딥러닝팀

1팀

정승민
변석주
이정환
송승현
최용원

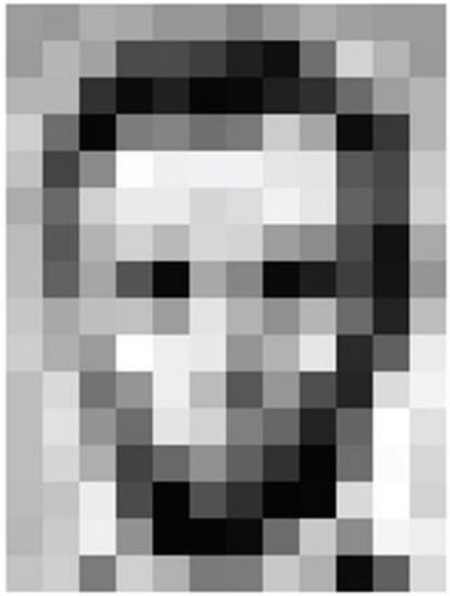
CONTENTS

1

이미지 데이터의 특징

1 이미지 데이터의 특징

컴퓨터에서의 이미지 데이터



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

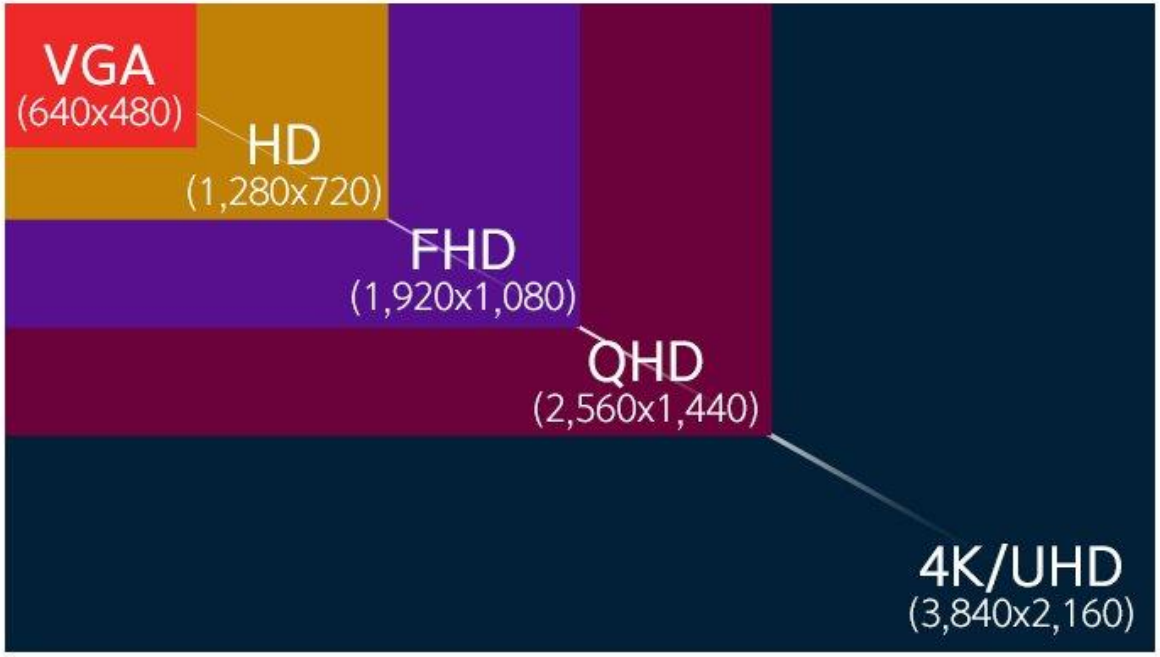
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

모든 이미지는 픽셀이라는 사각형의 점으로 구성됨

각 픽셀에는 값이 저장되어 있고, 색을 나타냄

1 이미지 데이터의 특징

컴퓨터에서의 이미지 데이터



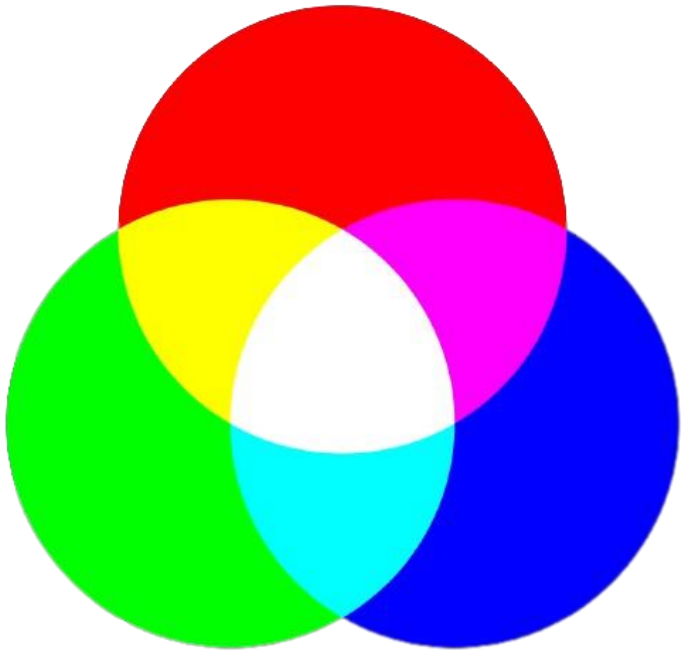
FHD는 1920×1080, QHD는 2560×1440개의 픽셀로 구성

픽셀이 많을수록 화질이 더 좋아짐

1 이미지 데이터의 특징

채널

Channel



컬러 이미지는 빛의 3원색인 R, G, B 3개의 채널로 이루어짐

각 채널에서 0~255의 값으로 색의 선명도 표현

1 이미지 데이터의 특징

채널

행렬 3개로 모든 이미지 표현 가능

Pytorch에서 (Channel, Height, Width)로 표현

Tensor에서는 (Height, Width, Channel)로 표현

2

CNN

기존 신경망의 문제

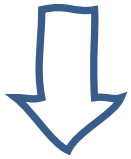
기존 신경망은 1차원 벡터를 input으로 사용

2차원 이상의 데이터 사용 시 1차원으로 임베딩 해야함

공간 정보 손실 발생!

기존 신경망의 문제

기존 신경망은 1차원 벡터를 input으로 사용



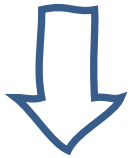
2차원 이상의 데이터 사용 시 1차원으로 임베딩 해야함

임베딩: 고차원 데이터를 저차원으로 변환하는 것

공간 정보 손실 발생!

기존 신경망의 문제

기존 신경망은 1차원 벡터를 input으로 사용



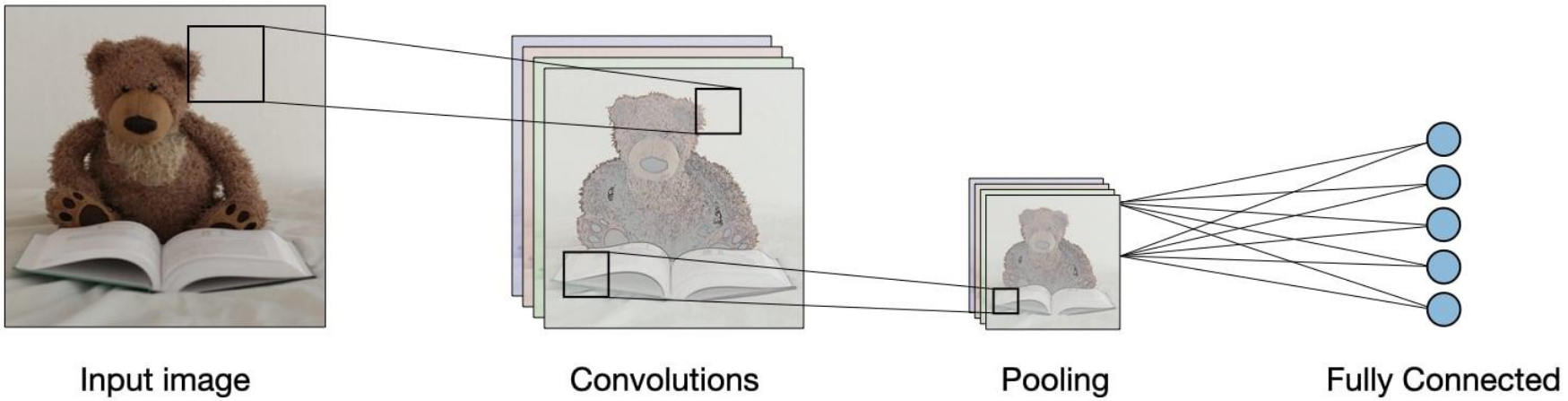
2차원 이상의 데이터 사용 시 1차원으로 임베딩 해야함

임베딩: 고차원 데이터를 저차원으로 변환하는 것

공간 정보 손실 발생!

CNN

Convolution Neural Network

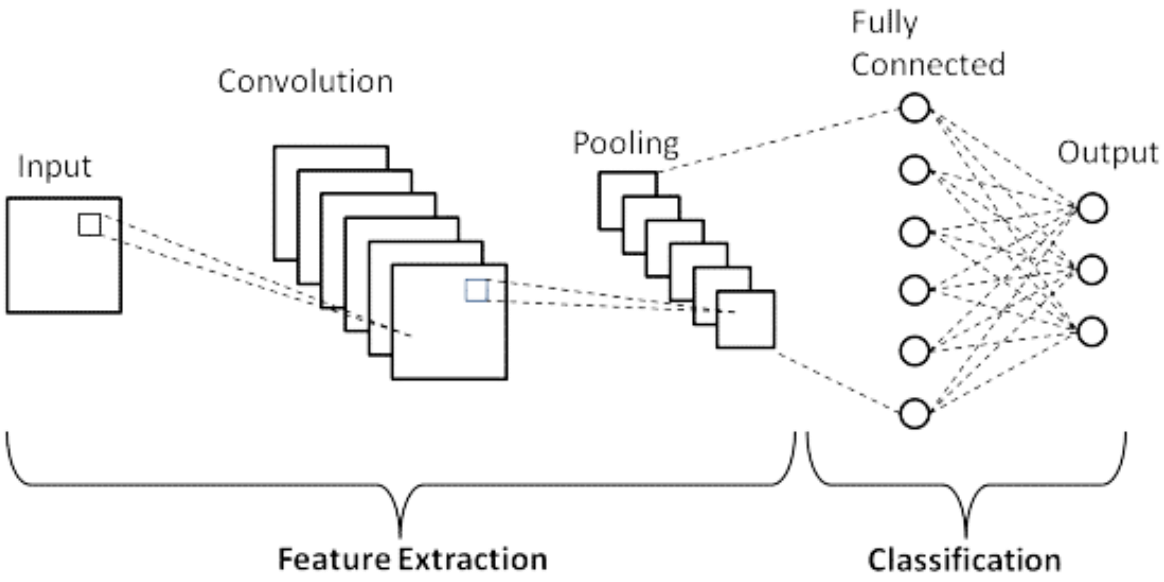


공간 정보의 보존

검정색 상자가 변환은 되나, 다음 층에서 남아있는 것을 확인!

CNN

Layers of CNN



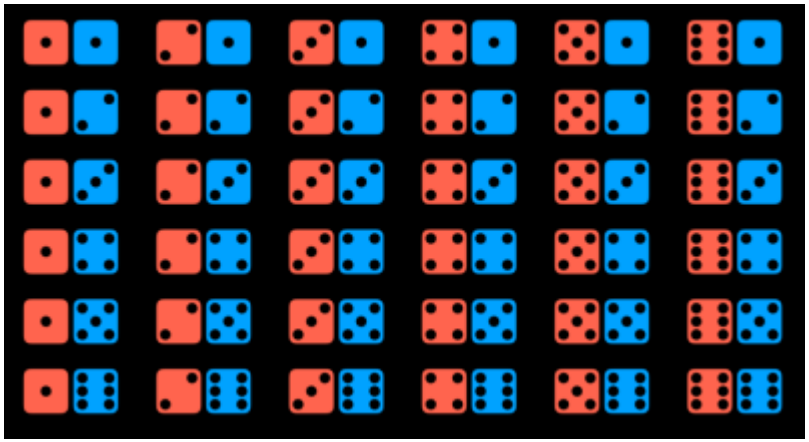
Convolution Layer

Pooling Layer

Fully Connected Layer

Convolution

1차원 합성곱



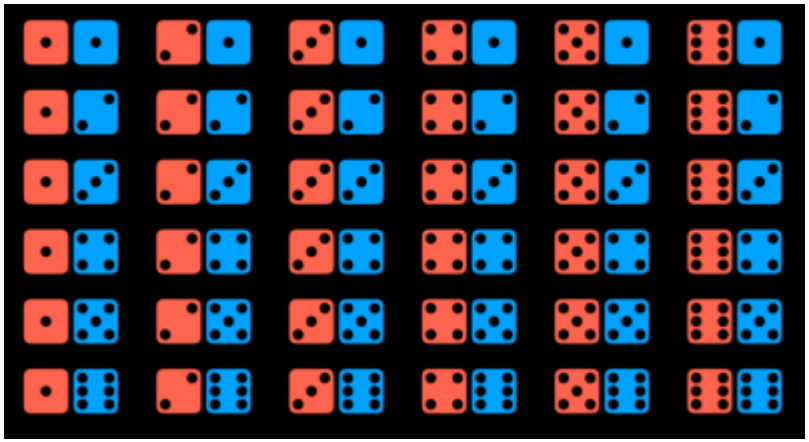
1차원 합성곱 정의

$$(f * g)[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k]$$

input 커널

Convolution

1차원 합성곱

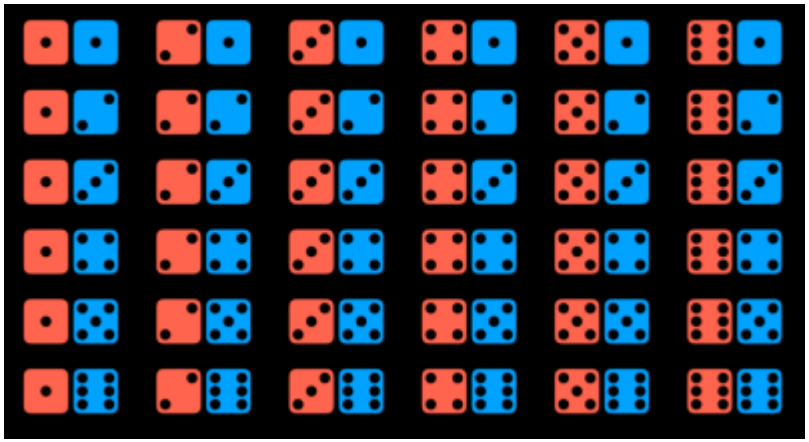


두 눈의 합이 10이 되는 경우

$$P(X+Y=10) = P(X=4)P(Y=6)$$
$$+P(X=5)P(Y=5) + P(X=6)P(Y=4)$$

Convolution

1차원 합성곱

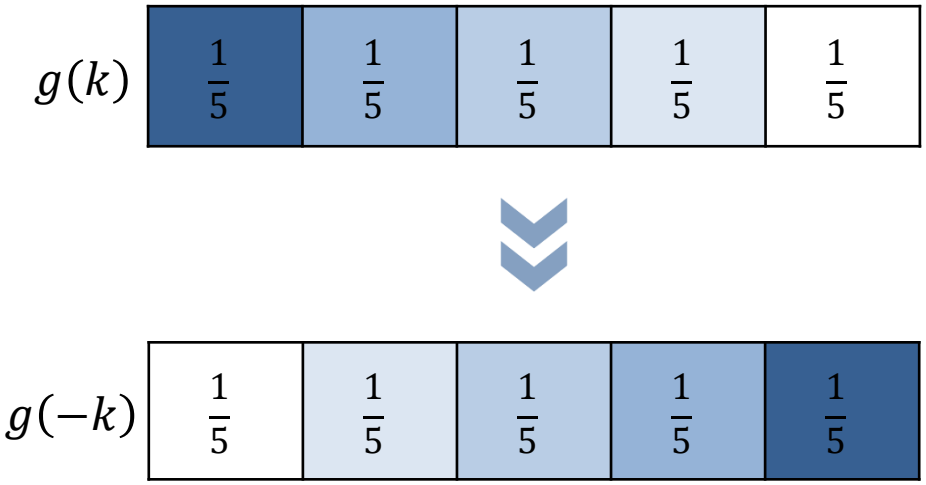


시그마를 이용하여 정리

$$\mathrm{P}(\mathrm{X}+\mathrm{Y}=10)=\sum_{k=4}^{6}P(X=k)P(Y=10-k)$$

$$=\sum_{k=-\infty}^{\infty}P(X=k)P(Y=10-k)$$

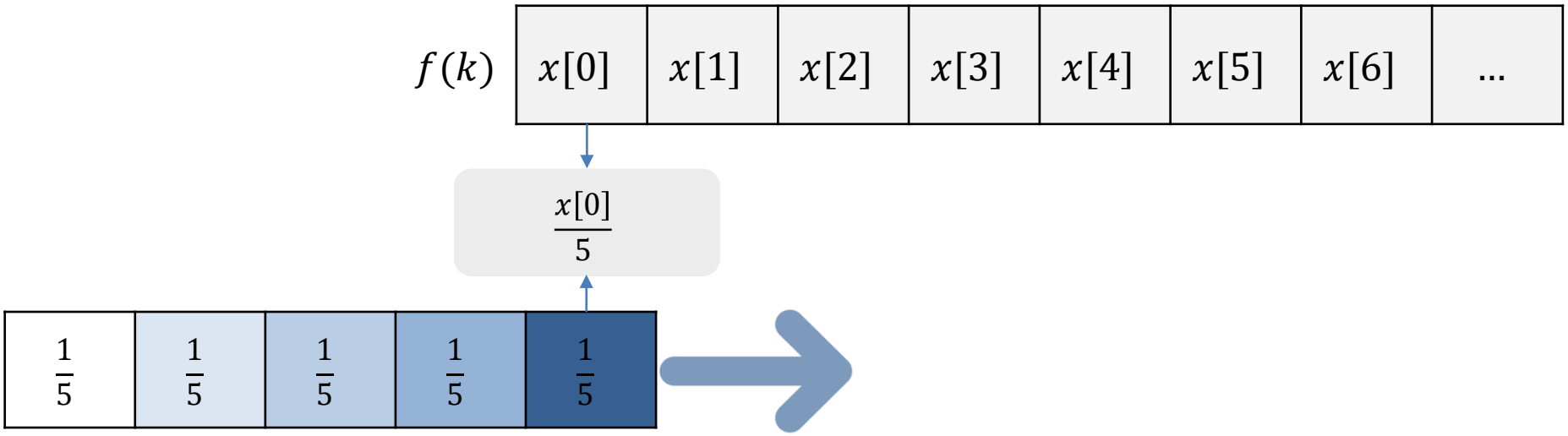
1차원 합성곱



같은 값으로 초기화하여 이동평균 효과부여

커널의 값이 역순으로 곱해짐

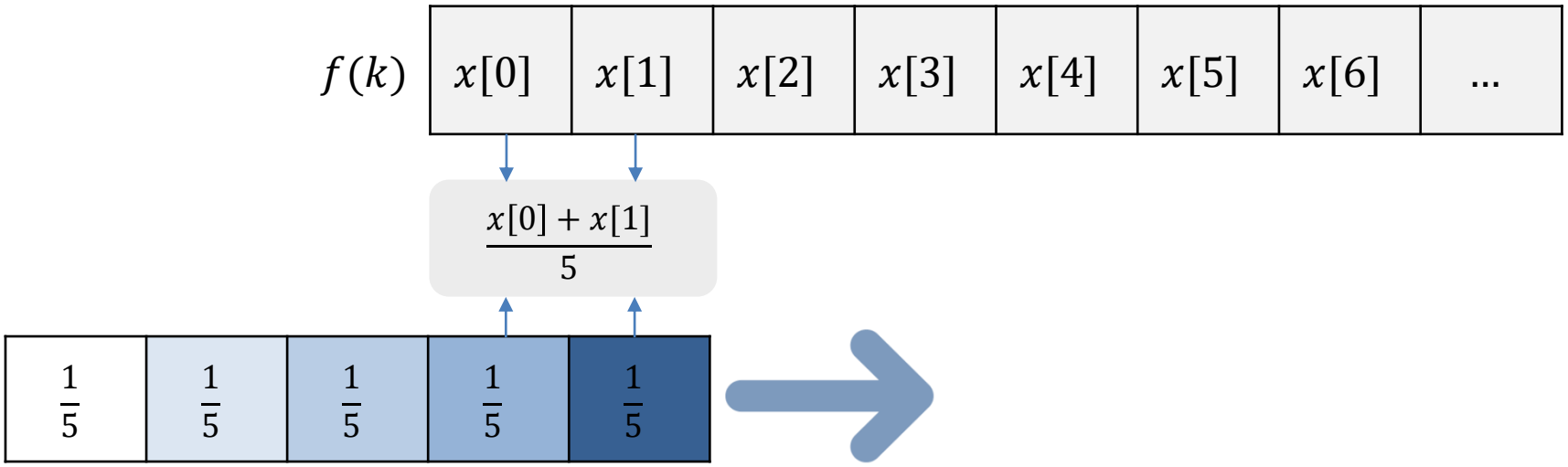
1차원 합성곱



같은 값으로 초기화하여 이동평균 효과부여

커널의 값이 역순으로 곱해짐

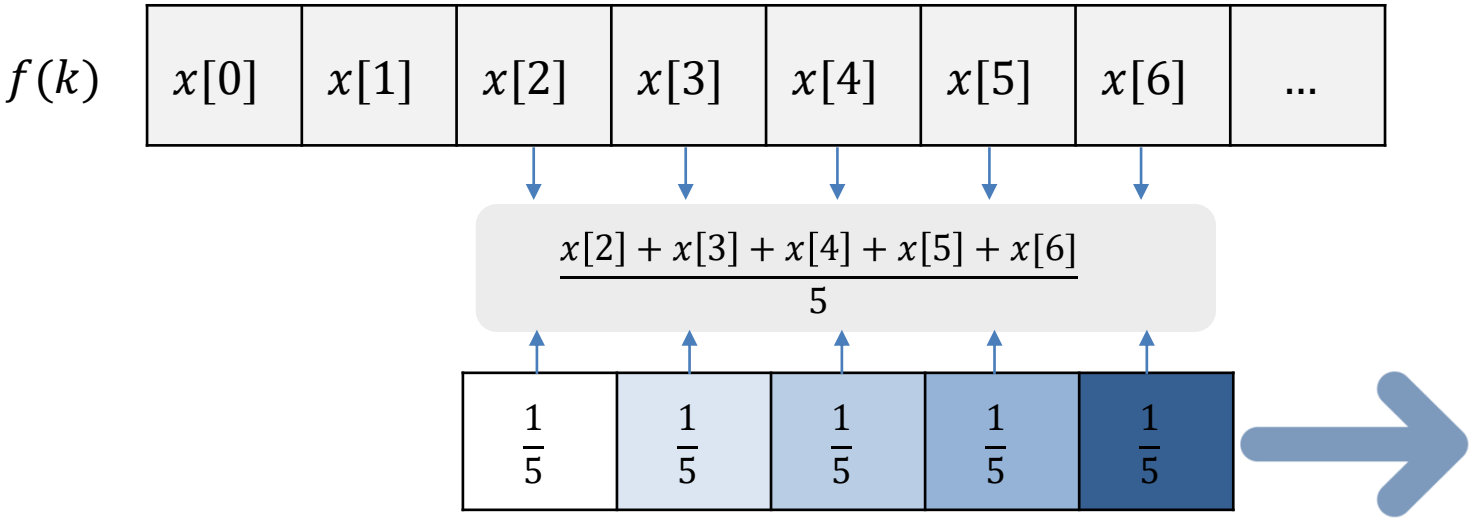
1차원 합성곱



같은 값으로 초기화하여 이동평균 효과부여

커널의 값이 역순으로 곱해짐

1차원 합성곱



같은 값으로 초기화하여 이동평균 효과부여

커널의 값이 역순으로 곱해짐

2차원 합성곱

1/100	···	1/100
···	···	···
1/100	···	1/100

(10×10)

Convolution

주변 픽셀의 평균값으로
이미지가 흐려지는 효과 부여

2차원 합성곱

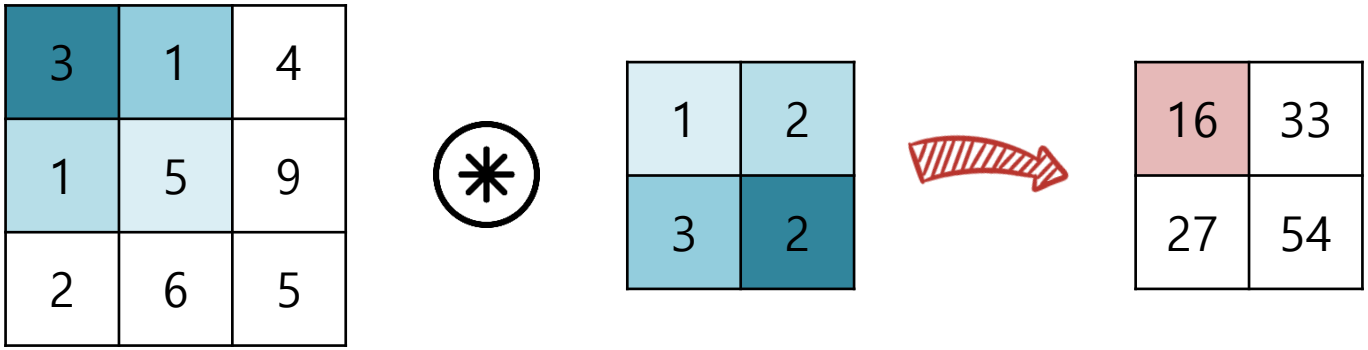
1/100	···	1/100
···	···	···
1/100	···	1/100

(10×10)

Convolution

주변 픽셀의 평균값으로
이미지가 흐려지는 효과 부여

2차원 합성곱



2차원에도 커널이 뒤집혀져 계산됨

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

2차원 합성곱

커널이 뒤집히는 것은 CNN에서 중요한 성질이 아님

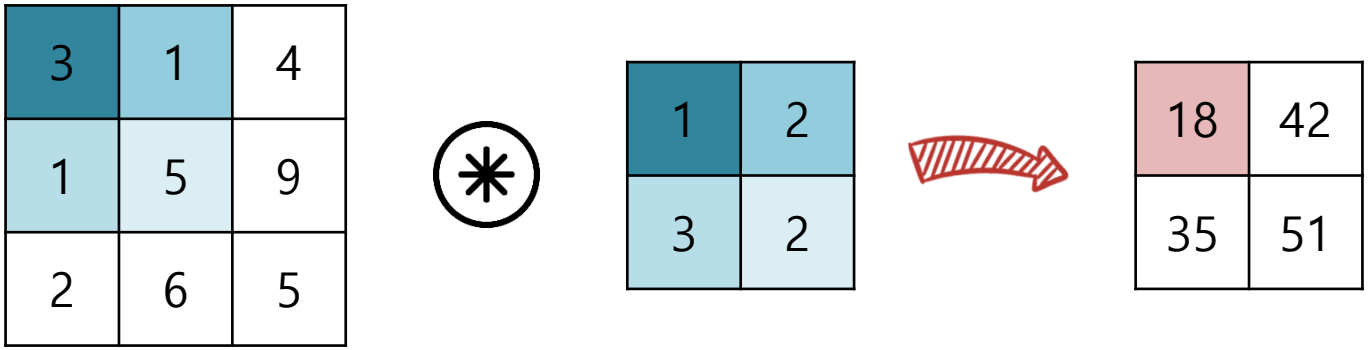
여러 라이브러리에서
같은 위치의 성분끼리 곱하는 교차상관 사용!

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

I: 이미지

K: 커널

2차원 합성곱



2차원 교차상관 정의

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

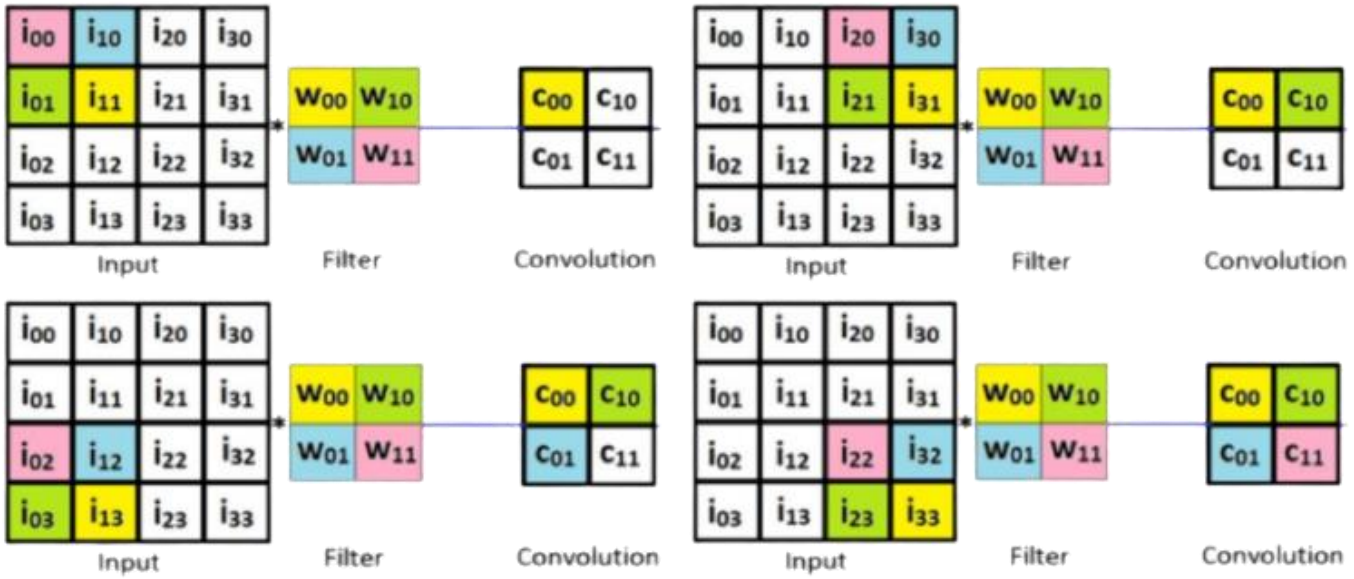
i, j에 대한 덧셈으로 변화

커널의 인덱스 고정

Convolution Layer

특징

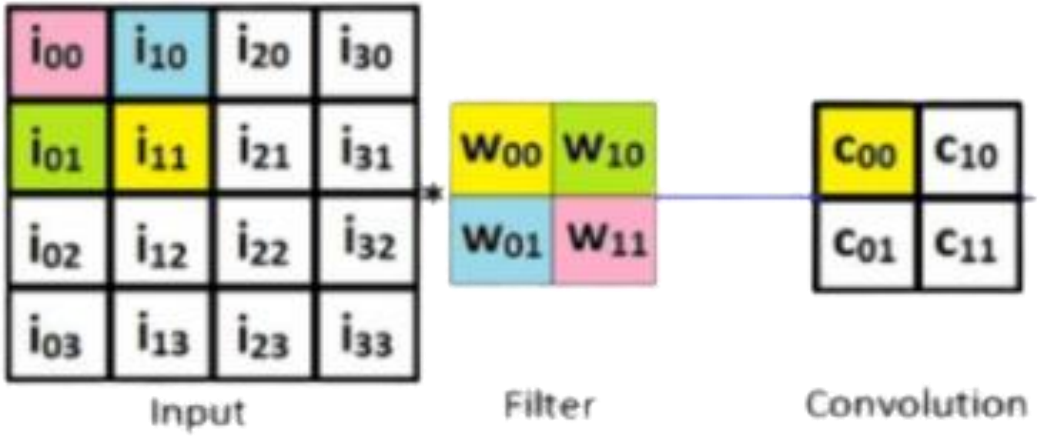
교차상관이 아닌 합성곱을 사용한 이미지



입력 데이터의 공간 정보 보존, 특징 추출

Convolution Layer

특징



$$i_{00}w_{11} + i_{10}w_{01} + i_{01}w_{10} + i_{11}w_{00} = c_{00}$$

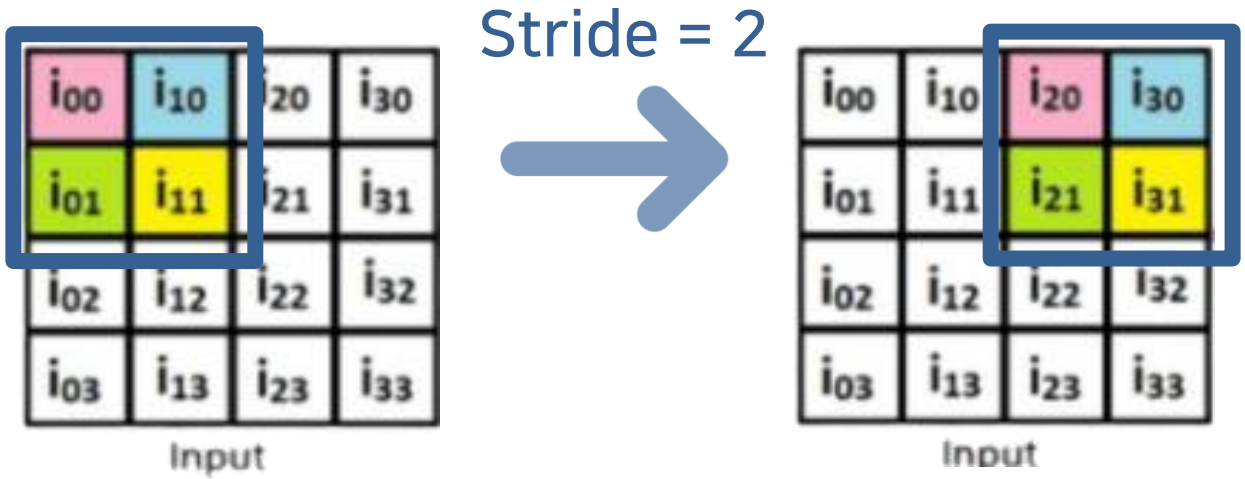
필터

커널의 다른 이름

입력 데이터와 합성곱되어 결과 반환

Convolution Layer

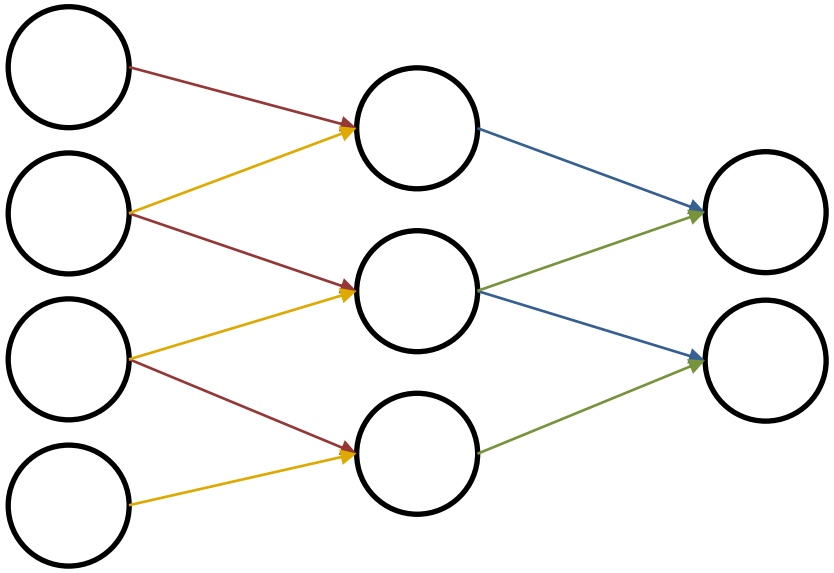
작동원리



Stride

필터가 입력데이터를 따라 얼마나 이동할지 결정하는 하이퍼 파라미터

Convolution Layer



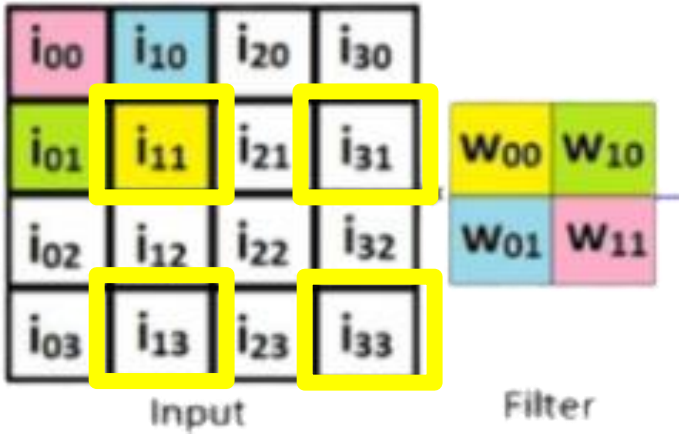
희소 상호작용

기존 신경망은 완전 연결(Fully Connected)인 반면

Convolution Layer는 하나의 입력데이터에 일부의 가중치만 곱해짐

완전 연결에서 연결하지 않는 부분의 가중치를 전부 0으로 처리

Convolution Layer

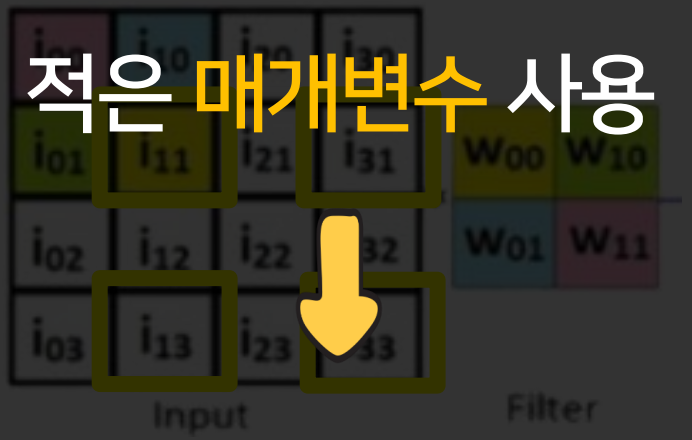


w_{00}를 네 개의 픽셀이 공유

매개변수 공유

하나의 가중치가 여러 입력데이터에 동일하게 곱해짐

Convolution Layer



적은 매개변수 사용

메모리 사용량 감소 ⬇

통계적 효율성 증가 ⬆

매개변수 공유

하나의 가중치가 여러 입력데이터에 동일하게 곱해짐

Convolution Layer

Feature Map

하이퍼 파라미터	Pytorch	Tensorflow 표현	역할
Input Channel	in_channels	미리 지정	입력 데이터의 채널 개수 지정
Output Channel	out_channels	filters	출력의 채널 개수 지정
필터 사이즈	kernel_size	kernel_size	필터의 크기 지정
Stride	stride	strides	필터의 이동 간격 지정
Padding	padding	padding	입력 데이터 주변에 붙일 padding 수 지정

하이퍼 파라미터로 Feature Map이 구성됨

Convolution layer에서 반환된 결과

Convolution Layer

Input channel | Output Channel

Input Channel

입력 데이터의 채널 개수

컬러 이미지 데이터: R, G, B

3개의 채널

흑백 이미지 데이터: 1개의 채널

Output Channel

convolution layer가

반환하는 채널 개수

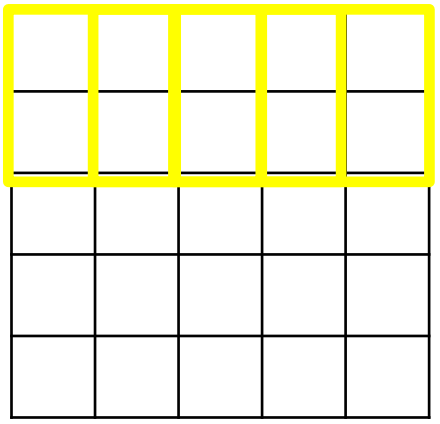
Convolution layer의

필터의 개수와 동일

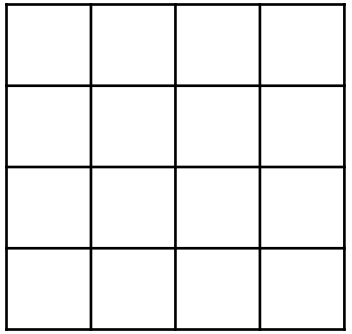
Convolution Layer

필터의 사이즈

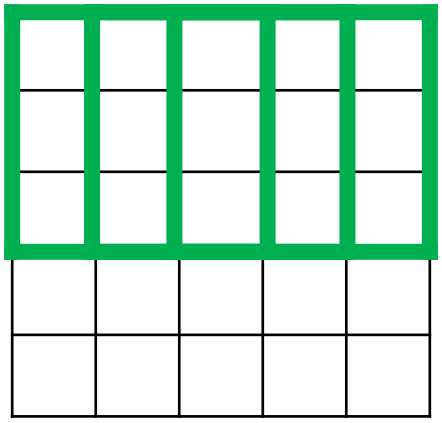
2x2필터



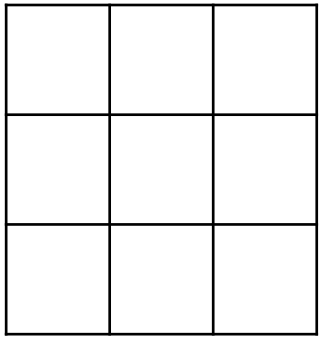
4x4 feature map



3x3필터



3x3 feature map



Convolution Layer

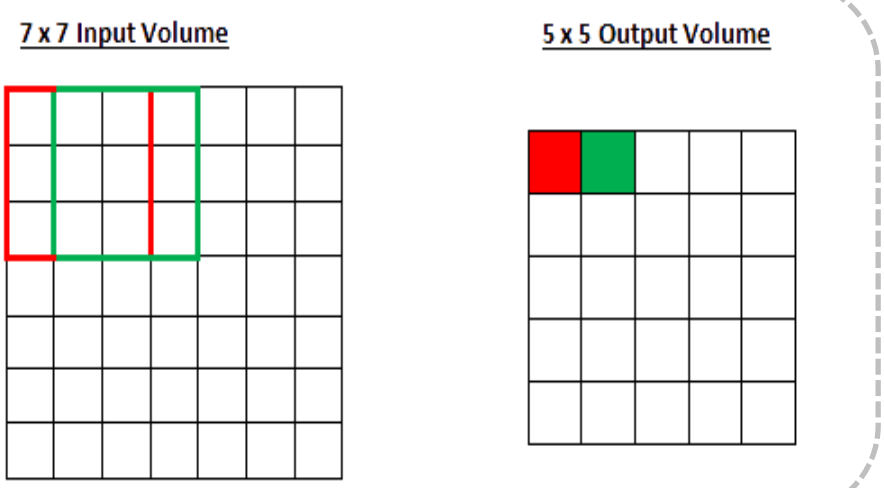
필터의 사이즈

필터의 크기가 클수록 feature map의 크기가 작아짐

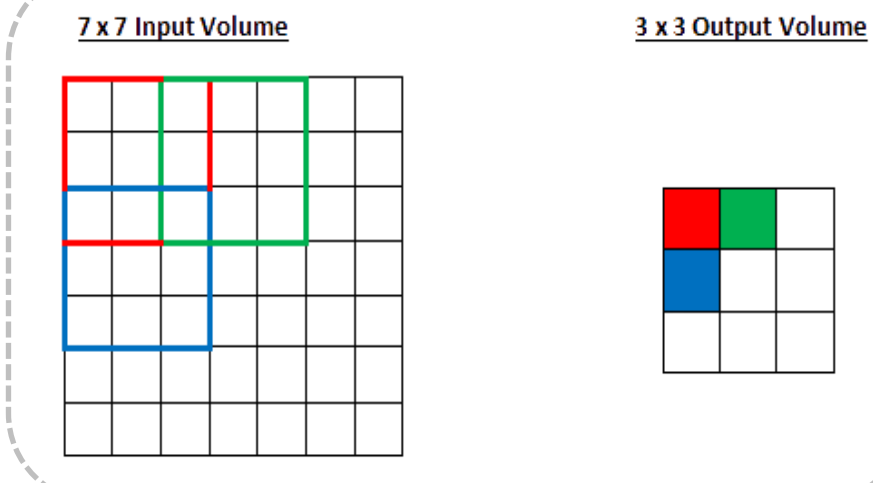
일반적으로 홀수의 필터 사이즈 사용

Convolution Layer

Stride



stride = 1



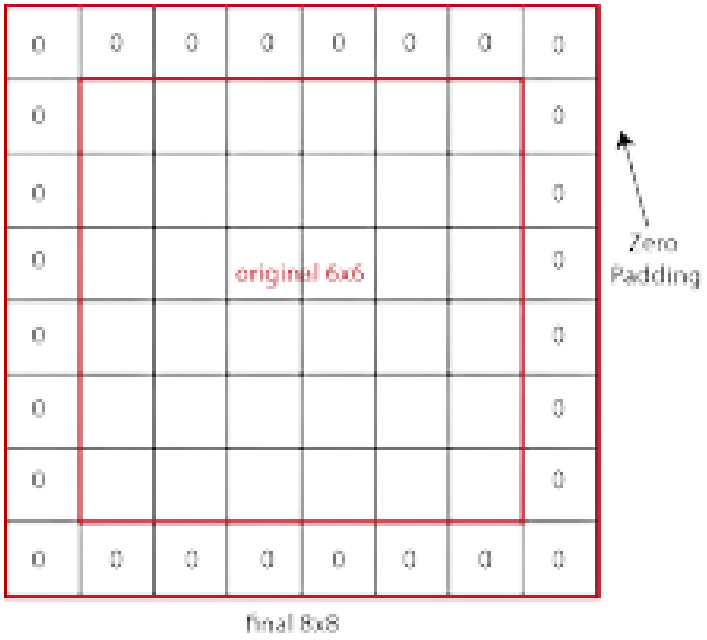
stride = 2

stride가 큼 → 입력 데이터의 연산 감소

→ feature map의 크기 작아짐

Convolution Layer

padding



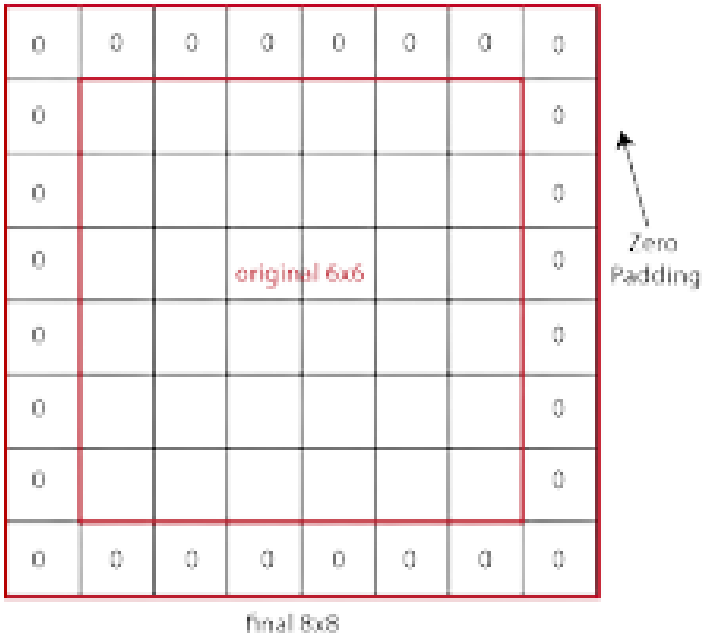
padding = 1

가장자리 입력 데이터는 중심부에 비해 필터에 통과되는 횟수가 적음

원본 데이터 주변 테두리에 새로운 값 추가해 가장자리의 값들도 필터를 여러 번 통과

Convolution Layer

padding



padding = 1

중심부 입력 데이터는 가장자리에 비해 필터에 통과되는 횟수가 많음

zero padding

: 가장자리에 모두 0을 대입하는 방법

원본 데이터 주변 테두리에 새로운 값 추가해 가장자리의 값들도 필터를 여러 번 통과

Convolution Layer

padding

유효한 (valid) 합성곱

zero padding을 사용하지 않음

출력 이미지가 급격하게 작아짐

동일 (same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 모델에 덜 반영됨

완전 (full) 합성곱

각 방향에서 모든 픽셀이 k번 방문되도록 충분히 많은 0을 채움

Convolution Layer

padding

유효한 (valid) 합성곱

zero padding을 사용하지 않음

출력 이미지가 급격하게 작아짐

동일 (same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 모델에 덜 반영됨

완전 (full) 합성곱

각 방향에서 모든 픽셀이 k번 방문되도록 충분히 많은 0을 채움

Convolution Layer

padding

유효한 (valid) 합성곱

zero padding을 사용하지 않음

출력 이미지가 급격하게 작아짐

동일 (same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 모델에 덜 반영됨

완전 (full) 합성곱

각 방향에서 모든 픽셀이 k번 방문되도록 충분히 많은 0을 채움

Convolution Layer

padding

유효한 (valid) 합성곱

zero padding을 사용하지 않음

출력 어느 차원이 급격하게 작아짐

동일 (same) 합성곱

출력과 입력이 같은 크기로 나오게 함

가장자리 픽셀들이 모델에 덜 반영됨

완전 (full) 합성곱

각 방향에서 모든 픽셀이 k번 방문되도록 충분히 많은 0을 채움

주로 유효한 합성곱과 동일 합성곱 사이의 개수 사용

Convolution Layer

Feature Map의 크기 계산

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

O_n = 출력의 가로 길이

I_n = 입력의 가로 길이

P = padding의 크기

F = 필터의 크기

S = stride의 크기

(3, 32, 32)의 입력 데이터

(4,4) 의 필터 10개 적용

stride = 2, padding = 1

$$\frac{32 + 2 - 4}{2} + 1 = 16$$

Convolution Layer

Feature Map의 크기 계산

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

O_n = 출력의 가로 길이

I_n = 입력의 가로 길이

P = padding의 크기

F = 필터의 크기

S = stride의 크기

(3, 32, 32)의 입력 데이터

(4,4) 의 필터 10개 적용

stride = 2, padding = 1

$$\frac{32 + 2 - 4}{2} + 1 = 16$$

Convolution Layer

Feature Map의 크기 계산

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

O_n = 출력의 가로 길이

I_n = 입력의 가로 길이

P = padding의 크기

F = 필터의 크기

S = stride의 크기

(3, 32, 32)의 입력 데이터

(4,4) 의 필터 10개 적용

stride = 2, padding = 1

$\frac{32 + 2 - 4}{2}$

(10, 16, 16) 의 feature map

Pooling Layer

입력 이미지의 중요한 특징 추출

Convolution Layer

여러 개의 필터 활용

필터 사이즈만큼의 가중치 존재

가중치와 입력의 연산을 통해

중요한 특징 추출

Pooling Layer

데이터의 크기를 줄임

가중치 사용하지 않음

중요한 특징 더욱 강조

Pooling Layer

입력 이미지의 중요한 특징 추출

Convolution Layer

여러 개의 필터 활용

필터 사이즈만큼의 가중치 존재

가중치와 입력의 연산을 통해

중요한 특징 추출

Pooling Layer

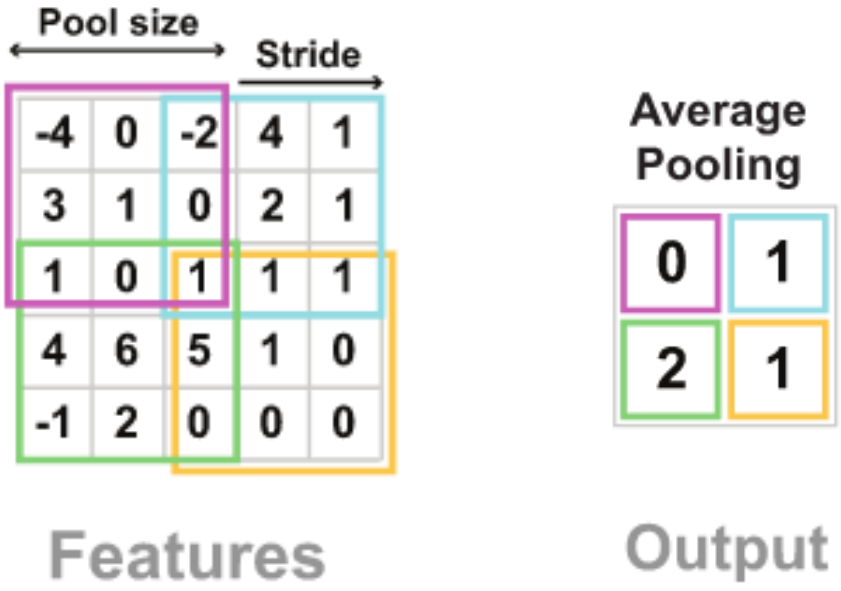
데이터의 크기를 줄임

가중치 사용하지 않음

중요한 특징 더욱 강조

Pooling Layer

Pooling의 종류

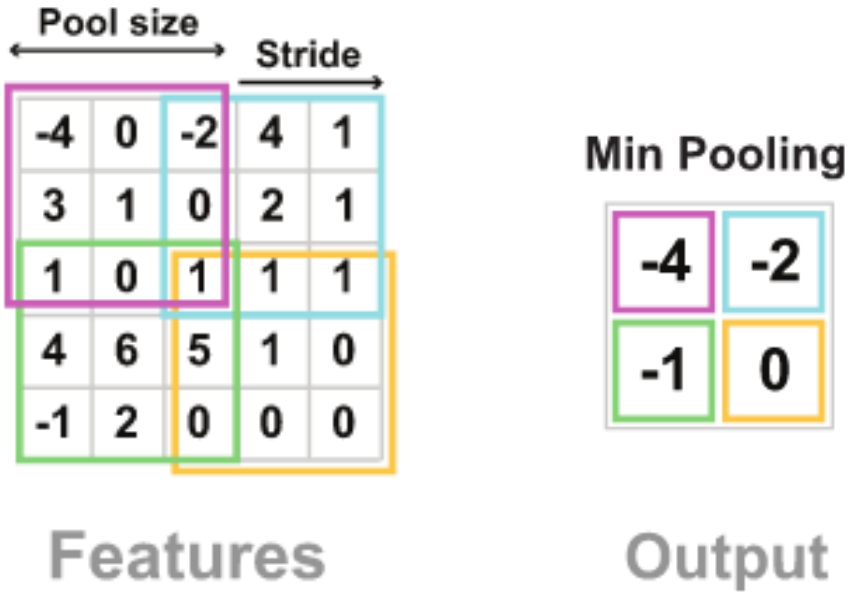


Average Pooling

필터가 덮고 있는 값의 평균 출력

Pooling Layer

Pooling의 종류



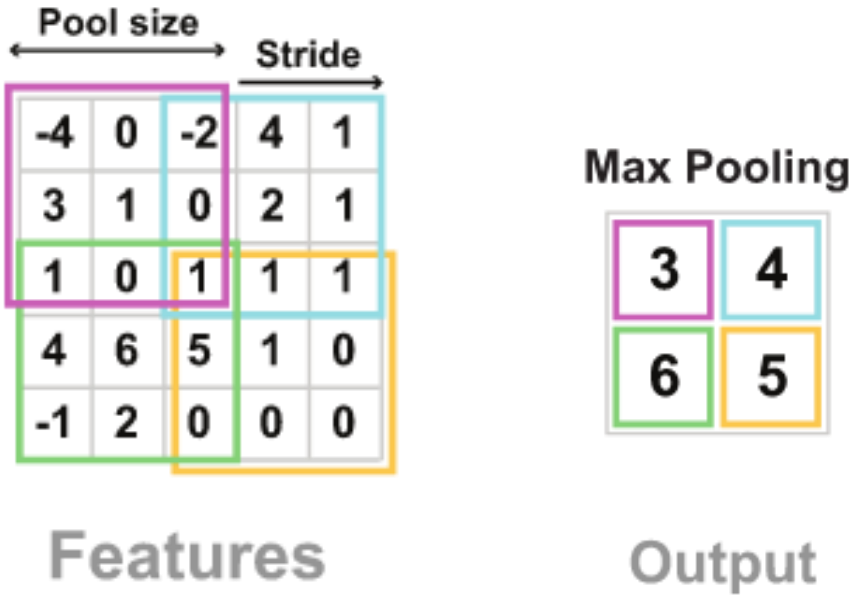
Min Pooling

필터가 덮고 있는 값 중 가장 작은 값 출력

0을 반환할 가능성이 높기 때문에 잘 사용하지 않음

Pooling Layer

Pooling의 종류

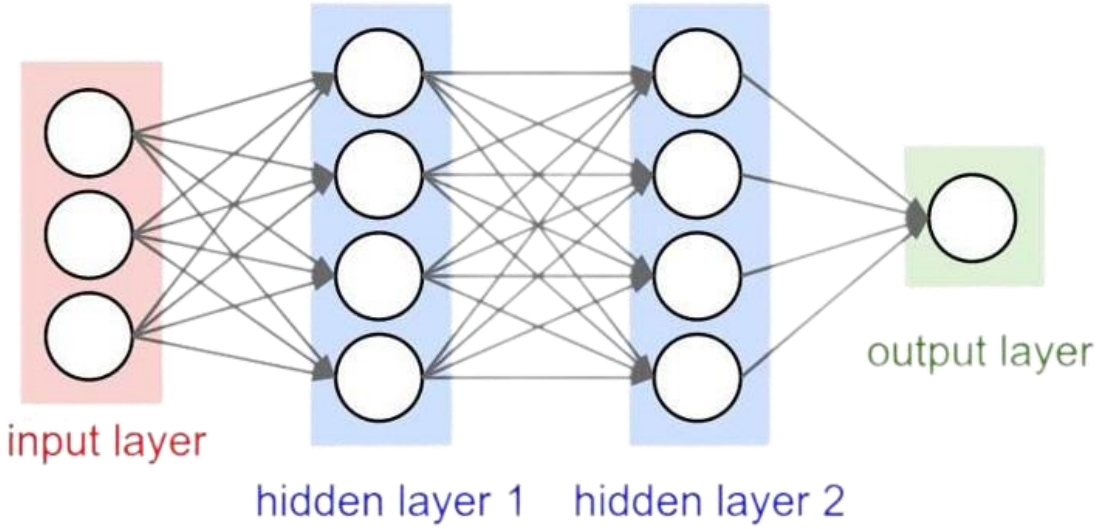


Max Pooling

필터가 덮고 있는 값 중 가장 큰 값 출력

위치에 따라 결과값이 불변하여 주로 사용

Fully Connected Layer



2차원 이상 입력시 벡터로의 변환 필수

FC Layer

최종 단계에서 라벨 예측을 진행하는 층

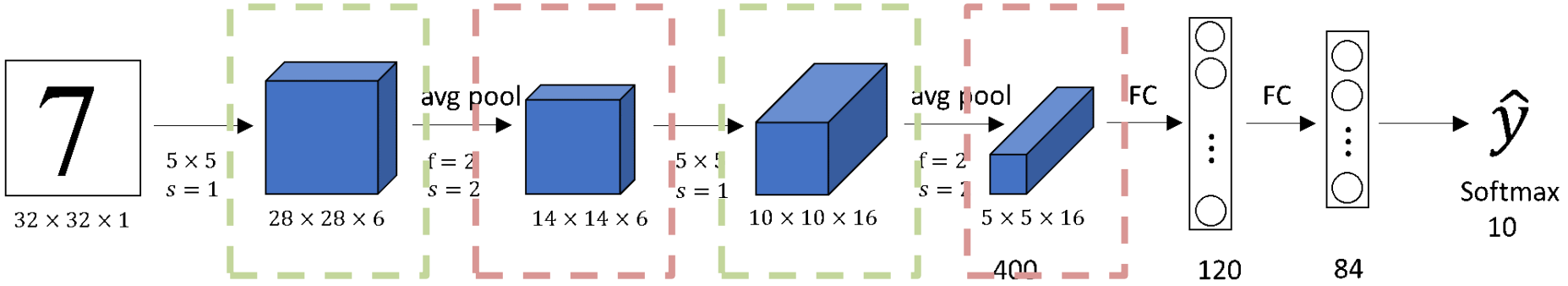
여러 개의 퍼셉트론이 겹쳐 있는 형태

3

CNN의 발전과정

CNN의 발전과정

LeNet-5



CNN의 태동과 같은 모델

2개의 FC Layer를 통과시켜 손글씨를 분류하는 모델

CNN의 발전과정

LeNet-5

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

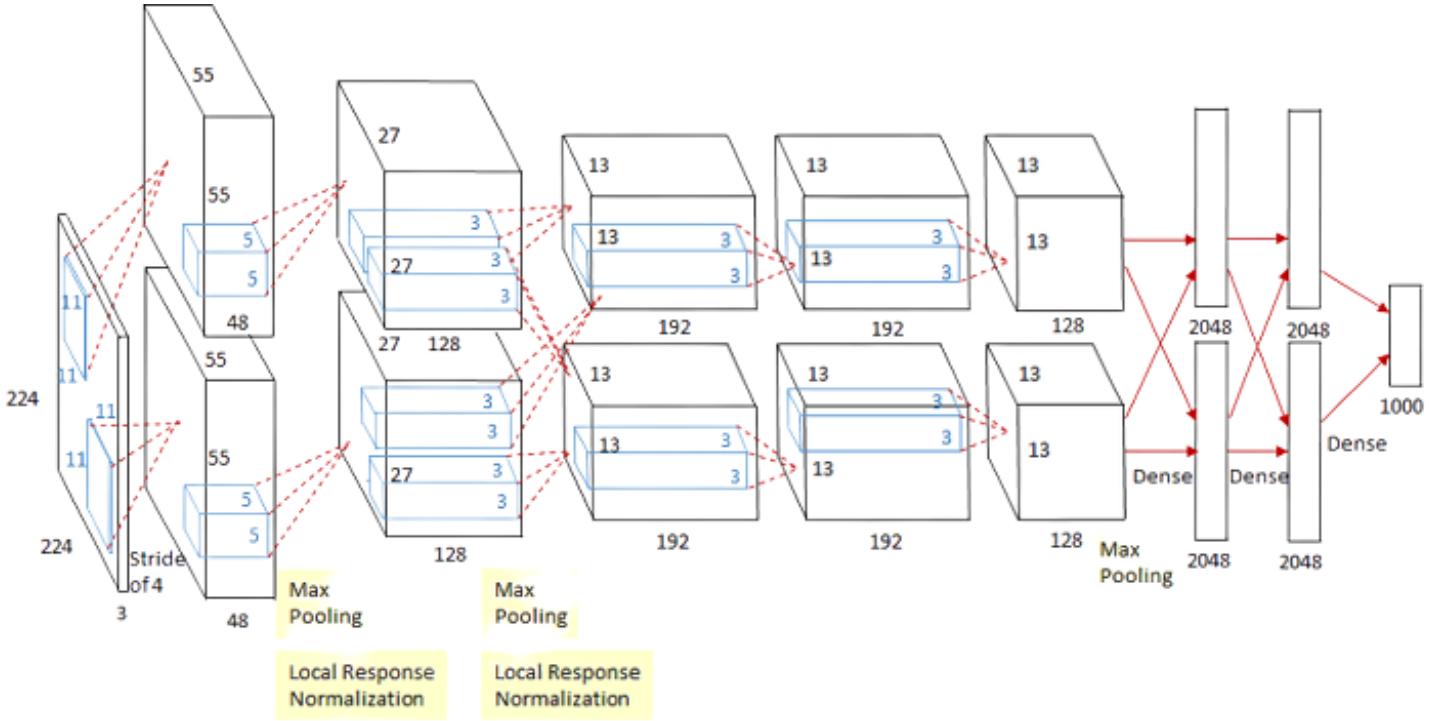
활성화 함수로 tanh 사용

(1, 32, 32)의 입력 이미지가 120으로 축소됨

Average pooling 사용

CNN의 발전과정

AlexNet



GPU의 한계로 병렬적 구조의 데이터

컬러 이미지로 채널과 픽셀 수 증가 → 학습의 연산량 증가

CNN의 발전과정

AlexNet

	LeNet-5	AlexNet
활성화 함수	tanh	ReLU
Pooling layer	Average pooling	Max pooling

ReLU 와 Max Pooling 사용

→ 연산 획기적으로 감소, 이미지 특징 효과적 추출

1주차 클린업 내용 참고!

CNN의 발전과정

AlexNet

	LeNet-5	AlexNet
활성화 함수	tanh	ReLU
Pooling layer	Average pooling	Max pooling

Local Response Normalization : 측면억제

Local Response Normalization과 Dropout 기법 사용

→ 성능 향상

CNN의 발전과정

AlexNet

Data Augmentation (데이터 증강 기법)

Data Augmentation

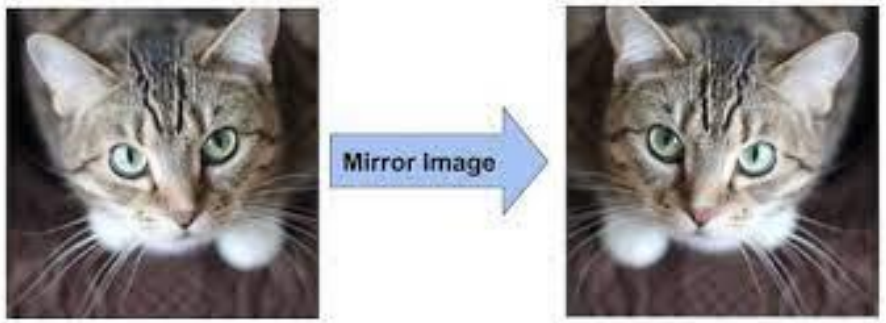
하나의 이미지를 사용해

여러 비슷한 이미지를 만들어내는 기법

CNN의 발전과정

AlexNet

Data Augmentation (데이터 증강 기법)



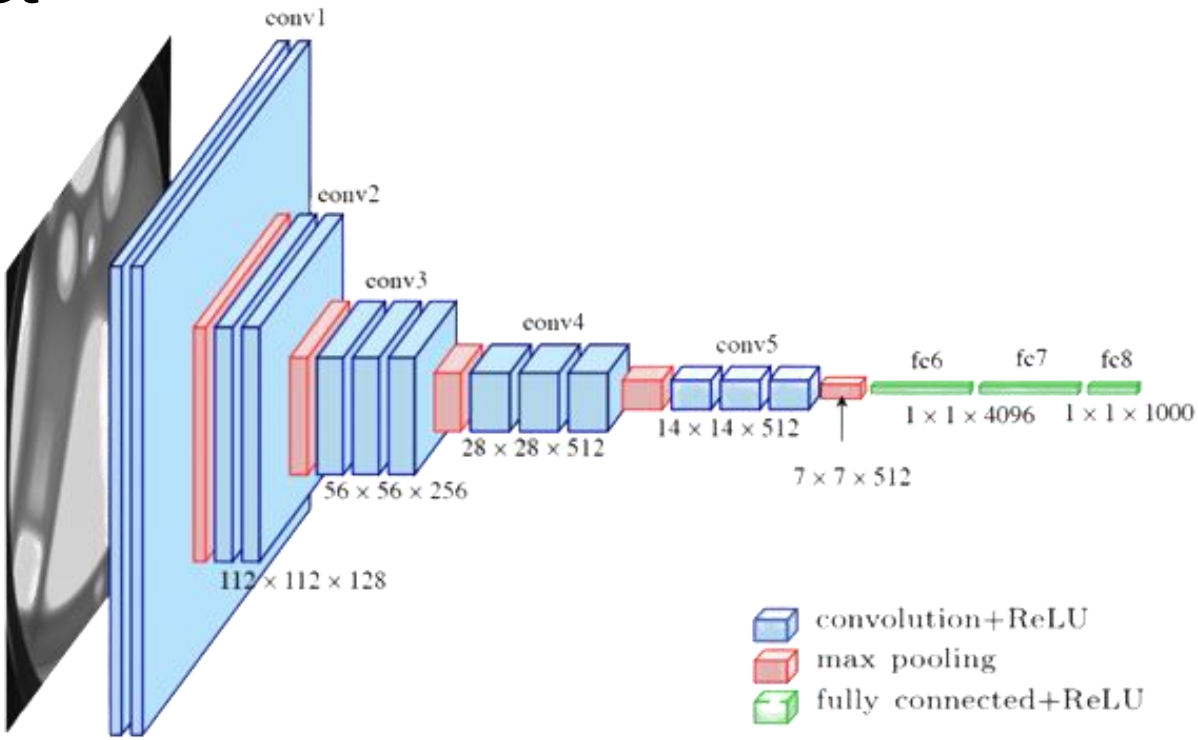
데이터를 좌우반전시키거나,

입력 이미지 크기보다 큰 이미지를 서로 다르게 잘라서 여러 장 만듦

→ 과적합 방지

CNN의 발전과정

VGGNet

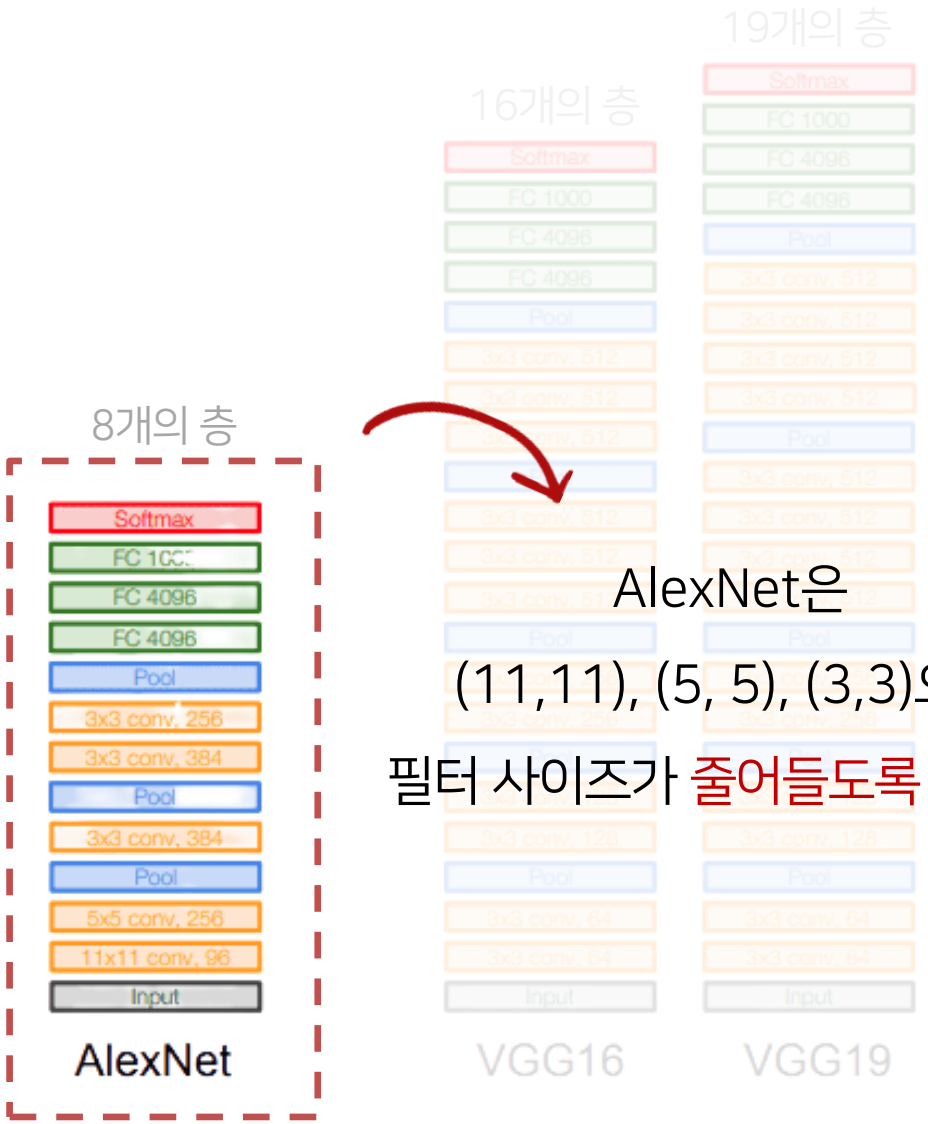


간단한 구조임에도 좋은 결과를 보인 모델

CNN에서 layer를 더 깊게 많이 쌓을 수 있다면 성능 향상이 가능함을 보여줌

CNN의 발전과정

VGGNet



AlexNet은

(11,11), (5, 5), (3,3)으로

필터 사이즈가 줄어들도록 구성됨

CNN의 발전과정

VGGNet

VGGNet은

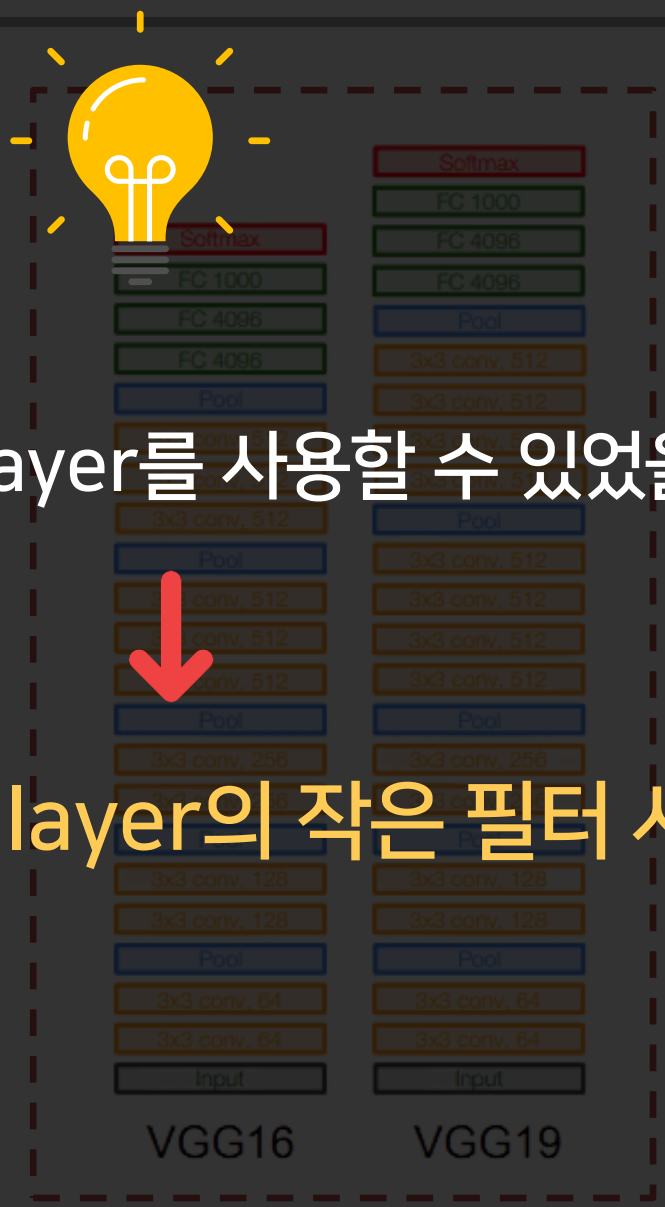
(3,3)의 필터

처음부터 끝까지 사용

Max pooling만 사용

CNN의 발전과정

VGGNet



어떻게 더 많은 layer를 사용할 수 있었을까?

VGGNet은

Convolution layer의 작은 필터 사이즈

처음부터 끝까지 사용

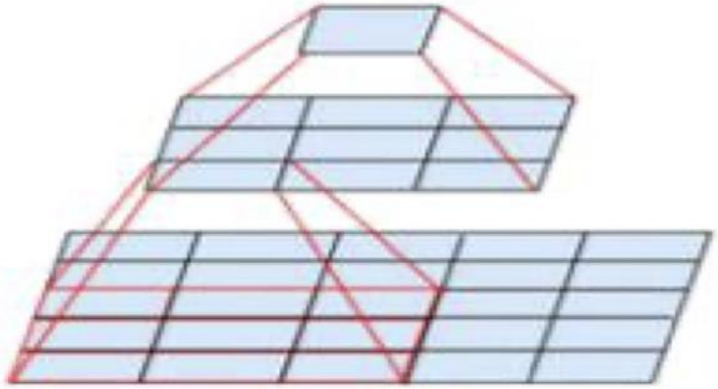
pooling만 사용

AlexNet

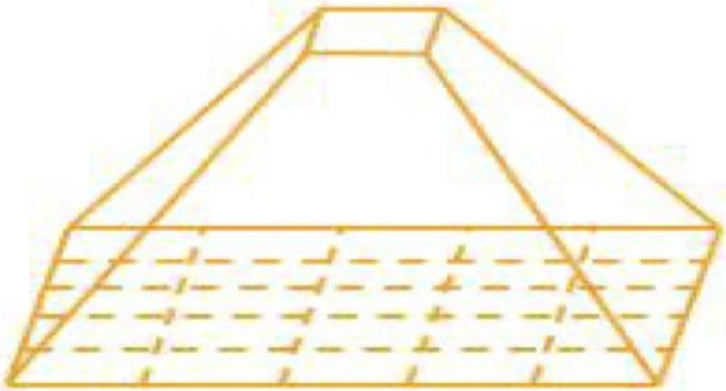
CNN의 발전과정

VGGNet

둘 다 (5,5) 데이터를 (1,1)로 변환



two successive
3x3 convolutions



5x5 convolution

Convolution layer 2번 통과

→ 비선형성 2번 추가

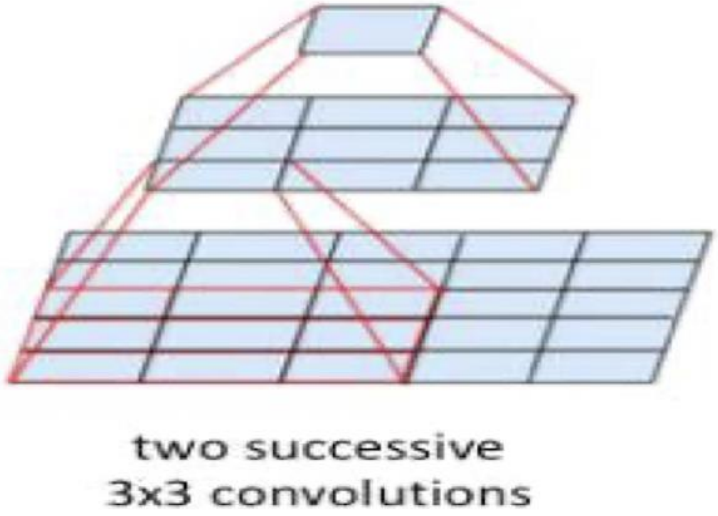
Convolution layer 1번 통과

→ 비선형성 1번 추가

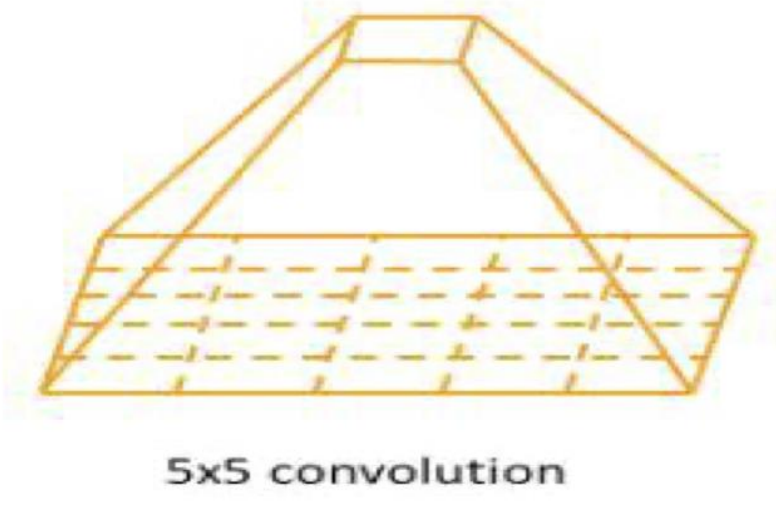
CNN의 발전과정

VGGNet

둘 다 (5,5) 데이터를 (1,1)로 변환



Convolution layer 2번 통과
파라미터 개수: 2x3x3xC
→ 비선형성 2번 추가



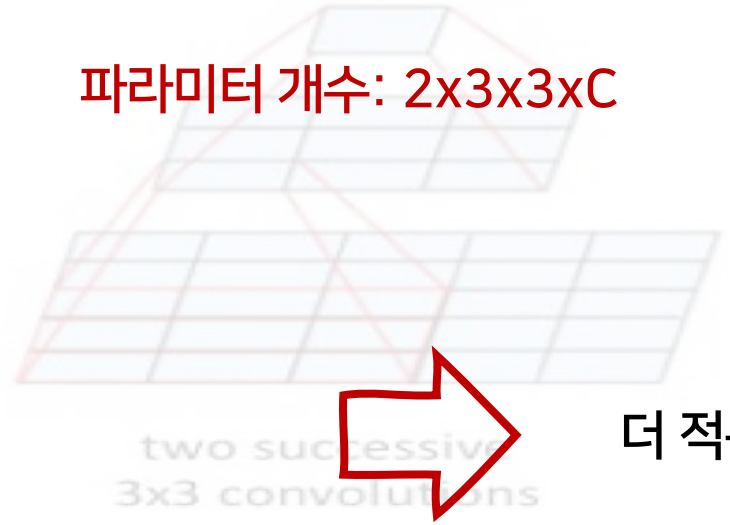
Convolution layer 1번 통과
파라미터 개수: 5x5xC
→ 비선형성 1번 추가

CNN의 발전과정

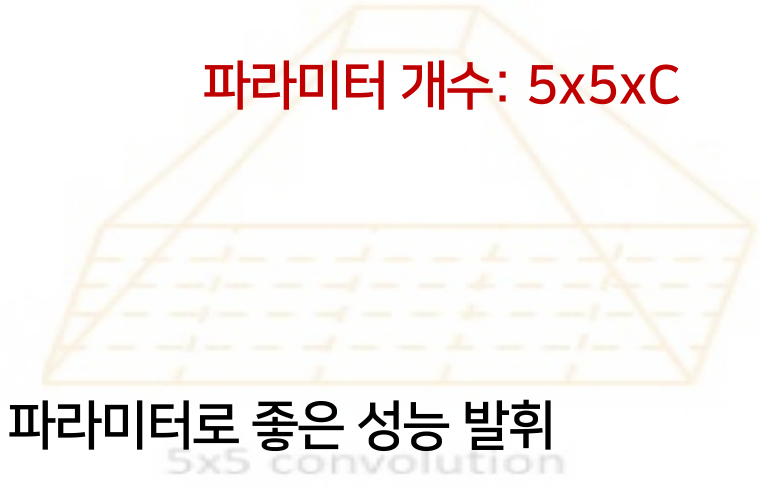
VGGNet

둘 다 (5,5) 데이터를 (1,1)로 변환

파라미터 개수: 2x3x3xC



파라미터 개수: 5x5xC



더 적은 파라미터로 좋은 성능 발휘

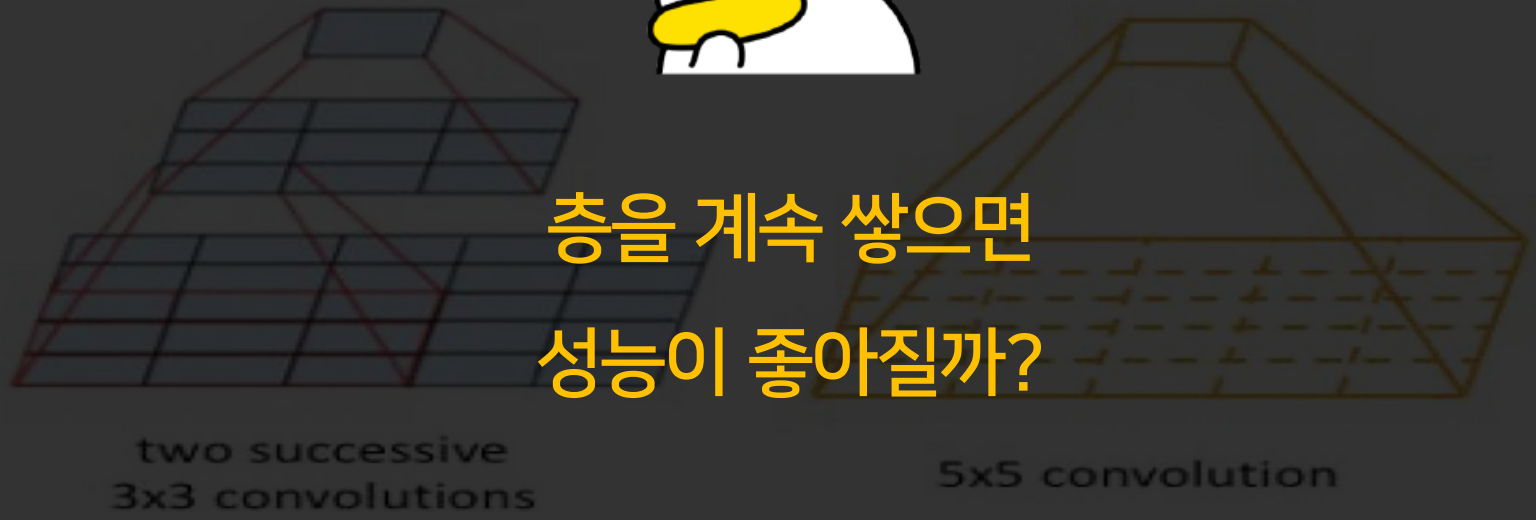
Convolution layer 2번 통과
→ 비선형성 2번 추가

Convolution layer 1번 통과
→ 비선형성 1번 추가

CNN의 발전과정

VGGNet

둘 다 (5,5) 데이터를 (1,1)로 변환



층을 계속 쌓으면

성능이 좋아질까?

Convolution layer 2번 통과

파라미터 개수: 3x3xC

→ 비선형성 2번 추가

Convolution layer 1번 통과

파라미터 개수: 5x5xC

→ 비선형성 1번 추가

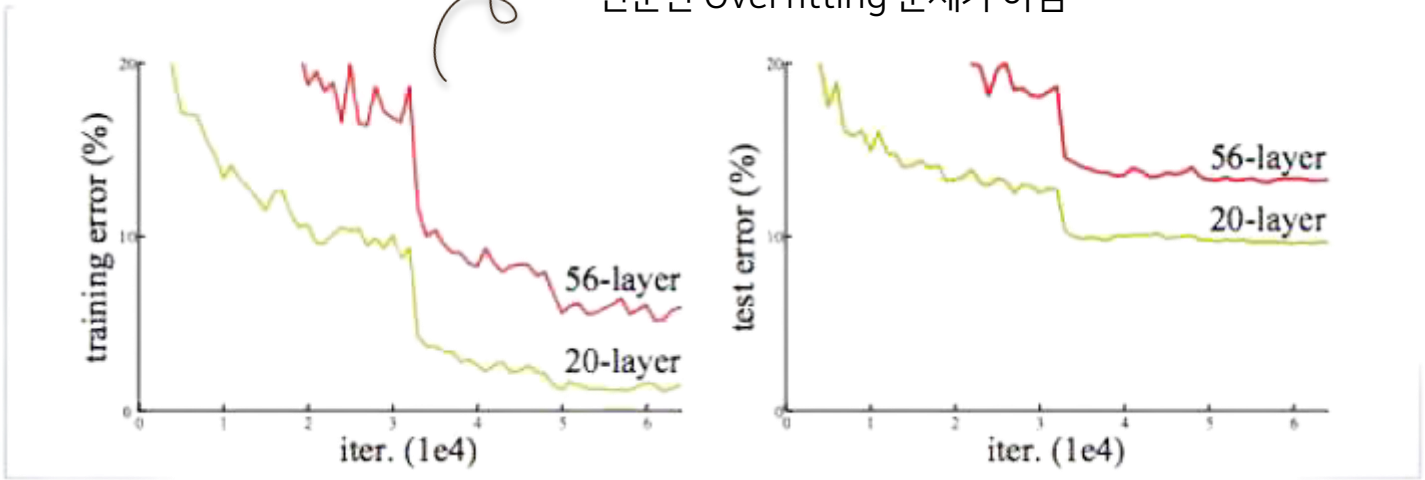
CNN의 발전 과정

ResNet

ResNet의 배경

Train, Test Error 모두 증가

단순한 Overfitting 문제가 아님



층이 많아졌지만 오히려 성능의 하락 발생

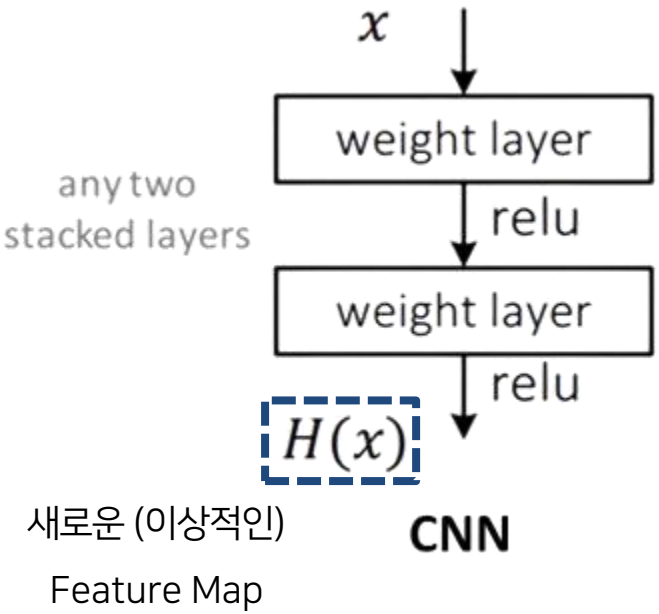
최적화 문제

Gradient Vanishing / Exploding 문제

CNN의 발전 과정

ResNet

Residual Learning



기존의 CNN

Input

(이전의 결과)

Convolution Layer 통과

새로운 Feature Map

$H(x)$ 반환

CNN의 발전 과정

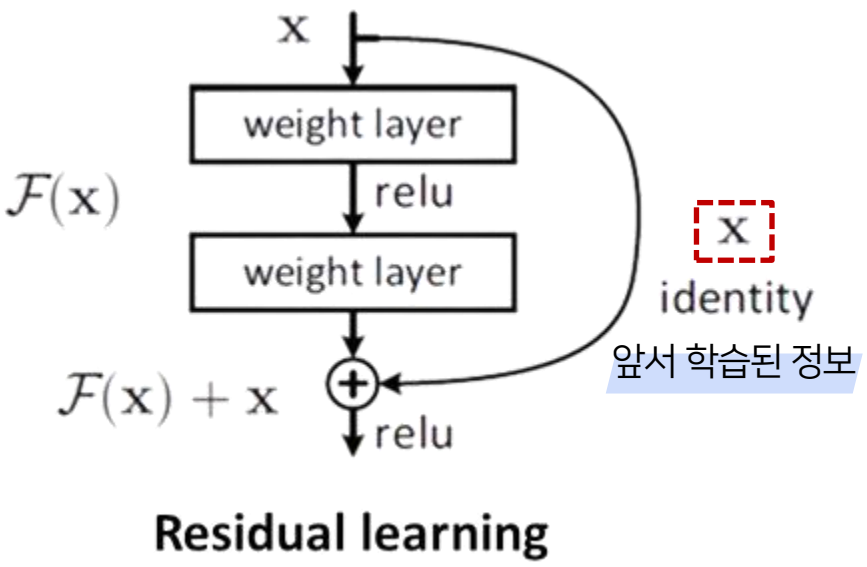
ResNet

Residual Learning

ResNet

이전 layer의 결과(x)를

그대로 출력층으로 전달



Residual learning

CNN의 발전 과정

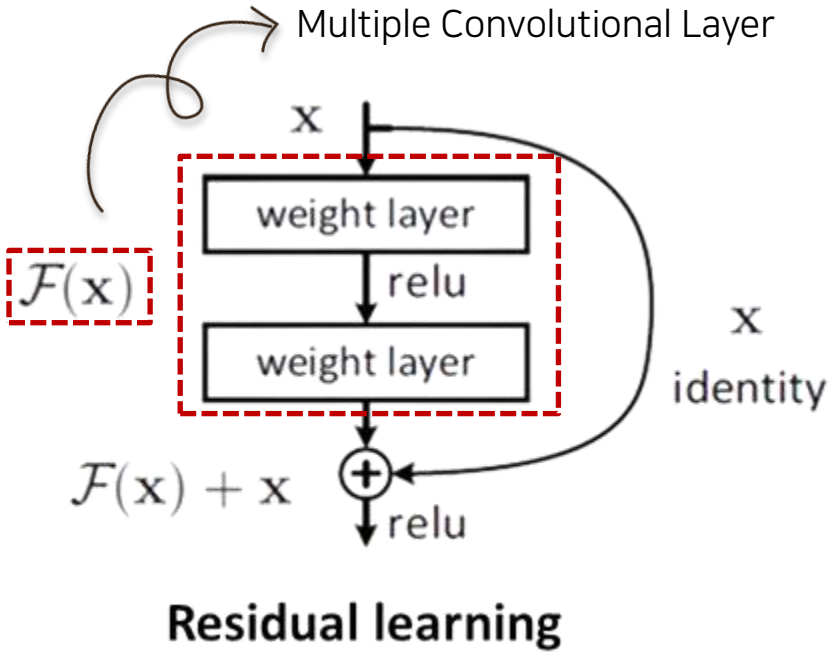
ResNet

Residual Learning

ResNet

x를 입력으로

Convolution Layer에 통과 ($F(x)$)



CNN의 발전 과정

ResNet

Residual Learning

ResNet

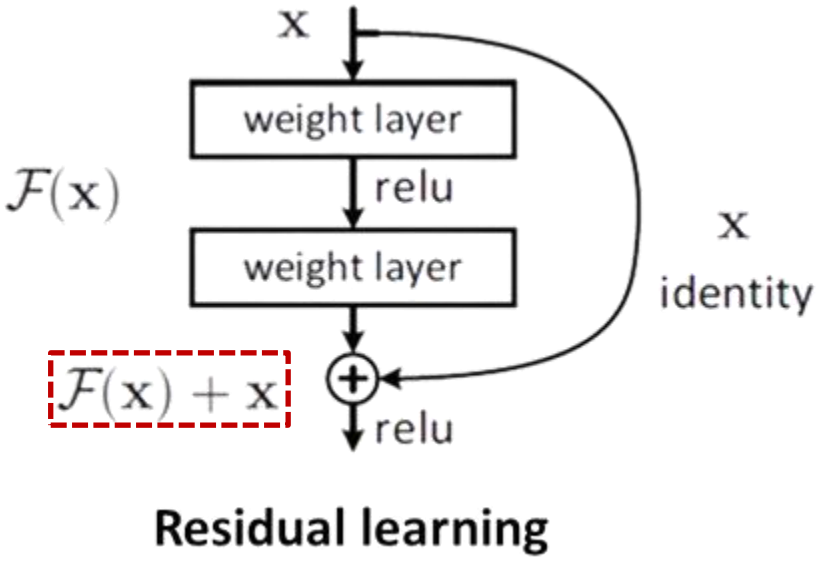
이전 layer의 결과(x)를
그대로 출력층으로 전달

x를 입력으로
Convolution Layer에 통과 ($F(x)$)

최종 Feature Map

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

학습하지 못한 추가 정보 학습



Residual learning

ResNet

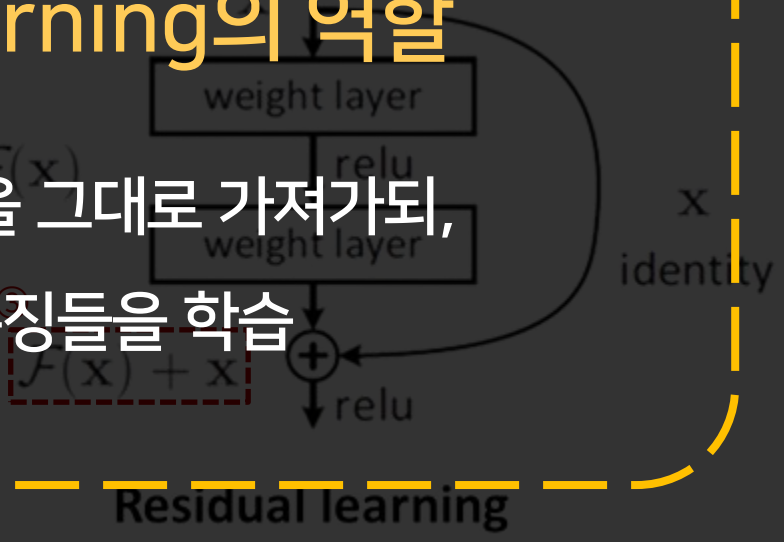
Residual Learning

ResNet

Residual Learning의 역할

이전 Layer의 출력을 그대로 가져가되,

부가적으로 특징들을 학습



③ $H(x) = F(x) + x$

학습하지 못한 추가 정보 학습

CNN의 발전 과정

ResNet

Residual Learning

ResNet

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

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

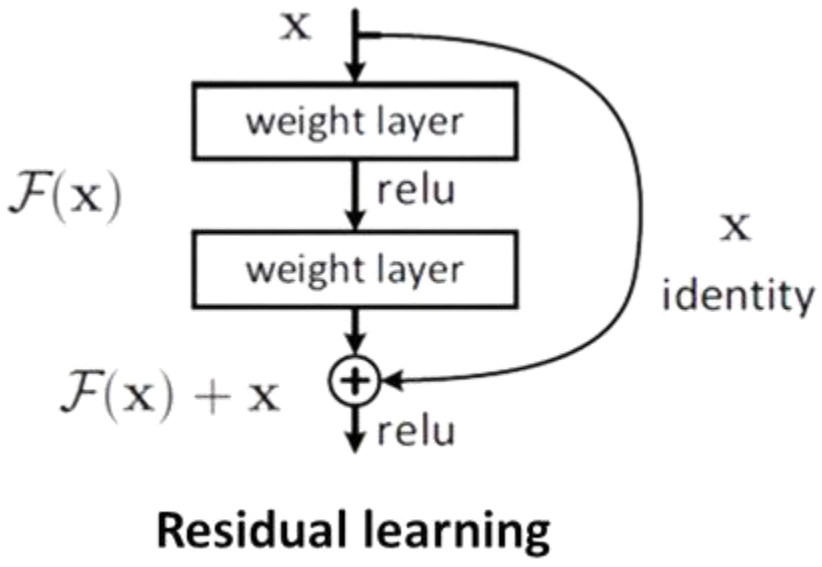
F(x) = 부가적으로 필요한 정보

= 학습의 대상

$$e = y - \hat{y}$$

Residual의 형태

이름이 ResNet인 이유!



Residual learning

ResNet

Residual Learning

역전파시 미분값

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

이전 Feature Map에서 학습되지 못한 $F(x)$을

최적화하는 방향으로 학습이 진행

CNN의 발전 과정

ResNet

Residual Learning

역전파시 미분값

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

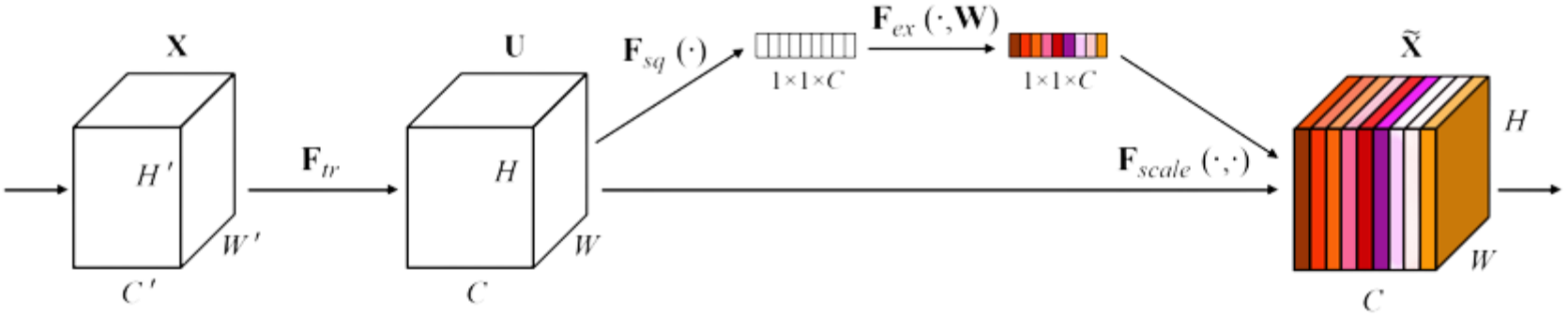
기존 x의 미분 값 = 1, Gradient Vanishing 예방

더 깊은 층 형성 가능

CNN의 발전 과정

SENet

SE Block



채널 간 상호작용을 통한 성능 향상

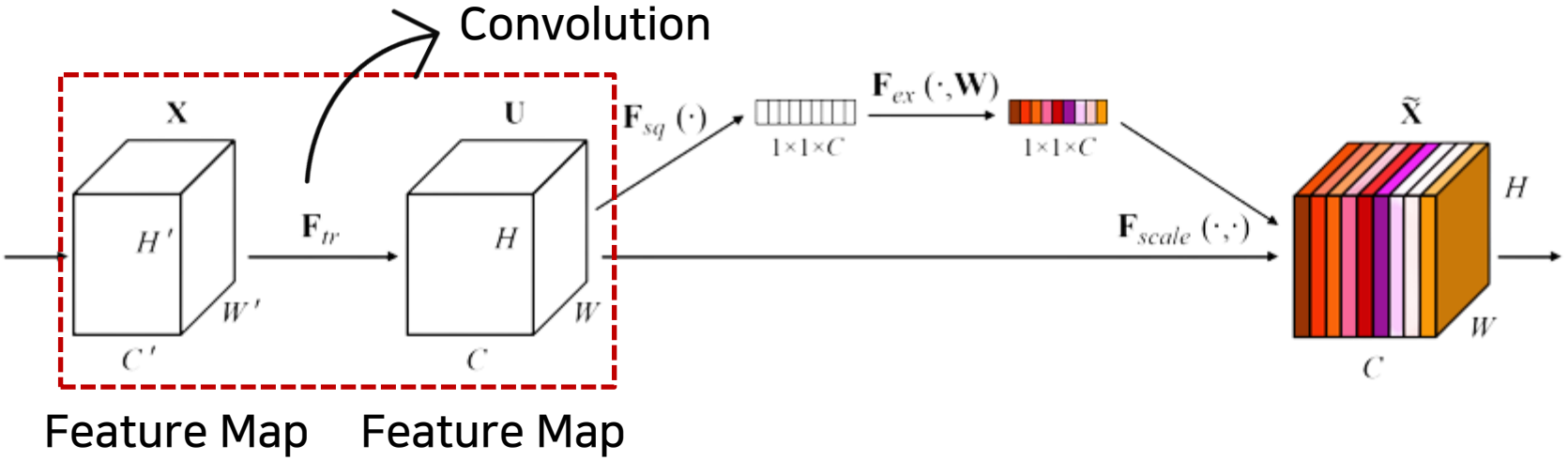
SE Block을 사용해 다양한 CNN 모델에 적용 가능

획기적인 성능 향상에 비해 연산이 많지 않음

CNN의 발전 과정

SENet

SE Block

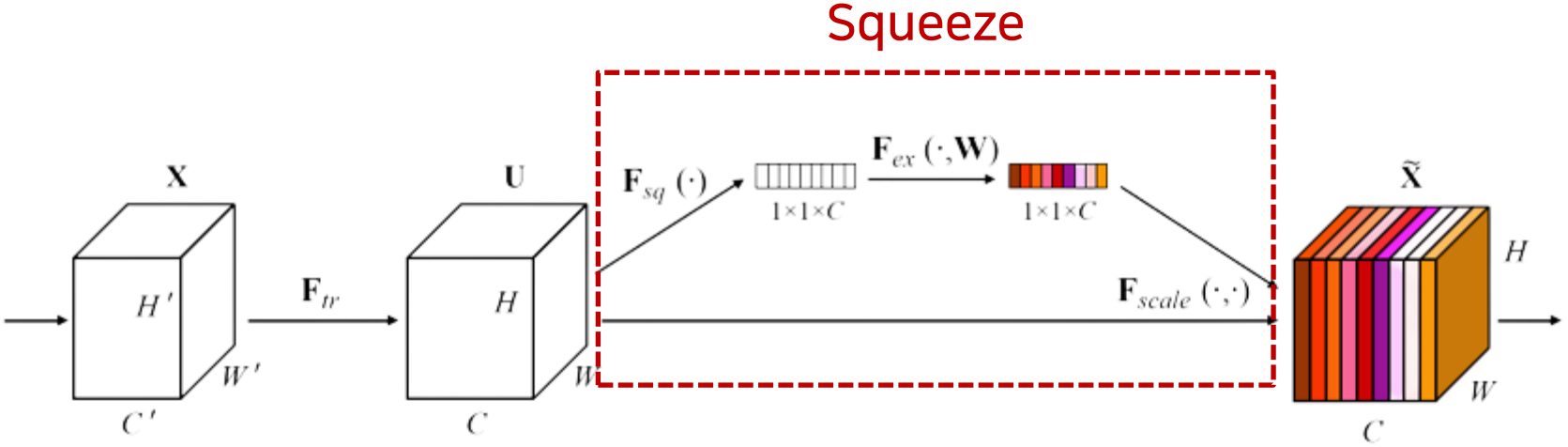


Convolution 통해 크기 U의 Feature map으로 변환

CNN의 발전 과정

SENet

SE Block



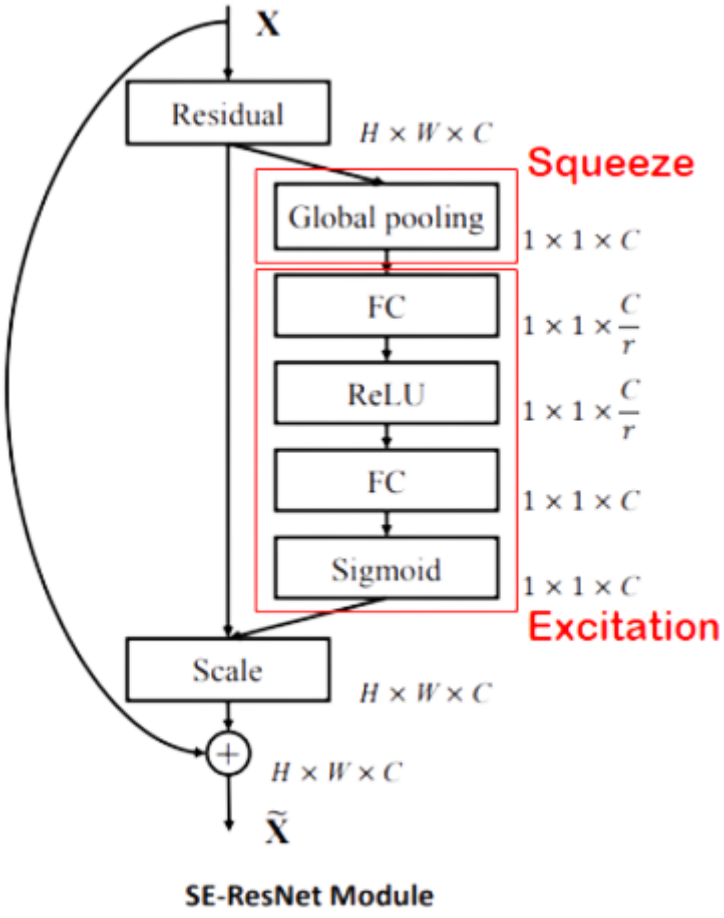
Squeeze!

각 채널을 1차원의 scalar로 변환

CNN의 발전 과정

SENet

SE Block



SE-ResNet Module

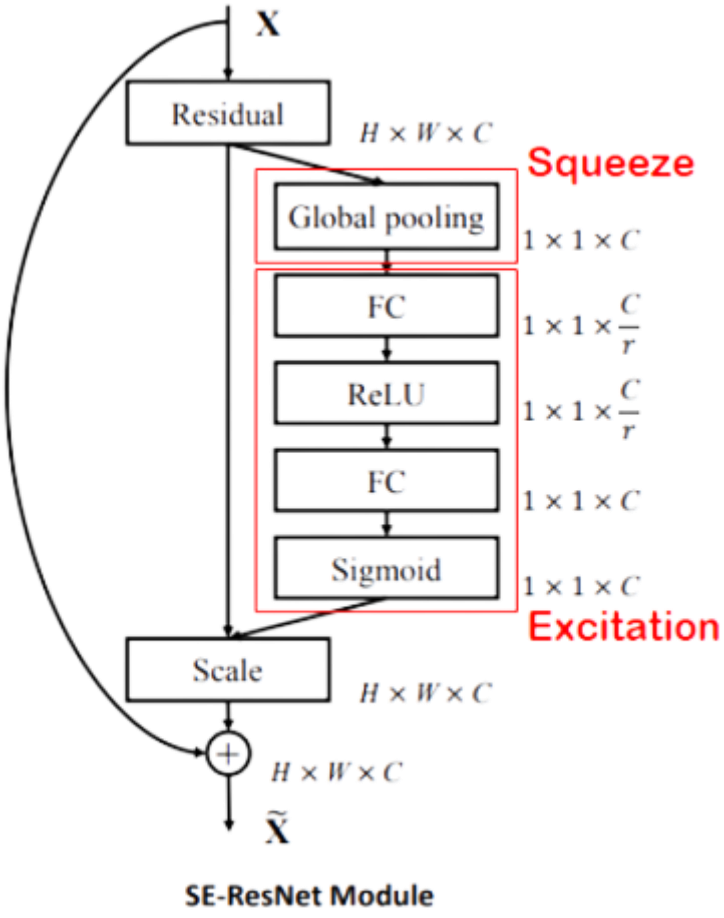
활성화 작업 시작

1x1xC의 벡터를 정규화해
가중치를 부여

CNN의 발전 과정

SENet

SE Block



SE-ResNet Module

FC1

1x1xC의 벡터를 입력받아

C개의 채널을

C/r개의 채널로 축소

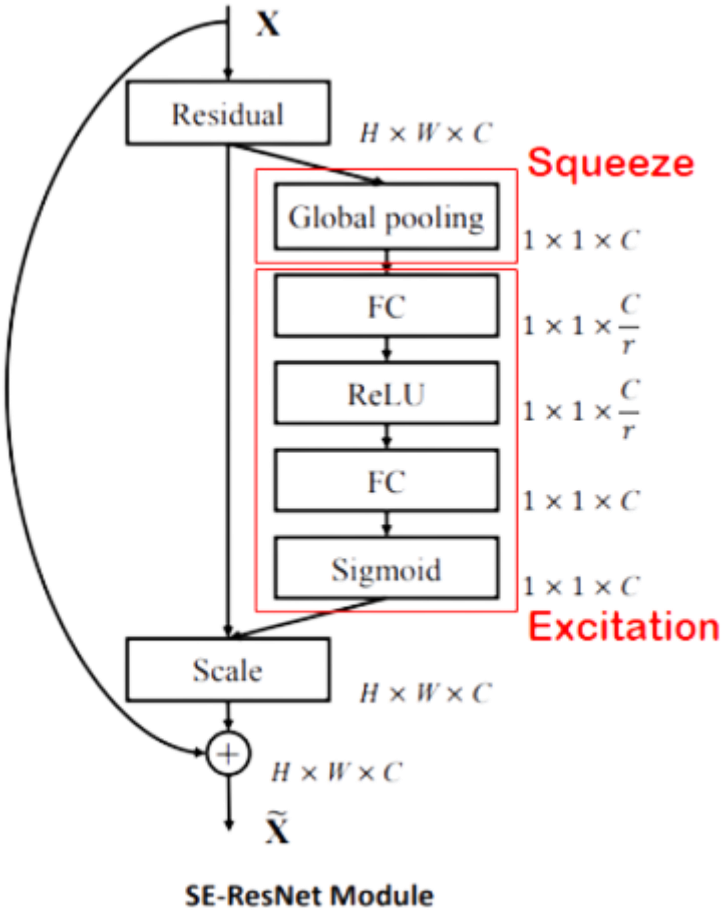
연산량 제한과 일반화 효과

r : 하이퍼 파라미터 값

CNN의 발전 과정

SENet

SE Block



SE-ResNet Module

ReLU

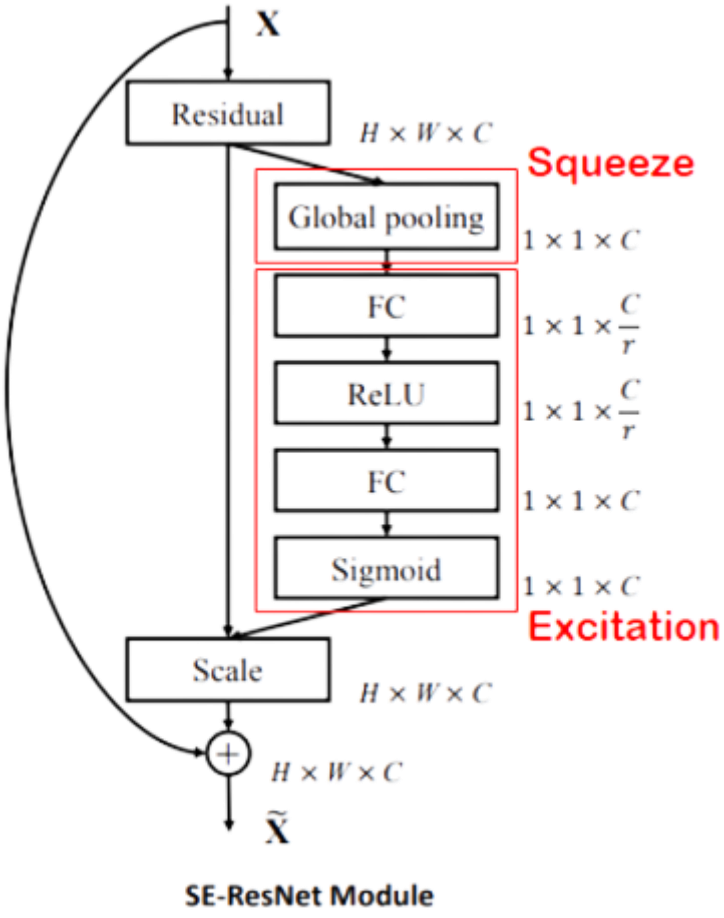
1x1xC/r의 벡터를

ReLU로전달

CNN의 발전 과정

SENet

SE Block



SE-ResNet Module

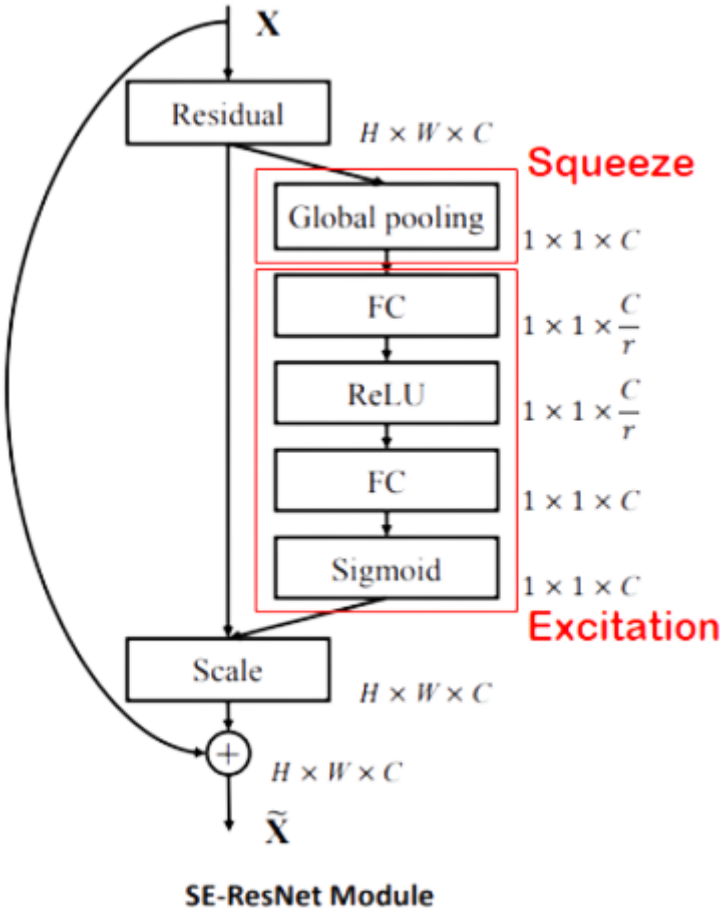
FC2

FC2를 통과하며
1x1xC/r의 벡터의 채널 수를
다시 C로 되돌림

CNN의 발전 과정

SENet

SE Block



SE-ResNet Module

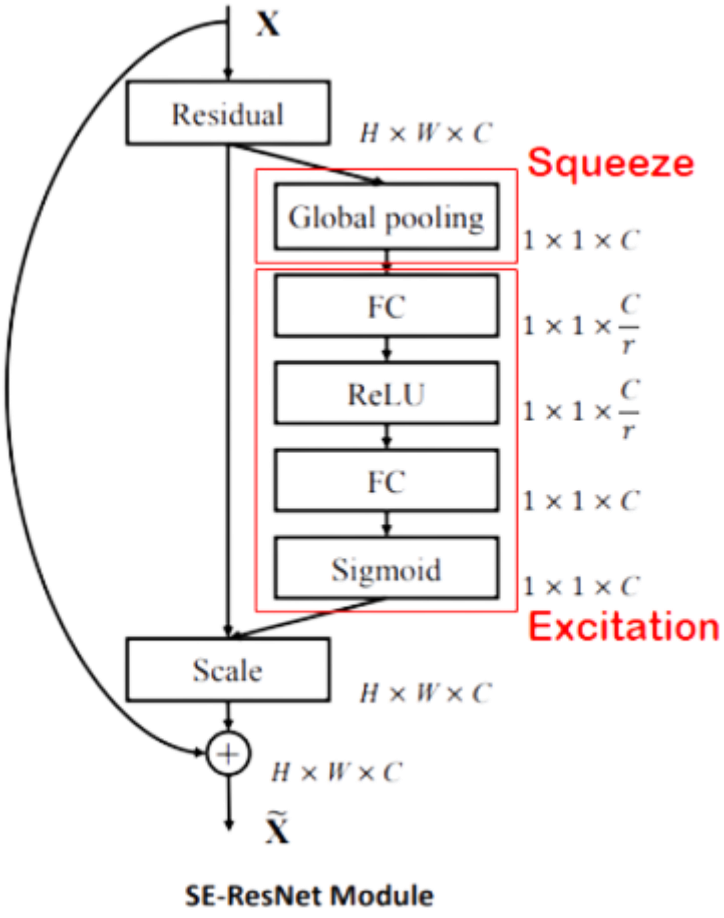
Sigmoid

Sigmoid를 통과하며

0 ~ 1 사이의 값 반환

SENet

SE Block



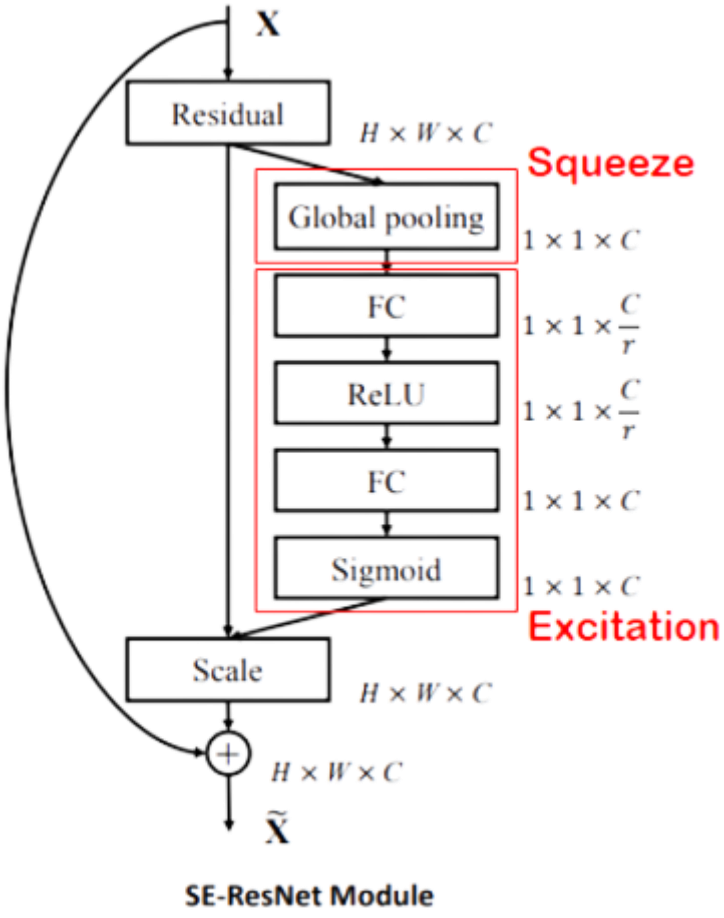
SE-ResNet Module

Sigmoid를 통과한 값은

각 채널 사이의 중요도를 의미

SENet

SE Block



SE-ResNet Module

반환된 중요도를

Feature Map과 곱해

Feature Map 채널에 가중치 부여

SENet

SE Block

SE-ResNet Module

SENet의 의의

기존의 CNN에 SE Block을 붙이며

Convolution을 통해 생성된 특성을

채널 당 중요도를 고려해서 재보정한 것

반환된 중요도를

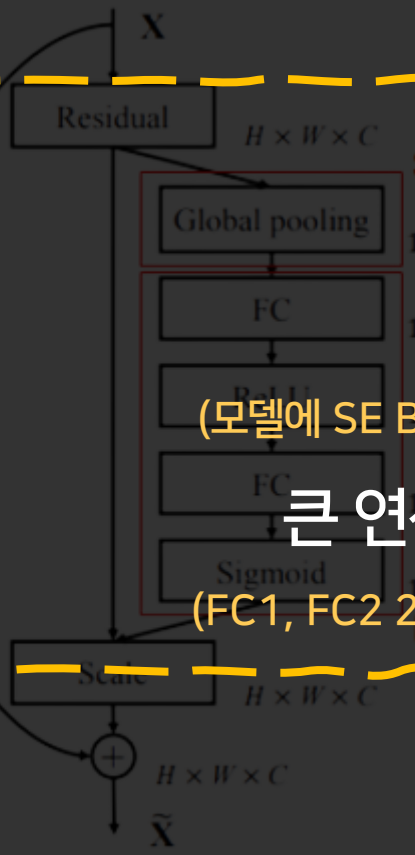
Feature Map과 곱해

각 채널에 가중치 부여

CNN의 발전 과정

SENet

SE Block



SE-ResNet Module

SENet의 의의

반환된 중요도를

높은 확장성

Feature Map과 곱해

(모델에 SE Block을 추가하는 것만으로 간편하게 생성)

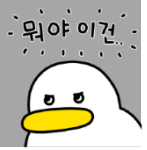
Feature Map의 채널에 가중치 부여

큰 연산의 증가 없이 성능 향상

(FC1, FC2 2개의 Fully Connected Layer만을 추가)

4

컴퓨터 비전



컴퓨터 비전

컴퓨터가 시각적인 체계를 이해하고 해석할 수 있도록
컴퓨터를 학습시키는 인공지능의 연구 분야

Object Detection

객체 탐지

여러 객체가 존재할 때도 분류할 수 있어야 함

이미지가 주어졌을 때, 어떤 물체가 어디에 있는지 탐지하는 것

Localization

객체가 존재하는

범위 탐지

Classification

탐지된 개체에 대한

분류 담당

Object Detection

객체 탐지

여러 객체가 존재할 때도 분류할 수 있어야 함

이미지가 주어졌을 때, 어떤 물체가 어디에 있는지 탐지하는 것

Localization

객체가 존재하는

범위 탐지

+

Classification

탐지된 개체에 대한

분류 담당

두 작업이 모두 일어나야 함

Object Detection

Bounding Box (bbox)

객체의 위치를 표시하는 경계 상자

Object Detection

종류

이미지가 주어졌을 때, 어떤 물체가 어디에 있는지 탐지하는 것

1-Stage Detector

Localization과
Classification 문제를
동시에 해결

빠른 수행 속도
상대적으로 떨어지는 정확도

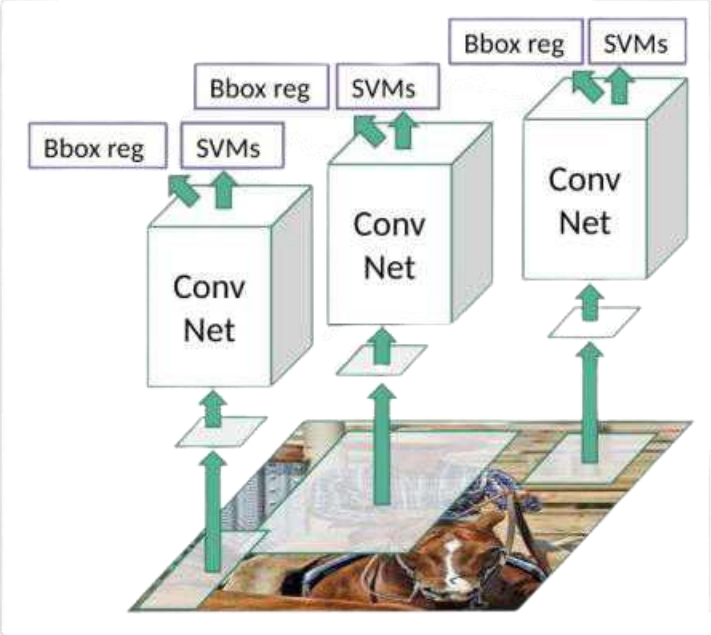
2-Stage Detector

Localization으로
객체의 위치 탐지 후
Classification을 통해 대상 분류

상대적으로 느린 속도
높은 정확도

RCNN

Regions with Convolutional Neuron Networks features



2-Stage Detector 모델의 가장 기본적인 형태

RCNN

작동 알고리즘

RoI(Region of Interest): 이미지에서 객체가 존재할 것으로 예상되는 위치

Region Proposal

이미지에서 ROI 생성

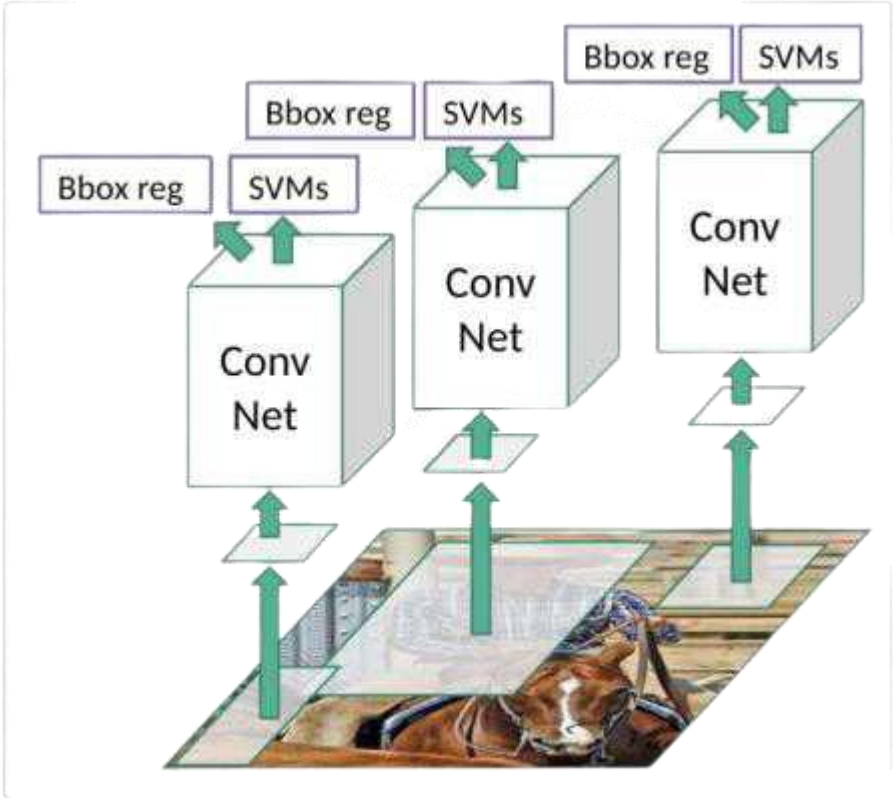
Selective Search 사용

Selective Search

작고 랜덤한 bbox를 많이 생성해,

계층적 그룹핑 알고리즘을 통해

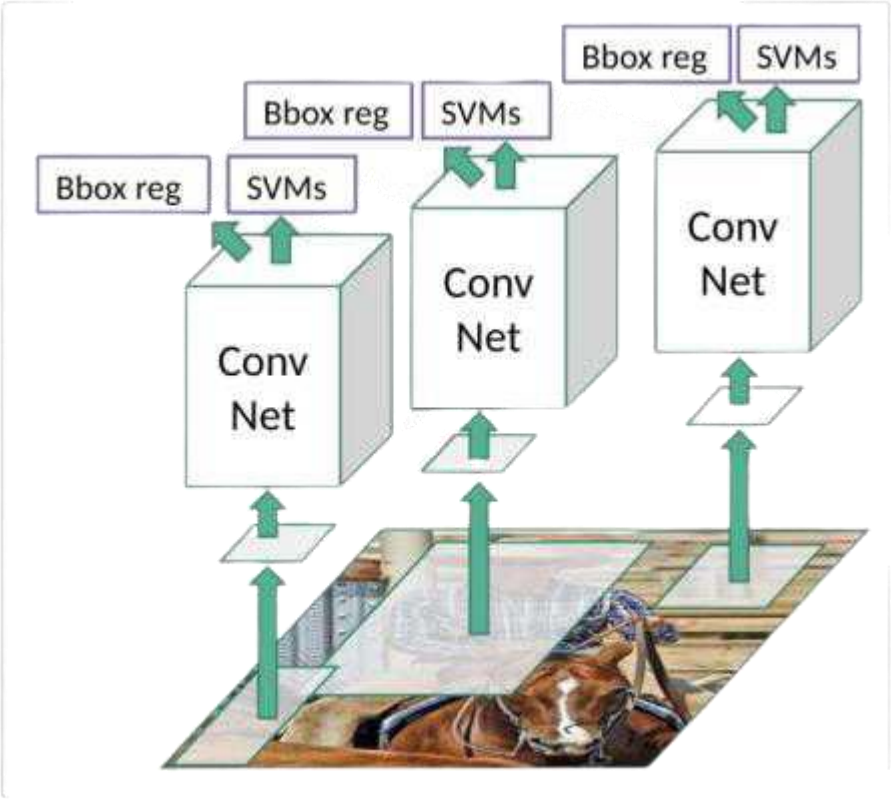
조금씩 합치며 ROI 생성



RCNN

작동 알고리즘

ROI를 CNN에 통과시켜
Feature map을 반환해
회귀와 분류 작업 진행



RCNN

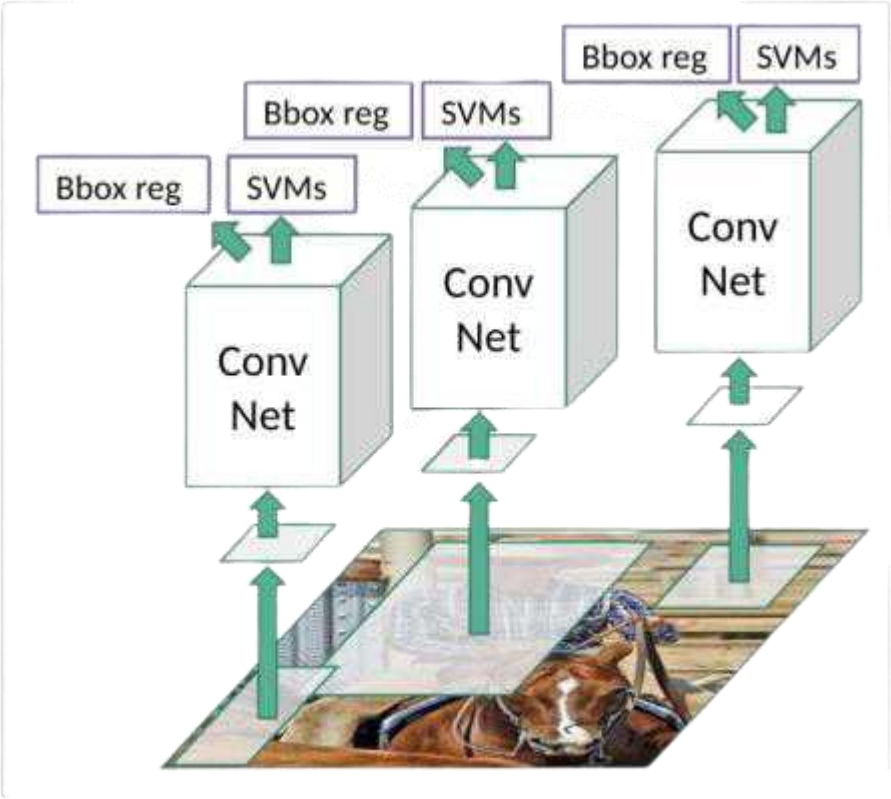
작동 알고리즘

SVM을 통해 각 ROI의 라벨 예측

CNN feature를 이용해

localization 에러 감소를 위해

bbox regression model 수정



RCNN

작동 알고리즘

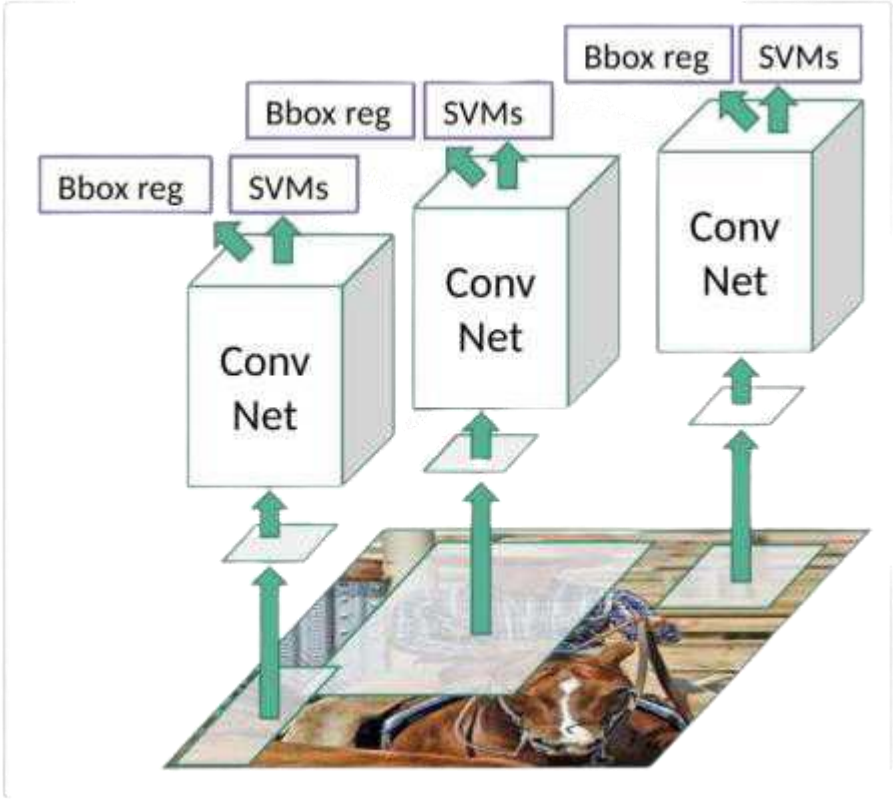
SVM을 통해 각 ROI 레벨 예측

CNN 사용해

localization 에러를 감소시키며

bbox regression model 수정

과정 반복!!



RCNN

작동 알고리즘

RCNN의 단점

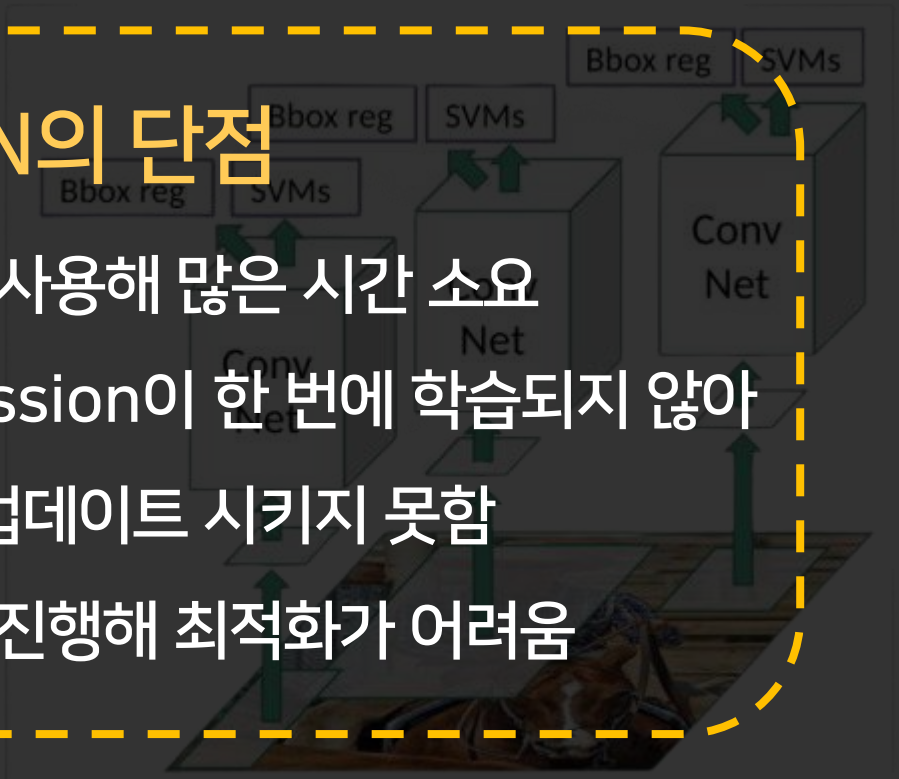
여러 CNN 모델을 사용해 많은 시간 소요

과정 반복!!

CNN, SVM, bbox regression이 한 번에 학습되지 않아

결과가 CNN을 업데이트 시키지 못함

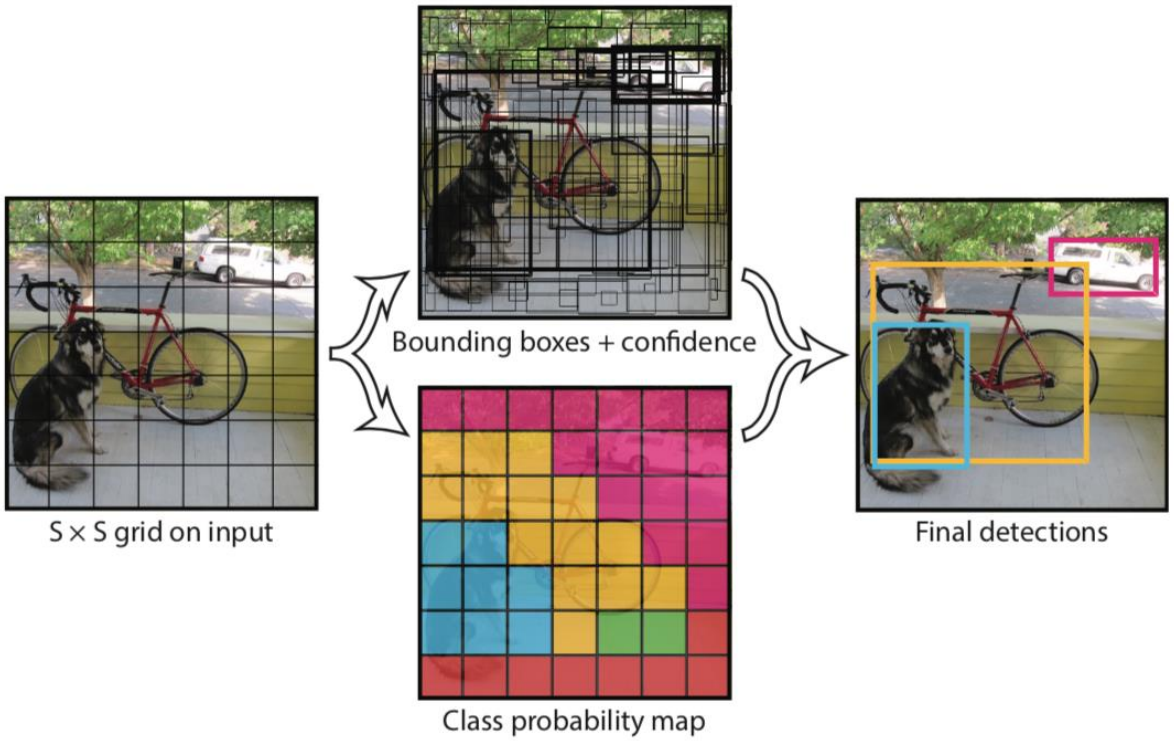
회귀와 분류를 모두 진행해 최적화가 어려움



YOLO

You Only Look Once

1-Stage Detector 모델의 가장 기본적인 형태



YOLO

You Only Look Once

이미지 전체에 하나의 모델이 한 번의 연산을 통해
bbox와 객체의 라벨 확률을 동시에 반환

CNN

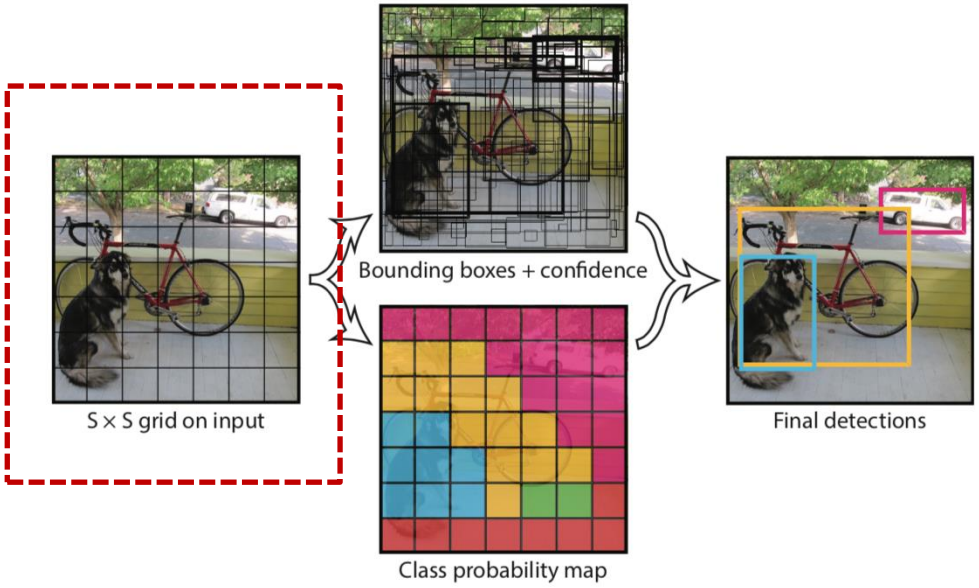
이미지를
여러 장으로
분할해 해석

YOLO

이미지 전체를 한 번만 보고
객체의 위치와
그 객체를 예측

YOLO

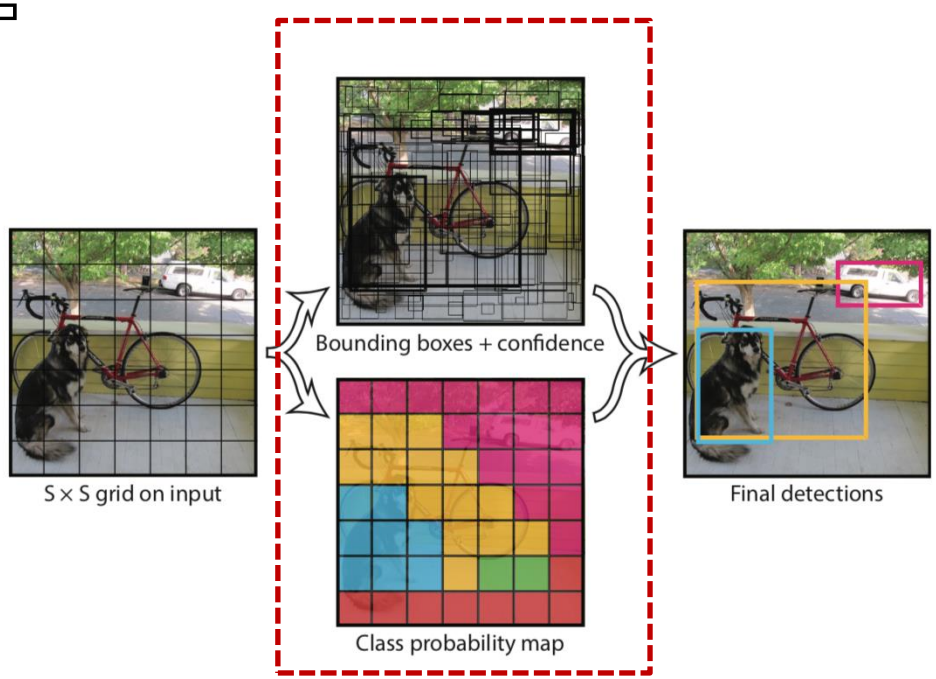
작동 알고리즘



Step 1. 원본 이미지를 동일한 그리드로 분할

YOLO

작동 알고리즘

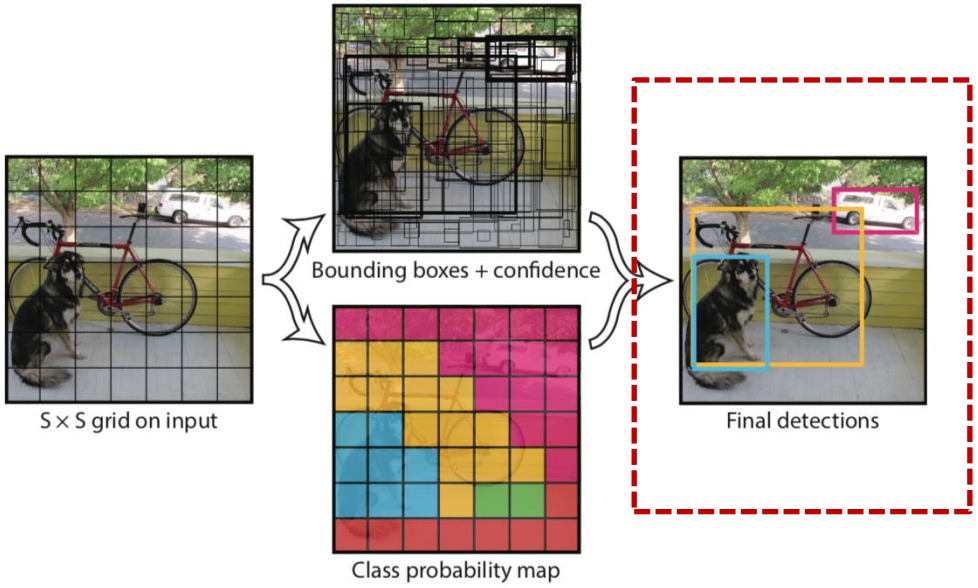


Step 2. 각 그리드별로 Bbox와 Confidence score 계산

동시에 가장 높은 확률의 클래스 선별

YOLO

작동 알고리즘



Step 3. 일정 확률 이상의 셀을 연결해 bbox와 라벨을 반환

YOLO

특징

장점

모델이 통합되어 있어 간단

기존의 모델보다 빠른 계산 시간

단점

2-Stage Detector 모델에 비해 떨어지는 정확도

작은 개체의 탐지가 어려움

Image Segmentation

정확한 위치를 요구하므로, Object Detection보다 더 복잡한 계산을 요구

픽셀 단위로 이미지의 Classification을 계산

INSTANCE SEGMENTATION

SEMANTIC SEGMENTATION

Image Segmentation

픽셀 단위로 이미지의 Classification을 계산

Semantic Segmentation

물체가 어디에 속하는지에
대해서만 분류를 진행

Image Segmentation

픽셀 단위로 이미지의 Classification을 계산

Instance Segmentation

같은 class 내에서도
더 세부적으로 분류 진행

겹쳐 있는 물체에 대해서도 세밀한 분류가 가능

Image Segmentation

입력된 이미지의 각 픽셀별로 class에 대한 출력을 반환

입력 이미지 크기 = 출력 이미지 크기

입력

segmented

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

출력

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

Segmentation map

Image Segmentation

CNN : 층을 거치면서 이미지의 크기 감소해

Image Segmentation을 그대로 사용할 수 없음

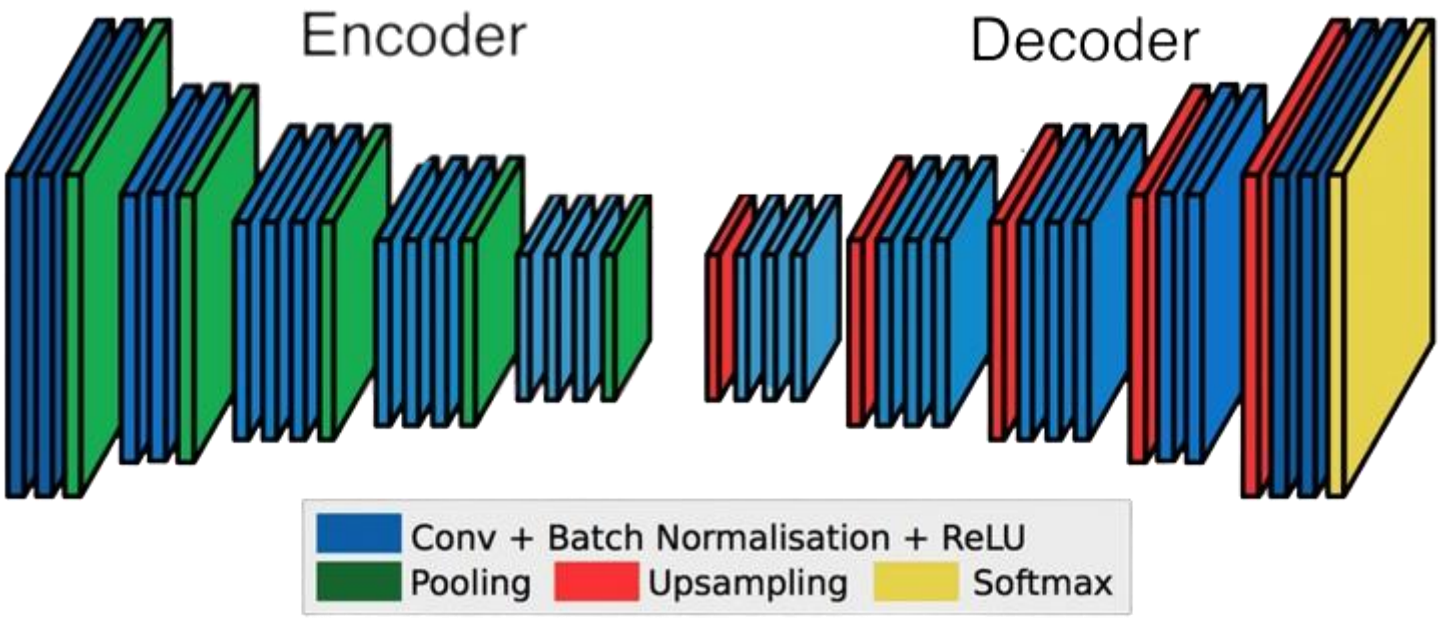


Image Segmentation

Convolution과 pooling을 역으로 연산해
이미지의 크기를 키우는 방법 사용!

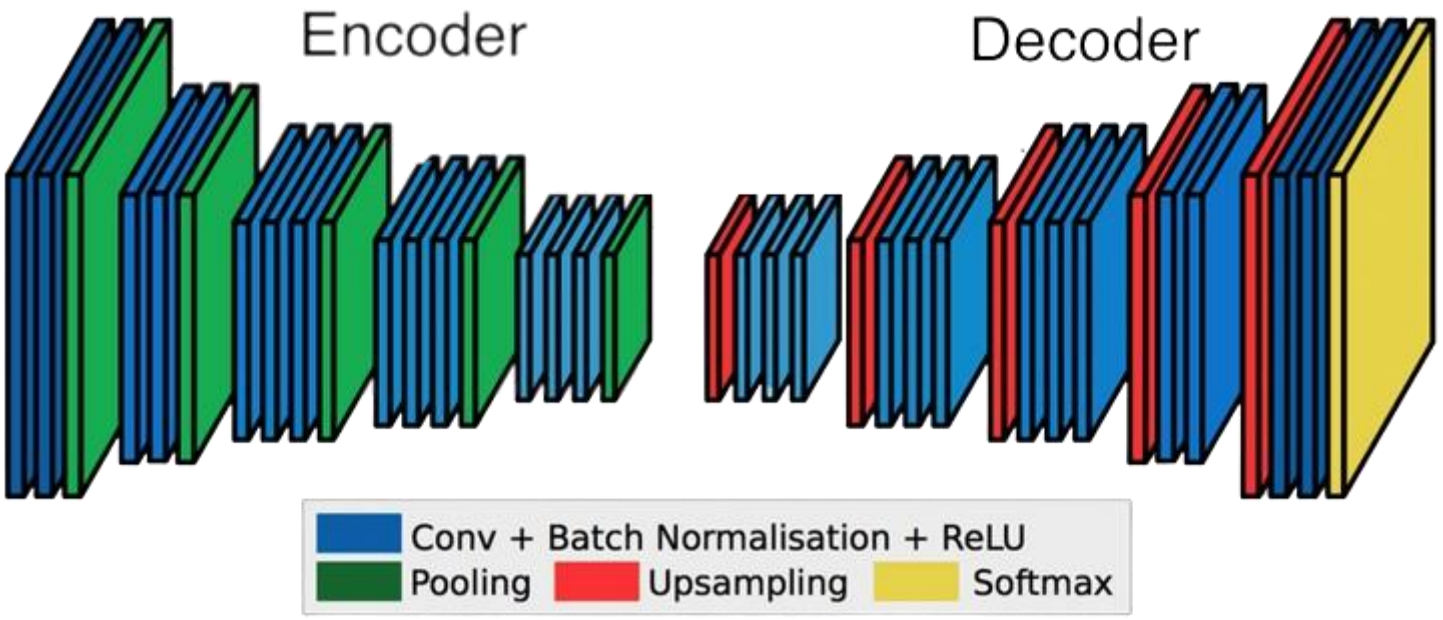
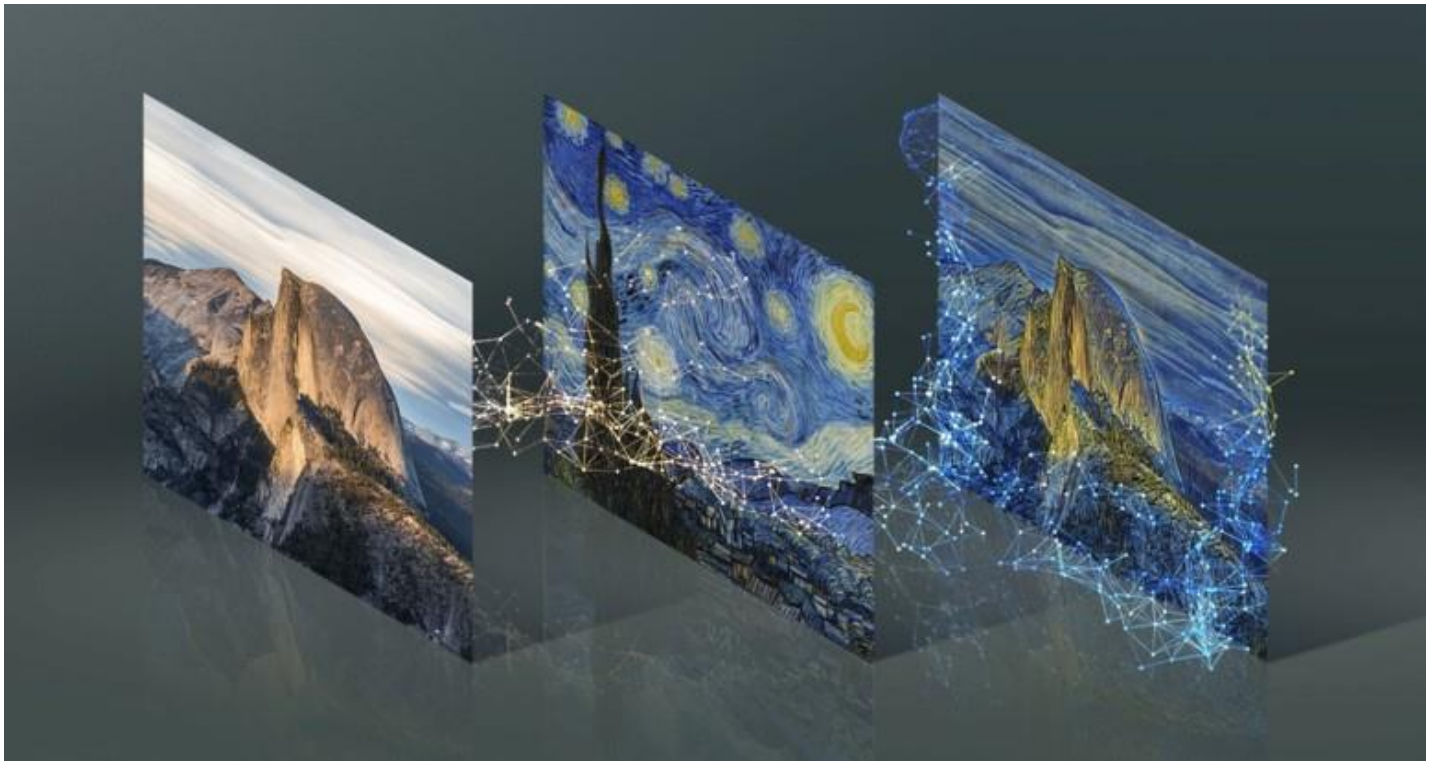


Image Generation

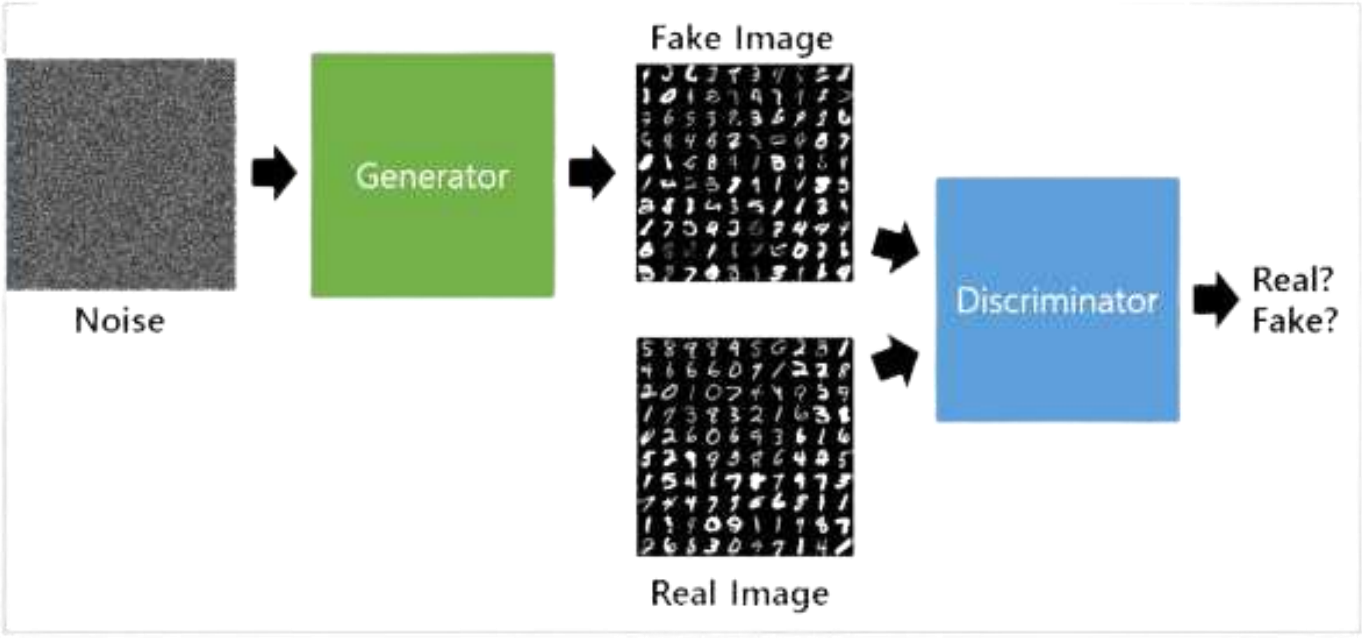
비지도 학습 딥러닝 모델 등장

주어진 이미지를 바탕으로 새로운 이미지 생성



GAN

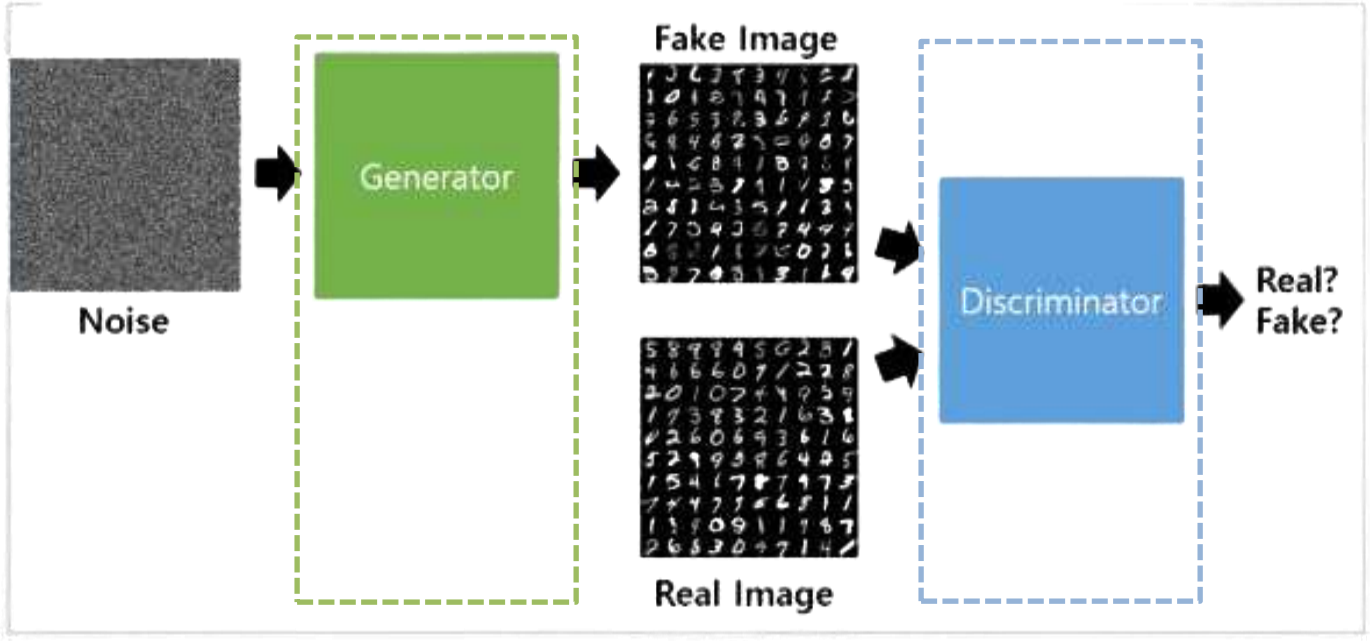
Generative Adversarial Network



Generator와 Discriminator를 통해, 진짜 같은 가짜 이미지 생성

GAN

Generative Adversarial Network



Generator

Discriminator를 속일 수 있는 가짜 이미지 생성

Discriminator

실제 이미지와 가짜 이미지를 구별

컴퓨터 비전

GAN

Generative Adversarial Netwo

서로 적대적으로 학습하며 성능 향상

궁극적으로 진짜 같은 가짜 이미지 생성

Generator

가짜 이미지 생성

Discriminator

실제 이미지와

가짜 이미지를 구별

Style Transfer

Content target

입력 (content)

+

Style reference

특징 추출

=

Combination image

이미지 생성

Style 이미지의 특징을 추출해 Content 이미지에 적용하여
새로운 이미지 생성

Style Transfer

Content target

입력 (content)

+

Style reference

특징 추출

=

Combination image

이미지 생성

손실함수

Content Difference

Output 이미지와 Content 이미지의 차이,

Output 이미지와 Style 이미지의 차이를

Style Difference

모두 줄이는 것이 목표

THANK YOU