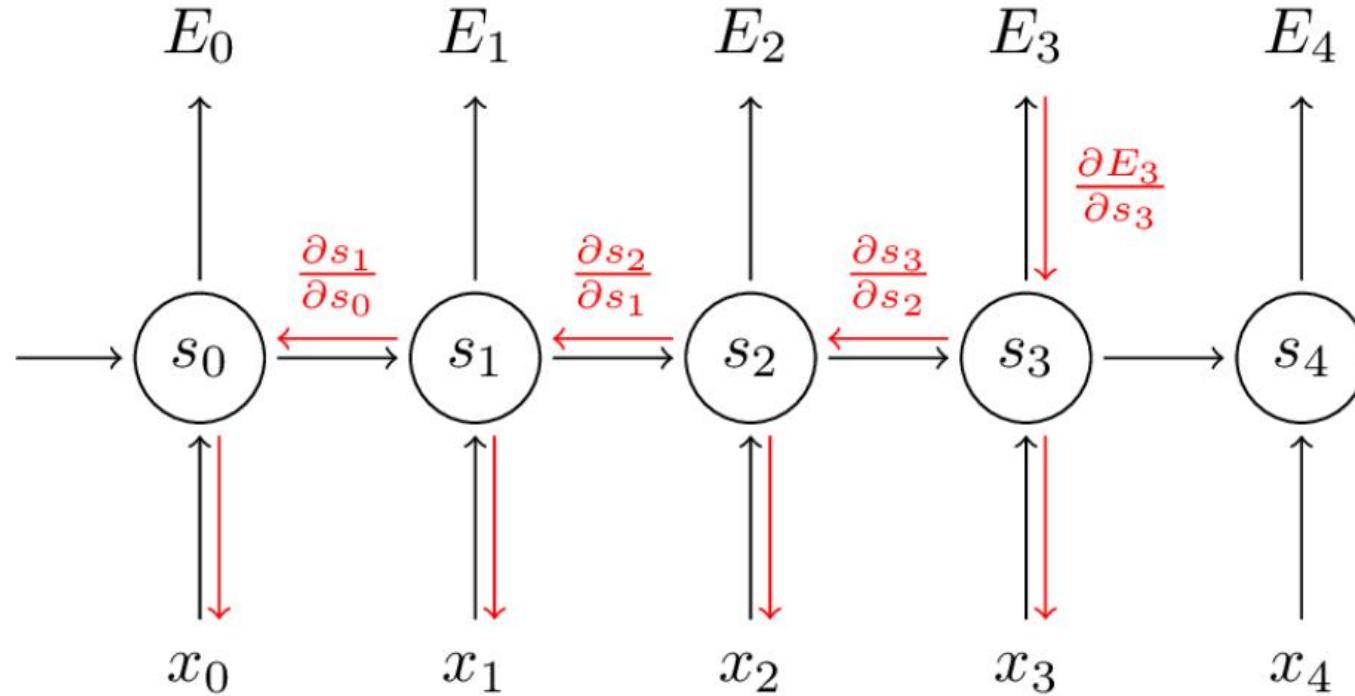
Training RNN

- BackPropagation Through Time (BPTT)

- 시간축으로 Unroll 해서 Gradient Descent 방법 사용
- 학습에 의해서 가중치 매트릭스 W , U , V 가 업데이트 되는데 RNN에서는 모든 타임 스텝에서 동일한 가중치를 사용한다.
- 즉, 현재의 경사(gradient) 변화는 과거 스텝의 계산에도 영향을 미친다.



RNN Backpropagation



- 시퀀스의 처음부터 끝까지를 모두 에러를 역전파하면 계산량이 많기 때문에 이를 줄이기 위해 현재 time step에서 일정시간 이전까지만 (보통 5 time step이전) 계산하는 Truncated-Backpropagation Through Time(생략된-BPTT)를 사용

Problems in RNN

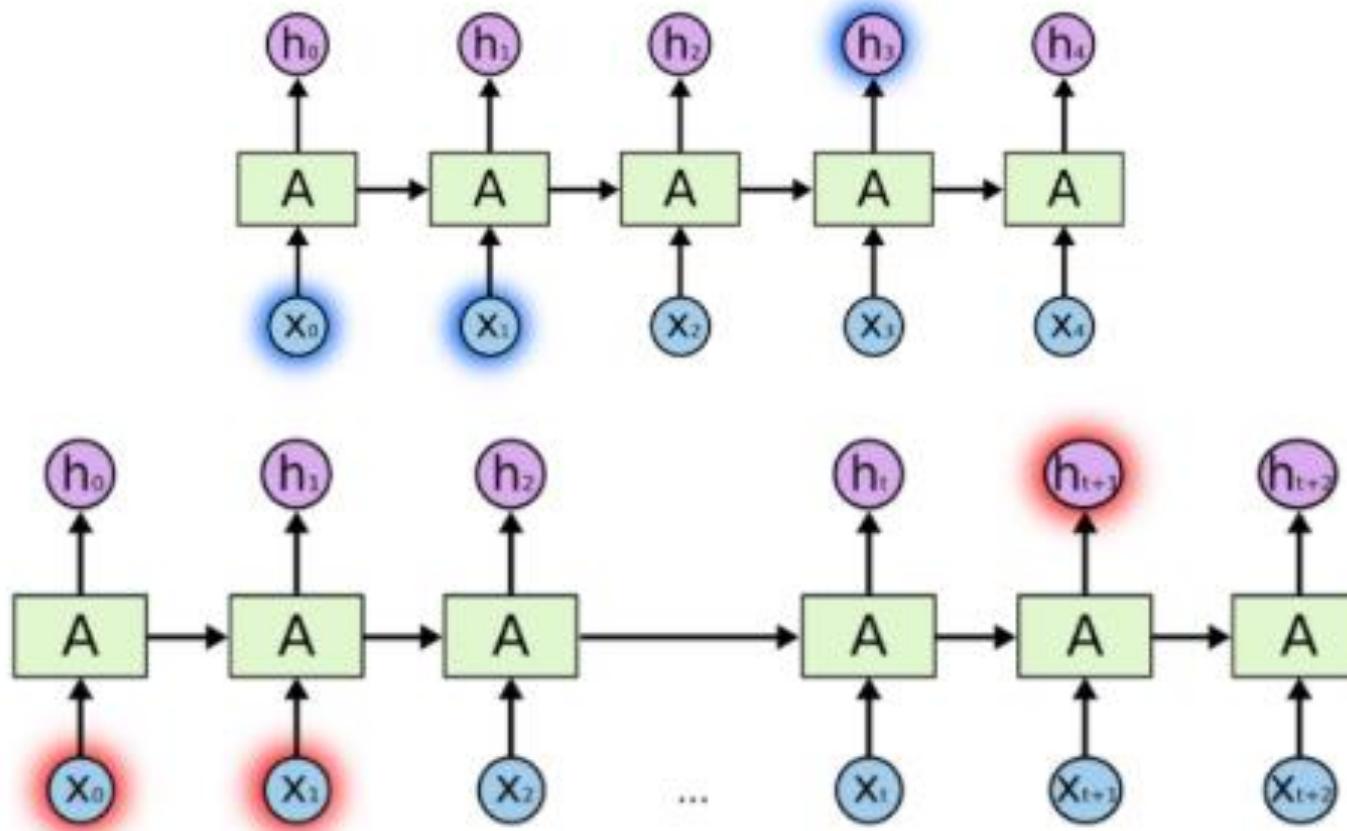
- RNN에서 여러 계층을 거치는 경우, 예를 들어 단어가 수십 개로 된 문장을 해석하는 경우, 오차 역전파를 하면서 경사값이 거의 사라지거나 또는 너무 큰 값으로 발산하여 RNN이 제대로 동작하지 못하기 쉽다.
 - 오래된 정보를 모두 중요시하면 정보가 너무 많이 축적되어 **발산**할 우려
 - 오래된 정보를 약하게 반영하면 오래되었지만 중요한 정보를 캐치하지 못하는 즉, **소실**되는 우려
- 이를 해결하기 위해서,
 - LSTM(Long-Short Term Memory), GRU 기법이 제안
 - LSTM에서는 오랜 기간 중요도를 유지할 정보와 그럴 필요 없이 망각해야 할 신호를 구분하여 따로 처리

LSTM

- 단순 RNN 구조 (케라스의 SimpleRNN)
 - 실제로는 **경사 소실-발산** 등의 문제로 인해 잘 사용하지 않는다
 - 즉, 오래된 정보가 마지막 출력단에서는 매우 약해져서 학습에 사용하기 부족해진다.
따라서 step 수가 늘면 학습이 잘 되지 않는다.
- LSTM
 - RNN의 단점을 극복하기 위해서 제안
 - 여러 스텝 앞의 정보를 놓치지 않고 따로 뒷단으로 보내주는 **채널**을 하나 더 **추가**(오래된 정보가 스텝을 지나면서 사라지지 않고 뒤에 영향을 미치도록 하는 것이 목적)
 - 우리가 대화를 할 때에도 바로 최근의 단어들을 듣고 뜻을 파악하지만 오래 전에 한 말을 통해서 전체적인 맥락이나 목적 등을 꾸준히 파악하는 것과 같은 의미
 - 즉, 시퀀스로 입력되는 데이터의 단기(short) 정보와 함께, 오래된(long) 정보를 병행해서 사용하고 학습한다는 의미로 LSTM이라는 이름을 붙임

LSTM - Vanishing Gradient Problem

- RNN은 관련 정보와 그 정보를 사용하는 지점 사이의 거리가 멀 경우
 - 역전파 시, gradient 정보가 점차 줄어 들어 학습능력이 현저히 저하

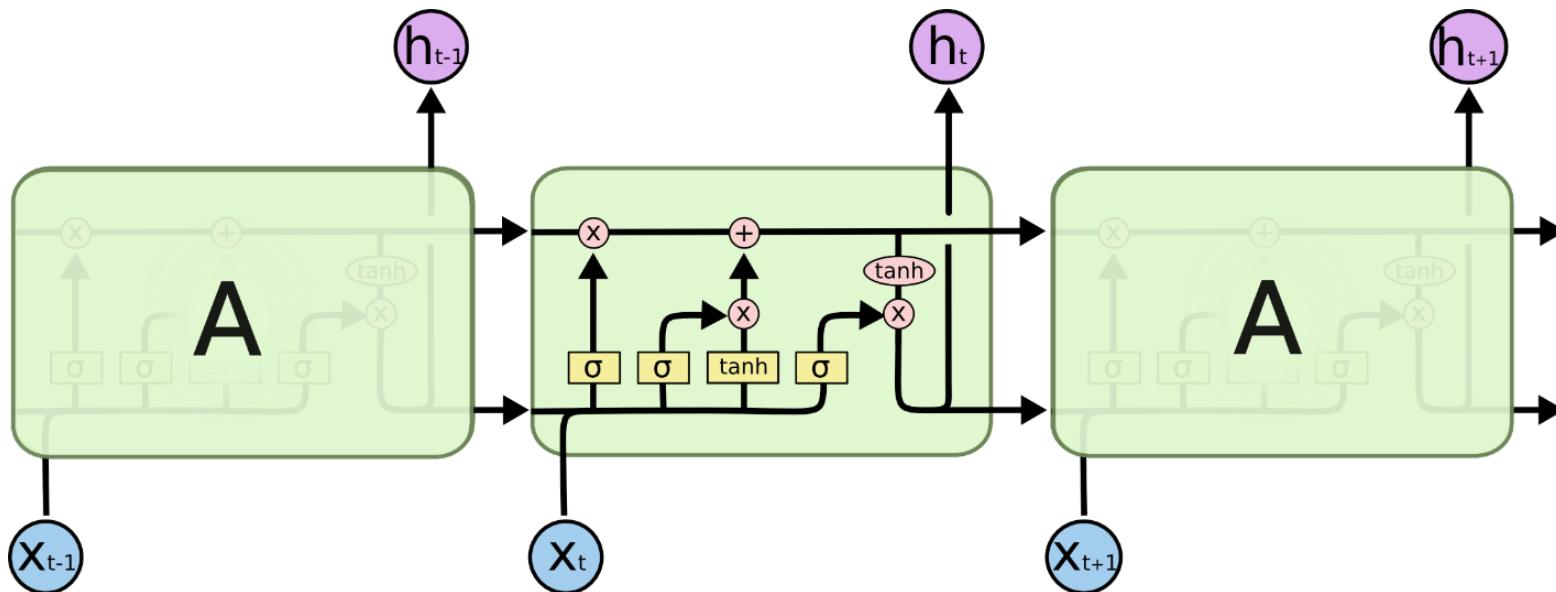


LSTM - Concept

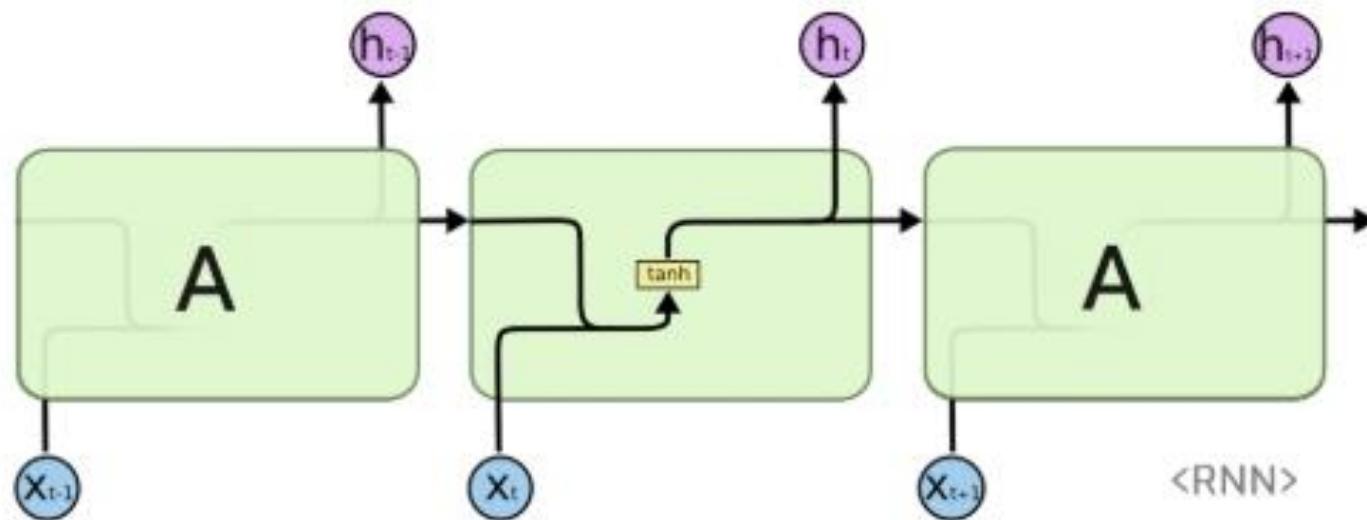
- 각 cell은 장, 단기 2개의 채널 사용
 - 단기적으로 학습한 정보가 전달되는 채널 (**short-term state, $h(t)$**): 현재의 입력 정보와 상태 정보에서는 필요한 부분을 선택
 - 장기적으로 살아남아 전달되는 채널 (**long-term state, $C(t)$**): 과거의 전달 정보에서도 필요한 정보를 필터링하는 작업을 수행
- LSTM에서 선택해야 할 하이퍼 파라미터
 - 임베딩 차원, 출력 차원
- 망각(forget), 입력(input), 출력(output) 게이트를 사용
 - 장기적인 상호작용을 학습시키는데 유용
 - 새롭고 관련성이 있는 정보를 선호하여 기억하도록 하고, 관련이 적은 정보를 잊도록 학습
- 문서 번역, 질의 응답(QA), 대화 서비스 (챗봇) 등에서 좋은 성능

LSTM Cell Structure

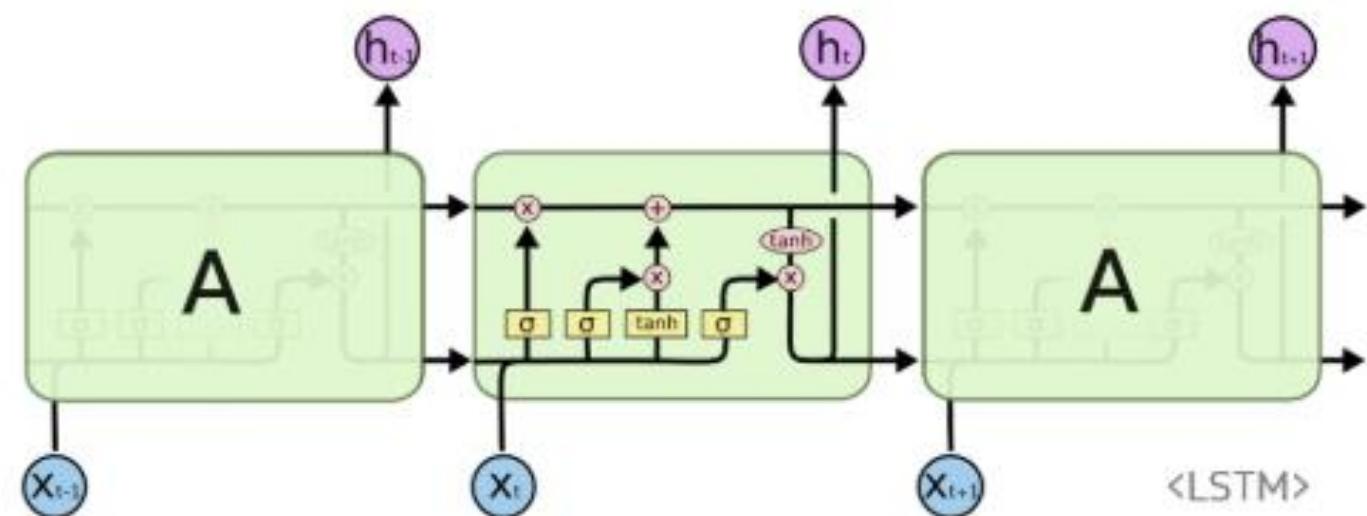
- 오래된 정보를 전달하는 Long-term state (Carry) 별도의 채널이 있다.
- 각 셀에서는,
 - 입력 신호(x), 이전 단계의 상태 정보(t), 그리고 이 전달 정보(c) 세 가지 정보의 가중치 합을 구하고,
 - 활성화 함수를 통과하여 출력(t)을 만든다



RNN and LSTM



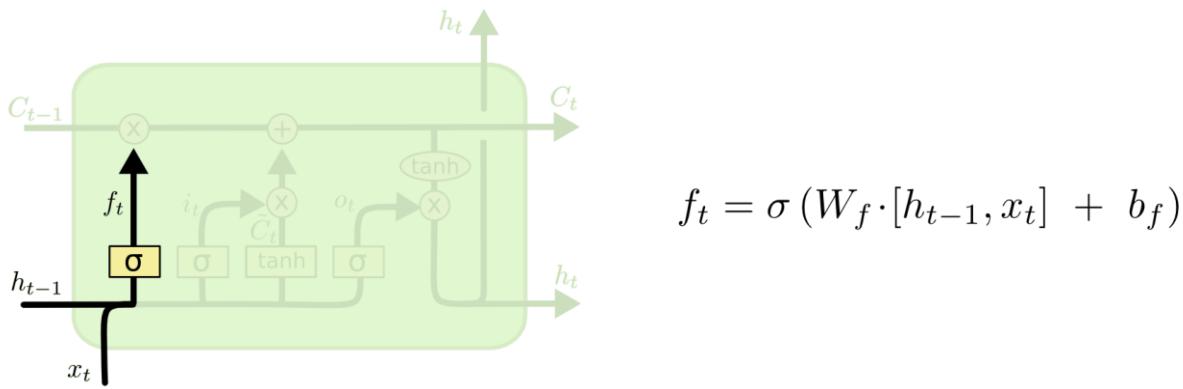
<RNN>



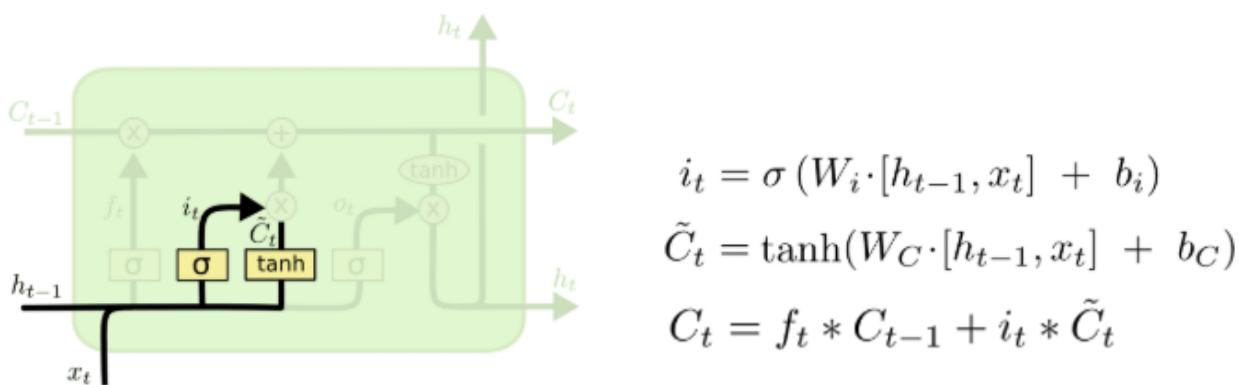
<LSTM>

LSTM Cell

- forget gate: cell state (C_{t-1})로부터 어떤 정보를 버릴 것인지를 결정 (sigmoid 사용)

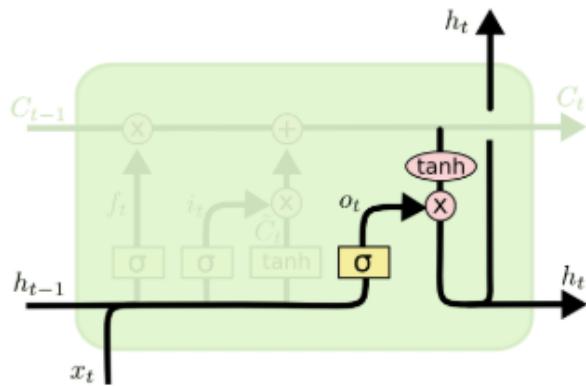


- input gate: 앞으로 들어오는 새로운 정보 중 어떤 것을 cell state에 저장할 것인지를 결정



LSTM

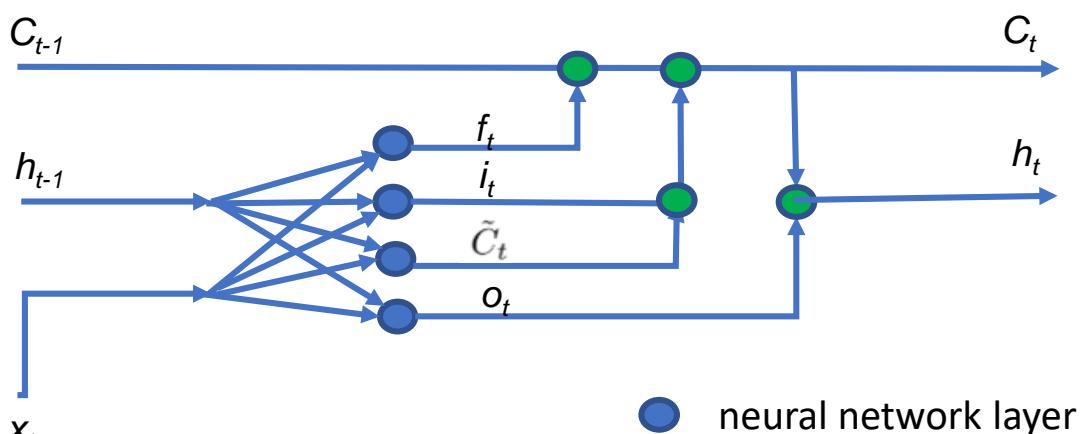
- output gate: 무엇을 output으로 내 보낼지를 결정



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

- Summary



- neural network layer
- pointwise operation

$$f_t = \sigma_g (W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g (W_i x_t + U_i h_{t-1} + b_i)$$

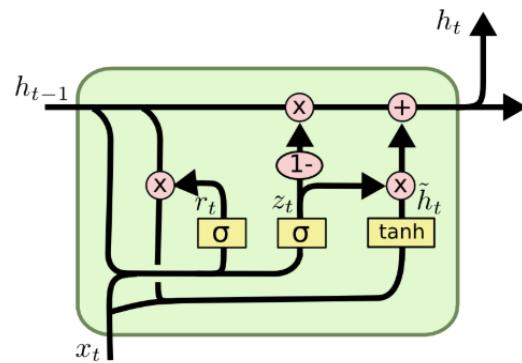
$$o_t = \sigma_g (W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c (W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \circ \sigma_h (c_t)$$

GRU

- Simplified version of LSTM (KyungHoon Cho, 2014)
 - 응용에 따라서 LSTM보다 성능이 우수하기도 하고 떨어지기도 함
- 2개의 게이트 사용
 - 리셋 게이트
 - 업데이트 게이트 : forget과 input 게이트를 합한 기능을 수행



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

임시 cell state $\tilde{c}_t = \tanh(W_{xh_g} x_t + W_{hh_g}(r_t \otimes h_{t-1}) + b_{h_g})$

최종 cell state (=hidden state) $h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes \tilde{c}_t$

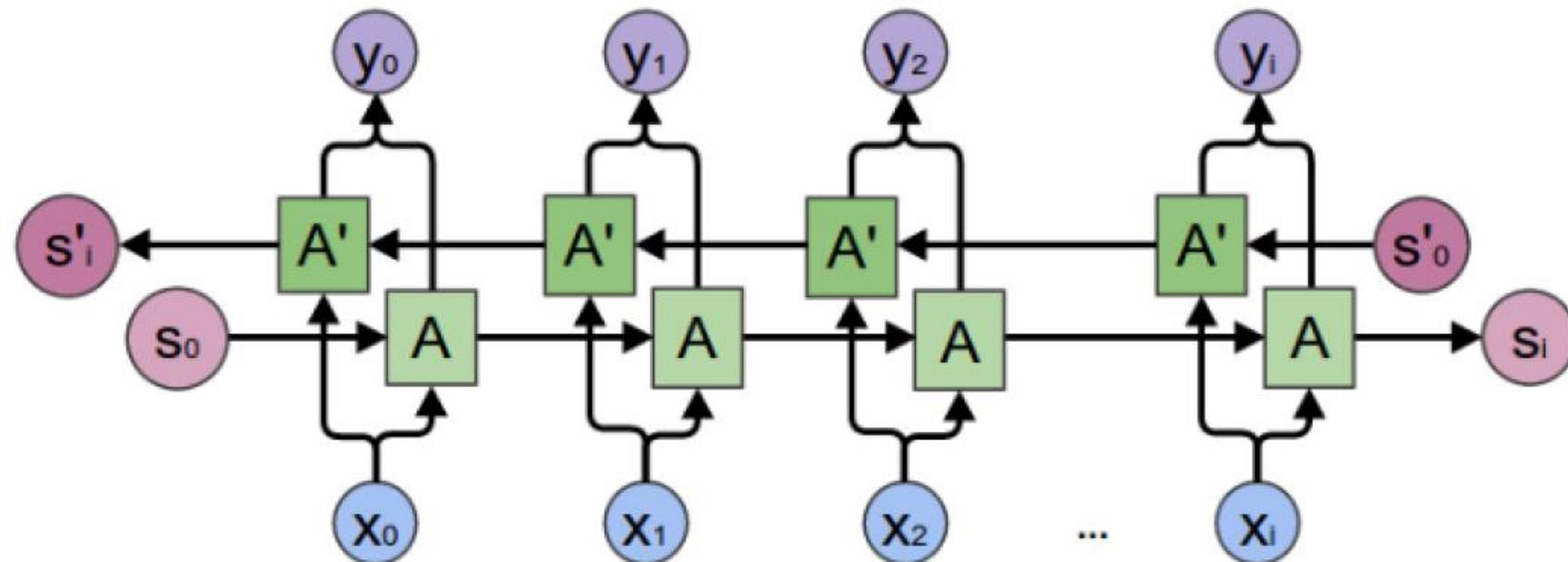
forget gate 역할
(과거정보)

input gate 역할
(현재정보)

- It combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state and makes some other changes, making it simpler than standard LSTM models.

Bidirectional RNN

- 시퀀스를 양쪽 방향으로 처리하여(즉, 두 번 프로세싱함), 한쪽 방향으로만 볼 때 놓치기 쉬운 패턴을 찾아보는 방법
- 즉, 현재에서 과거로의 추정과 함께 과거로부터 현재로의 추정을 동시에 진행하여 이 중에 좋은 패턴을 활용한다는 개념



Bidirectional RNN

- 양방향 RNN이 항상 잘 성능을 개선하는 것은 아니나 좋은 성능을 나타내는 경우가 있다. 특히 자연어 처리에서 좋은 성능을 낸다.
- 문장과 같이 단어의 나열의 경우에 반대 방향으로 단어의 순서를 뒤집어서 예측하는 경우도 비슷한 성능을 보임. 이는 언어에서 반대의 순으로 말을 하여도 컴퓨터를 거의 비슷하게 학습할 수 있다고 볼 수 있다.
- 같은 정보를 다른 방법으로 표현하는 것을 활용하여 앙상블을 취하면 더 좋은 성능을 낼 수 있다는 가정에서 양방향 RNN을 도입.
- Keras에서는 양방향 RNN을 구축하기 위한 Bidirectional 계층을 지원한다. 그러나 두 배 많은 파라미터를 사용하게 되어 과대적합이 될 가능성도 더 높아진다.

Seq2Seq and Attention Mechanism

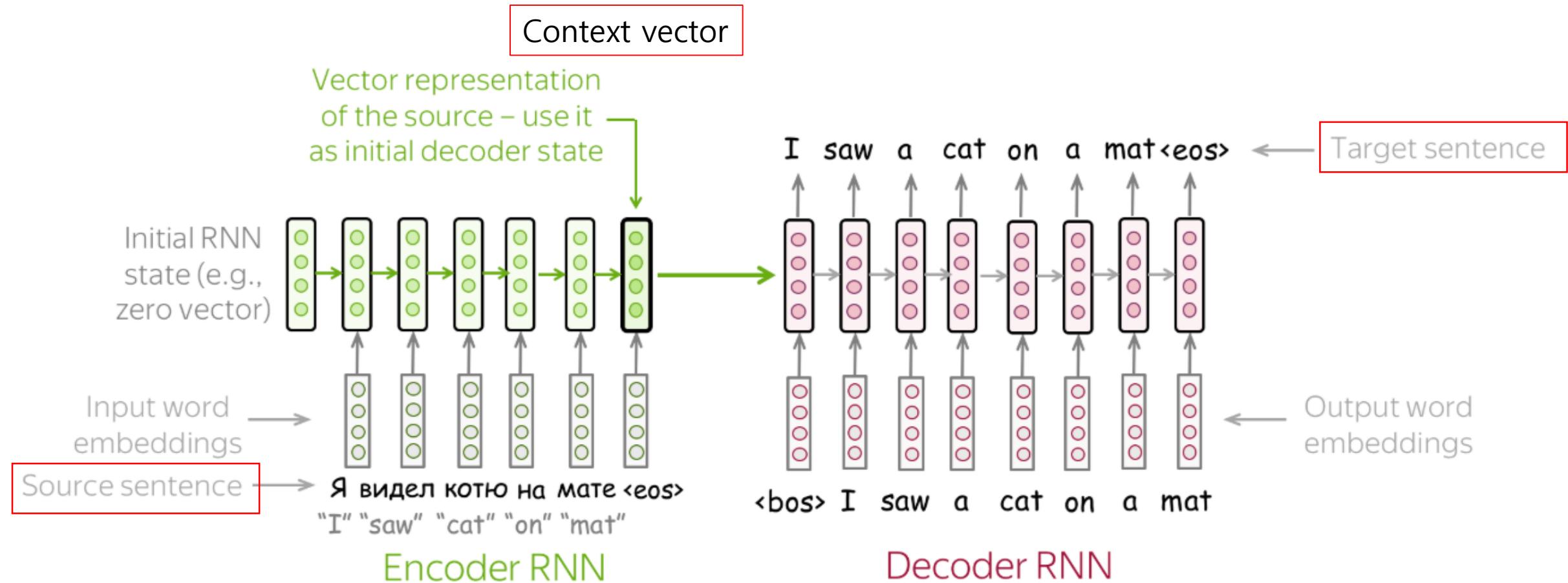
Seq2Seq 모델

- 개념
 - 입력 시퀀스를 받아 다른 시퀀스로 변환하는 딥러닝 구조로, 대표적으로 기계 번역·문장 요약·챗봇 등에 사용.
 - 핵심은 '인코더(Encoder)'와 '디코더(Decoder)' 두 부분으로 나뉘어 입력을 압축하고, 그 정보를 바탕으로 새로운 시퀀스를 생성.
- 인코더 (Encoder)
 - 입력 시퀀스를 받아 의미를 함축한 벡터(**컨텍스트 벡터**)로 변환
 - RNN(LSTM, GRU)이나 Transformer 블록을 사용
 - 예: 영어 문장 "I love you" → 벡터 표현
- 디코더 (Decoder)
 - 컨텍스트 벡터를 기반으로 출력 시퀀스를 순차적으로 생성
 - 예: "나는 너를 사랑해"라는 한국어 문장으로 변환
 - 디코더는 이전에 생성한 단어를 입력으로 받아 다음 단어를 예측하는 방식으로 진행.
- 고정 문맥 벡터의 단점
 - 고정된 길이의 문맥 벡터를 사용하므로, 긴 시퀀스를 기억하는 데 한계가 있다.
 - 입력 시퀀스가 길어질수록 컨텍스트 벡터에 모든 정보가 압축되기 어려워져 성능이 저하.

Seq2Seq 모델

- 주요 특징
 - 입력과 출력 길이가 달라도 처리 가능
 - 번역, 요약, 질의응답 등 다양한 NLP 문제에 적합
- 학습 방식
 - **Teacher Forcing**: 디코더는 학습 시 모델의 예측값 대신 이전 시점의 **실제 정답 단어**를 입력으로 사용 (학습을 안정화하고 빠르게 수렴시키지만, 학습과 추론 환경이 달라지는 단점도 있음)
 - **실제 추론 시에는 정답이 없고 모델의 예측값을 사용해야 함.**
- Attention 도입
 - 단순 컨텍스트 벡터 대신, 입력 시퀀스의 각 위치에 가중치를 두어 더 정밀한 정보 전달
 - Transformer 구조는 Attention을 기반으로 발전한 Seq2Seq 모델
- 활용 분야
 - 기계 번역 (Machine Translation): 영어 → 한국어, 한국어 → 일본어 등
 - 문장 요약 (Summarization): 긴 문서를 짧게 요약
 - 챗봇 (Dialogue Systems): 질문 → 답변 생성
 - 음성 인식 (Speech-to-Text): 음성 시퀀스 → 텍스트 시퀀스
 - 이미지 캡션 생성: CNN으로 이미지 특징 추출 후 Seq2Seq로 문장 생성

Seq2Seq



Attention Mechanism

- 등장 배경: Seq2Seq 모델의 고정된 문맥 벡터가 긴 시퀀스에서 정보를 손실하는 문제를 해결하기 위해 Attention 메커니즘이 등장.
- 핵심 아이디어:
 - 디코더에서 출력 단어를 예측하는 매 시점(time step)마다, 인코더에서의 전체 입력 문장을 다시 한 번 참고한다.
 - 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야 할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중(attention)해서 보게 된다.
- 가중치 부여 및 집중
 - 고정된 문맥 벡터 대신, 디코더가 출력을 생성하는 매 시점(y_t)마다, 인코더의 모든 중간 상태(h_j)를 참고하여 새로운 문맥 벡터를 구성.
 - 디코더의 현재 상태와 인코더의 각 상태 간의 정렬을 측정하여 가중치($\alpha_{t,j}$)를 계산
 - 새로운 문맥 벡터(c_t)는 인코더의 은닉 상태를 이 가중치로 가중 합(Weighted Sum).

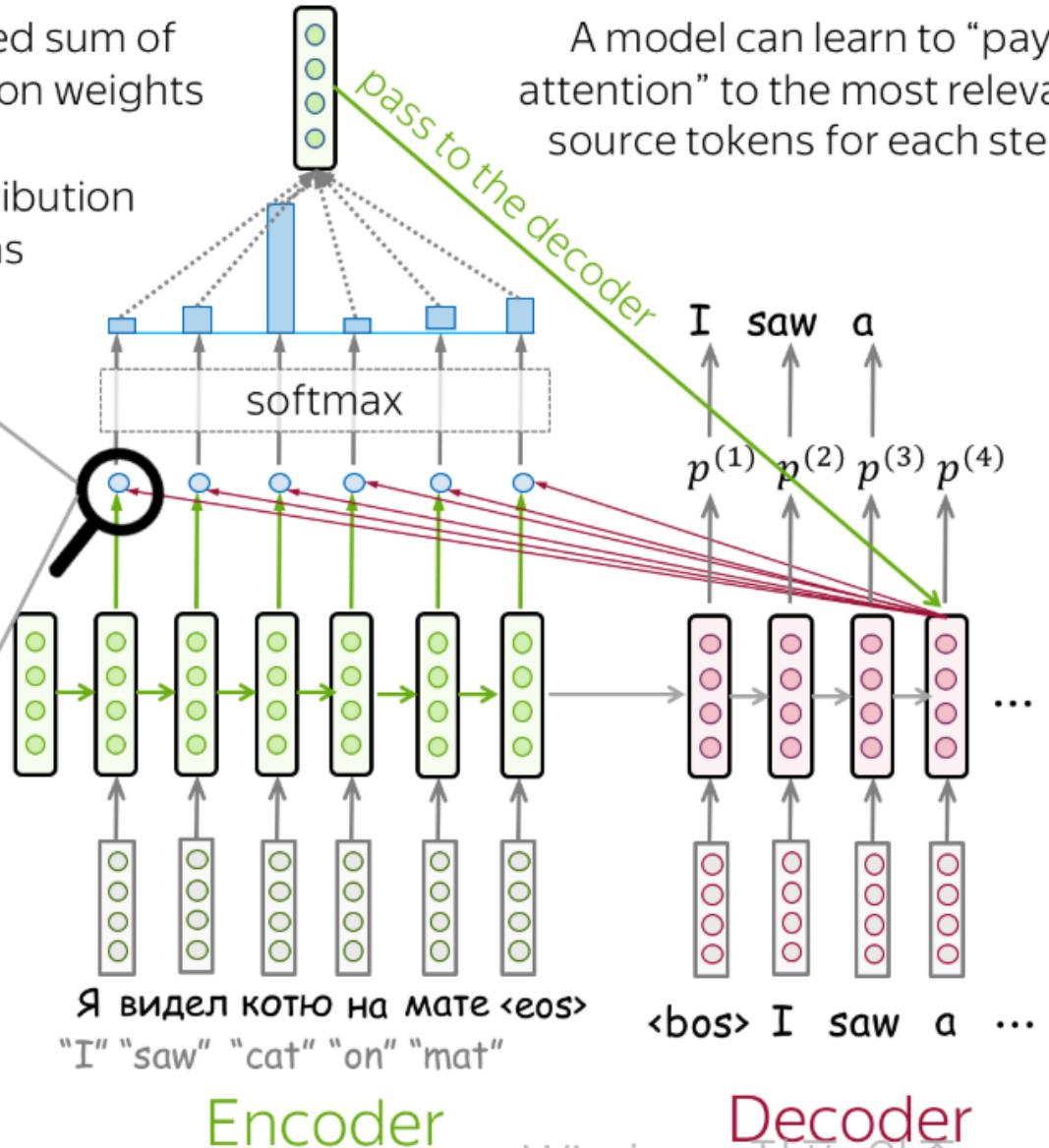
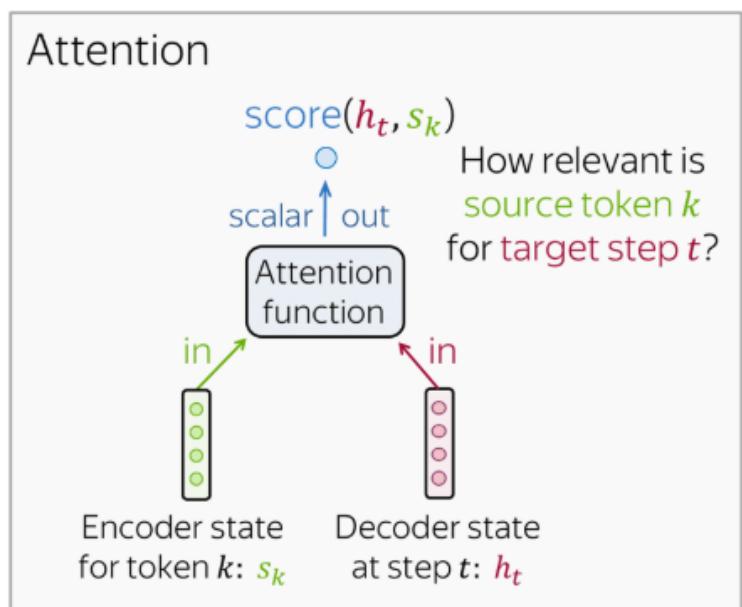
$$c_t = \sum_{j=1}^{T_x} \alpha_{t,j} h_j$$

Attention

Attention output: weighted sum of encoder states with attention weights

A model can learn to “pay attention” to the most relevant source tokens for each step

Attention weights: distribution over source tokens

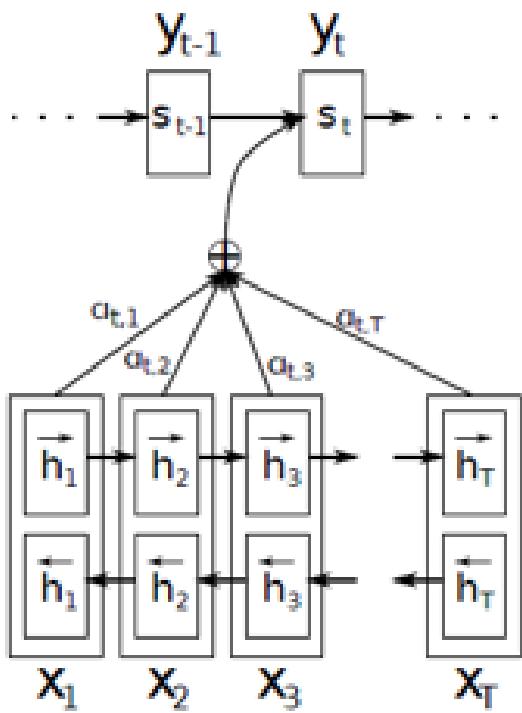


Attention

- 인코더는 마지막 은닉 상태만 전달하는 대신, 모든 은닉 상태($h_1, h_2, h_3 \dots$)를 디코더에 제공한다.
- 디코더는 매 단계에서,
 - 자신의 현재 은닉 상태 (own current hidden state)와 인코더의 모든 은닉 상태를 비교한다.
 - 이 비교를 통해 각 입력 단어가 현재 출력 단어와 얼마나 관련 있는지를 나타내는 **어텐션 점수(attention scores)**를 계산한다. (유사도(Similarity)를 기반으로 계산)
 - 이 점수는 소프트맥스(softmax)를 통해 확률 형태의 가중치로 변환된다.
 - 디코더는 이 가중치를 이용해 인코더 은닉 상태들의 **가중합(weighted context vector)**을 새로 계산한다.
 - (출력은 인코더 hidden state들의 가중합으로 표현되지만, 그 가중치를 구할 때는 디코더의 이전 상태를 사용한다.)
- 즉, 어텐션 메커니즘은 디코더가 출력 단어를 생성할 때 입력 문장의 중요한 부분에 집중하도록 도와주는 장치이다.

Attention

- Attention Mechanism (from original paper – bidirectional)



$$h_j = [\vec{h}_j^\top; \overleftarrow{h}_j^\top]^\top$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

$$e_{ij} = a(\underline{s_{i-1}}, h_j)$$

- Sequence of annotations (h_1, h_2, \dots, h_T) for each input sequence (T : number of words in the input sequence)
- Context vector for the output word y_i : simply a weighted sum of hidden states
- Weight (also learned by a feed-forward network)
- output score of a feedforward network (function a attempts to capture the alignment between input at j and output at i)

Attention 계산 과정

- Attention 계산은 기본적으로 “유사도(Similarity)”를 기반으로 한다. 디코더의 현재 상태(또는 Query)와 인코더의 출력(또는 Key)을 비교하여 얼마나 관련성이 있는지를 점수로 매기고, 그 점수를 Softmax로 변환해 가중치로 사용한다.
- 계산 과정:
 - Query, Key, Value 정의
 - Query(Q): 디코더의 현재 은닉 상태
 - Key(K): 인코더의 각 은닉 상태
 - Value(V): 인코더의 은닉 상태(실제 정보)
 - Similarity 계산 (Score Function): Query와 Key를 비교하여 관련성을 측정
 - 대표적인 방식: **Dot-product (점곱)**: $\text{score}(Q, K) = Q \cdot K$
 - Softmax 변환: 점수를 확률 분포 형태로 변환 → 각 입력 단어가 현재 출력 단어에 얼마나 중요한지 나타냄
 - Weighted Sum (Context Vector)
 - Attention 가중치를 Value에 곱해 합산 → 디코더가 집중해야 할 정보만 추출

Scaled Dot-product: $\frac{Q \cdot K}{\sqrt{d_k}}$ (Transformer에서 사용)

Attention 계산 과정

- Query, Key, Value의 역할
 - Query (Q): 디코더의 현재 상태 → "지금 내가 어떤 정보를 필요로 하는가?"
 - Key (K): 인코더의 각 은닉 상태에서 추출 → "내가 가진 정보는 이런 특징이다"
 - Value (V): 인코더의 각 은닉 상태에서 추출 → "실제로 전달할 정보"
- Key와 Value의 관계
 - 같은 인코더 은닉 상태에서 생성
 - 보통 인코더의 출력 벡터에 서로 다른 선형 변환(가중치 행렬)을 적용해서 Key와 Value를 얻음. 즉, 원천 데이터는 같지만, 변환을 달리해서 역할을 분리
 - 차이점
 - Key는 Query와 비교되어 유사도 점수(score)를 계산하는 데 사용
 - Value는 그 점수(가중치)에 의해 실제 컨텍스트 벡터를 구성하는 데 사용
- 예시: (Transformer의 Scaled Dot-Product Attention)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- QK^T : Query와 Key의 유사도 계산
- Softmax: 가중치로 변환
- V: 그 가중치로 합쳐져 최종 컨텍스트 벡터 생성

Attention 기반 디코더 출력 계산

1. Score 계산 (유사도 측정)

디코더의 현재 hidden state s_t 와 인코더 hidden state h_i 를 비교해 관련성 점수를 구함

- Bahdanau Attention (Additive):

$$e_{t,i} = v_a^T \tanh(W_s s_t + W_h h_i)$$

- Luong Attention (Dot-product):

$$e_{t,i} = s_t^T W h_i \quad \text{또는} \quad e_{t,i} = s_t^T h_i$$

2. Softmax 변환 (가중치 계산)

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}$$

→ 각 인코더 hidden state에 대한 중요도를 확률 분포 형태로 변환

3. Context Vector (가중합)

$$c_t = \sum_i \alpha_{t,i} h_i$$

→ 인코더 hidden state들의 weighted sum

4. 디코더 출력 계산

디코더는 자신의 hidden state s_t 와 context vector c_t 를 결합해 최종 출력을 예측

$$y_t = \text{softmax}(W_o[s_t; c_t] + b_o)$$

- 여기서 $[s_t; c_t]$ 는 두 벡터를 연결(concatenation)한 것
- W_o, b_o 는 출력층의 파라미터
- Softmax를 통해 단어 분포를 얻고, 가장 확률이 높은 단어를 선택

Transformer 는 scaled dot-product attention 사용

- 디코더 출력 y_t 는 디코더 hidden state s_t 와 인코더 hidden state들의 가중합 c_t 를 함께 사용해 계산됨.
- 즉, 디코더 상태는 문맥을 반영하고, Attention은 입력 문장의 중요한 부분을 반영하여 최종적으로 단어를 예측하는 구조.

기술 추이 요약: RNN → Attention → Transformer

1. 기본 Seq2Seq (2014)
 - 인코더 RNN(LSTM/GRU)이 입력 문장을 읽고 마지막 은닉 상태를 "컨텍스트 벡터"로 전달
 - 디코더 RNN이 이 벡터를 기반으로 출력 문장을 생성
 - 문제: 긴 문장을 하나의 벡터에 압축해야 해서 정보 손실이 큼 → "정보 병목(bottleneck)" 발생
2. Seq2Seq + Attention (2015)
 - 인코더의 모든 은닉 상태를 디코더가 참조할 수 있도록 함 (Bahdanau, Luong 등이 제안)
 - 디코더는 현재 상태(Query)와 인코더 상태(Key)를 비교해 **유사도 점수(Attention Score)**를 계산
 - Softmax로 가중치를 만든 뒤 Value를 가중합해 새로운 컨텍스트 벡터 생성
 - 효과: 긴 문장에서도 특정 단어에 집중 가능 → 번역 품질 크게 향상
3. Self-Attention (2017)
 - "Attention is All You Need" 논문에서 Transformer 제안
 - RNN이나 CNN 없이, 입력 시퀀스 내 토큰들이 서로를 바라보며 관계를 학습
 - 병렬 연산이 가능해 학습 속도가 RNN보다 훨씬 빠름
 - 장기 의존성 문제 해결: 멀리 떨어진 단어 관계도 쉽게 포착
4. Transformer의 확장 (2018 이후)
 - BERT, GPT, T5 등 대형 모델들이 모두 Transformer 기반으로 발전
 - NLP뿐 아니라 이미지, 음성, 멀티모달 분야까지 확장
 - 현재는 사실상 RNN을 대체하는 표준 구조로 자리 잡음