

딥러닝팀

1팀

이정환
김동환
권가민
박준영
박채원

CONTENTS

1. 이미지 데이터의 특징

2. CNN

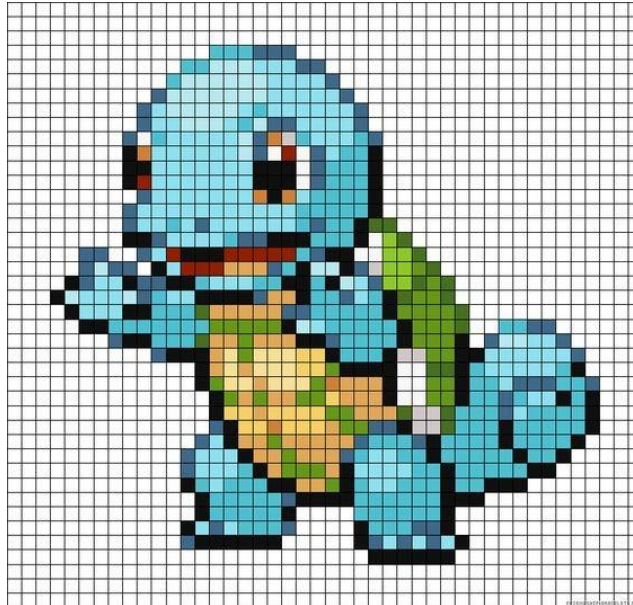
3. CNN의 발전과정

4. 컴퓨터 비전

1

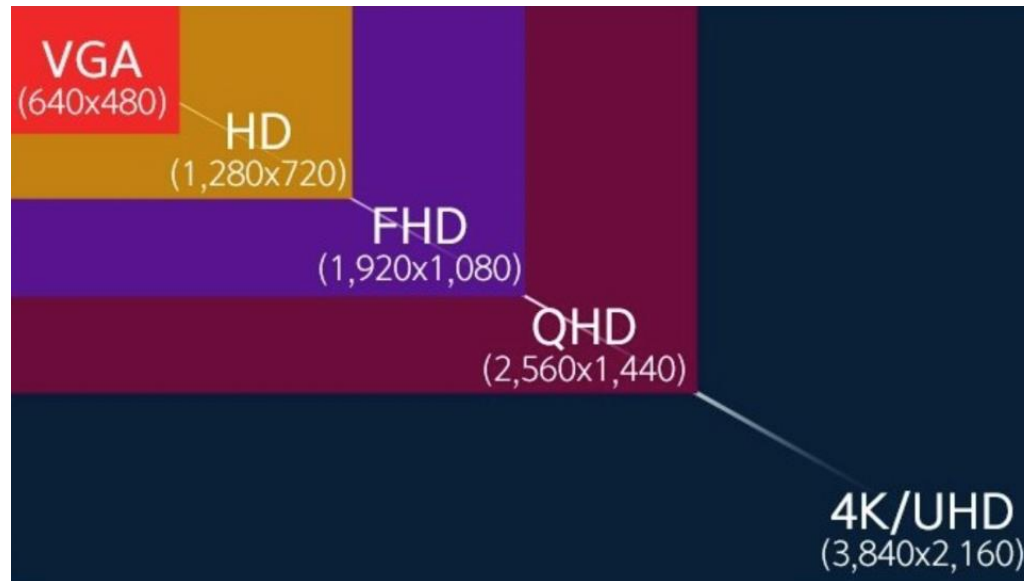
이미지 데이터의 특징

컴퓨터에서의 이미지 데이터



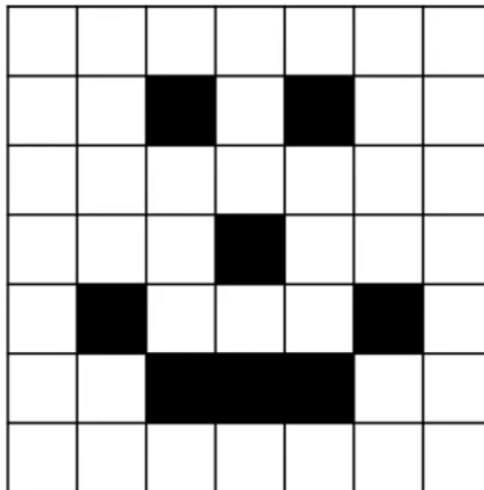
이미지는 **픽셀**이라 불리는 사각형의 집합
픽셀에는 **값**을 이용하여 **색**을 지정할 수 있음

컴퓨터에서의 이미지 데이터



픽셀 수가 많을수록 **고화질**의 이미지임
 $\text{FHD}(1920 \times 1080) < \text{QHD}(2560 \times 1440)$

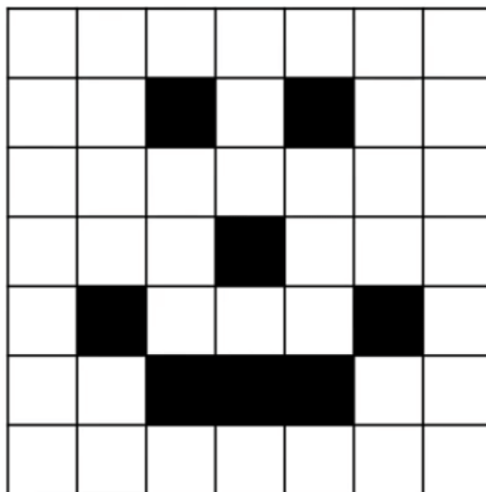
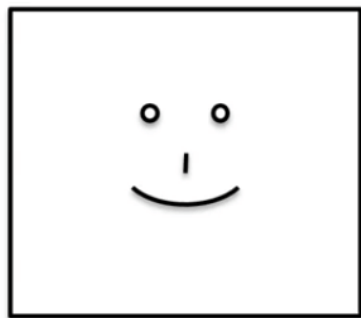
컴퓨터에서의 이미지 데이터



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

흑백 이미지의 경우 검정색이면 1, 흰색이면 0

컴퓨터에서의 이미지 데이터



0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

흑백 이미지의 경우 검정색이면 1, 흰색이면 0
이미지를 행렬의 형태로 나타낸 것이 비트맵 이미지

컴퓨터에서의 이미지 데이터

비트맵 이미지의 표현

(Height : 세로 픽셀) X (Width : 가로 픽셀) X (Channel : 차원)

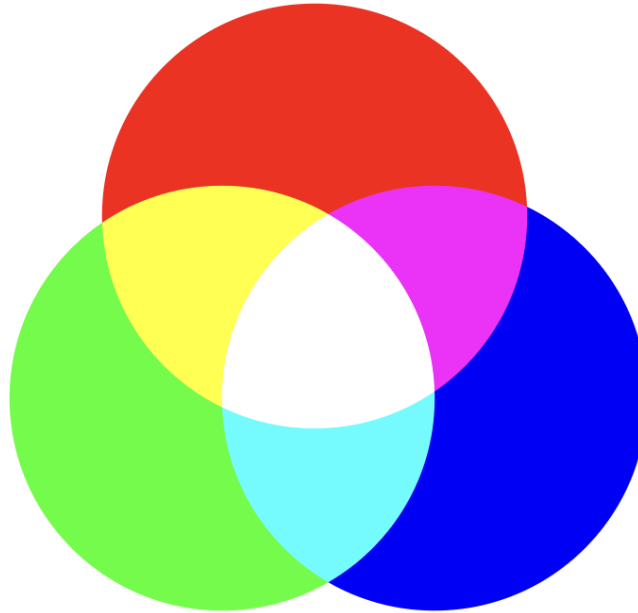
Pytorch : (C , H , W) ex. (3 , 256 , 256)

Tensor : (H , W , C) ex. (256 , 256 , 3)

흑백 이미지의 경우 검정색이면 **1**, 흰색이면 **0**
 이미지를 행렬의 형태로 나타낸 것이 **비트맵 이미지**

채널

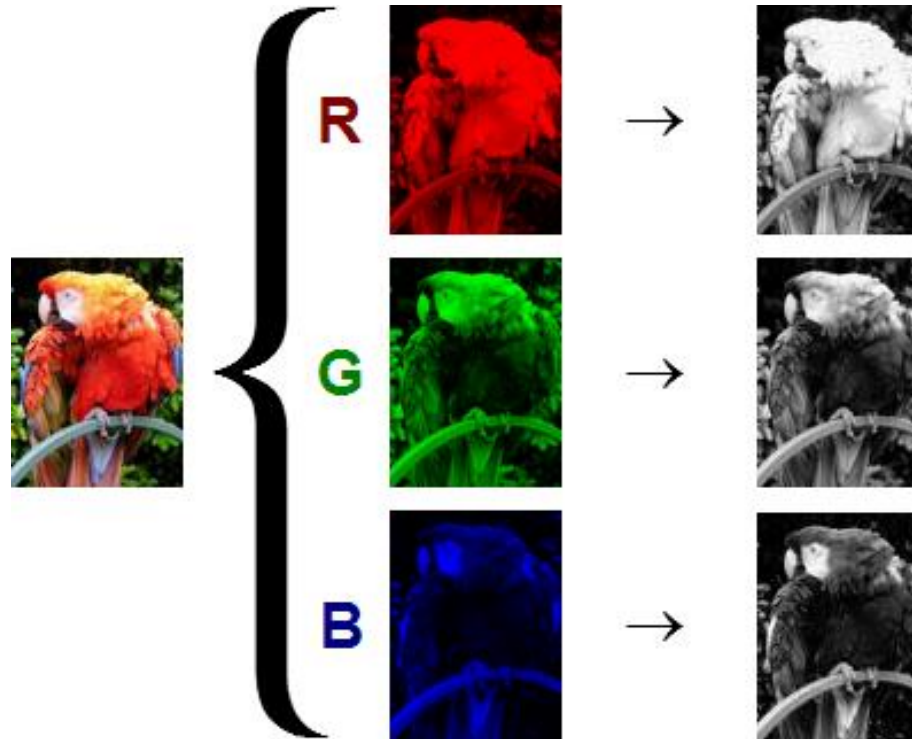
Channel



이미지 데이터는 R, G, B 3개의 채널로 구성
각각은 0~255의 값을 가지며 이에 따라 색의 선명도 결정

채널

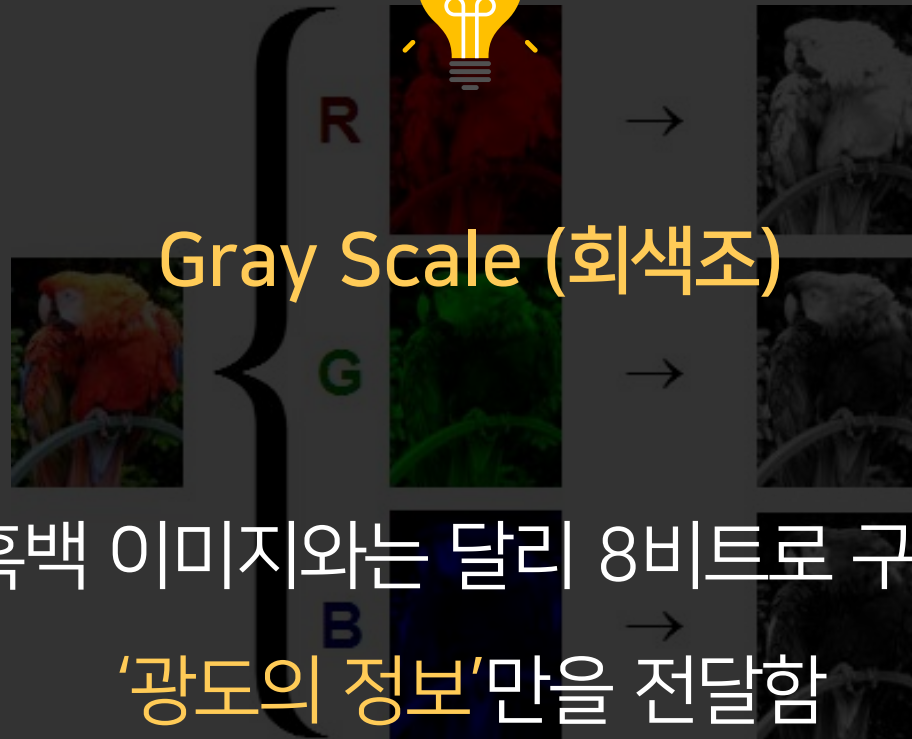
Channel



많이 포함된 색일수록 해당 채널이 상대적으로 밝음
Grayscale 역시 R채널이 가장 밝음

채널

Channel



1비트인 흑백 이미지와는 달리 8비트로 구성되어있고
'광도의 정보'만을 전달함

많이 분포된 색일수록 채널이 상대적으로 밝음

Grayscale 역시 R채널이 가장 밝음

이미지 데이터의 가정

이동에 대한 불변성 (Invariance)



입력이 조금 이동해도 출력이 대부분 변하지 않는 것
특정 위치보다 **특징 그 자체**가 중요할 때 유용한 가정

2

CNN

CNN의 등장 배경

기존 신경망은 1차원 자료만을 입력으로 받아들임

고차원의 자료들을 저차원으로 변경하는 것

고차원 데이터를 1차원 벡터로 변경해주는 임베딩 작업 필요

이 과정에서 이미지의 공간적인 정보 손실되는 문제가 발생

CNN의 등장 배경

기존 신경망은 1차원 자료만을 입력으로 받아들임

고차원의 자료들을 저차원으로 변경하는 것

고차원 데이터를 1차원 벡터로 변경해주는 임베딩 작업 필요

이 과정에서 이미지의 공간적인 정보 손실되는 문제가 발생

CNN의 등장 배경

기존 신경망은 1차원 자료만을 입력으로 받아들임

고차원의 자료들을 저차원으로 변경하는 것

고차원 데이터를 1차원 벡터로 변경해주는 임베딩 작업 필요

이 과정에서 이미지의 공간적인 정보 손실되는 문제가 발생

CNN의 등장 배경

기존 신경망은 1차원 자료만을 입력으로 받아들임

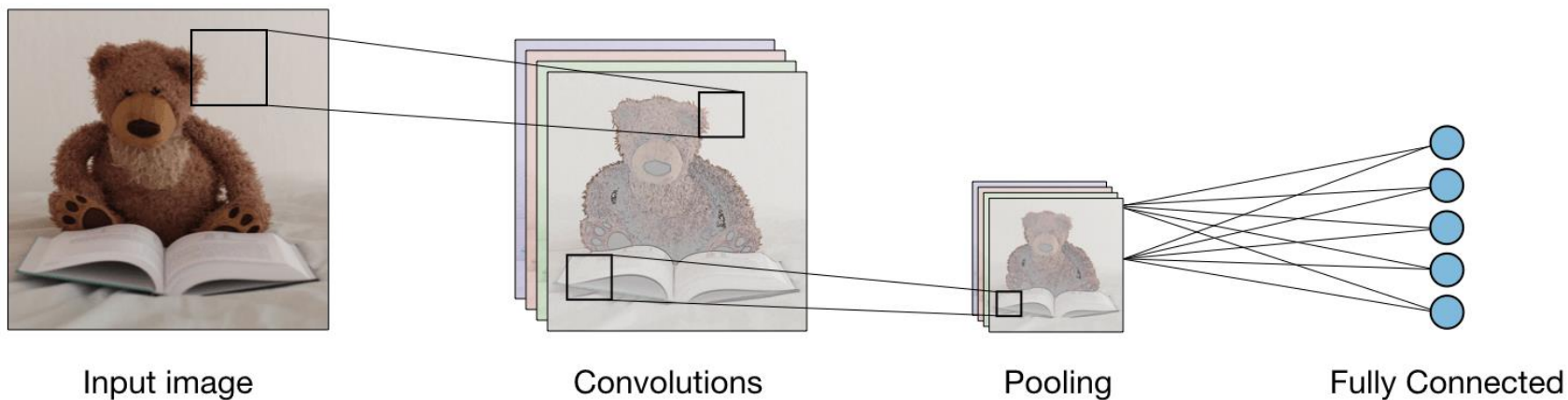
고차원의 자료들을 저차원으로 변경하는 것

고차원 데이터를 1차원 벡터로 변경해주는 임베딩 작업 필요

이 과정에서 이미지의 공간적인 정보 손실되는 문제가 발생

CNN

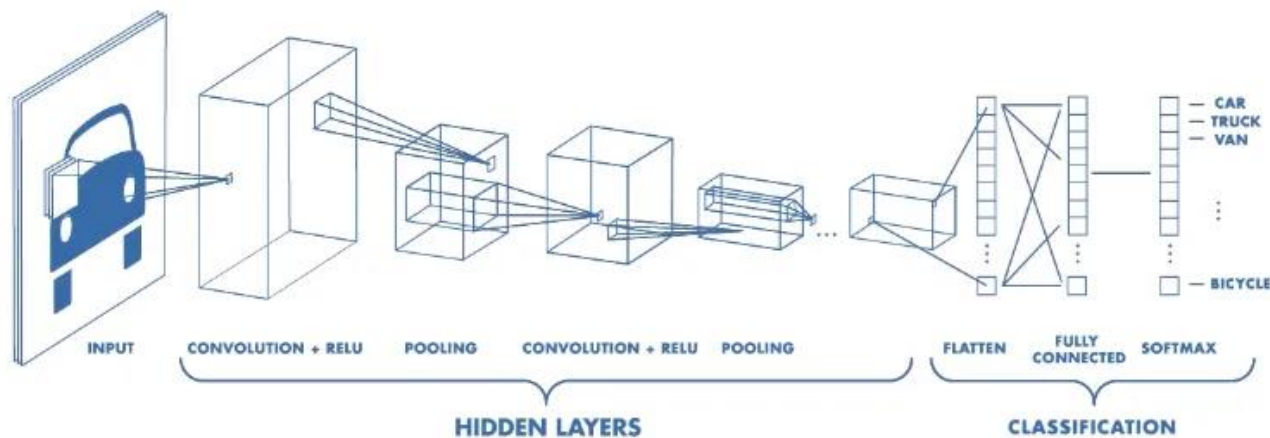
Convolution Neural Network



기존 신경망과 달리 **공간 정보가 보존됨**
검은색 상자의 위치가 다음 층에서도 그대로 유지

CNN

CNN의 구조



Convolution Layer

Pooling Layer

Fully Connected
Layer

Convolution Layer

합성곱 (Convolution)

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

입력 커널

입력함수를 커널 함수의 반전 이동한 값에 곱하고
구간에 적분하여 새로운 함수를 구하는 연산자

Convolution Layer

합성곱 (Convolution)

(f)
↓
입력



$d\tau$

입력함수를 커널 함수의 반전 이동한 값에 곱하고
구간에 적분하여 새로운 함수를 구하는 연산자

Convolution Layer

1차원 합성곱 (1D-Convolution)

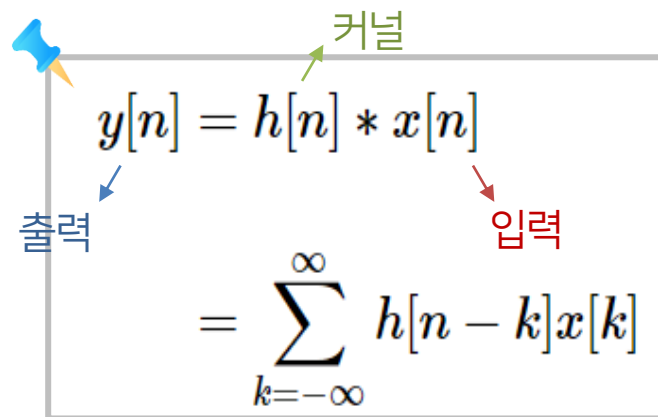


Diagram illustrating the 1D convolution equation with annotations:

- 출력** (Output): Points to $y[n]$
- 커널** (Kernel): Points to $h[n]$
- 입력** (Input): Points to $x[n]$

$$y[n] = h[n] * x[n]$$

$$= \sum_{k=-\infty}^{\infty} h[n-k]x[k]$$

$h[k]$

1/5	1/5	1/5	1/5	1/5
-----	-----	-----	-----	-----

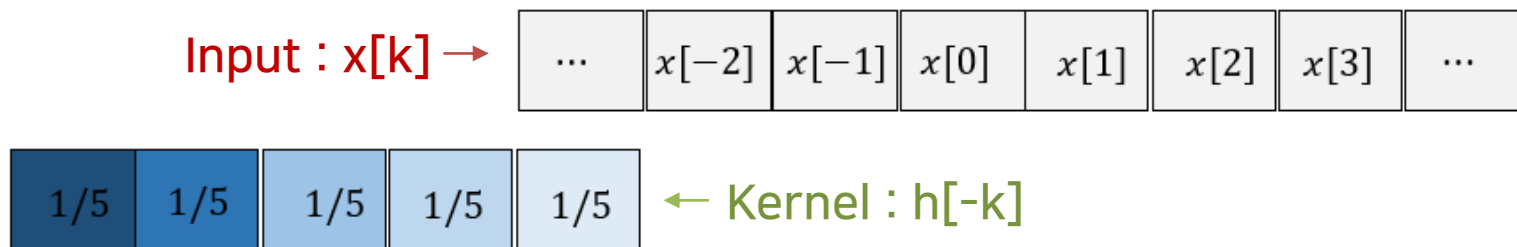
$h[-k]$

1/5	1/5	1/5	1/5	1/5
-----	-----	-----	-----	-----

커널에 같은 값을 주면 **이동 평균** 효과를 낼 수 있음

Convolution Layer

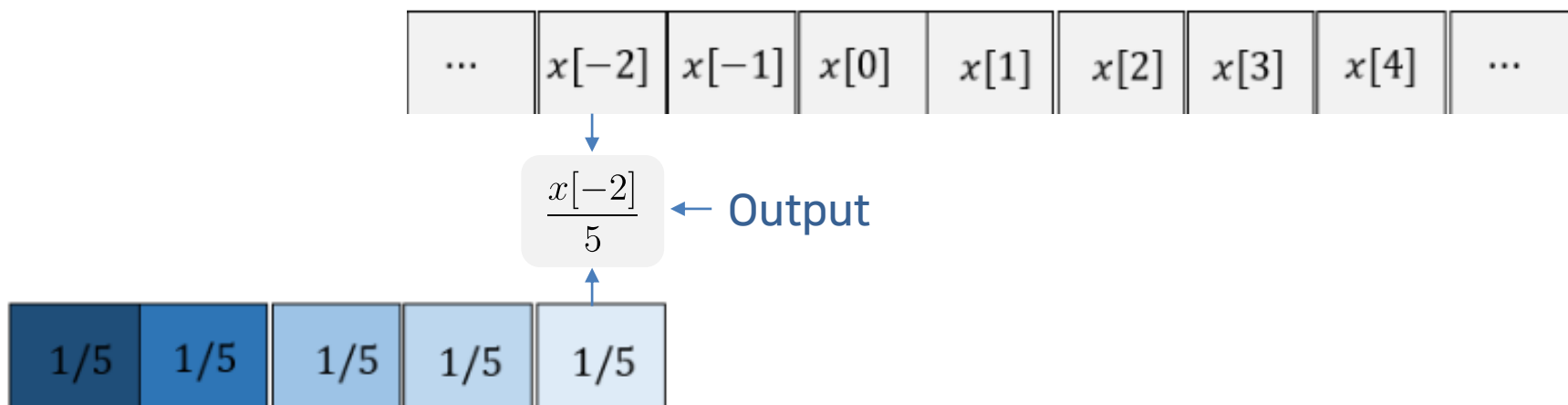
1차원 합성곱 (1D-Convolution)



커널이 지나가면서
커널의 값이 **역순**으로 곱해짐

Convolution Layer

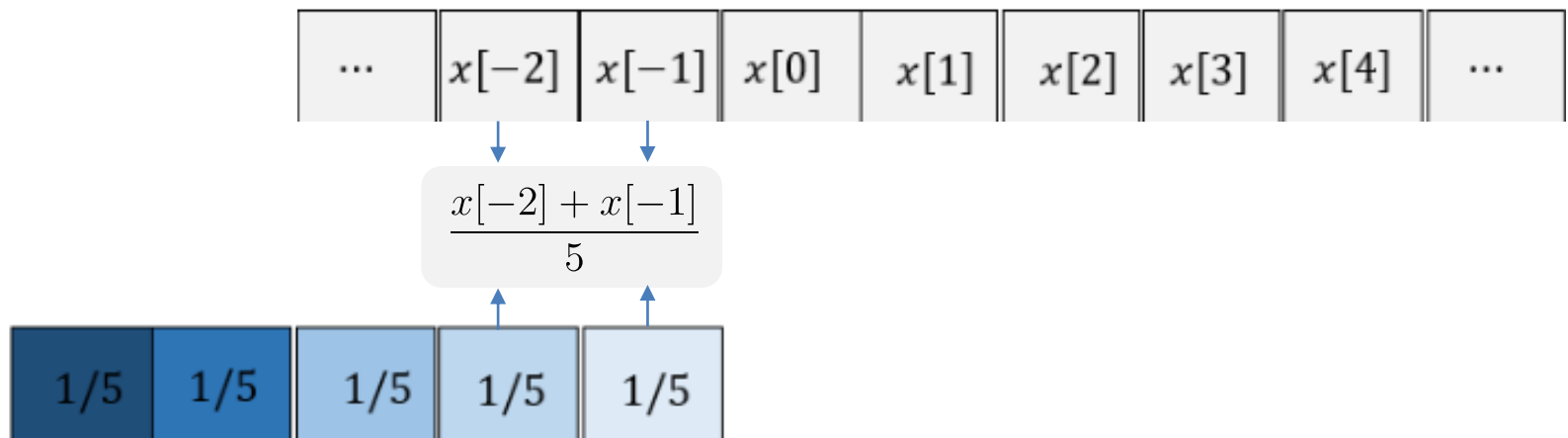
1차원 합성곱 (1D-Convolution)



커널이 지나가면서
커널의 값이 **역순**으로 곱해짐

Convolution Layer

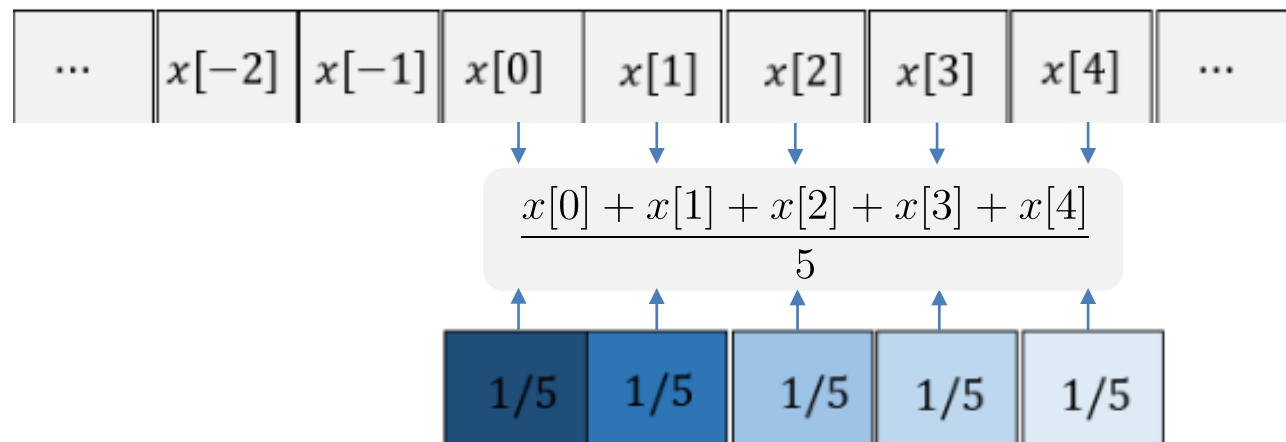
1차원 합성곱 (1D-Convolution)



커널이 지나가면서
커널의 값이 역순으로 곱해짐

Convolution Layer

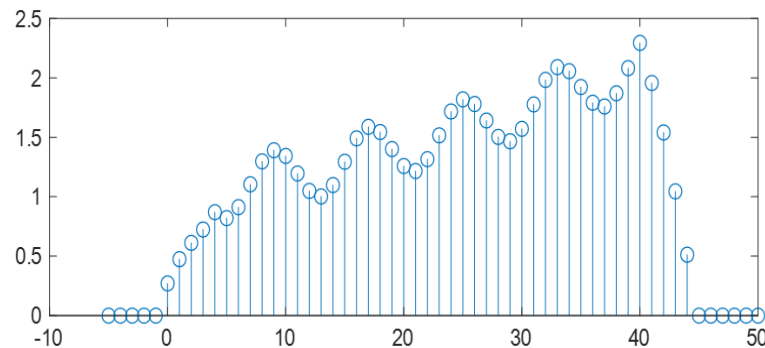
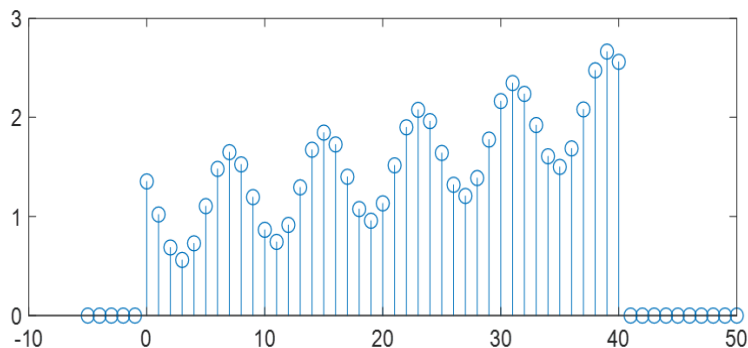
1차원 합성곱 (1D-Convolution)



커널이 지나가면서
커널의 값이 **역순**으로 곱해짐

Convolution Layer

1차원 합성곱 (1D-Convolution)



합성곱 결과 이동평균의 효과가 적용되어
출력의 그래프가 **부드러워지는** 것을 확인

Convolution Layer

2차원 합성곱 (2D-Convolution)



$$\begin{array}{|c|c|c|} \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array} =$$



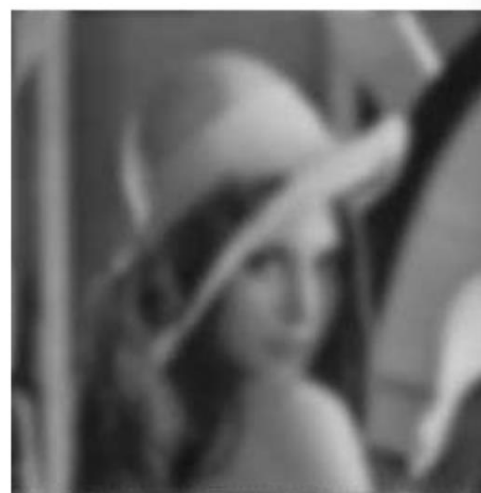
모든 커널에 **같은 값**을 주어 합성곱 진행

Convolution Layer

2차원 합성곱 (2D-Convolution)



$$\begin{array}{|c|c|c|} \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array} =$$



1차원과 마찬가지로 이동평균 효과가 적용되어
사진이 흐려지는 것을 알 수 있음

Convolution Layer

2차원 합성곱 (2D-Convolution)



$$\begin{array}{|c|c|c|} \hline +1 & +2 & +1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} =$$

* 미분 필터 Sobel Filter

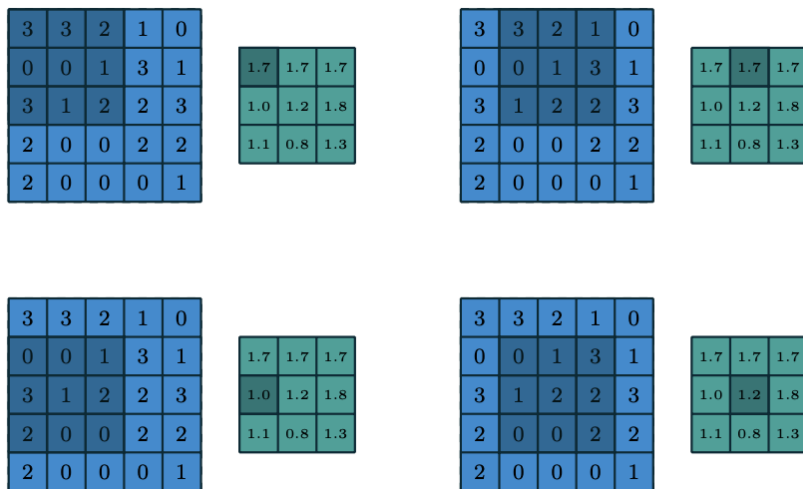


미분 필터를 이용하여 합성곱 진행
윤곽선 성분을 추출하는 효과

급격하게 값이 변하는 부분을 테두리로 판단할 수 있음

Convolution Layer

2차원 합성곱 (2D-Convolution)



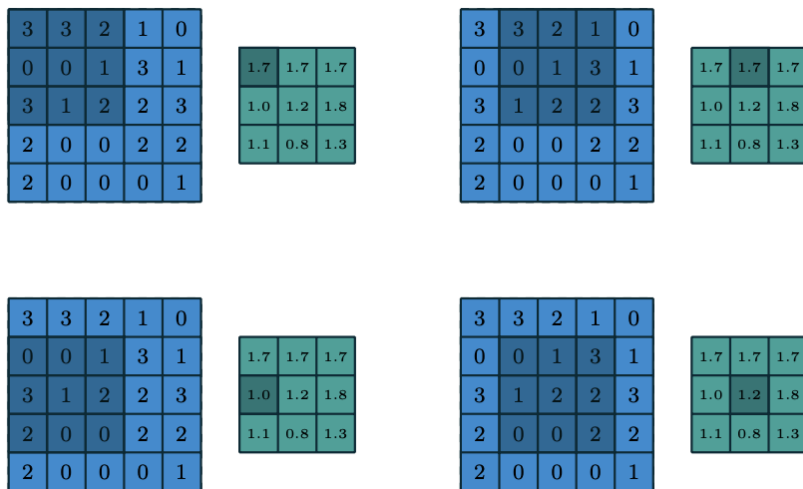
2차원 합성곱 정의

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n)$$

I : 이미지
 K : 커널

Convolution Layer

2차원 합성곱 (2D-Convolution)



2차원 합성곱 정의

m, n 에 대해 커널이 뒤집힘

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n)$$

I : 이미지

K : 커널



Convolution Layer CNN에서의 합성곱

2차원 합성곱 (2D-Convolution)

모델 초기화 과정에서 **가중치 배열**을

무작위로 초기화함

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

합성곱 연산 시에 커널을 뒤집는 것은 **큰 의미가 없음**

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

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

1.7	1.7	1.7
1.0	1.2	1.8
1.1	0.8	1.3

2차원 합성곱 정의

m, n 에 대해 커널이 뒤집힘

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n)$$

I : 이미지

K : 커널



Convolution Layer CNN에서의 합성곱

2차원 합성곱 (2D-Convolution)

모델 초기화 과정에서 **가중치 배열**을

0	0	1	3	1	1.7	1.7	1.7	0	0	1	3	1	1.7	1.7	1.7
3	1	2	2	3				3	1	2	2	3	1.0	1.2	1.8
2	0	0	2	2				2	0	0	2	2	1.1	0.8	1.3
2	0	0	0	1				2	0	0	0	1			

무작위로 초기화함

합성곱 연산 시에 커널을 뒤집는 것은 **큰 의미가 없음**

3	3	2	1	0	1.7	1.7	1.7	3	3	2	1	0	1.7	1.7	1.7
0	0	1	3	1				0	0	1	3	1	1.0	1.2	1.8
3	1	2	2	3	1.0	1.2	1.8	3	1	2	2	3	1.1	0.8	1.3
2	0	0	2	2	1.1	0.8	1.3	2	0	0	2	2			
2	0	0	0	1				2	0	0	0	1			



2차원 합성곱 정의

m, n 에 대해 커널이 뒤집힘

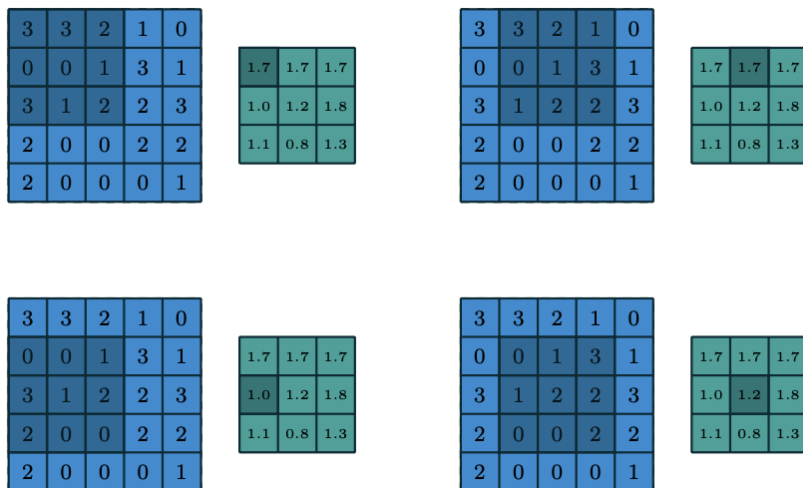
대부분의 라이브러리에서

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i-m, j-n)$$

I : 이미지
같은 위치의 성분끼리 곱하는 교차 상관 적용!

Convolution Layer

2차원 합성곱 (2D-Convolution)

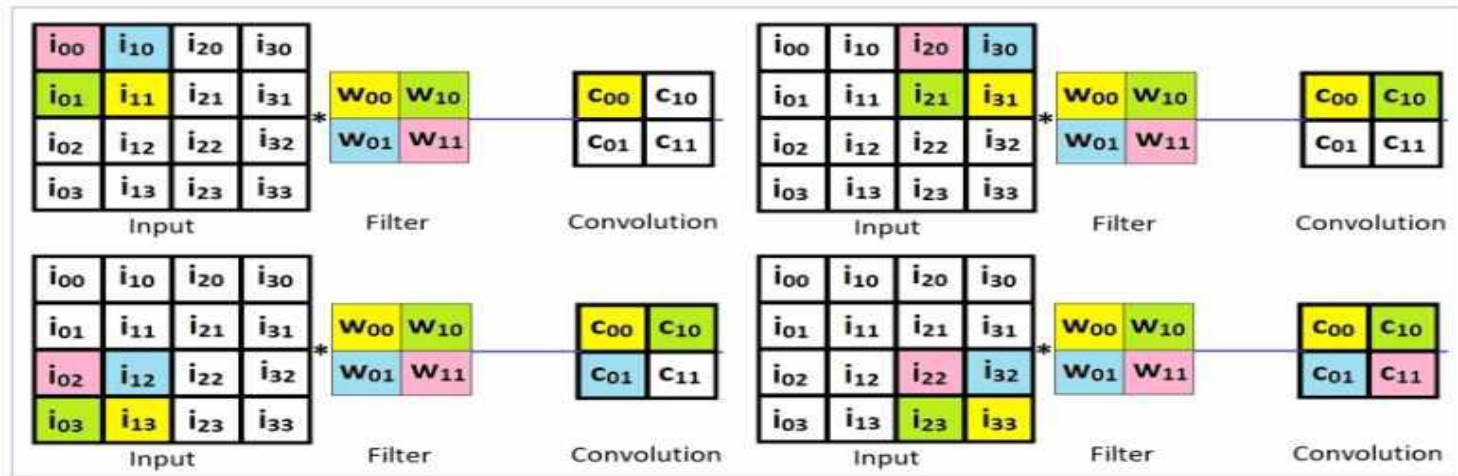


2차원 교차상관

i, j 에 대한 덧셈으로 바뀜

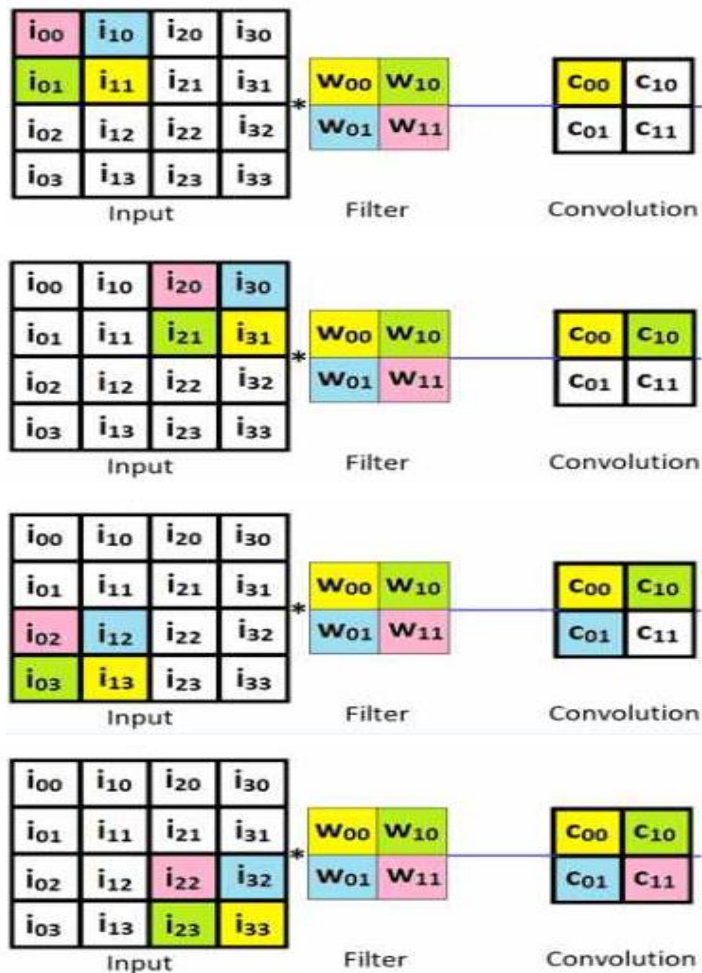
$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

Convolution Layer



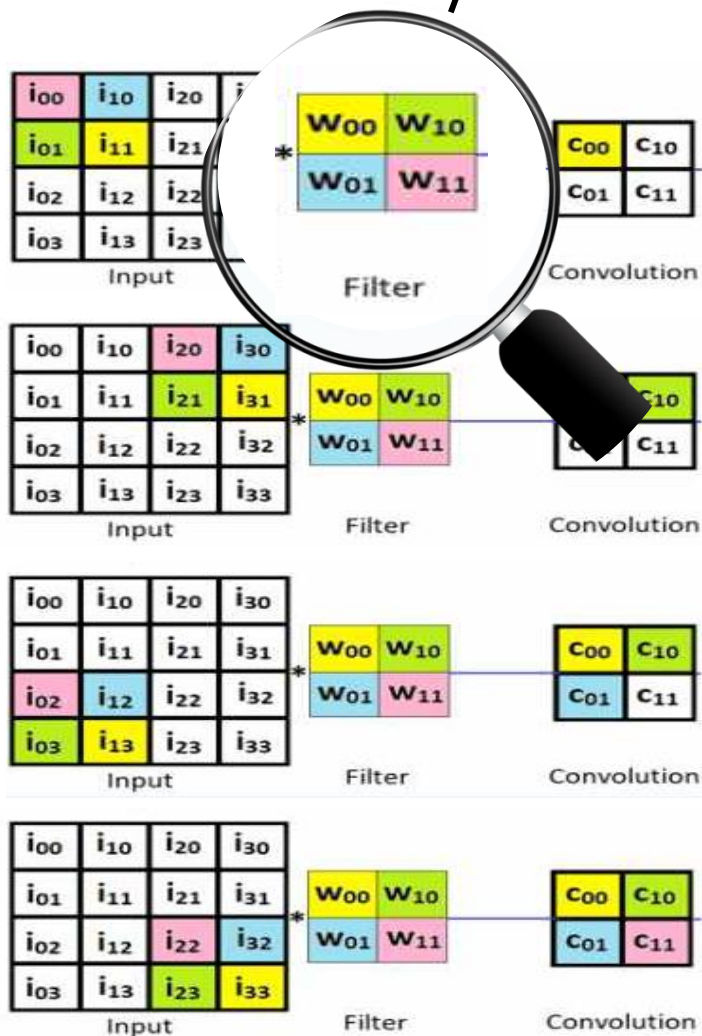
공간 정보를 보존하는 동시에
입력 데이터의 특징을 추출하는 역할

Convolution Layer



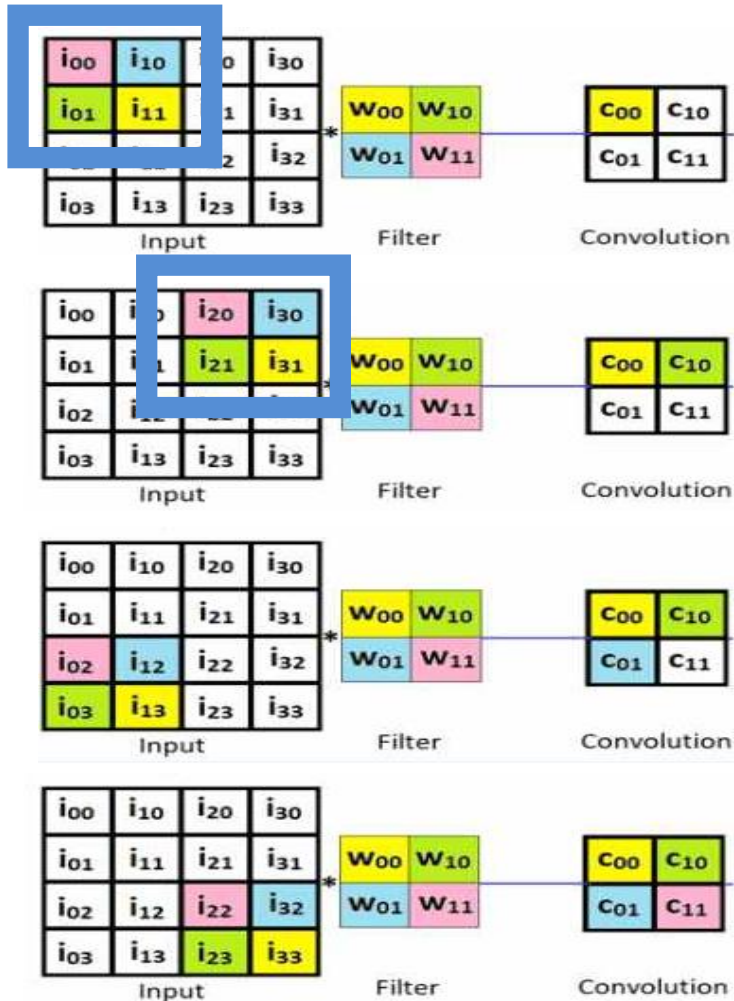
1. (2, 2) 크기의 필터를 설정
2. 두 칸 씩 움직이며
3. 모든 경우에 대해 합성곱 진행

Convolution Layer



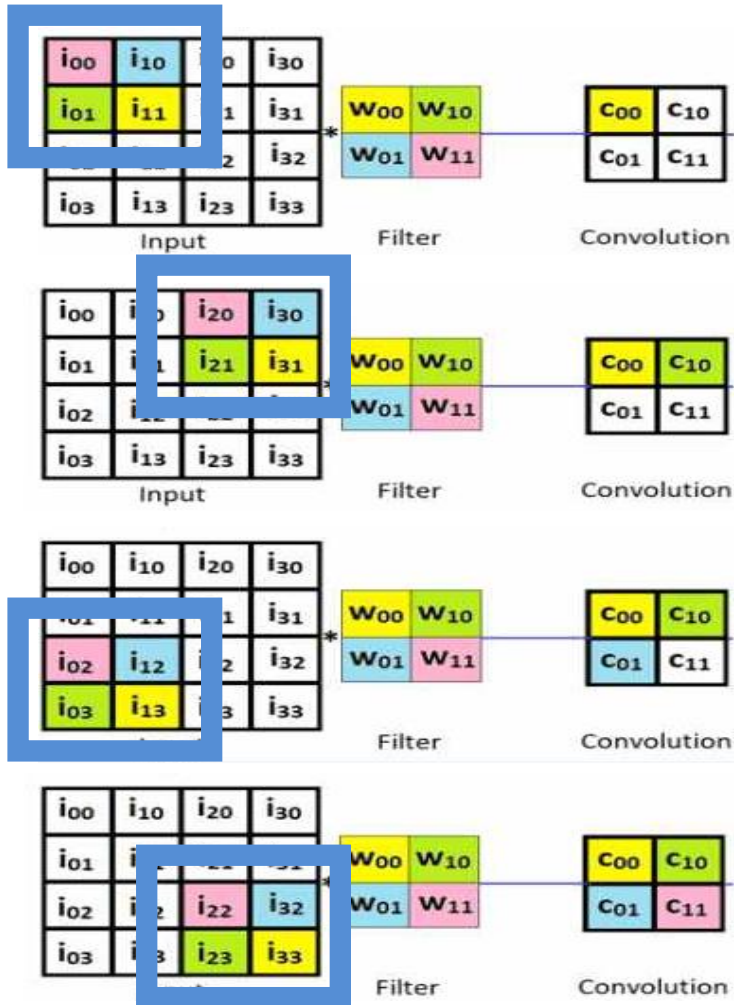
1. (2, 2) 크기의 필터를 설정
2. 두 칸 씩 움직이며
3. 모든 경우에 대해 합성곱 진행

Convolution Layer



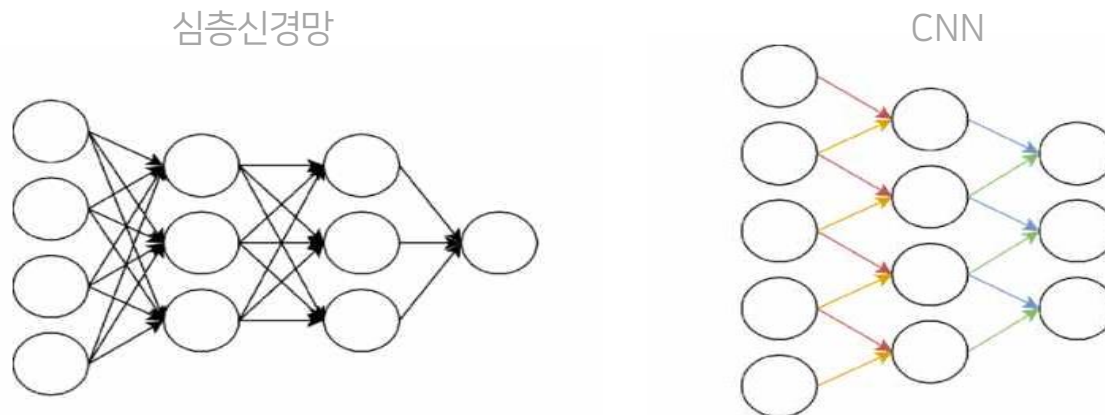
1. (2, 2) 크기의 필터를 설정
2. 두 칸 씩 움직이며
3. 모든 경우에 대해 합성곱 진행

Convolution Layer



1. (2, 2) 크기의 필터를 설정
2. 두 칸 씩 움직이며
3. 모든 경우에 대해 합성곱 진행

Convolution Layer



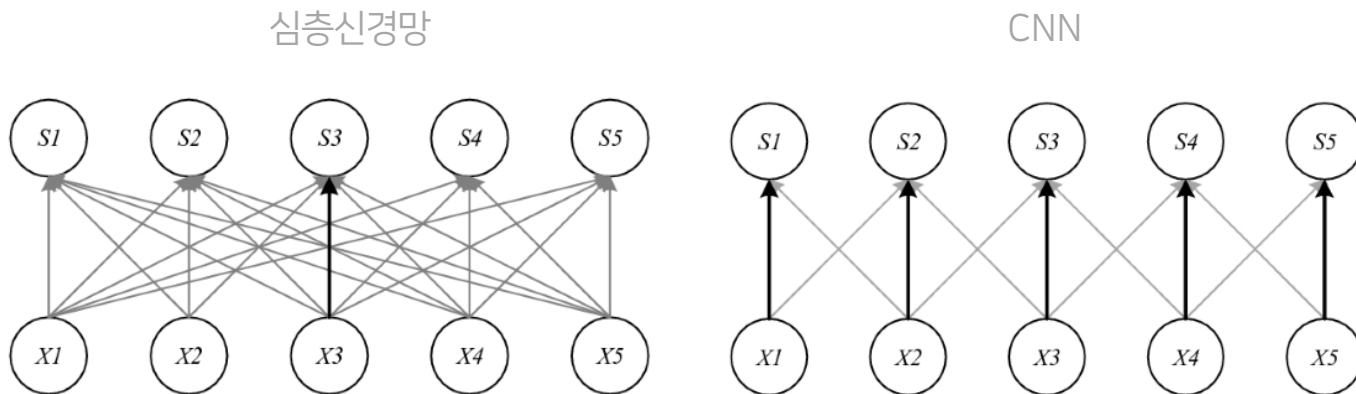
희소 상호작용 (Sparse interaction)

매개 변수들이 **완전 연결 되어있지 않은 것**을 의미

메모리 사용량 줄어듦고 통계적 효율성 높아지는 효과가 있음

연결되지 않는 부분은 가중치가 0으로 처리됨

Convolution Layer

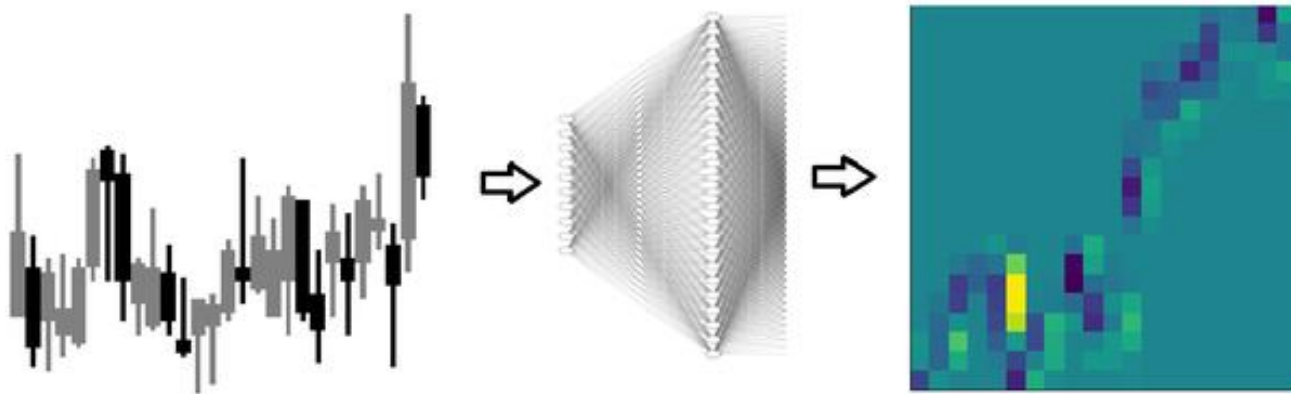


매개변수 공유 (Parameter Sharing)

필터가 이동하며 곱해질 때 **가중치 바뀌지 않는** 특성

가중치는 일반적으로 무작위 초기화

Convolution Layer



피쳐맵 (Feature Map)

Convolution Layer의 **출력**에 해당

Input/output channel, filter size, stride, padding 등으로 결정

Convolution Layer

Hyper parameter

Input Channel

입력 이미지 채널 개수
일반 이미지 3, 흑백 1

Output Channel

하나의 입력 데이터에
반환하는 채널 수

= Convolution Layer 필터의 개수

Filter Size

필터의 크기
필터 사이즈 ↑ feature map ↓

Stride

필터가 이동하는 간격
Stride ↑ feature map ↓

Convolution Layer

Hyper parameter

Input Channel

입력 이미지 채널 개수
일반 이미지 3, 흑백 1

Output Channel

하나의 입력 데이터에
반환하는 채널 수

= Convolution Layer 필터의 개수

Filter Size

필터의 크기
필터 사이즈 ↑ feature map ↓

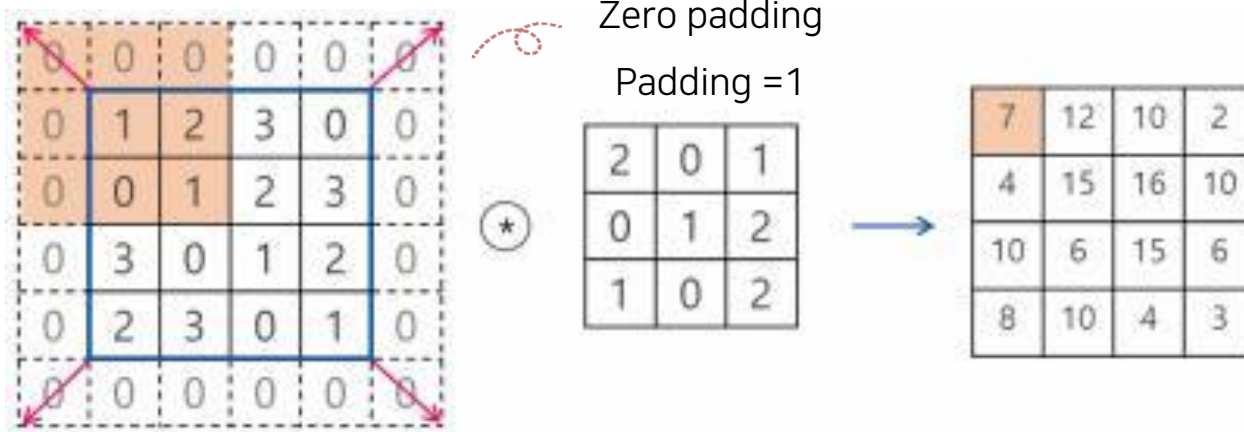
Stride

필터가 이동하는 간격
Stride ↑ feature map ↓

Convolution Layer

패딩 (Padding)

원본 데이터 주변에 새로운 값들을 더하여
가장자리의 값들도 필터를 여러 번 통과할 수 있게 해주는 방법



Convolution Layer

Zero padding

유효한(valid) 합성곱

Zero padding **아예 사용하지 않는 경우**

출력 이미지가 급격하게 작아짐

동일(same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 덜 반영되는 문제

완전(full) 합성곱

각 방향에서 **모든 픽셀이 k번 반복되도록**

충분히 많은 0들을 채움

Convolution Layer

Zero padding

유효한(valid) 합성곱

Zero padding **아예 사용하지 않는 경우**

출력 이미지가 급격하게 작아짐

동일(same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 덜 반영되는 문제

완전(full) 합성곱

각 방향에서 **모든 픽셀이 k번 반복되도록**

충분히 많은 0들을 채움

Convolution Layer

Zero padding

유효한(valid) 합성곱

Zero padding **아예 사용하지 않는 경우**

출력 이미지가 급격하게 작아짐

동일(same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 덜 반영되는 문제

완전(full) 합성곱

각 방향에서 **모든 픽셀이 k번 반복되도록**

충분히 많은 0들을 채움

Convolution Layer

Zero padding

유효한(valid) 합성곱

Zero padding **아예 사용하지 않는 경우**

출력 이미지가 급격하게 작아짐

동일(same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 덜 반영되는 문제

완전(full) 합성곱

각 방향에서 **모든 픽셀이 k번 반복되도록**

충분히 많은 0들을 채움

Convolution Layer

Zero padding

유효한(valid) 합성곱

Zero padding **아예 사용하지 않는 경우**

출력 이미지가 급격하게 작아짐



일반적으로 유효한 합성곱과
동일 합성곱 사이의 개수 사용

완전(full) 합성곱

각 방향에서 **모든 픽셀이 k번 반복되도록**

충분히 많은 0들을 채움

Convolution Layer

Feature map 크기

$$O_n = \frac{I_n + 2P - F}{S} + 1$$



Input 사이즈에 대한 output 사이즈를 계산

O_n = 출력 가로 길이

I_n = 입력 가로 길이

P = padding 크기

F = filter 크기

S = stride 크기

Pooling Layer

29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

100	184
12	45

Max Pooling :

필터가 덮고 있는 값 중 가장 큰 값

36	80
12	15

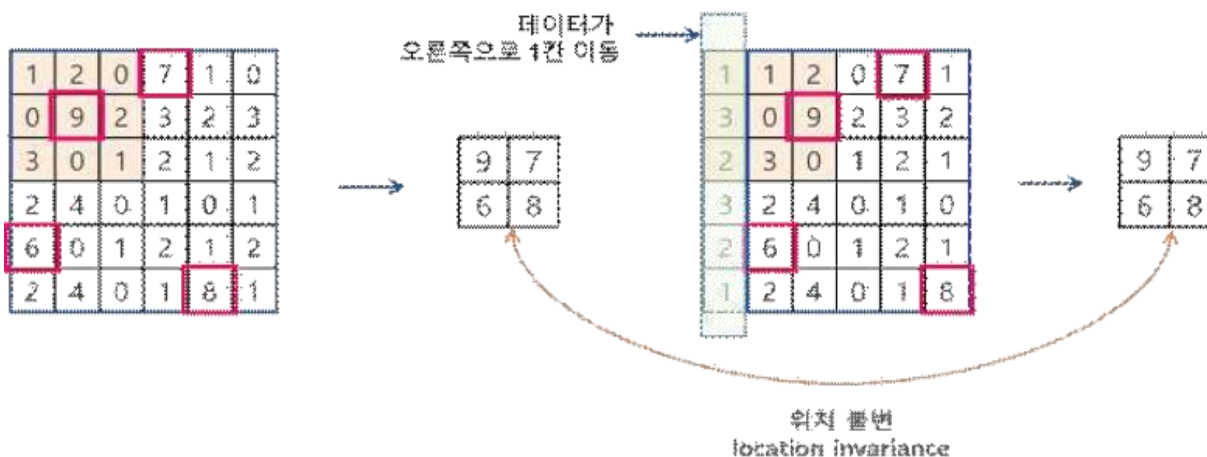
Average Pooling:

필터가 덮고 있는 값의 평균

가중치를 사용하지 않고 입력 이미지의 중요한 특징들 강조함
데이터 크기 줄이는 것 중점

Pooling Layer

Max/Min pooling

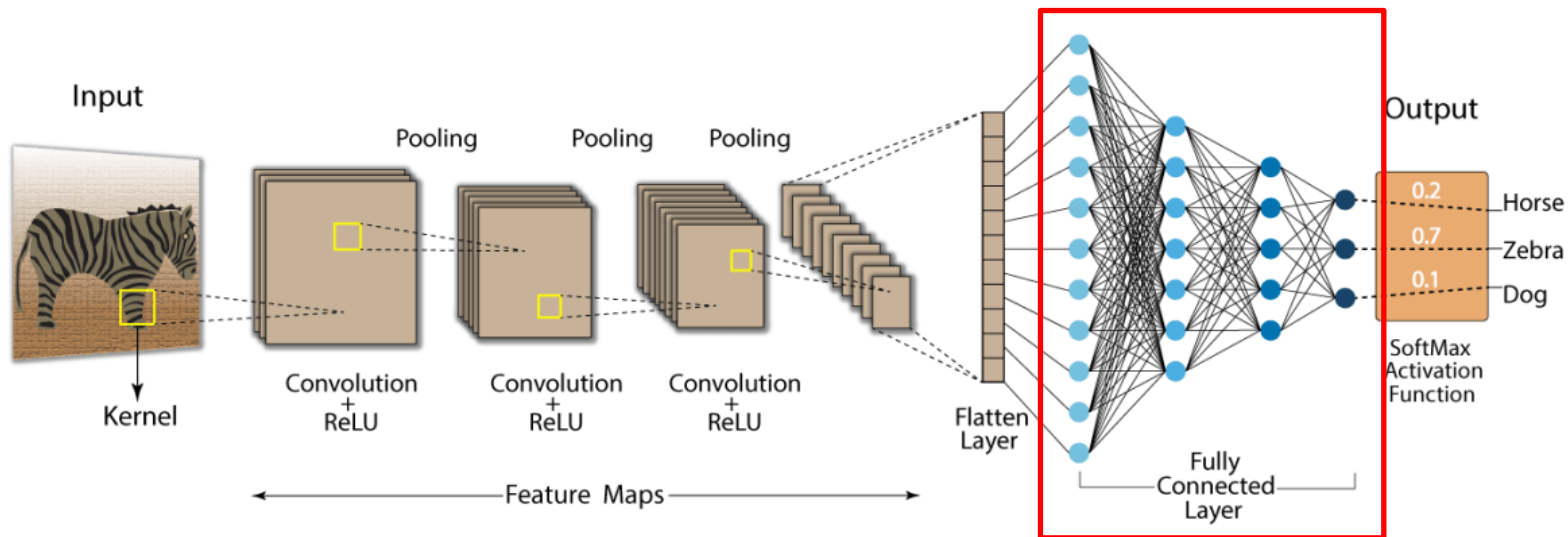


위치불변 (Location invariance)

데이터가 이동해도 Pooling Layer의 **결과값이 바뀌지 않음**
이런 특성으로 일반적으로 Max pooling을 이용함

Min pooling 일 경우 0을 반환할 가능성이 높아 잘 사용하지 않음

Fully connected Layer



2차원 이상 데이터에 적용 불가

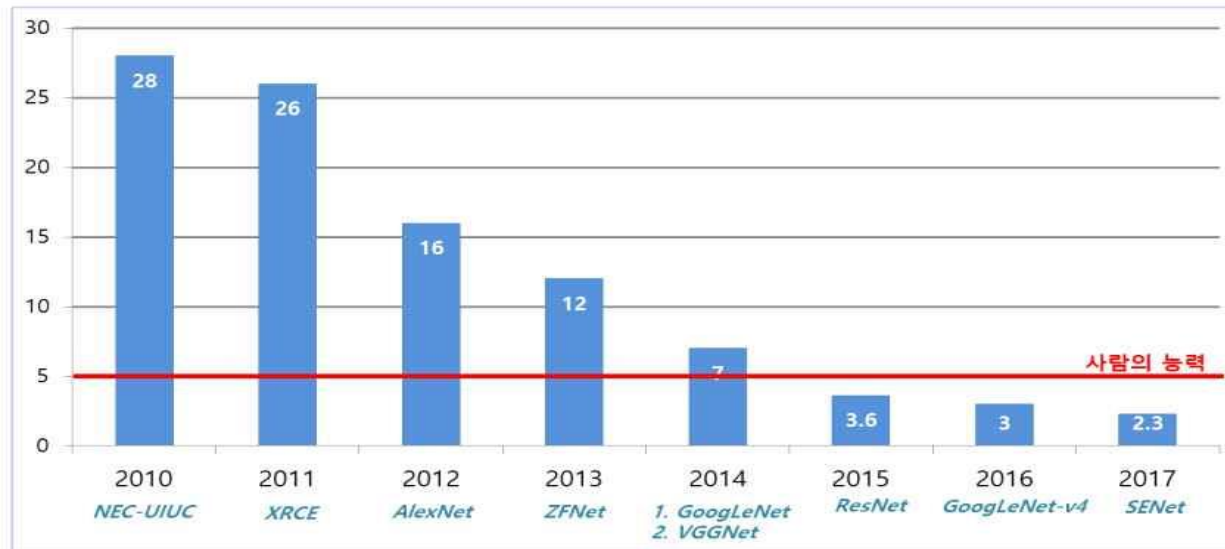
최종 단계에서 라벨 예측을 진행하는 layer
최종 feature map을 **벡터로 전환**하여 입력으로 넣음

3

CNN의 발전과정

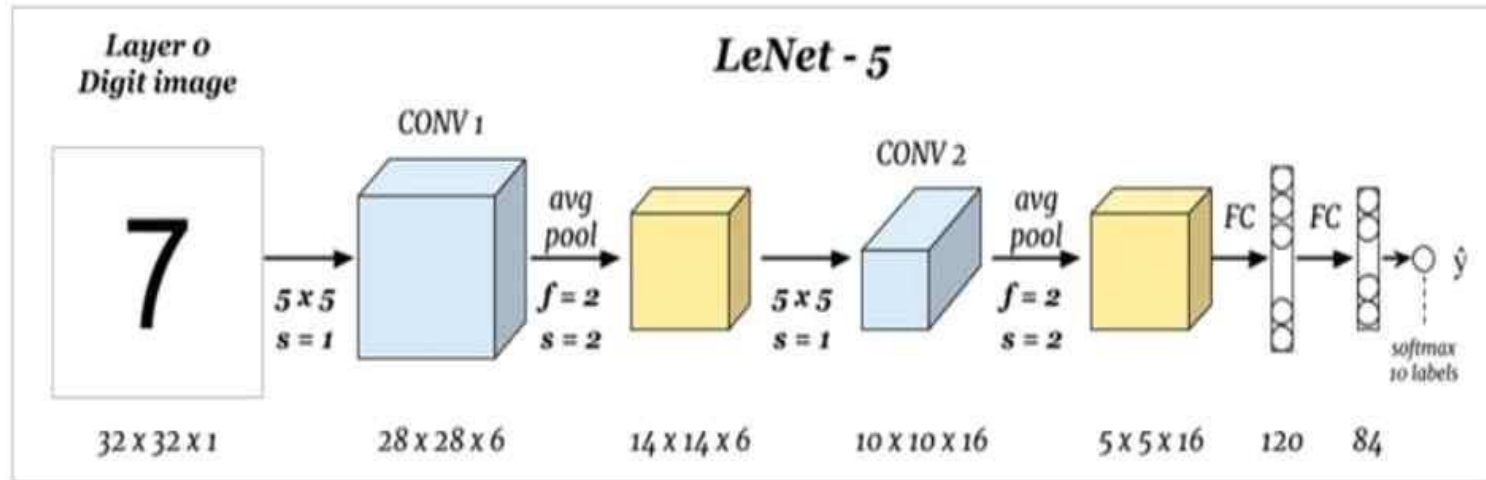
ILSVRC

대용량 이미지 데이터셋 ImageNet을 이용한 이미지 분류 문제 해결 대회



ResNet 부터,
사람의 오류율보다 인공지능 모델의 오류율이 더 낮아짐

LeNet-5



얀 르쿤 교수가 1998년 가장 처음 제안한 CNN 손글씨 분류 모델

LeNet-5

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

(1, 32, 32) 입력 이미지가 3개의 Convolution Layer와 2개의 Pooling Layer를 교차하여 통과한 후 120으로 축소됨

LeNet-5

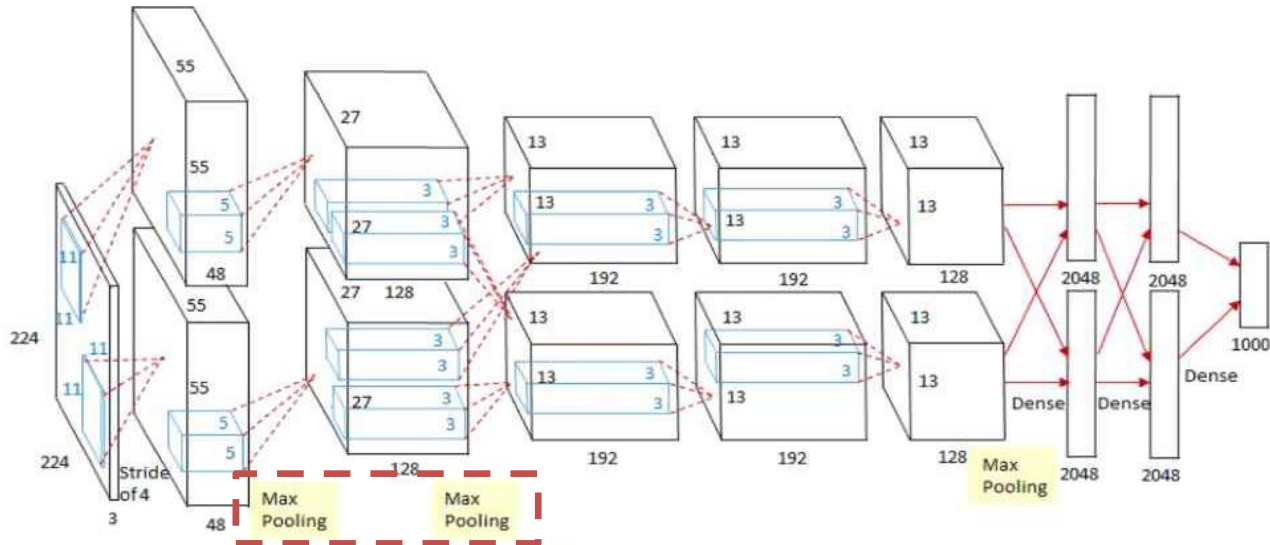
Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

이후 FC Layer를 통과하여 10종류 손글씨에 대해 분류를 진행

3

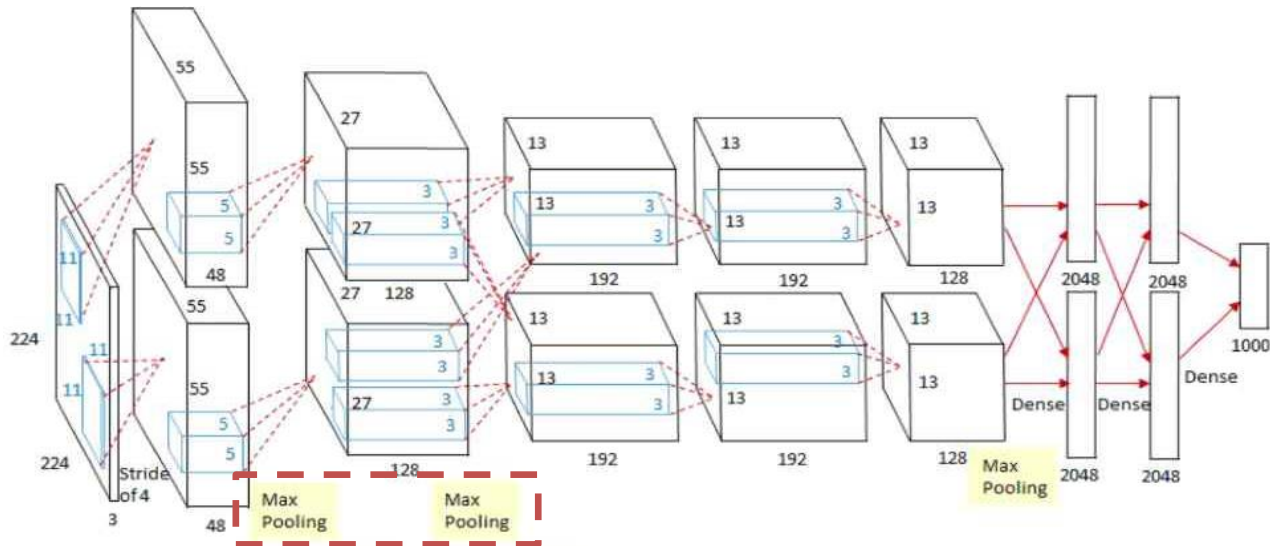
CNN의 발전 과정

AlexNet



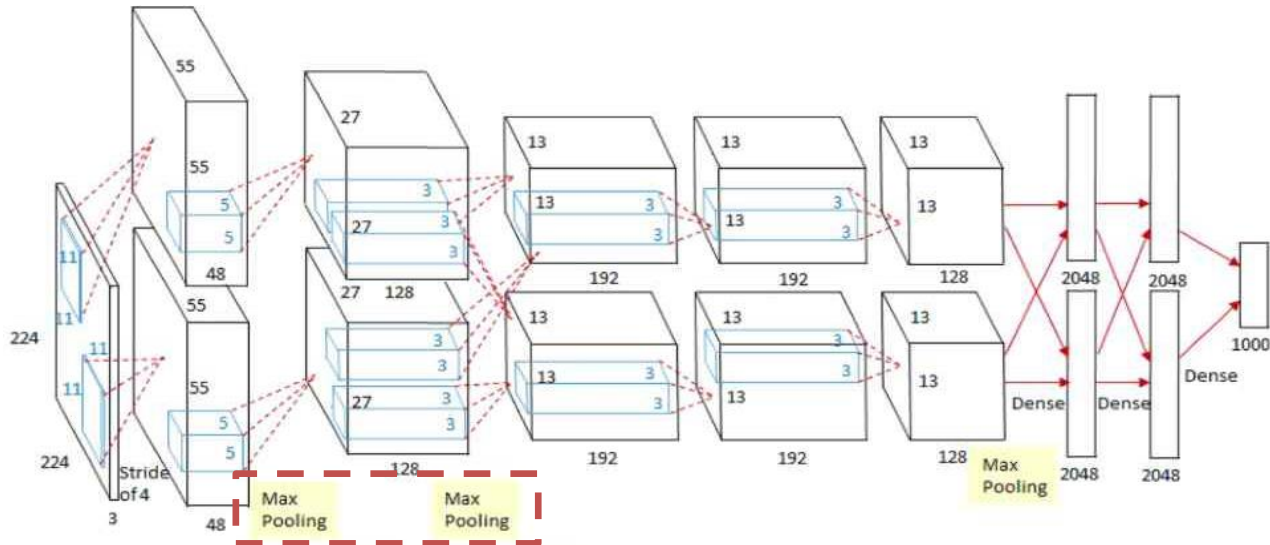
AlexNet은 LeNet-5과 달리 (3,224,224)의 컬러 이미지를 사용
입력 데이터의 크기가 크게 증가하여 많은 연산이 필요

AlexNet



ReLU 함수 와 Max Pooling 사용하여 연산을 줄이고 특징 효과적 추출
또한 여러 기법을 이용하여 성능 향상시킴

AlexNet



ReLU 함수 와 Max Pooling 사용하여 연산을 줄이고 특징 효과적 추출
또한 여러 기법을 이용하여 성능 향상시킴

배치 정규화, Dropout 등등 1주차 클린업 참고

AlexNet

Data Augmentation

하나의 이미지를 사용해 좌우반전, 회전 등의 방법으로
여러 비슷한 이미지를 만드는 **데이터 증강 방법**으로 과적합 방지 효과

Original



Rotation



Flip



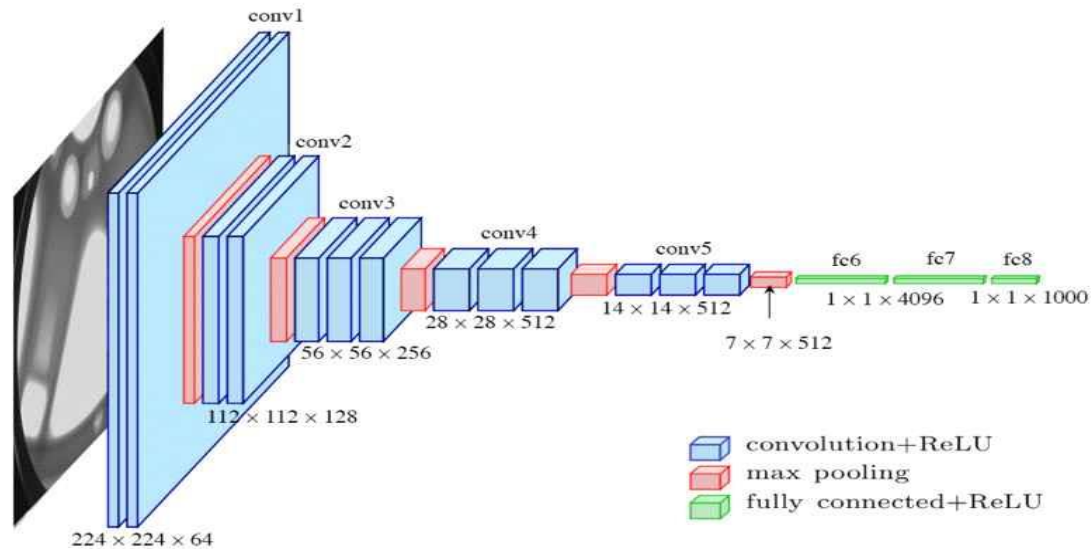
Scaling



Brightness



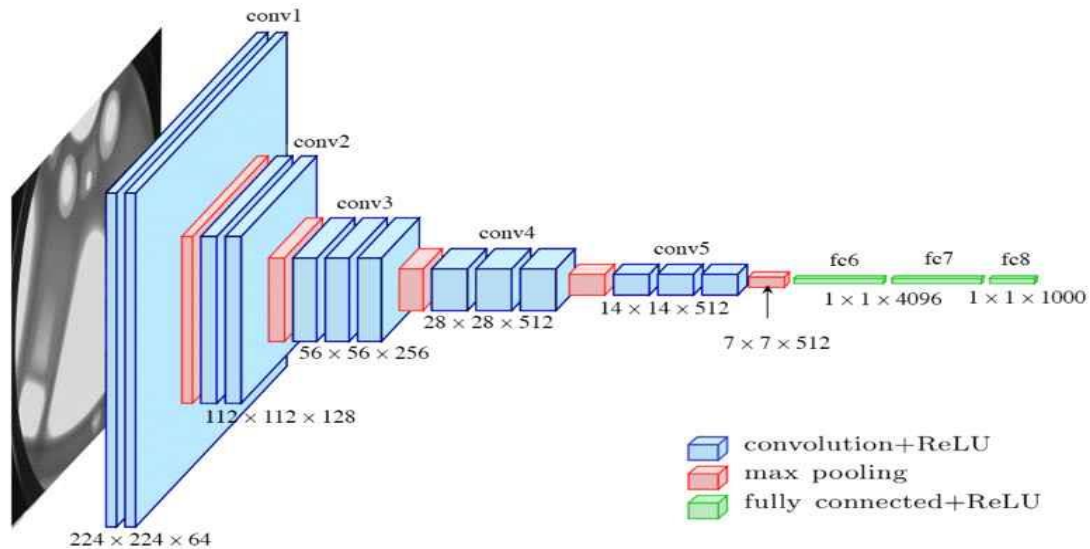
VGGNet



간단한 구조임에도 좋은 결과를 보인 모델

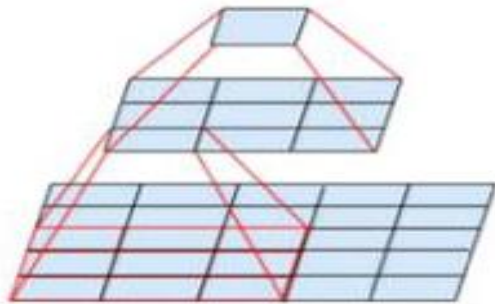
CNN에서 layer를 더 깊게 많이 쌓으면 성능을 향상시킬 수 있음을 보여줌

VGGNet



필터의 사이즈가 줄어드는 AlexNet 과 달리
VGGNet은 더 작은 사이즈인 (3,3) 필터를 처음부터 끝까지 사용함

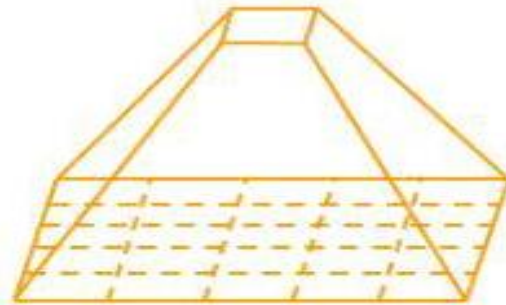
VGGNet



두번의

(3, 3) Convolution layer

비선형성 2번 추가



한번의

(5, 5) Convolution layer

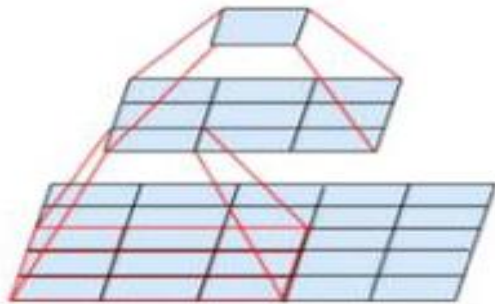
비선형성 1번 추가



양쪽 모두 입력 데이터를 1X1으로 반환함

왼쪽이 "층의 중첩을 통한 비선형성의 추가" 라는 목적에 더 부합

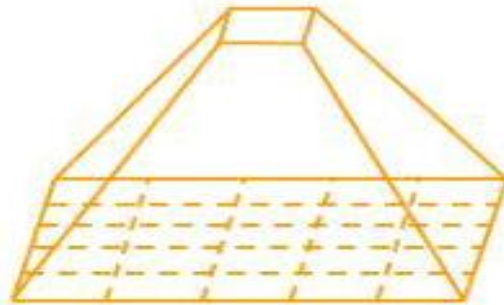
VGGNet



두번의

(3, 3) Convolution layer

파라미터 수 = $2 \times 3^2 \times C$



한번의

(5, 5) Convolution layer

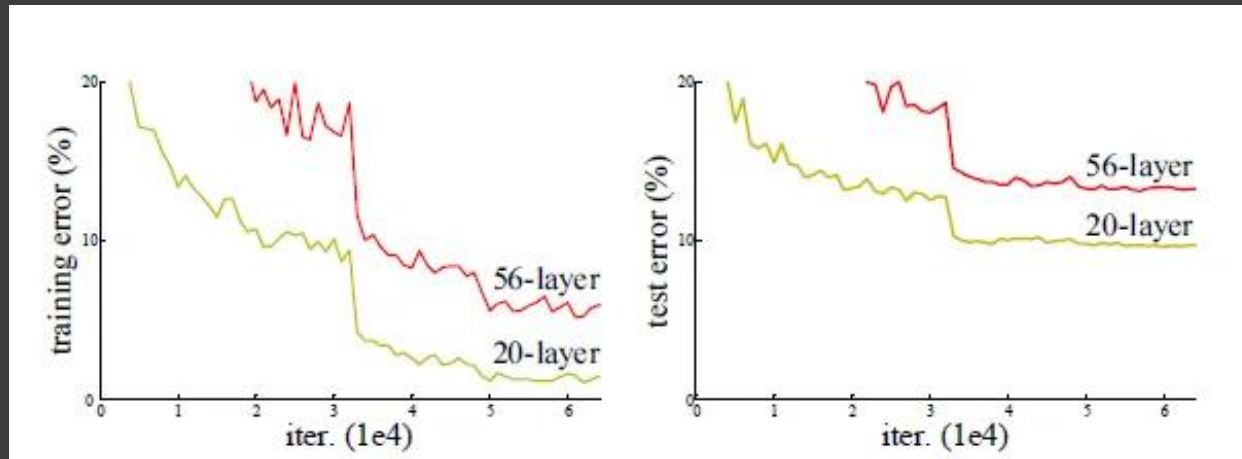
파라미터 수 = $5^2 \times C$

파라미터 수 또한 **왼쪽이 더 적은 것**을 확인 할 수 있음



VGGNet

층이 깊을수록 무조건 좋을까?



VGGNet의 Layer 개수에 따른 성능차이 그래프

(3, 3) Convolution layer

$$\text{파라미터 수} = 2 \times 3^2 \times C$$

(5, 5) Convolution layer

$$\text{파라미터 수} = 2 \times 3^2 \times C$$



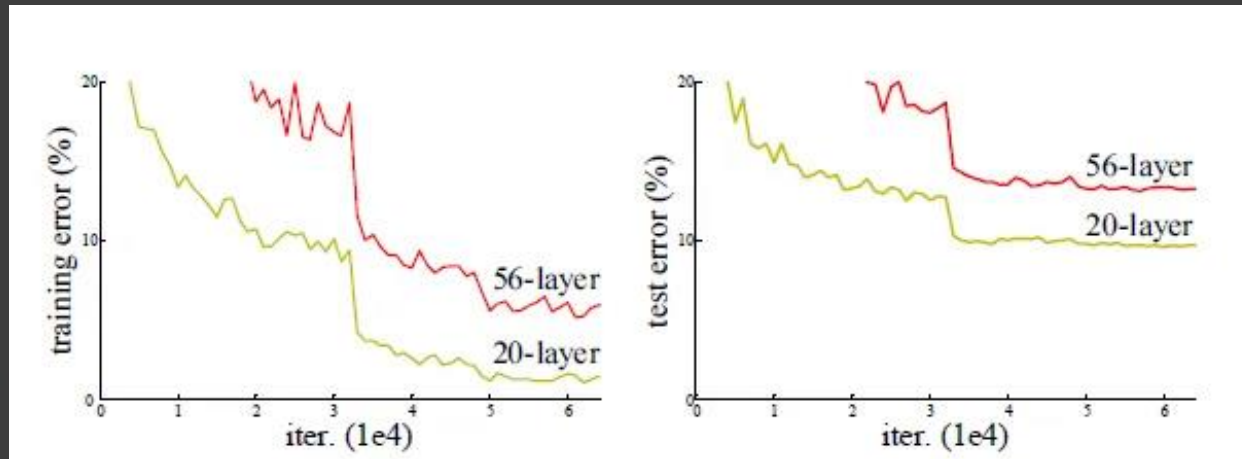
층이 과하게 많아지면 **train/test error가 모두 증가**

파라미터 수 또한 왼쪽이 더 적은 것을 확인 할 수 있음



VGGNet

층이 깊을수록 무조건 좋을까?



VGGNet의 Layer 개수에 따른 성능차이 그래프

(3, 3) Convolution layer

$$\text{파라미터 수} = 2 \times 3^2 \times C$$

(5, 5) Convolution layer

$$\text{파라미터 수} = 2 \times 3^2 \times C$$

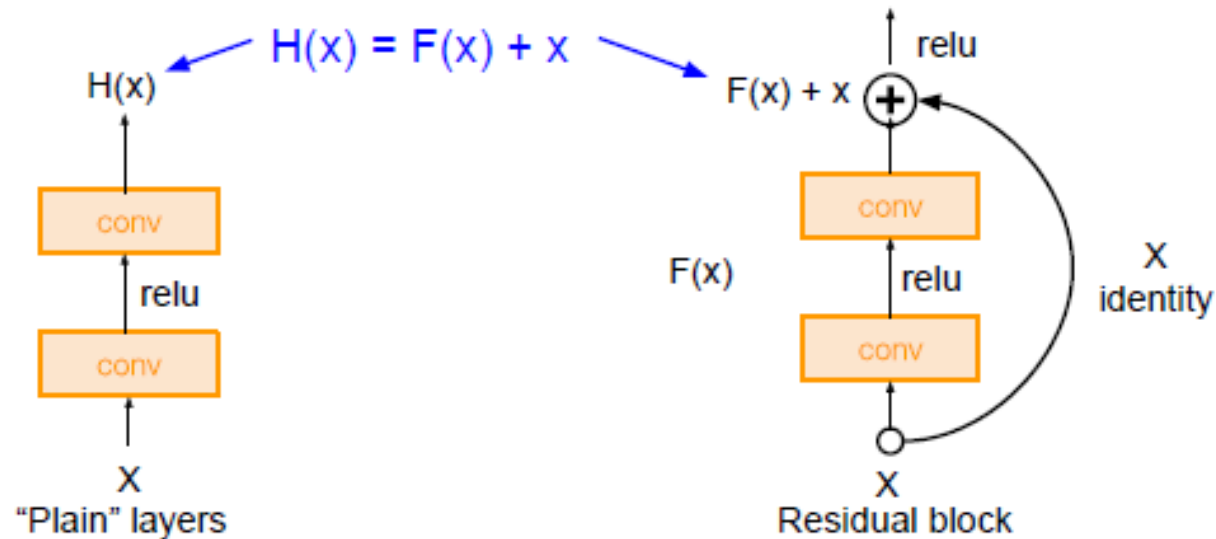
층이 과하게 많아지면 **train/test error가 모두 증가**

파라미터 수 또한 왼쪽이 더 적은 것을 확인 할 수 있음



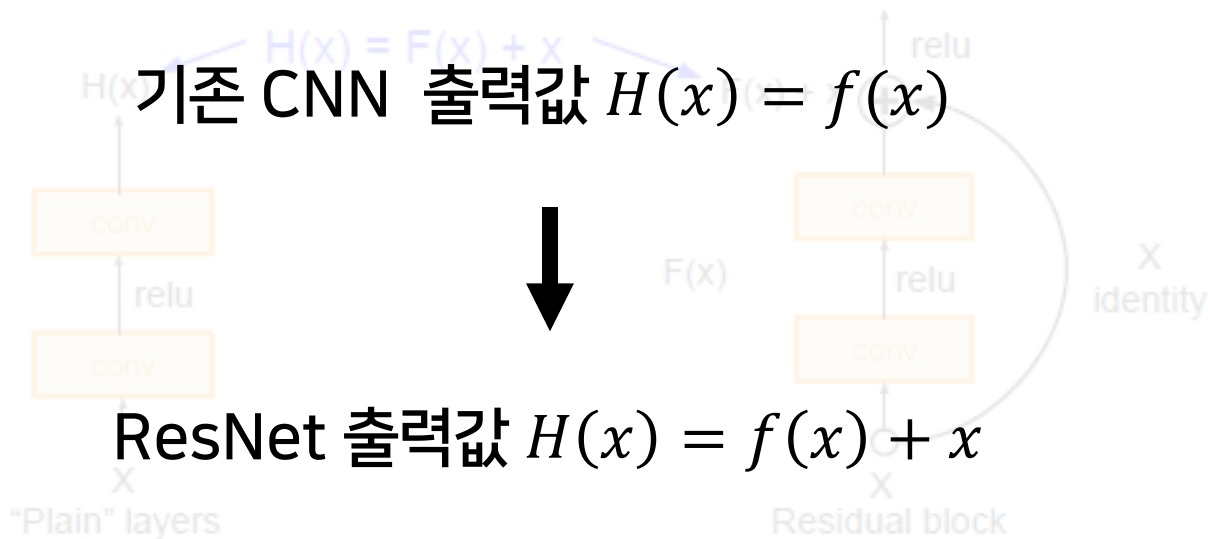
과적합보다는 최적화(기울기소실)의 문제!!

ResNet



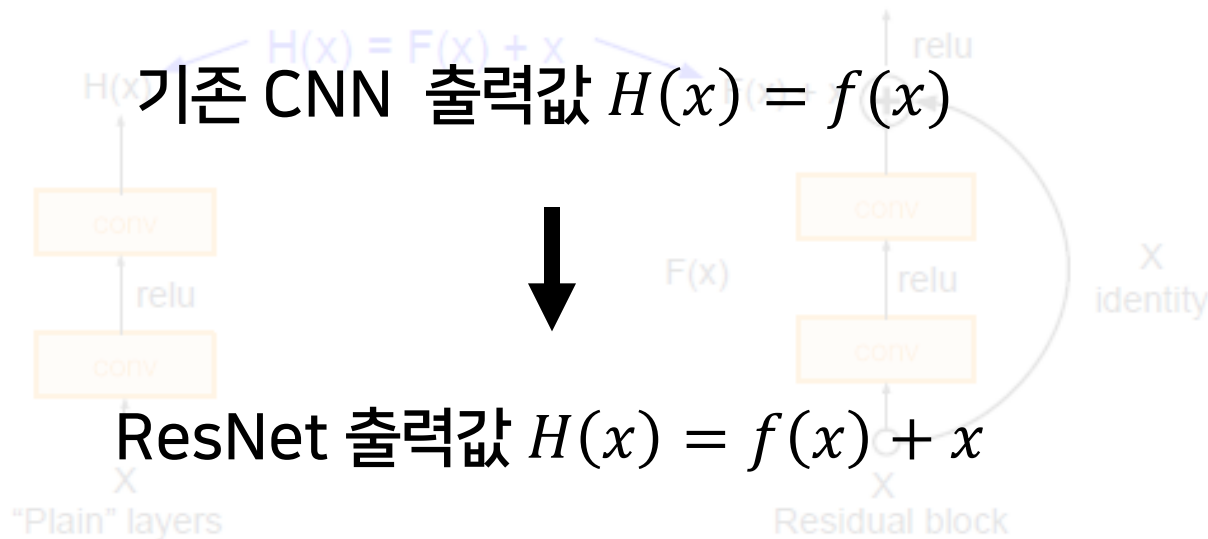
최적화 문제를 해결하기 위해 Residual Learning 고안

ResNet



이전의 CNN과 달리 ResNet은 **이전 layer의 출력값을 더해줌**
 이 때문에 Conv에서 추출한 특징과, 원본 이미지의 특징을 **동시에 반영**

ResNet



이전의 CNN과 달리 ResNet은 이전 layer의 출력값을 더해줌
 이 때문에 잔차와 같은 residual 형태로 변형 가능

Residual learning의 이유

$$f(x) = H(x) - x = y - \hat{y}$$

ResNet

$$\frac{\partial H(x)}{\partial x} = \frac{d}{dx} (F(x) + x) = F'(x) + 1$$

역전파 시 Gradient를 확인해보면 이전 출력에서 학습되지 못한 $F(x)$ 를 최적화하는 방향으로 학습이 진행됨

ResNet

$$\frac{\partial H(x)}{\partial x} = \frac{d}{dx} (F(x) + x) = F'(x) + 1$$

기울기 소실 문제 예방하여
레이어 깊게 쌓을 수 있음!!

역전파 시 Gradient를 확인해보면 이전 출력에서 학습되지 못한
 $F(x)$ 를 최적화하는 방향으로 학습이 진행됨

SENet

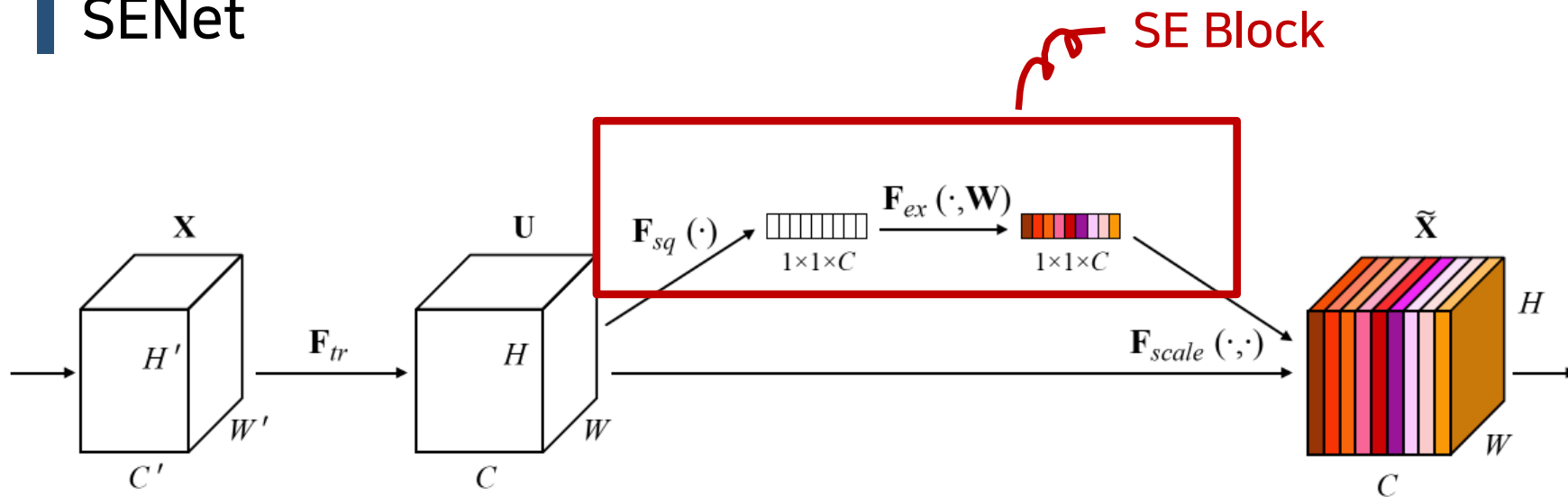


Figure 1: A Squeeze-and-Excitation block.

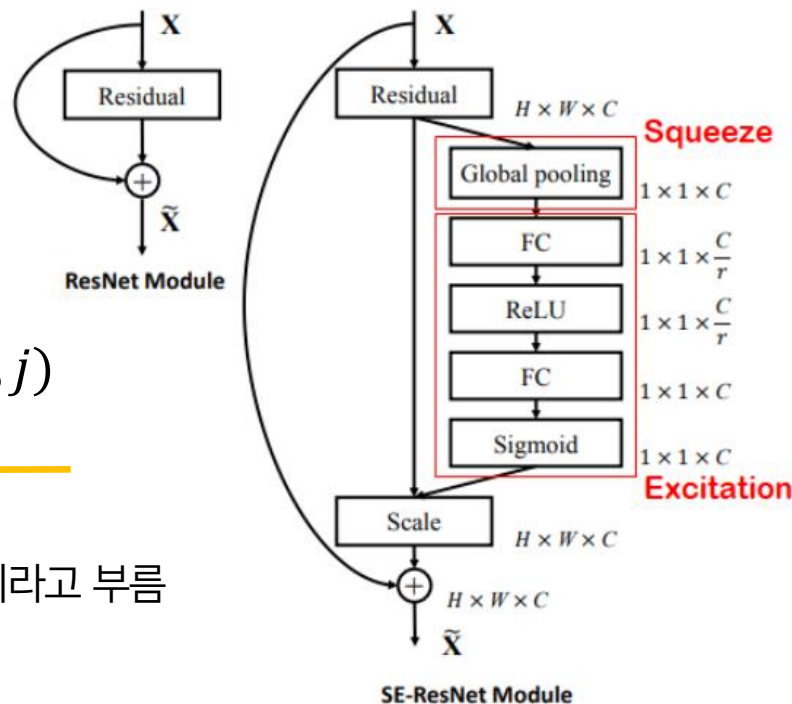
SE Block을 이용하여 채널 간의 상호작용에 집중하여 성능을 높인 모델
SE Block은 CNN의 어떤 모델에도 사용할 수 있음

SENet

$$F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j)$$



Global Average Pooling이라고 부름



Squeeze

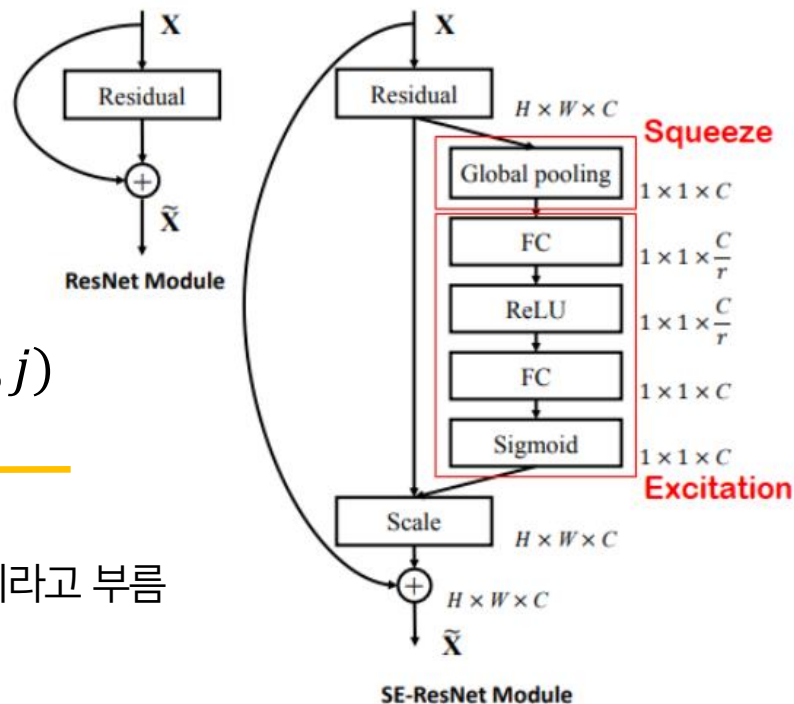
각 채널을 1차원으로 만드는 역할

Feature Map을 다 연결하면 $1 \times 1 \times C$ 크기가 됨

SENet

$$F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j)$$

Global Average Pooling이라고 부름



Excitation

Squeeze의 출력인 $1 \times 1 \times C$ 벡터를 정규화하여 가중치를 부여
4개의 층으로 구성되어 있음

SENet

FC1: $1 \times 1 \times C$ 벡터 입력 $\rightarrow C$ 채널을 C/r 채널로 축소

ReLU: $1 \times 1 \times C/r$ 벡터 전달

$$F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j)$$

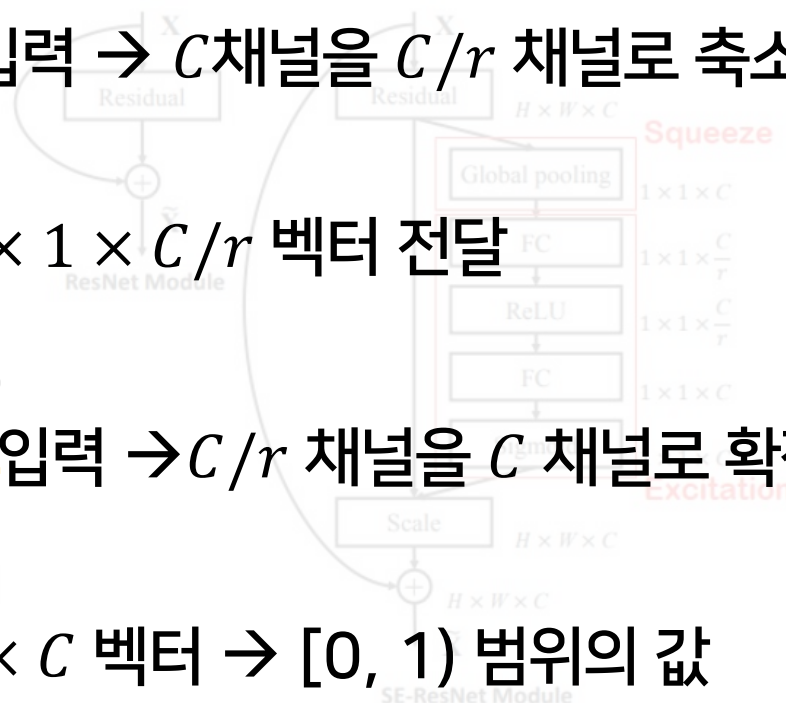
FC2: $1 \times 1 \times C/r$ 벡터 입력 $\rightarrow C/r$ 채널을 C 채널로 확장

Global Average Pooling

Sigmoid: $1 \times 1 \times C$ 벡터 $\rightarrow [0, 1)$ 범위의 값

Excitation

Squeeze의 출력인 $1 \times 1 \times C$ 벡터를 정규화하여 가중치를 부여
Feature Map과 곱해서 채널에 가중치 부여



SENet

FC1: $1 \times 1 \times C$ 벡터 입력 $\rightarrow C$ 채널을 C/r 채널로 축소

ReLU: $1 \times 1 \times C/r$ 벡터 전달

$$F_{sq}(u_c) = \frac{1}{H \times W} \sum_i \sum_j u_c(i, j)$$

FC2: $1 \times 1 \times C/r$ 벡터 입력 $\rightarrow C/r$ 채널을 C 채널로 확장

Global Average Pooling

Sigmoid: $1 \times 1 \times C$ 벡터 $\rightarrow [0, 1)$ 범위의 값

Excitation

각 채널의 중요도를 의미

Squeeze의 출력인 $1 \times 1 \times C$ 벡터를 정규화하여 가중치를 부여
Feature Map과 곱해서 채널에 가중치 부여

SENet

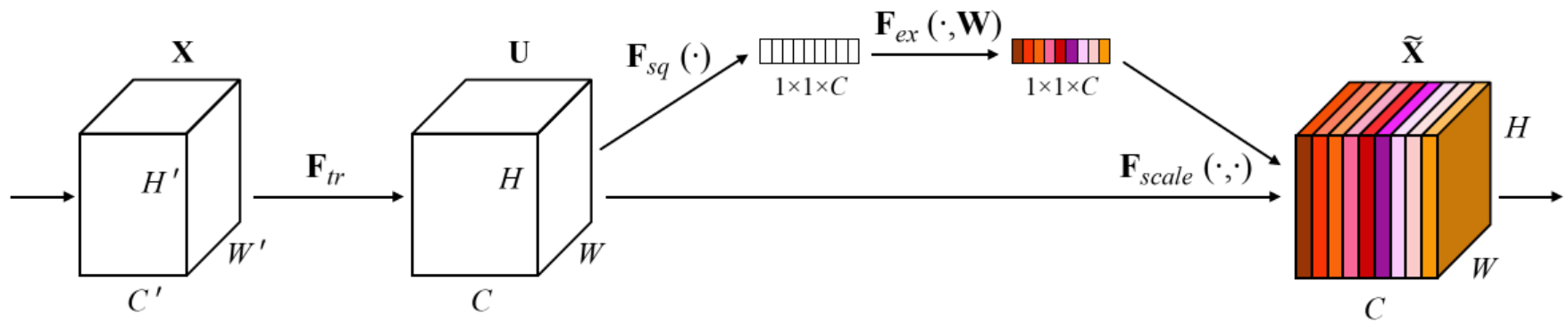


Figure 1: A Squeeze-and-Excitation block.

기존의 CNN 모델에 SE Block을 추가하면
연산량을 많이 증가시키지 않고 성능은 많이 향상 시킬 수 있음

4

컴퓨터 비전

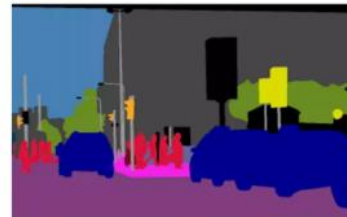
컴퓨터 비전 (Computer Vision)

컴퓨터 비전

인공지능 분야 중 컴퓨터가 시각적인 체계를
이해하고 해석할 수 있도록 컴퓨터를 학습시키는 연구 분야



Image



Semantic segmentation



Object detection & segmentation



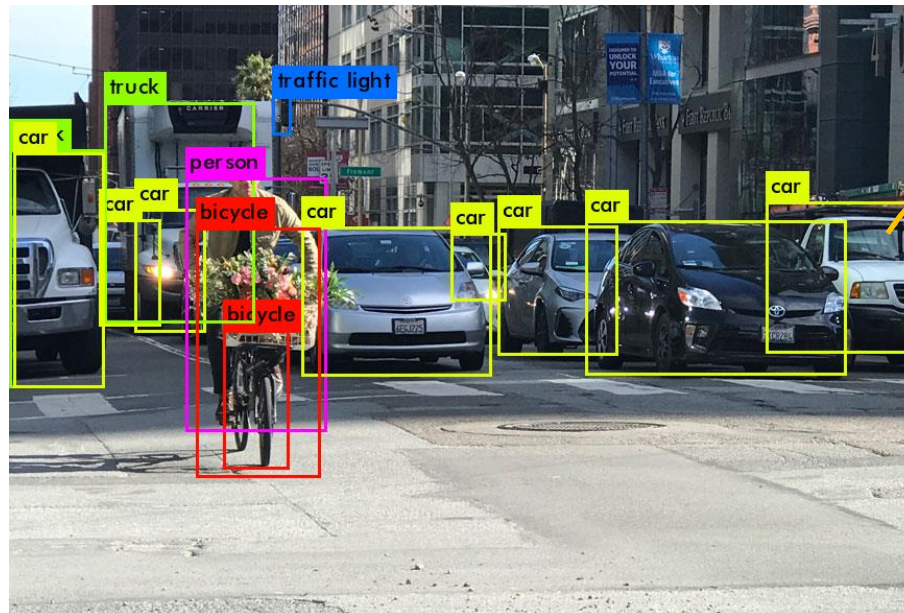
Panoptic segmentation

컴퓨터 비전 (Computer Vision)

객체 탐지 (Objective detection)

이미지가 주어졌을 때, 어떤 물체가 어디에 있는지 탐지하는 과제

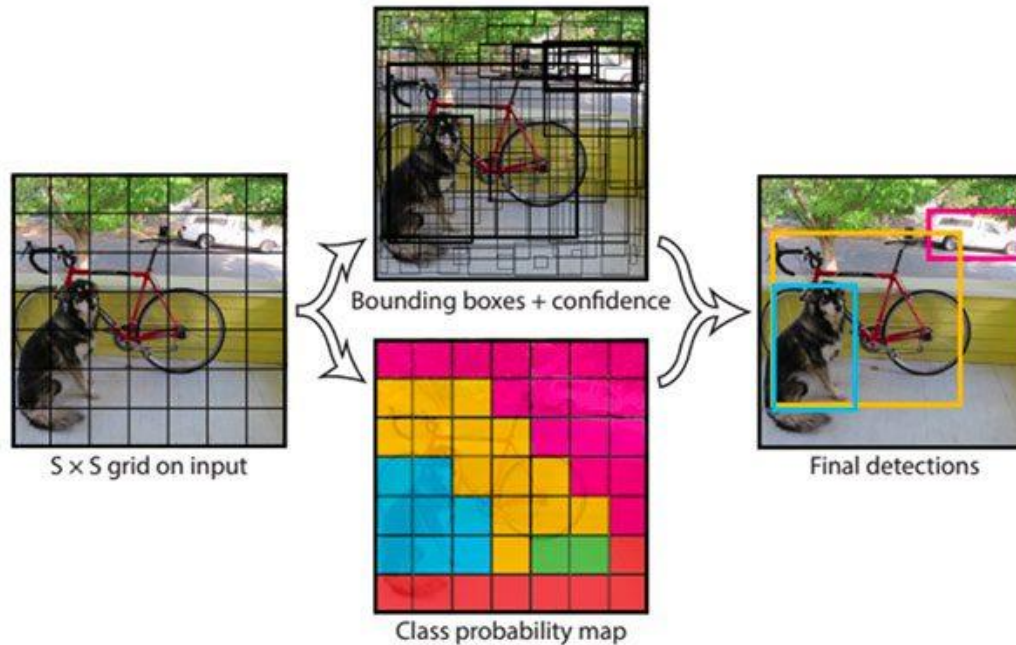
Localization과 Classification 과정으로 구성



객체의 위치를
나타내는 경계 상자
(Bounding Box/bbox)

Objection Detection

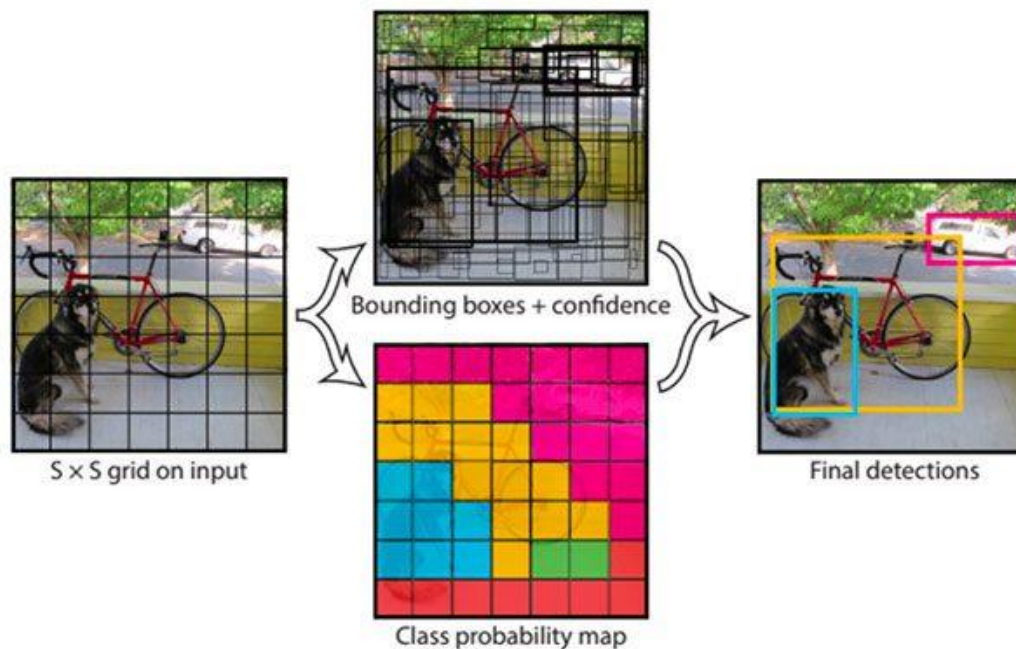
1-Stage Detector : YOLO



Localization과 Classification을 동시에 수행
속도는 빠르지만 상대적으로 정확도가 떨어짐

Objection Detection

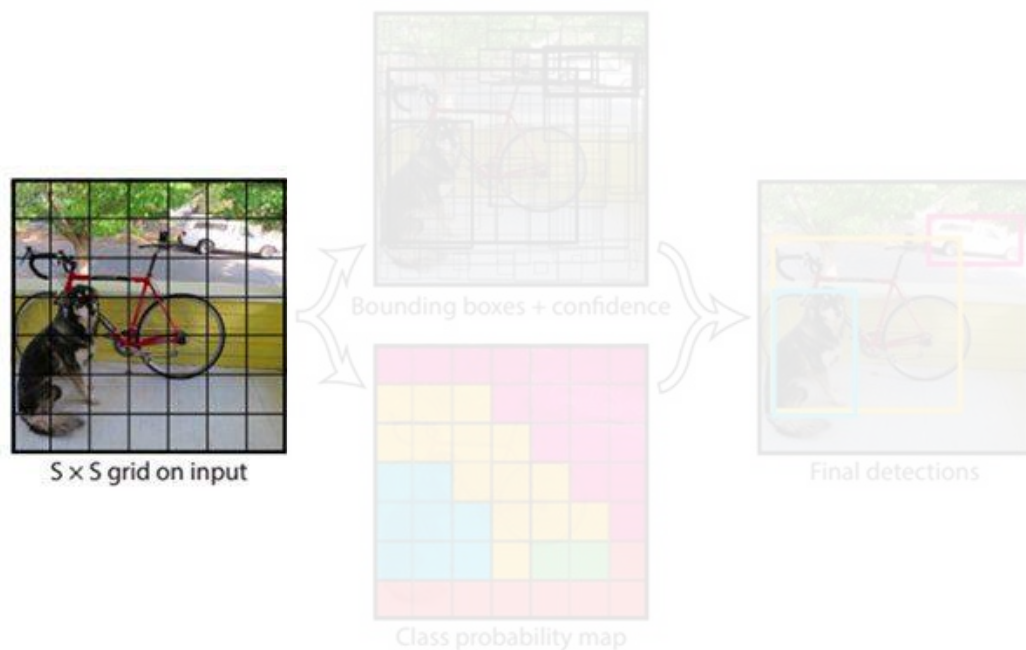
1-Stage Detector : YOLO



하나의 모델이 한번의 연산을 통해
이미지 전체의 bbox와 개체의 라벨 확률 동시에 반환

Object Detection

1-Stage Detector : YOLO

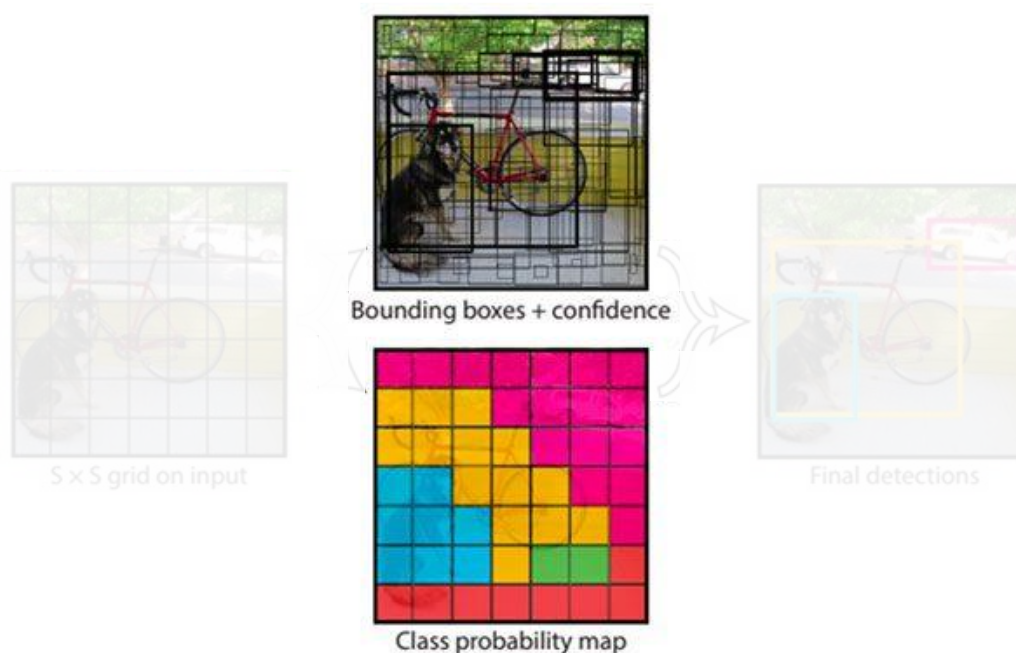


1단계: 원본 이미지를 동일한 크기의 그리드로 분할

이미지 전체의 bbox와 개체의 라벨 확률 동시에 반환

Objection Detection

1-Stage Detector : YOLO



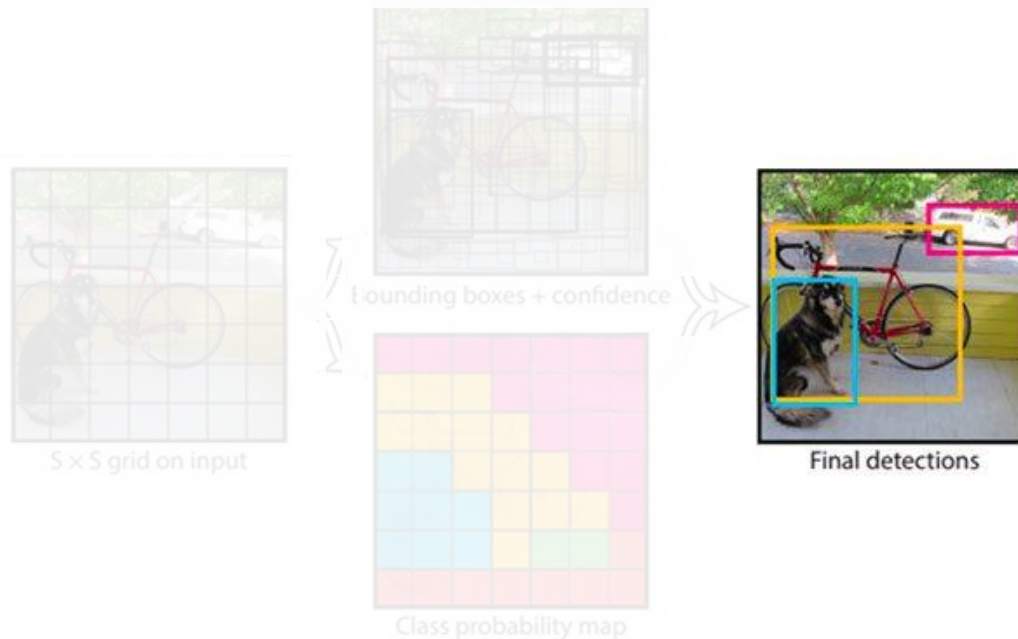
2단계: 각 그리드에 대해 bbox와 Confidence Score 계산
+ 확률이 일정 값 이상이 되는 셀을 서로 연결

이미지 전체의 bbox와 개체의 라벨 확률 동시에 만족하는 작은 개체의

인식률이 떨어짐

Objection Detection

1-Stage Detector : YOLO

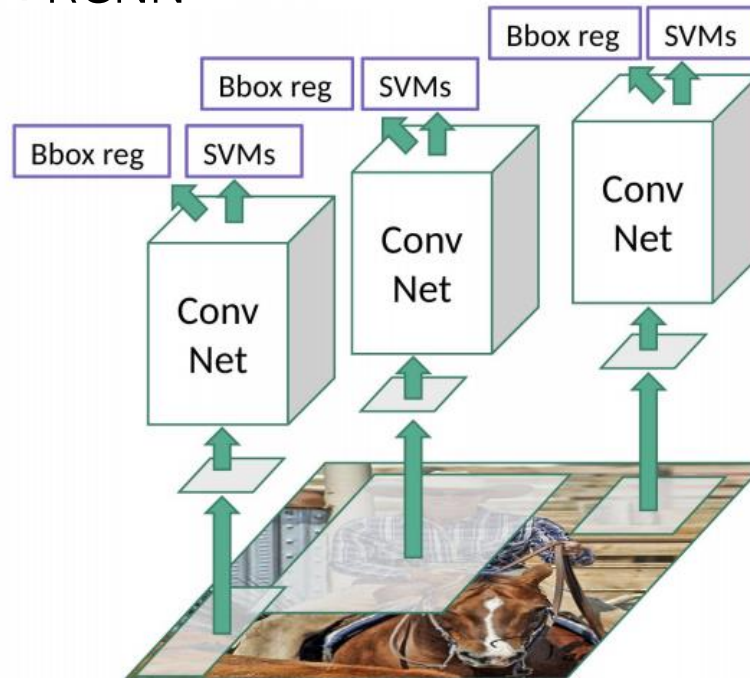


3단계: 최종적으로 bbox와 라벨을 반환

이미지 전체의 bbox와 개체의 라벨 확률 동시에 반환

Objection Detection

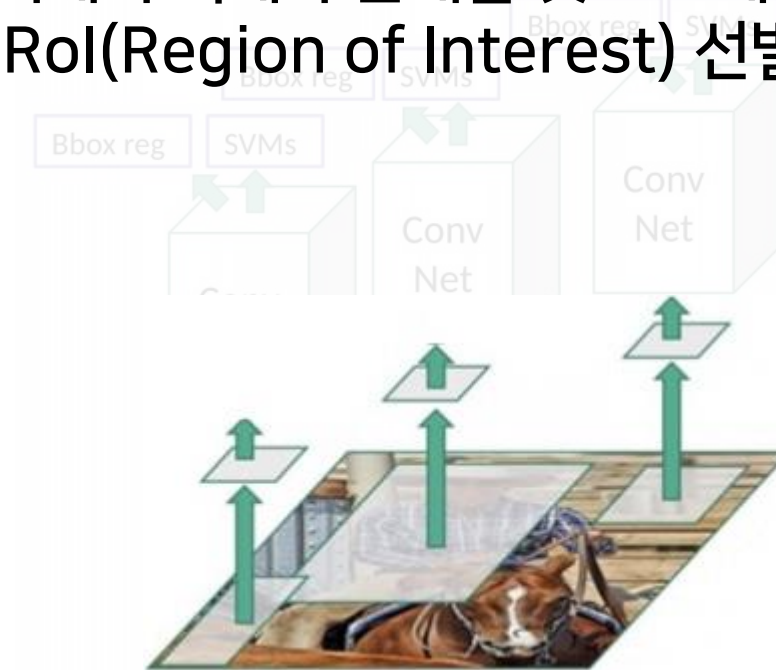
2-Stage Detector : RCNN



Localization과 Classification을 순차적으로 수행
상대적으로 느리지만 높은 정확도를 보임

Object Detection

1단계: 이미지에서 객체가 존재할 것으로 예상하는 위치인
RoI(Region of Interest) 선별



Localization과 Classification을 순차적으로 수행
상대적으로 느리지만 높은 정확도를 보임

Objection Detection

이 과정을 Region Proposal이라 부르고
Selective Search 알고리즘을 사용

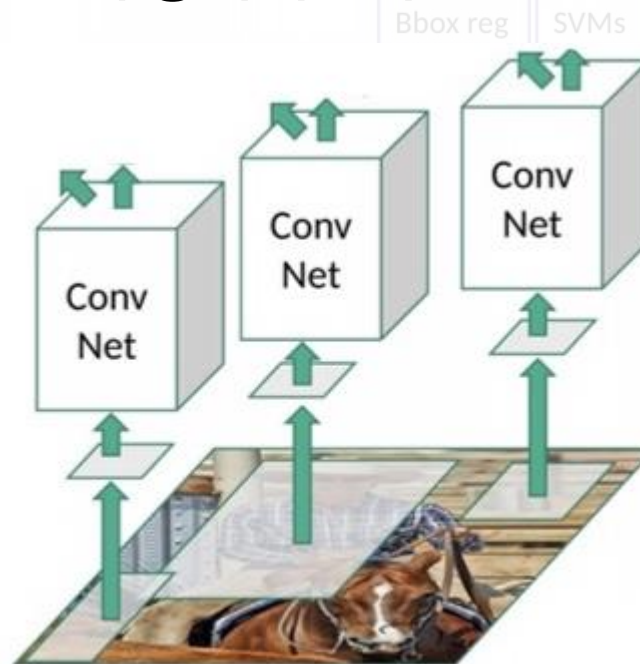
Bbox를 랜덤하고 작게 많이 생성한 후
계층적 알고리즘을 통해 Bbox를 합쳐서 Roi 생성



Localization과 Classification을 순차적으로 수행
상대적으로 느리지만 높은 정확도를 보임

Object Detection

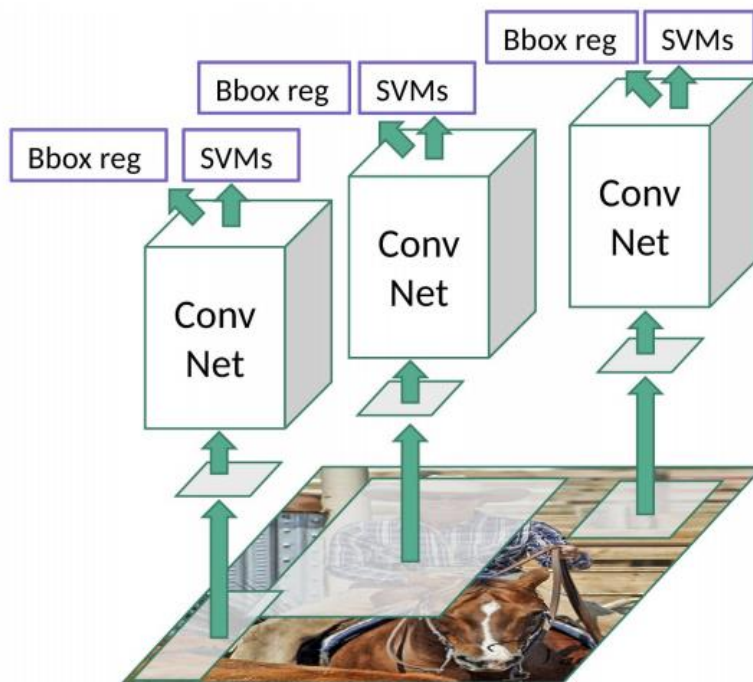
2-Stage: RoI를 CNN에 통과시킨 후 Feature Map 반환



Localization과 Classification을 순차적으로 수행
상대적으로 느리지만 높은 정확도를 보임

Object Detection

2-Stage Detector • RCNN

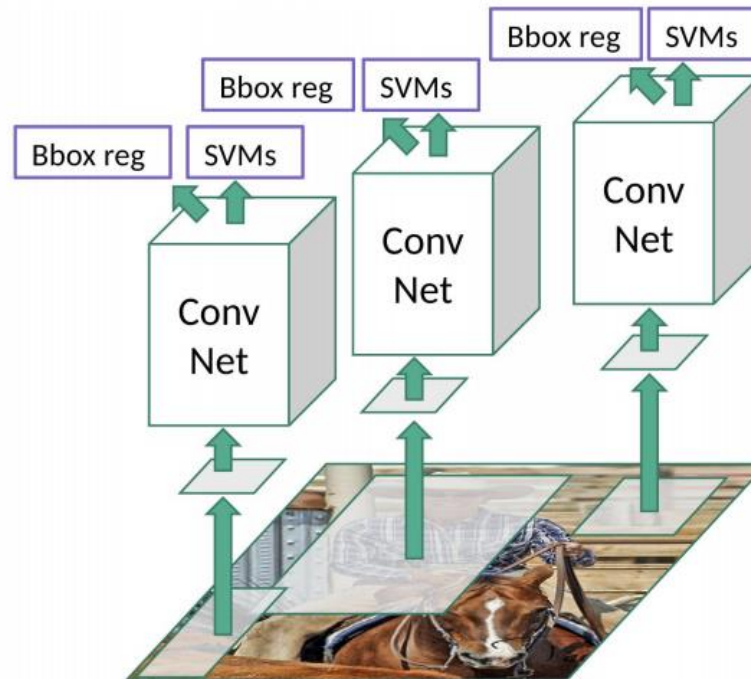


Localization과 Classification을 순차적으로 수행

3단계: 반환된 Feature Map을 바탕으로 회귀와 분류 진행

Object Detection

2-Stage Detector • RCNN

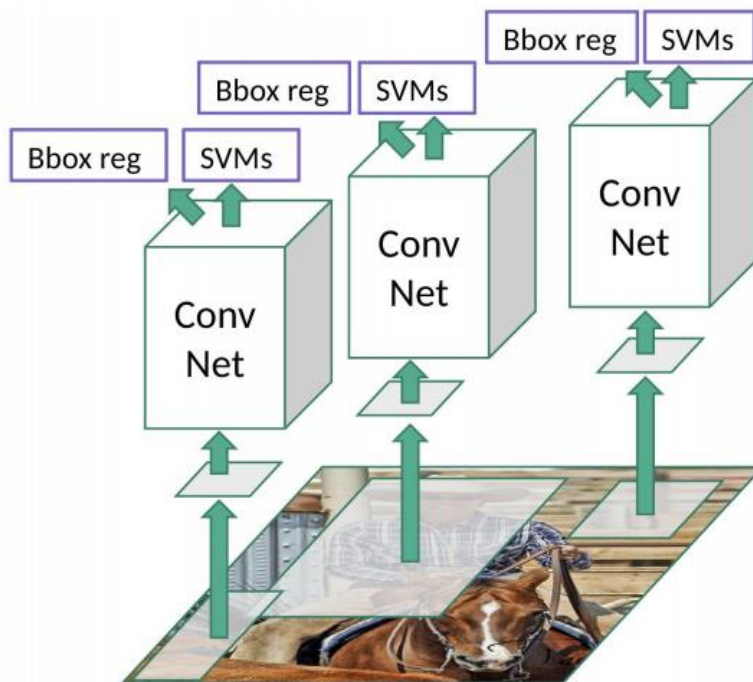


Localization과 Classification을 순차적으로 수행
상대적으로 느리지만 높은 정확도를 보임

**4단계: CNN feature를 이용하여
Bbox regression model 업데이트**

Object Detection

2-Stage Detector • RCNN



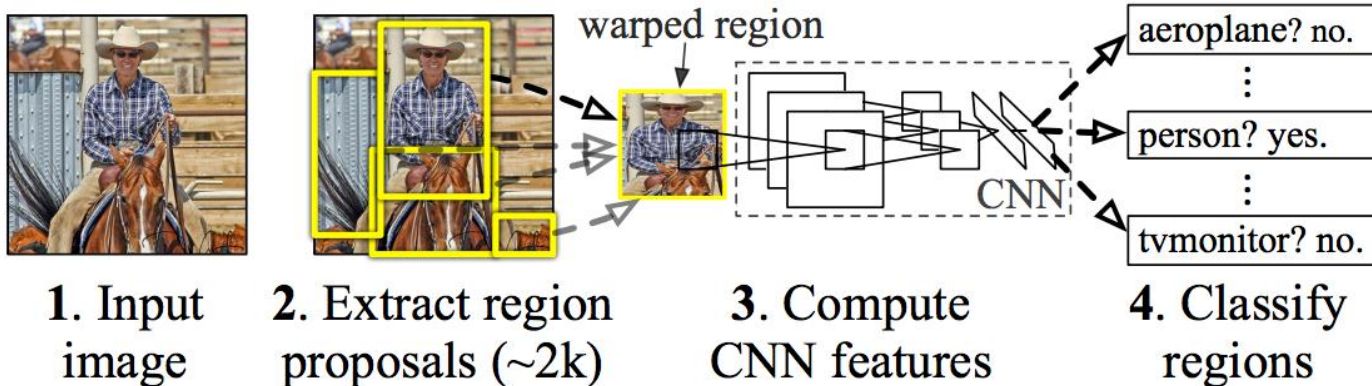
Localization과 Classification을 순차적으로 수행
상대적으로 느리지만 높은 정확도를 보임

4단계: SVM을 통해 각 RoI의 라벨 예측

Objection Detection

2-Stage Detector : RCNN

R-CNN: *Regions with CNN features*



시간이 매우 오래 걸림

분류와 회귀의 결과를 CNN에 반영할 수 없음

회귀와 분류 문제 모두 존재해 최적화가 어려움

시무룩..



Image Segmentation



Input



- 1: Person
- 2: Purse
- 3: Plants/Grass
- 4: Sidewalk
- 5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4

Semantic Labels

이미지를 구성하고 있는 모든 픽셀이 어느 class에 속하는지 분류
더 정확한 위치를 알아야 해서 객체 탐지보다 더 복잡

Image Segmentation



Input



- 1: Person
- 2: Purse
- 3: Plants/Grass
- 4: Sidewalk
- 5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4

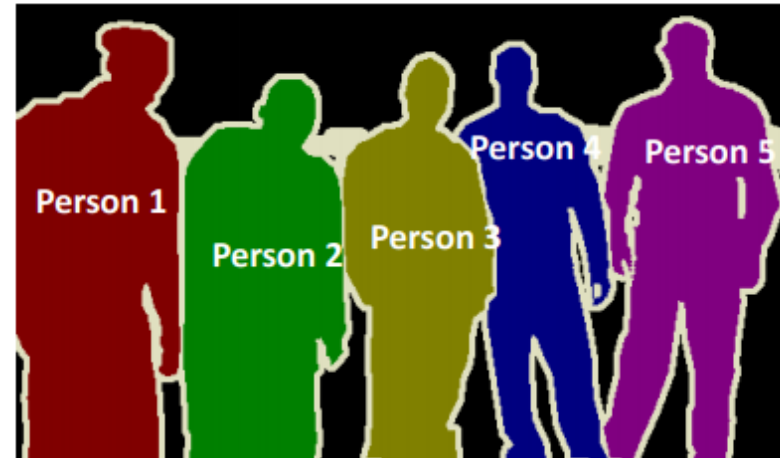
Semantic Labels

Convolution과 Pooling을 역으로 적용하는 방식으로
입력 이미지와 출력의 크기를 같게 만듦

Image Segmentation



Semantic Segmentation



Instance Segmentation

이 이미지를 구성하고 있는 모든 픽셀이 어느 어디에 속하는지 더 세부적으로 분류
정확한 위치를 알아야 해서 객체 인식보다 더 복잡
→ Semantic Segmentation보다 더 복잡

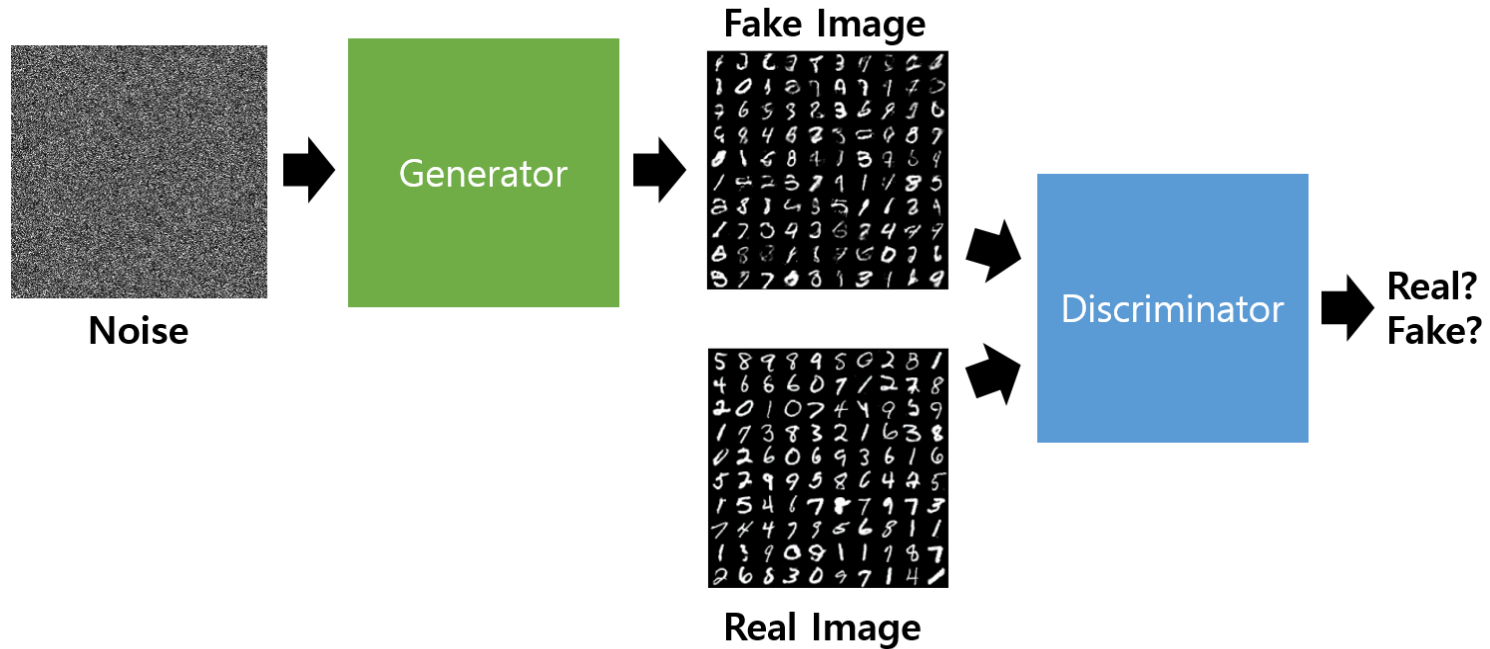
| Image Generation

Image Generation

주어진 이미지를 바탕으로 새로운 이미지를 만드는 모델
비지도 학습에 해당

Image Generation

GAN(Generative Adversarial Network)



Generator와 Discriminator로 구성됨

Image Generation

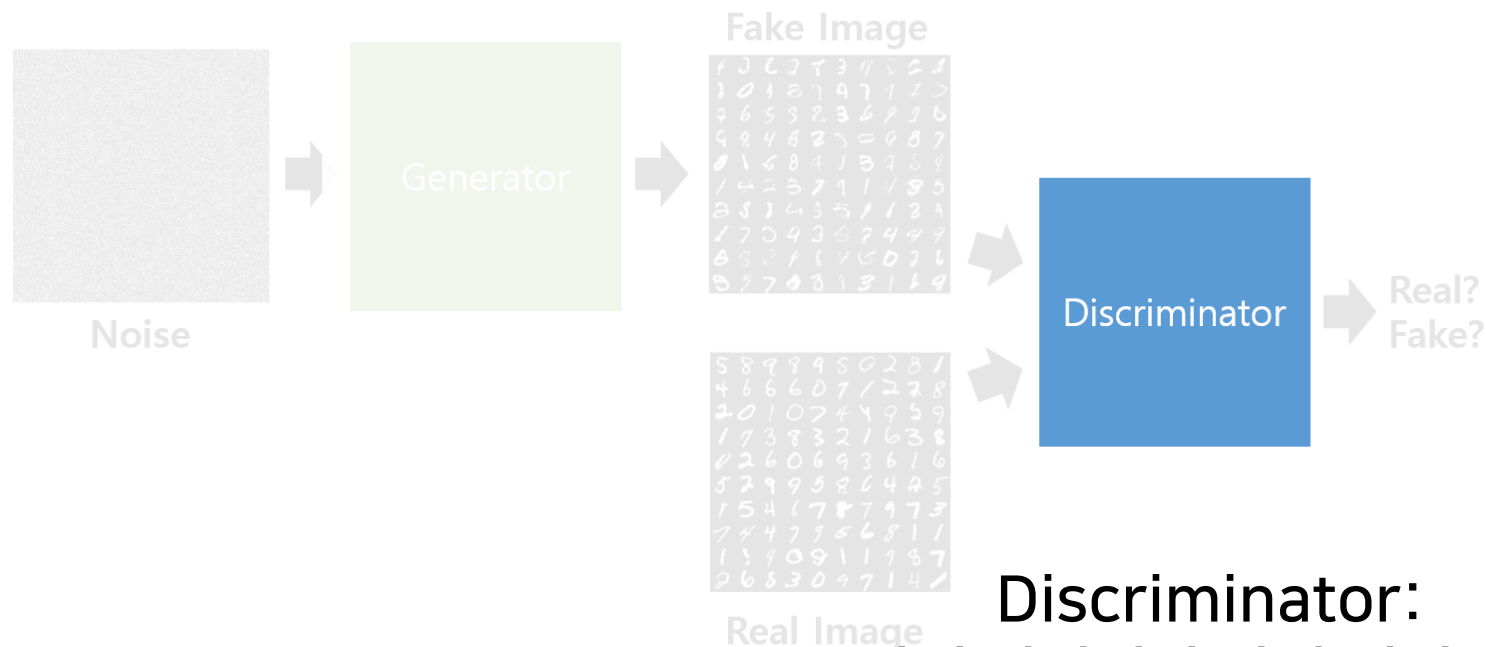
GAN(Generative Adversarial Network)



Generator와 Discriminator로 구성됨

Image Generation

GAN(Generative Adversarial Network)

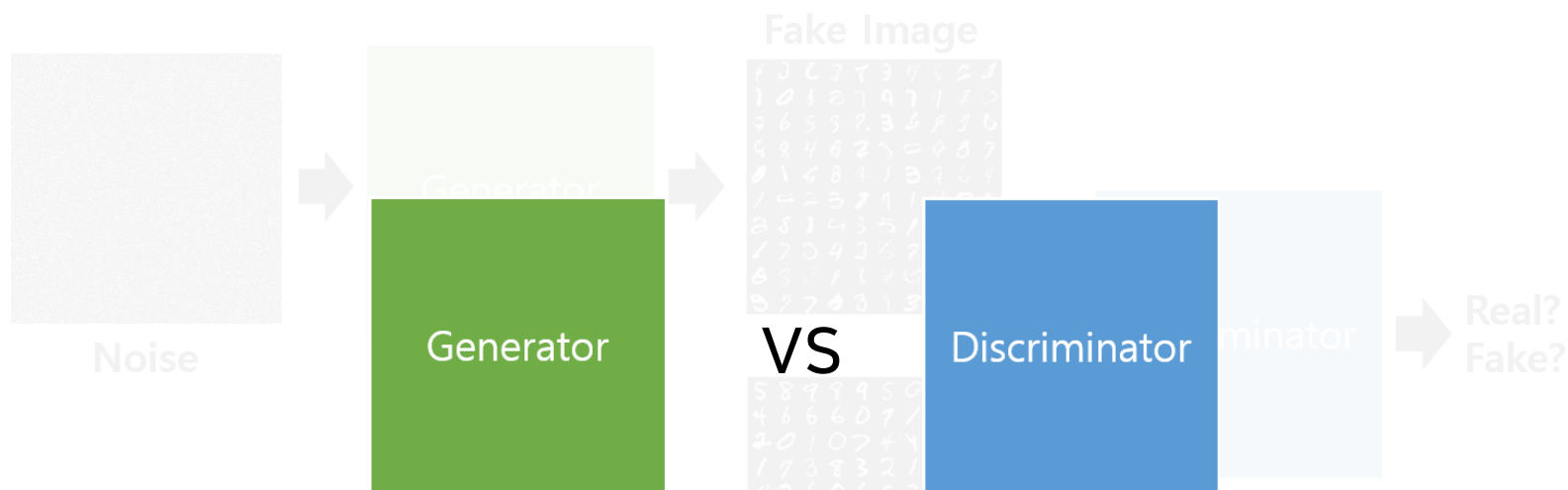


Discriminator:
실제 이미지와 가짜 이미지를
구별하는 모델

Generator와 Discriminator로 구성됨

Image Generation

GAN(Generative Adversarial Network)

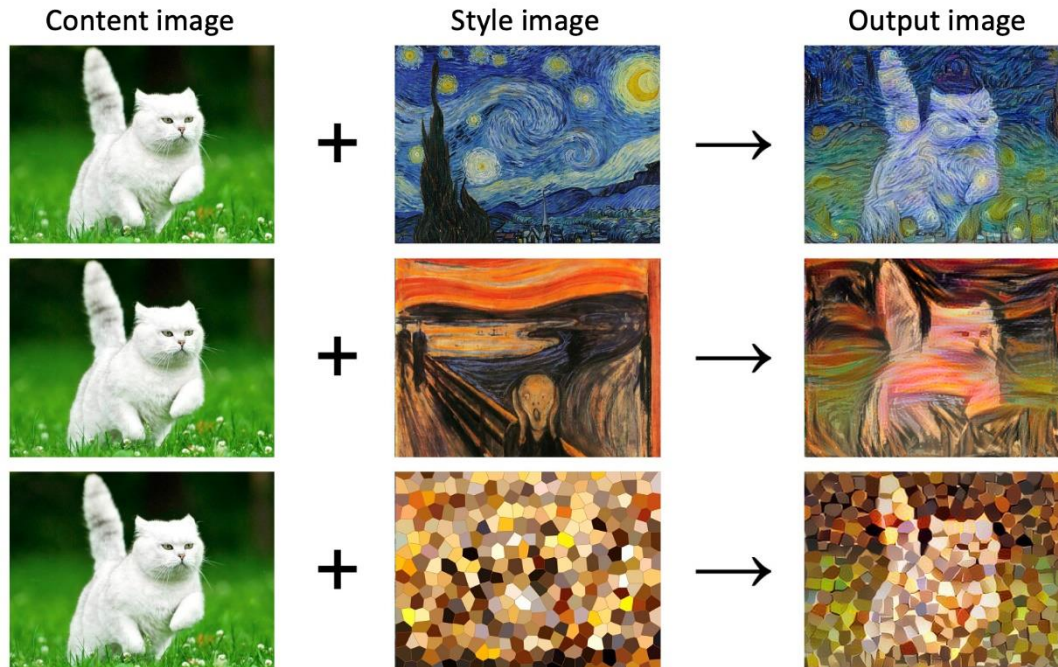


두 모델은 서로를 적대적으로 학습하며
서로의 성능을 향상시킴!

Generator와 Discriminator로 구성

Image Generation

Style Transfer




Style 이미지의 특징을 추출하여
입력에 해당하는 content image에 적용하는 알고리즘

Image Generation

Style Transfer

Style Difference

 : 스타일 이미지와 출력 이미지의 특성 차이

Content image



+

Style image



→

Output image



+



→



+



→



Content Difference
: 결과 이미지와
출력 이미지의 특성 차이

Content Difference, Style Difference를

모두 줄이는 것을 목적으로 손실함수 구성



THANK YOU

