

# CV팀

CV팀  
강철석  
박상훈  
박채원  
방건우  
이현진

# INDEX

---

1. 이미지 데이터
2. CNN
3. 발전된 CNN 모델
4. Object Detection

1

이미지 데이터

# 1

## 이미지 데이터

### 컴퓨터가 이미지를 저장하는 방법



우리가 컴퓨터를 통해 보고 있는 거의 모든 이미지는  
**픽셀**이라고 불리는 작은 점들 안의 숫자들로 표현됨

## 이미지 데이터

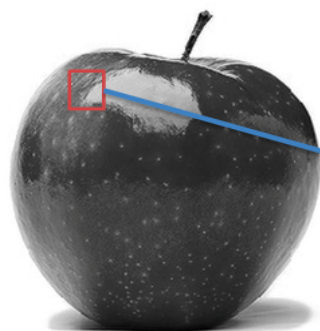
1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0
1	1	1	1	1	1	1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	1
1	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0
1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	1	0
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0
1	1	1	1	1	1	1	1	1	0	1	1	0	1	1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

Binary Image의 경우 **흰색을 1, 검은색을 0으로 표현함**

## 1

## 이미지 데이터

## 컴퓨터가 이미지를 저장하는 방법



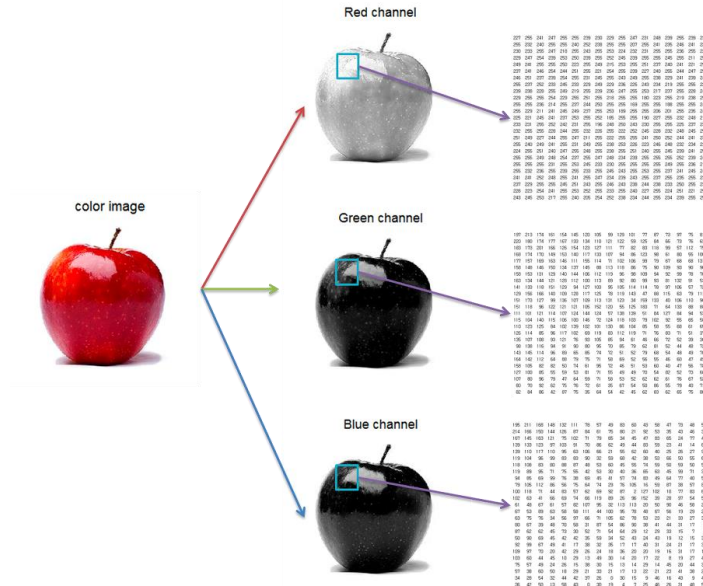
Gray-scale Image

206	225	193	185	182	174	151	137	132	161	140	119	132	120	141	121	125
230	194	191	197	189	160	164	143	156	157	99	160	127	113	121	122	107
195	188	213	185	148	178	153	160	148	116	123	123	155	142	107	151	117
183	191	190	170	177	167	148	164	145	134	127	158	140	112	128	97	146
194	177	189	184	171	139	179	149	108	142	146	140	122	114	115	109	164
177	171	170	175	161	165	172	121	149	153	127	116	131	148	133	133	140
180	177	157	156	168	171	137	145	153	134	138	149	133	128	137	123	119
185	160	171	149	157	142	132	147	124	129	118	138	132	118	165	138	104
165	159	145	176	159	125	159	137	131	142	152	152	116	135	147	106	122
153	180	186	168	139	160	151	158	114	155	172	83	125	154	107	124	152
176	191	153	127	166	140	144	149	164	158	71	184	166	81	147	150	132
177	145	124	151	152	154	140	179	156	92	161	201	108	101	165	128	131
139	131	152	146	140	158	173	159	92	170	171	89	123	161	124	136	99
145	136	169	150	141	134	175	106	158	155	142	121	144	137	102	112	107
141	157	158	121	139	169	137	135	165	124	145	129	105	104	118	112	118
158	149	122	135	153	140	107	156	121	152	156	118	124	129	118	104	94
165	142	145	132	156	117	135	146	127	138	107	95	116	120	102	94	93
130	168	151	132	132	134	125	139	116	132	126	111	129	106	99	102	123
171	173	149	136	133	111	130	121	120	102	104	127	120	111	106	102	118
185	171	150	109	133	125	120	114	105	121	109	111	111	103	115	100	96
181	138	124	129	102	123	107	138	119	101	108	109	114	95	102	109	125
155	137	131	109	114	105	128	119	104	102	103	121	104	129	103	124	110
140	120	139	128	103	116	110	122	110	106	103	112	110	108	124	120	104
119	111	136	112	125	125	122	115	90	119	105	98	132	101	126	91	122
125	127	132	91	134	121	82	117	109	96	97	112	130	109	113	126	129

Gray-scale Image의 경우 0과 255 사이의 값으로 밝기를 표현

## 이미지 데이터

1000000

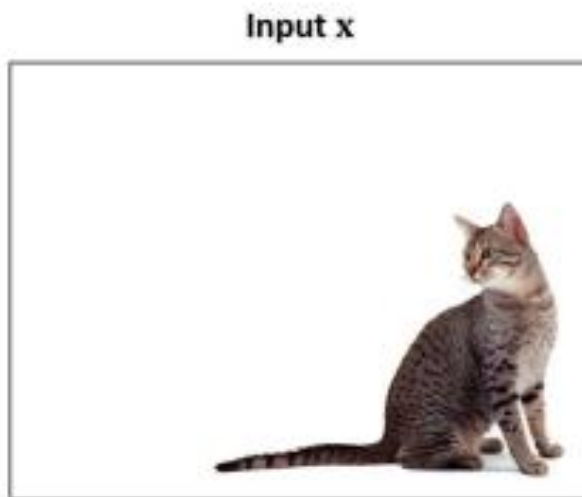


RGB 형식의 경우 빨강, 초록, 파랑 세 개의 채널을 통해 컬러 이미지를 표현

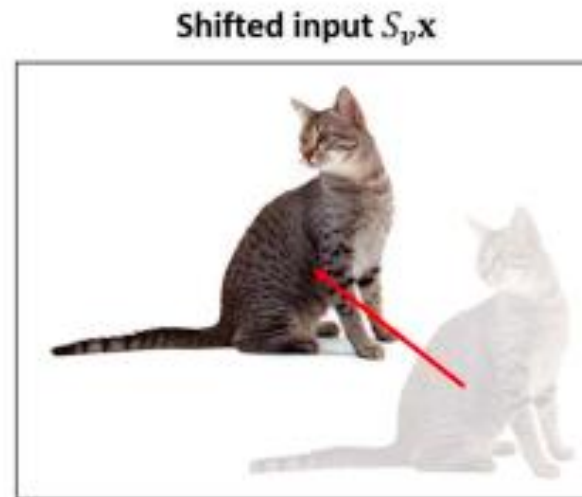
## 이미지 데이터의 가정

통계의 정상성 *Stationarity of Statistics*

시간이 지나더라도 데이터의 통계적 성질은 변하지 않음



Output  $f(x) = 1$



Output  $f(S_v x) = 1$

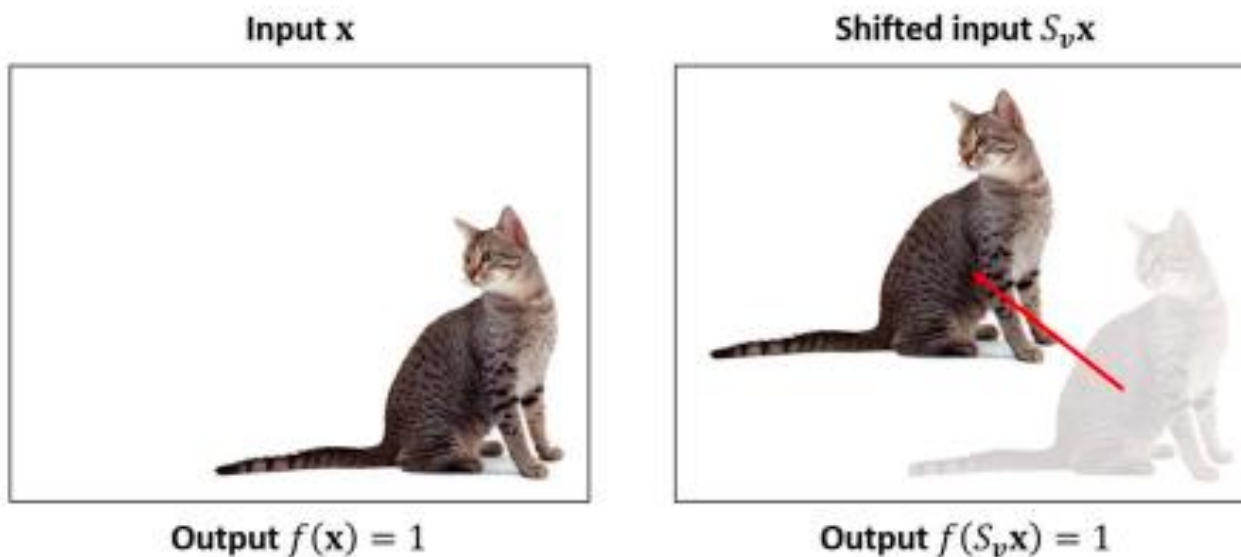


## 이미지 데이터의 가정

통계의 정상성 *Stationarity of Statistics*

이미지 데이터는 **Shift-Invariant**하기에

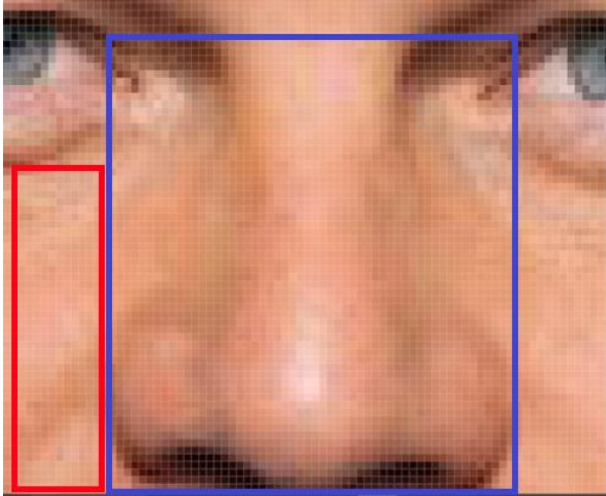
이미지의 픽셀들에 shifting이 발생해도 데이터의 핵심 패턴이 유지됨



## 이미지 데이터의 가정

### 픽셀 종속성의 국소성 *Locality of Pixel Dependencies*

각 픽셀들은 주변 픽셀에 대해서 높은 종속성을 가진다는 것



파랑 박스 내부의 픽셀은 서로 관련성이 큼

반면 박스 안과 밖의 픽셀은 관련성이 작음

## 이미지 데이터의 가정

픽셀 종속성의 국소성 *Locality of Pixel Dependencies*

각 픽셀들은 주변 픽셀에 대해서 높은 종속성을 가진다는 것

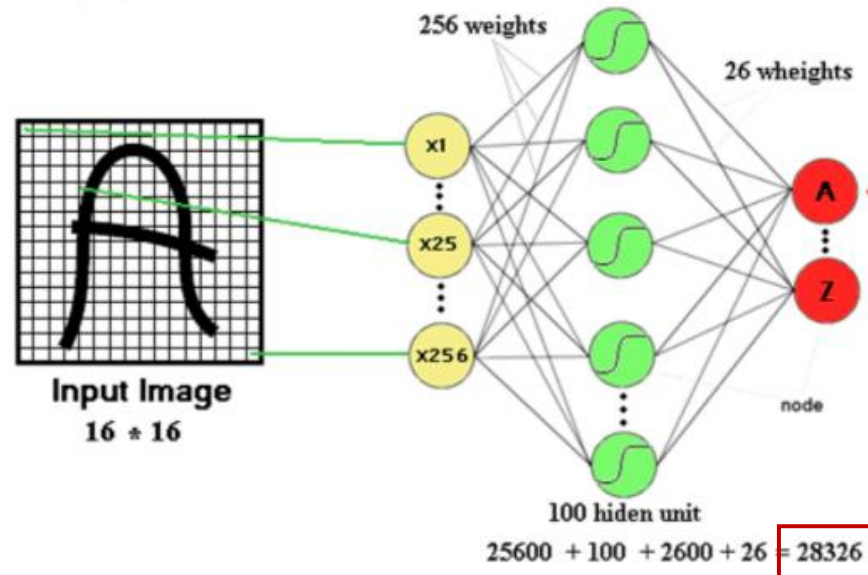
기존의 신경망으로 이미지 데이터를 학습하면  
데이터의 국소성을 보존하지 못하는 문제가 발생



2

CNN

## 기존 인공 신경망의 한계



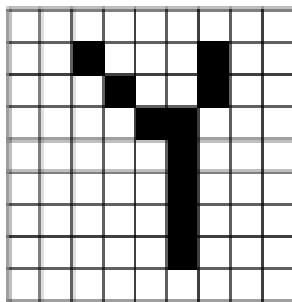
작은 size의 이미지를 인식하는데 필요한 파라미터의 개수가 상대적으로 많음

## 기존 인공 신경망의 한계



이미지가 shift되면 해당 픽셀의 파라미터는 무용지물이 됨  
정상성을 보존하지 못함

## 기존 인공 신경망의 한계

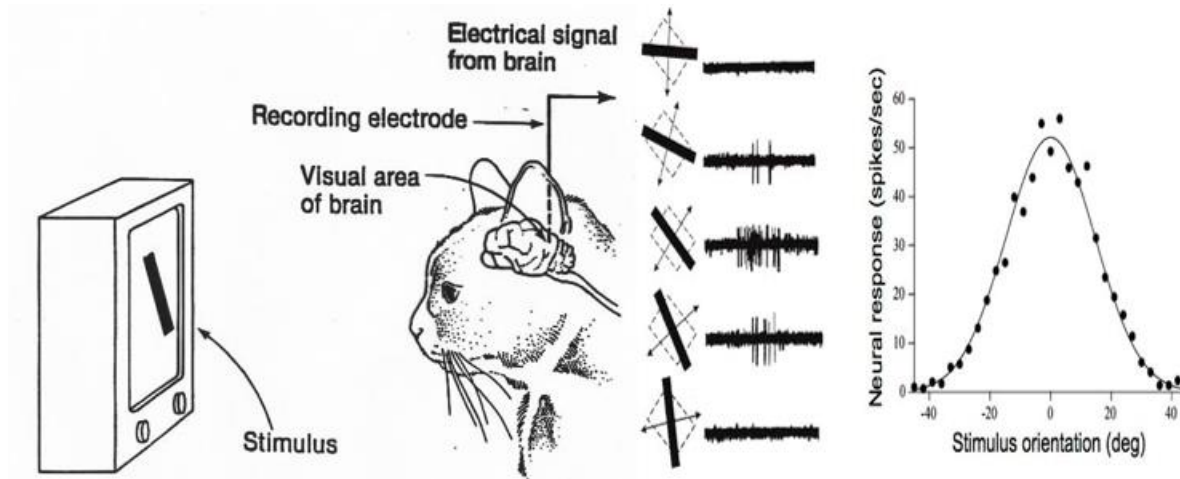


↓ 변환



학습 전 이미지 데이터를 Input layer에 넣기 위해  
1-D array로 변환해야 하기에 **국소성을 보존하지 못함**

## CNN의 배경

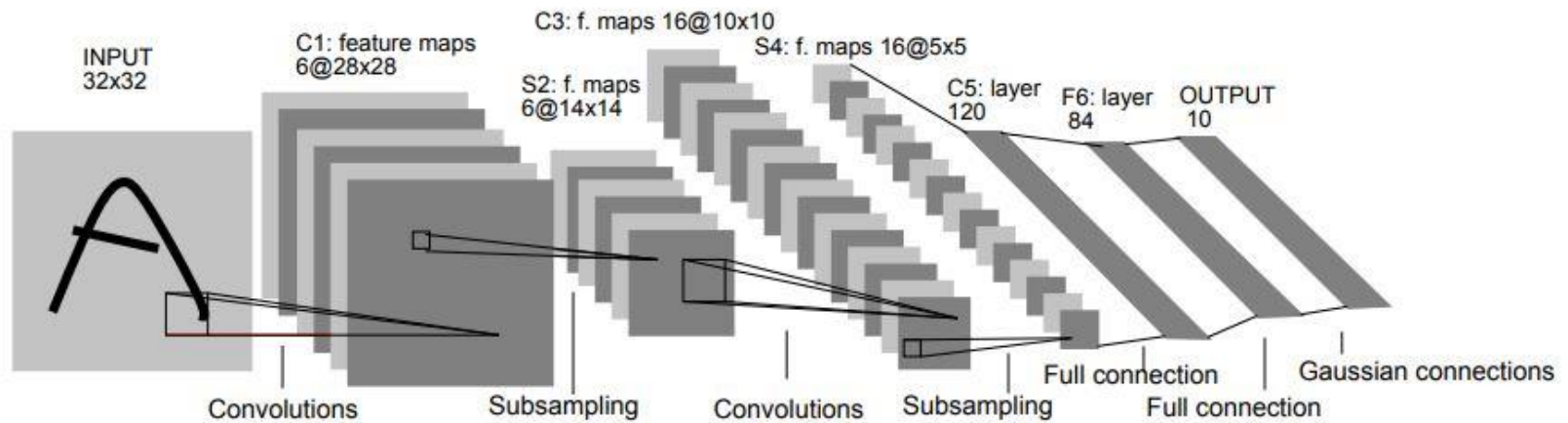


허블과 비셀의 고양이 시각 피질 실험 (1950s)

동물의 시각 피질 안 뉴런들이 일정 범위 안의 자극에만 활성화되는  
**근접 수용 영역**을 가진다는 사실이 밝혀짐



## CNN의 배경



LeNet-5 Architecture (1998)

이후 **특정 국소 영역**에 속하는 노드들의 연결로 구성된 인공신경망이 새로 고안됨  
CNN(Convolution Neural Network)의 시초가 됨

## Convolution Neural Network

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

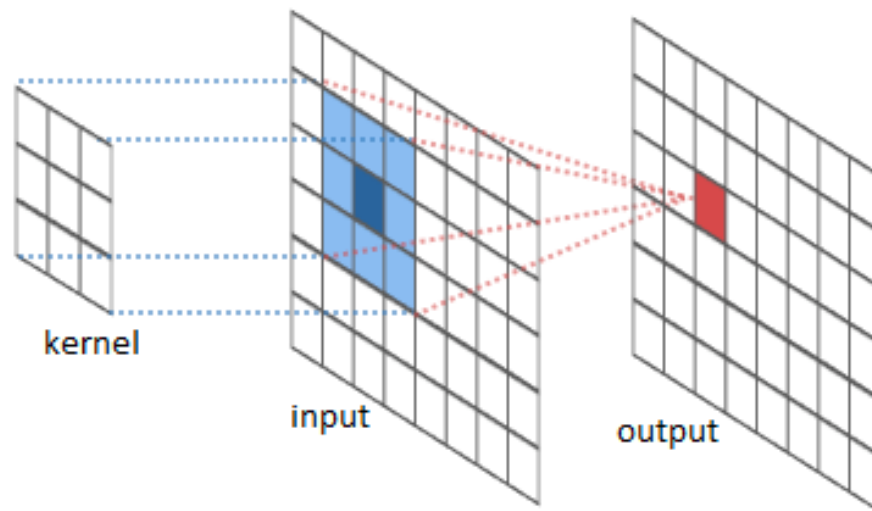
Convolution

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

Cross-Correlation

실제로 CNN 내부에서 돌아가는 연산은 합성곱과 유사한 **교차상관** 함수이지만  
편의상 합성곱(Convolution) 연산이라고 부름

## Convolution Neural Network

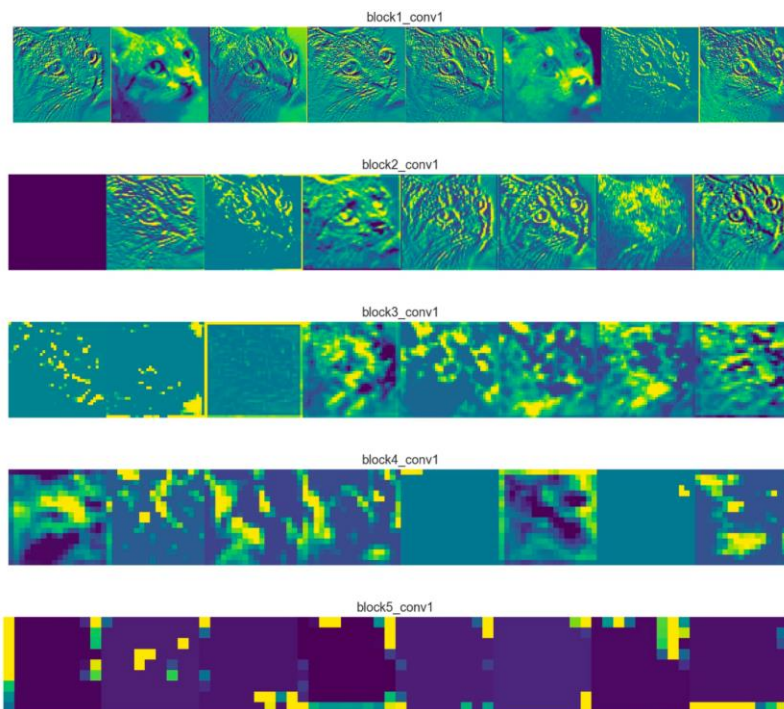


CNN의 합성곱 연산은 Convolutional Layer에서 진행되며  
output으로 **Feature map**이라는 새로운 이미지가 만들어짐

## Feature map

### Feature map

신경망이 학습을 하며 사용하는 **이미지의 특징이 담긴 맵**

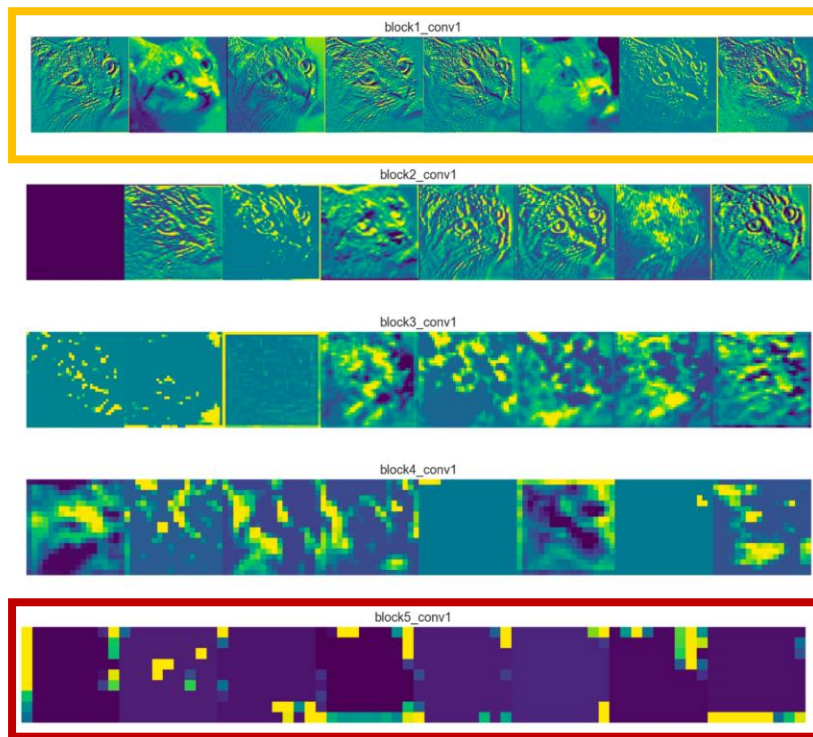


## Feature map

## Feature map

얇은 Layer에서는 **단순하고 반복적인 특징**을 반영  
깊은 Layer는 이미지 전체를 커버하는 **추상적인 특징**을 반영

얇은 Layer



깊은 Layer

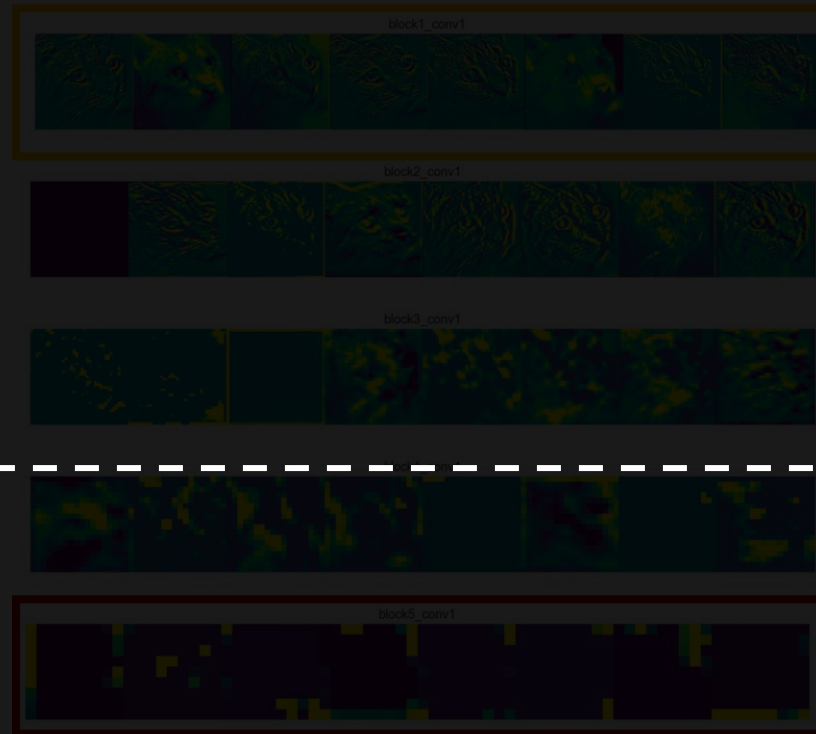


## Feature map

Feature map

얇은 Layer에서는 단순하고 반복적인 특징을 반영  
 이러한 Feature Map은 어떻게 만들어질까?

얇은 Layer



깊은 Layer



Feature map

Feature map

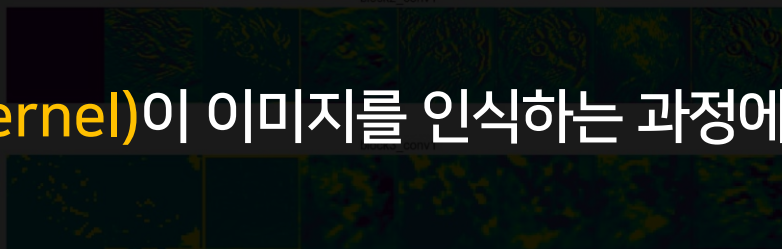
얇은 Layer에서는 단순하고 반복적인 특징을 반영  
이러한 Feature Map은 어떻게 만들어질까?

얇은 Layer



block2\_conv1

커널(Kernel)이 이미지를 인식하는 과정에서 생성!



block5\_conv1



깊은 Layer

## Convolutional Layer

### Kernel(Filter)

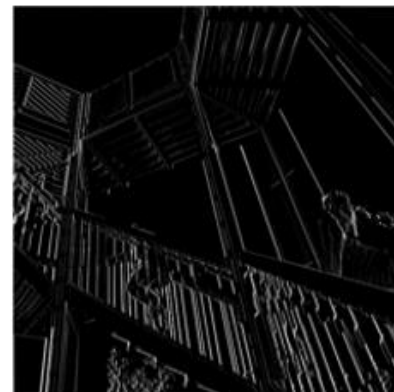
픽셀에 따른 **합성곱 연산**을 통해 Feature map을 생성하는 필터



\*

1	0	-1
1	0	-1
1	0	-1

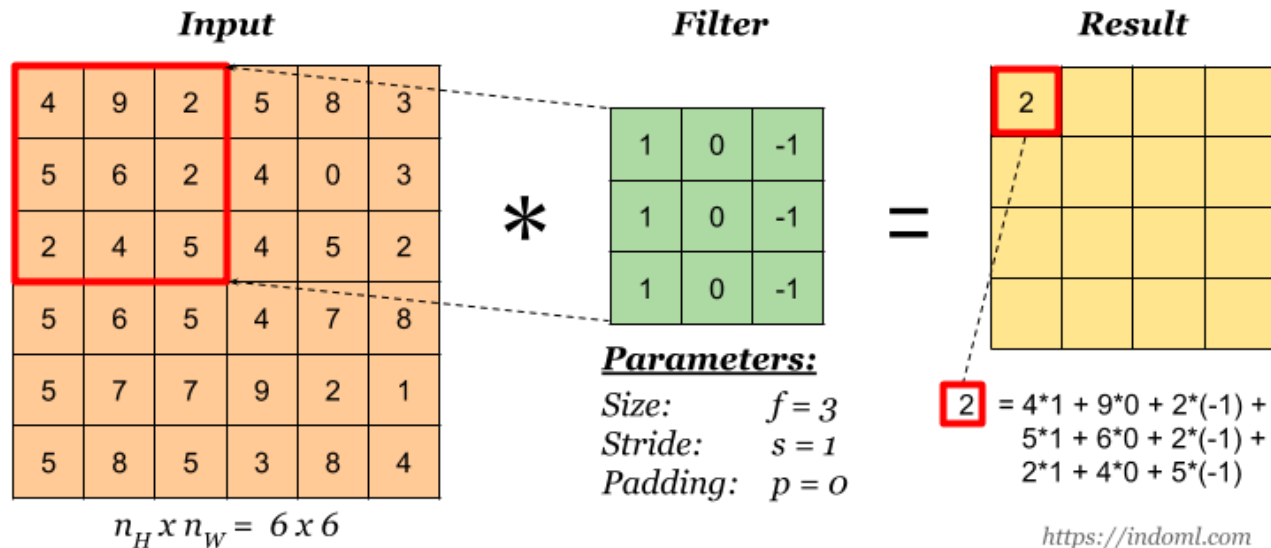
=



EX) 이미지에서 수직선의 특징을 뽑아내는 미분필터



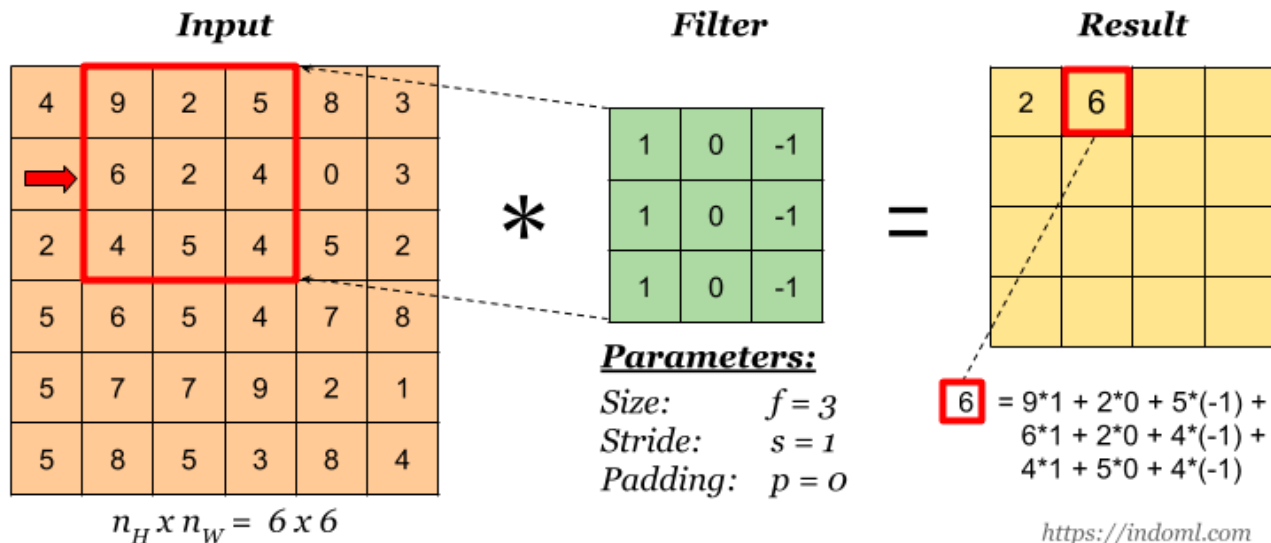
## Convolutional Layer



### Feature map 생성 과정

Kernel은 input과 같은 위치의 픽셀 값과 필터의 가중치 값을 모두 곱하여 더하는  
**아다마르 곱(Hadamard product)** 진행

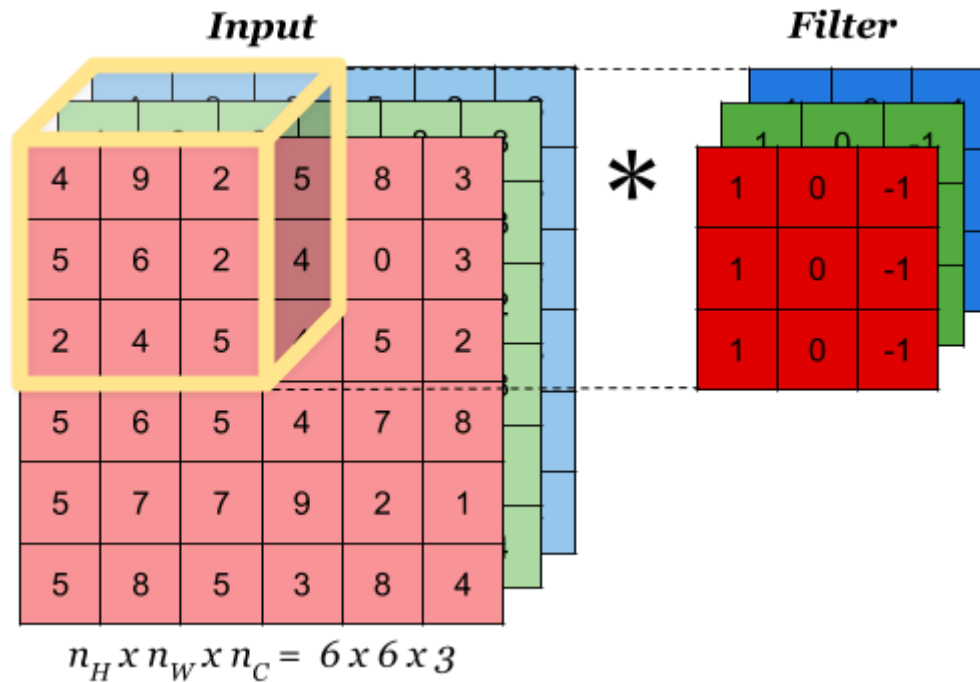
## Convolutional Layer



### Feature map 생성 과정

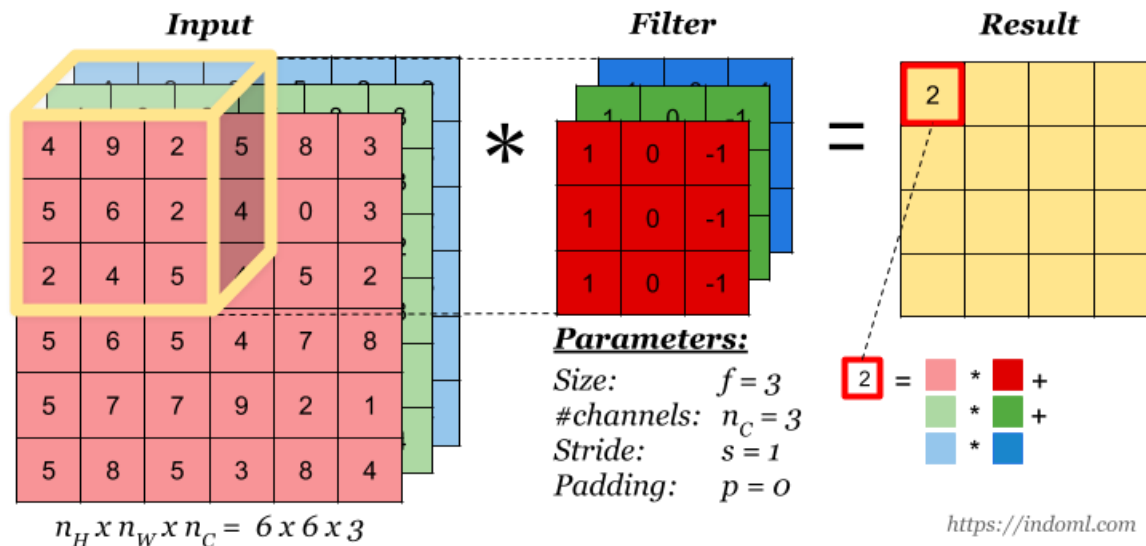
Output이 나올 때 마다 **픽셀을 따라 이동**하며 다시 동일한 연산 과정을 수행  
 모든 과정이 끝나면 하나의 Feature map이 생성됨

## Convolutional Layer



컬러 이미지는 채널이 3개이기 때문에 커널의 채널도 3개여야 함

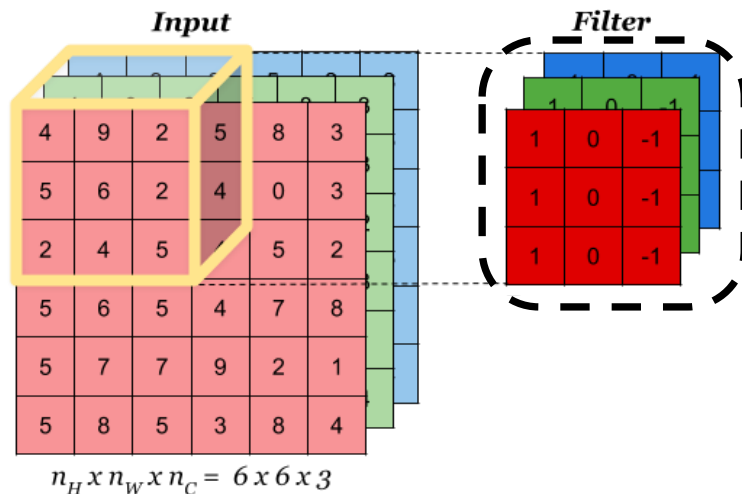
## Convolutional Layer



### Feature map 생성 과정 (Multiple Channels)

각 채널 별로 합성곱 연산을 진행한 결과들에 대해  
 같은 위치의 성분끼리 더하여 최종 output인 Feature map을 도출

## Convolutional Layer

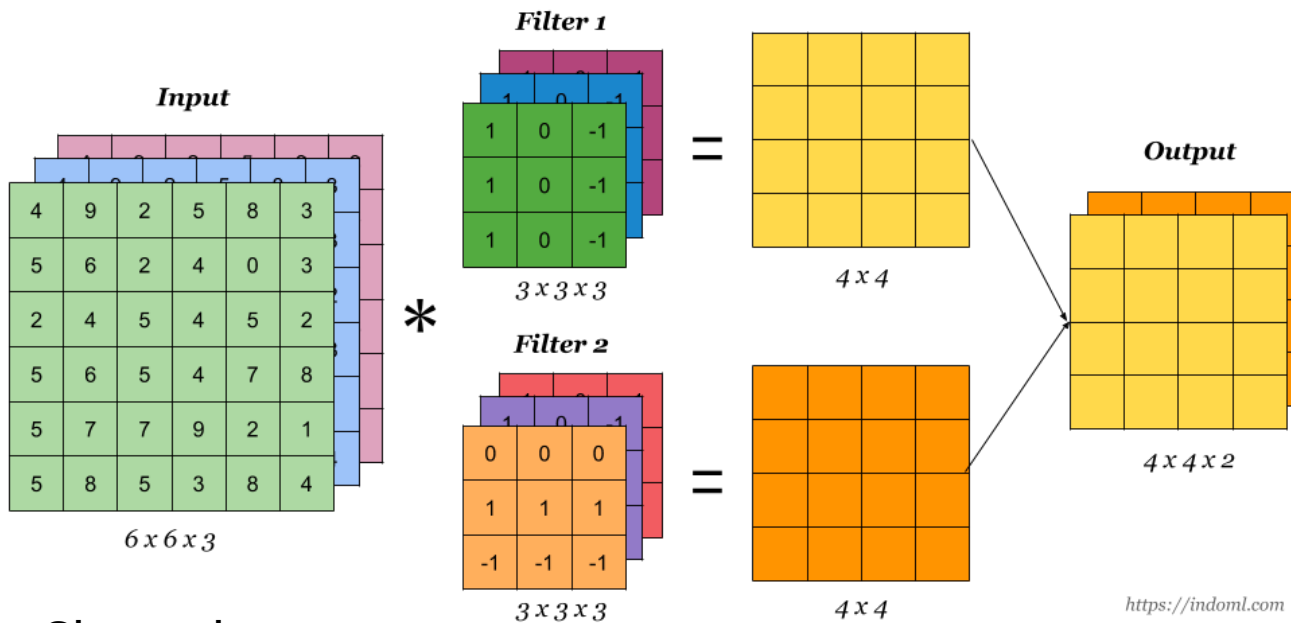


최적화해야 할 커널 내 파라미터

### CNN의 학습

CNN은 **커널 내 파라미터**를 학습을 통해 업데이트하며 최적의 값을 찾음  
 이는 기존의 신경망 학습 방식을 통해 이루어짐

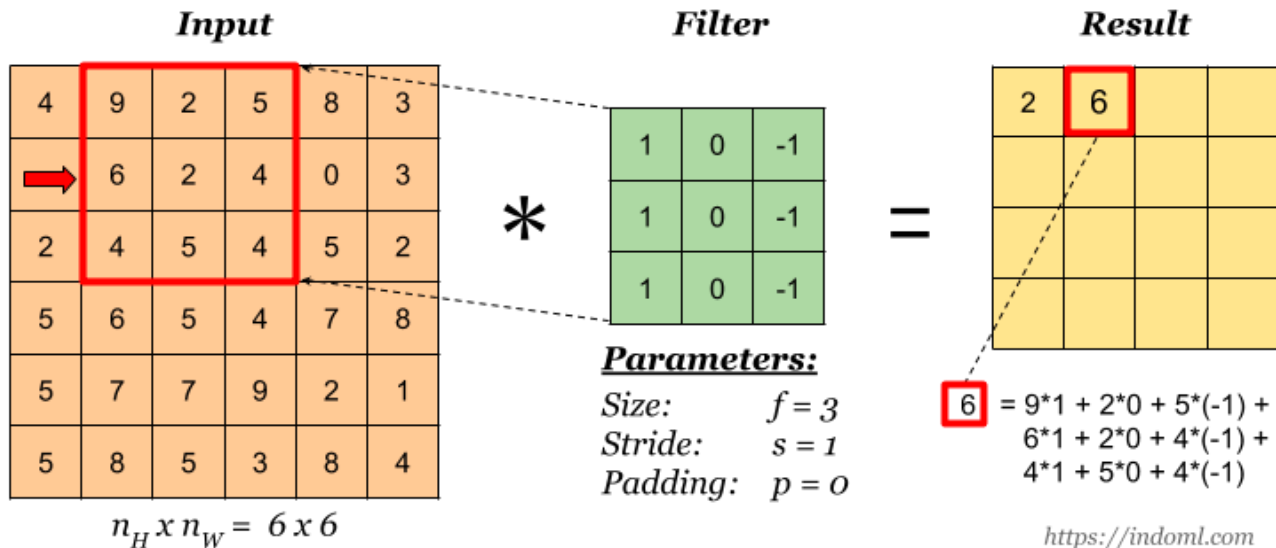
## Convolutional Layer



### Output Channel

Output의 채널 수를 늘리면 필요한 커널의 수도 늘어남  
 결국 학습해야 할 파라미터의 수가 증가하여 모델의 복잡도 증가

## Convolutional Layer

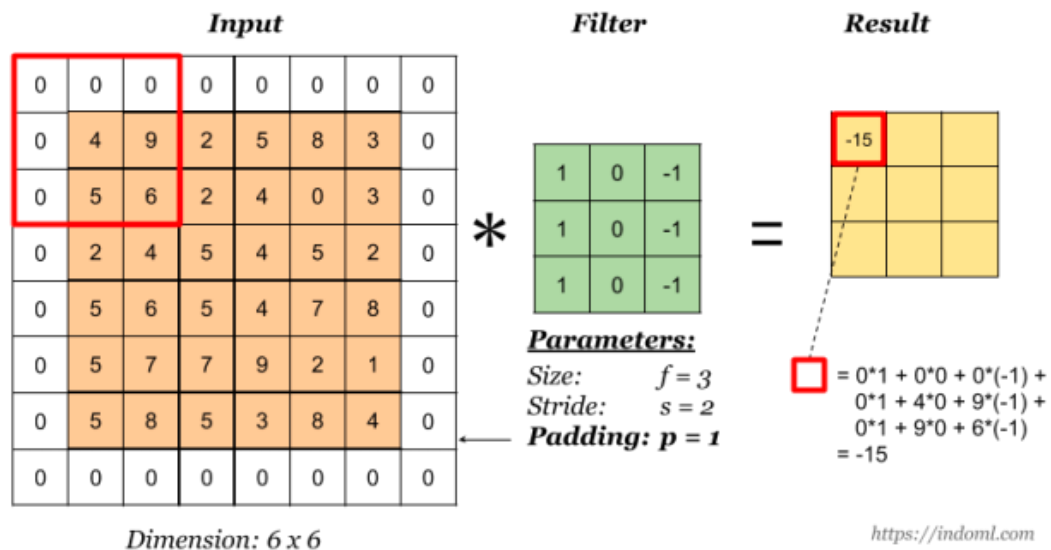


### Stride

커널이 한번 움직일 때마다 **지나는 픽셀의 수**

이미지의 크기, 메모리와 시간 요구량을 고려하여 사용자가 적절히 선택

## Convolutional Layer



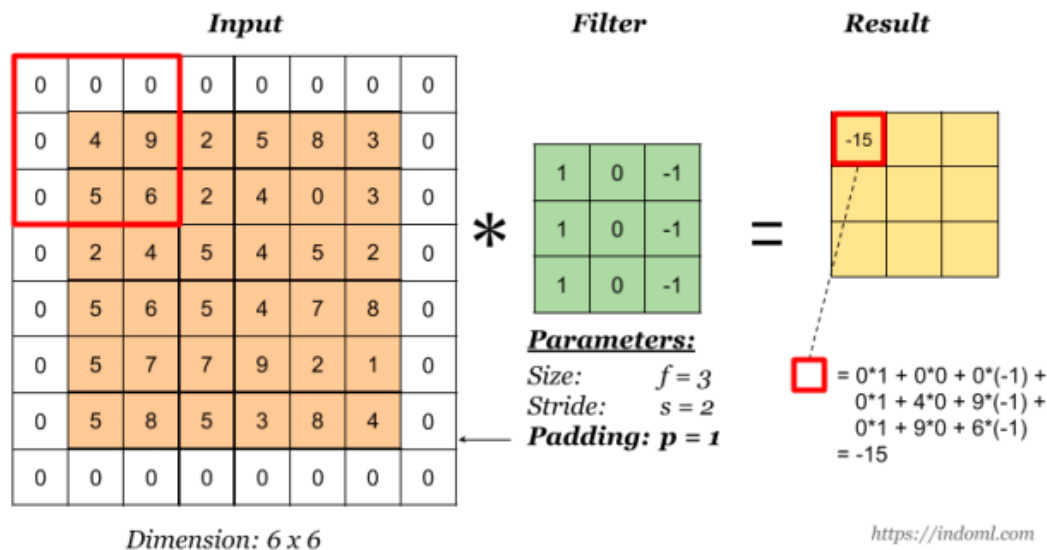
Zero - Padding

### Padding 기법

가장자리의 픽셀들은 필터를 통과하는 횟수가 상대적으로 적음  
 가장자리에 새로운 값들을 추가하여 **기존 값들이 여러 번 통과할 수 있도록 함**



## Convolutional Layer



Zero - Padding

### Padding 기법

Input의 크기보다 Output의 크기가 작아지는 기존의 방식과 달리,  
 가장자리에 Padding을 둘러 **Output의 크기가 너무 작아지는 것을 방지**

## 파라미터들의 관계

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

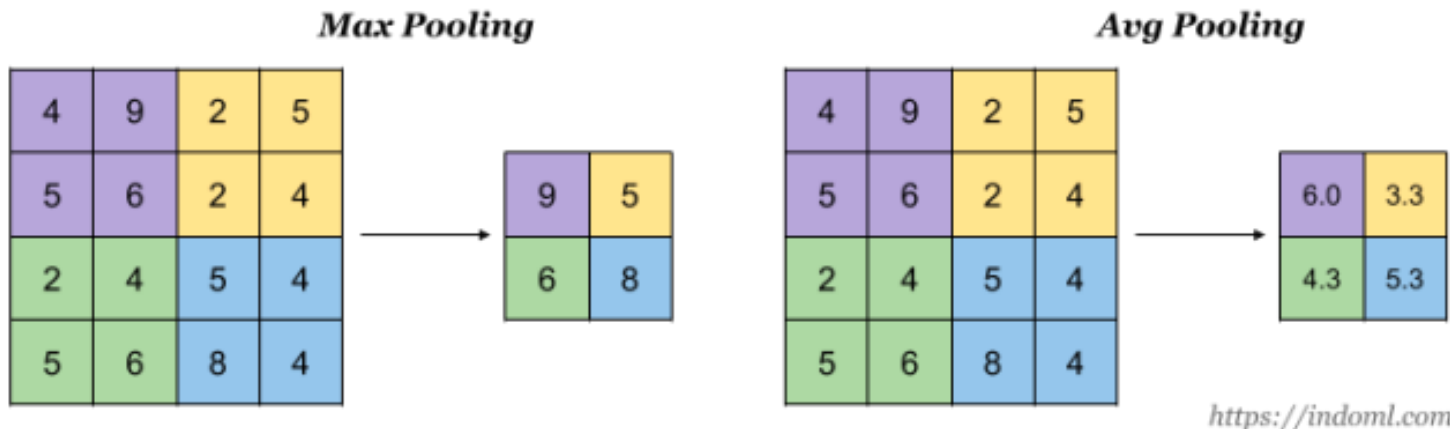
$$OH = \frac{H+2P-FH}{S} + 1$$

Feature map의 크기와 Filter의 크기의 관계를 이용해 어느 한 쪽을 통제할 수 있음

OW : output의 너비 OH : output의 높이 FW : 필터의 너비

FH : 필터의 높이 P : Padding의 크기 S : Stride의 크기

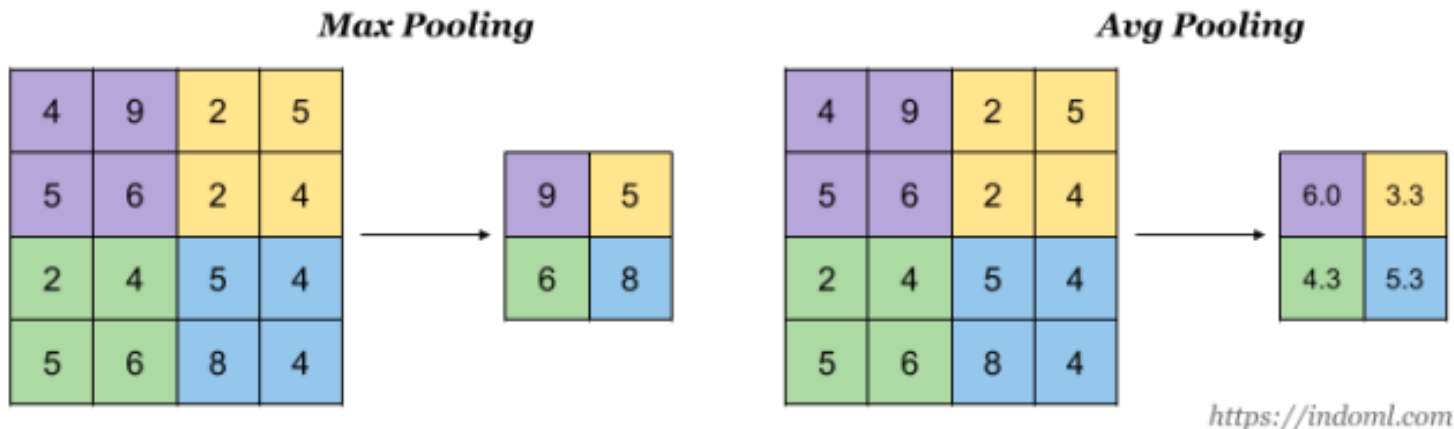
## Pooling Layer



### Pooling 기법

대상 영역의 최댓값 혹은 평균값을 사용하여 이미지를 압축함  
특징을 최대한 유지하면서도 **Feature map의 크기를 줄일 수 있음**

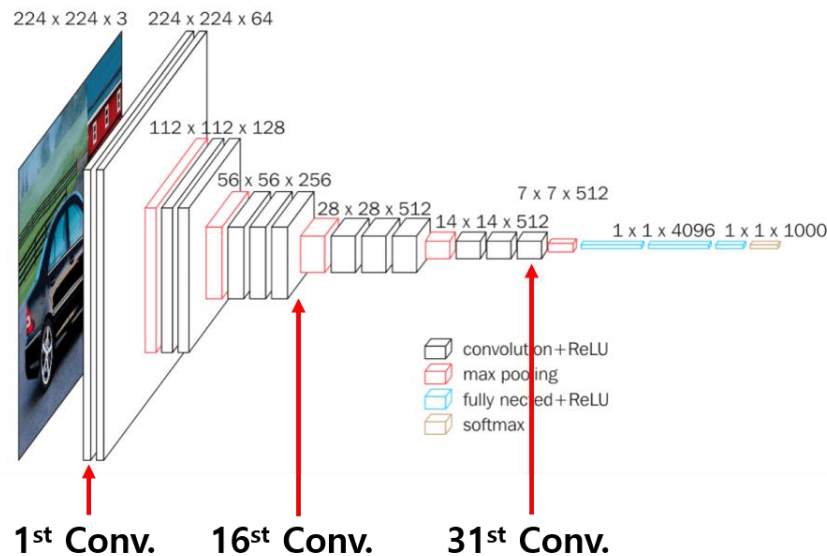
## Pooling Layer



### Pooling 기법

특히 Max Pooling은 영역의 **최댓값**을 대푯값으로 사용하여  
이미지의 크기를 줄이면서도 특징을 강하게 유지할 수 있음

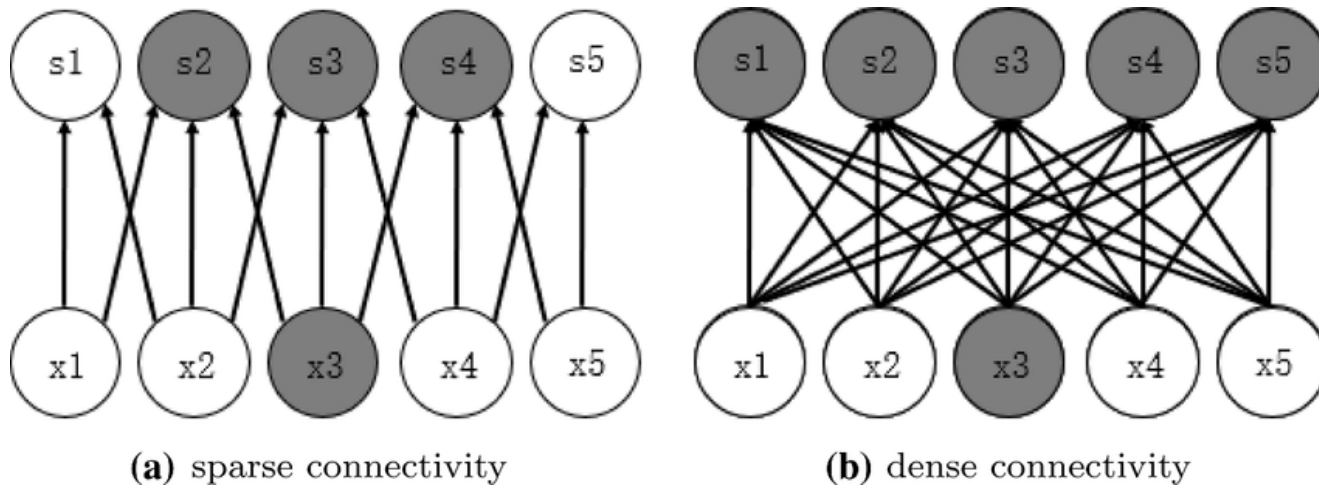
## Pooling Layer



### Pooling 기법

추가적인 학습이 없고 파라미터의 수 감소로 이어지기 때문에  
Overfitting 위험이 감소한다는 장점도 존재

## Sparse Connectivity



### Network의 Sparse Connectivity

연결된 특정 영역의 노드들에게만 가중치를 통해 정보를 제공하여  
**파라미터의 수를 줄여** Overfitting 위험성과 연산량을 감소시킴



## Sparse Connectivity

CNN이 Sparse Connectivity를 갖는 이유?



(a) sparse connectivity

(b) dense connectivity

CNN이 Sparse Connection을 갖는 이유

CNN은 작은 수의 가중치를 담은 필터 하나로  
국소 영역과의 연산을 수행하며 모든 픽셀들을 처리하기 때문



## Sparse Connectivity

CNN이 Sparse Connectivity를 갖는 이유?



작은 수의 가중치를 담은 필터 하나로  
국소 영역과의 연산을 수행하며 모든 픽셀들을 처리하기 때문

CNN이 Sparse Connection을 갖는 이유

CNN은 작은 수의 가중치를 담은 필터 하나로  
국소 영역과의 연산을 수행하며 모든 픽셀들을 처리하기 때문





## Sparse Connectivity

CNN이 Sparse Connectivity를 갖는 이유?



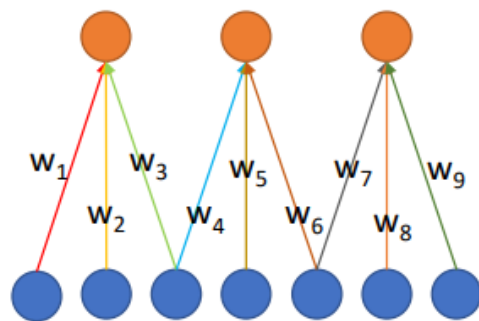
작은 수의 가중치를 담은 필터 하나로  
국소 영역과의 연산을 수행하며 모든 픽셀들을 처리하기 때문



CNN이 Sparse Connection을 갖는 이유

즉, CNN은 Weight Sharing을 통해  
국소 영역과의 연산을 수행하며 모든 픽셀들을 처리하기 때문  
Sparse Connection을 구현

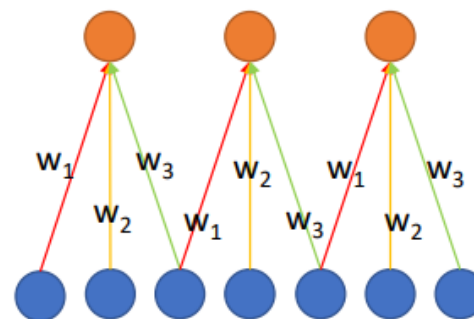
## Weight Sharing



**Without** weight sharing

Hidden layer

Input layer

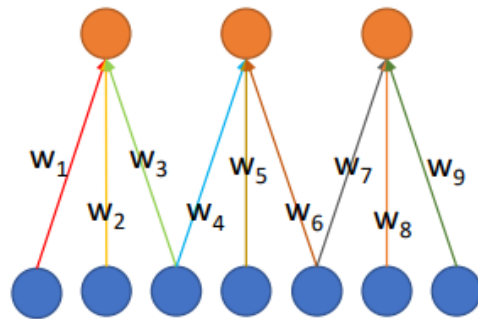


**With** weight sharing

## Weight Sharing

모든 국소 영역에 동일한 가중치의 필터를 사용하여  
필터가 전체 이미지를 스캔하며 **특정 패턴을 찾아내는** 기능을 가지게 됨

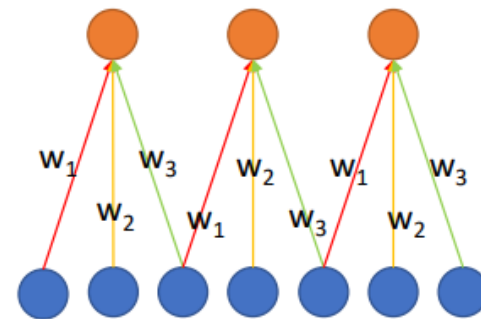
## Weight Sharing



**Without** weight sharing

Hidden layer

Input layer



**With** weight sharing

## Weight Sharing

즉, 커널 내의  $N$ 개의 가중치로 모든 input에 대해 연산을 진행함  
 더불어 특정 영역과만 연산하여 **Sparse Connectivity** 역시 만족

## CNN이 뛰어난 이유

### ☒ CNN



커널로 인접 픽셀들과만 연산을 수행하므로 픽셀 종속성의 국소성 만족



필터가 이동하며 픽셀들을 처리하므로 이미지의 Shift에도 대응 가능



기존 인공 신경망에 비해 작은 수의 파라미터로도 학습이 잘 됨

# 3

발전된 CNN 모델

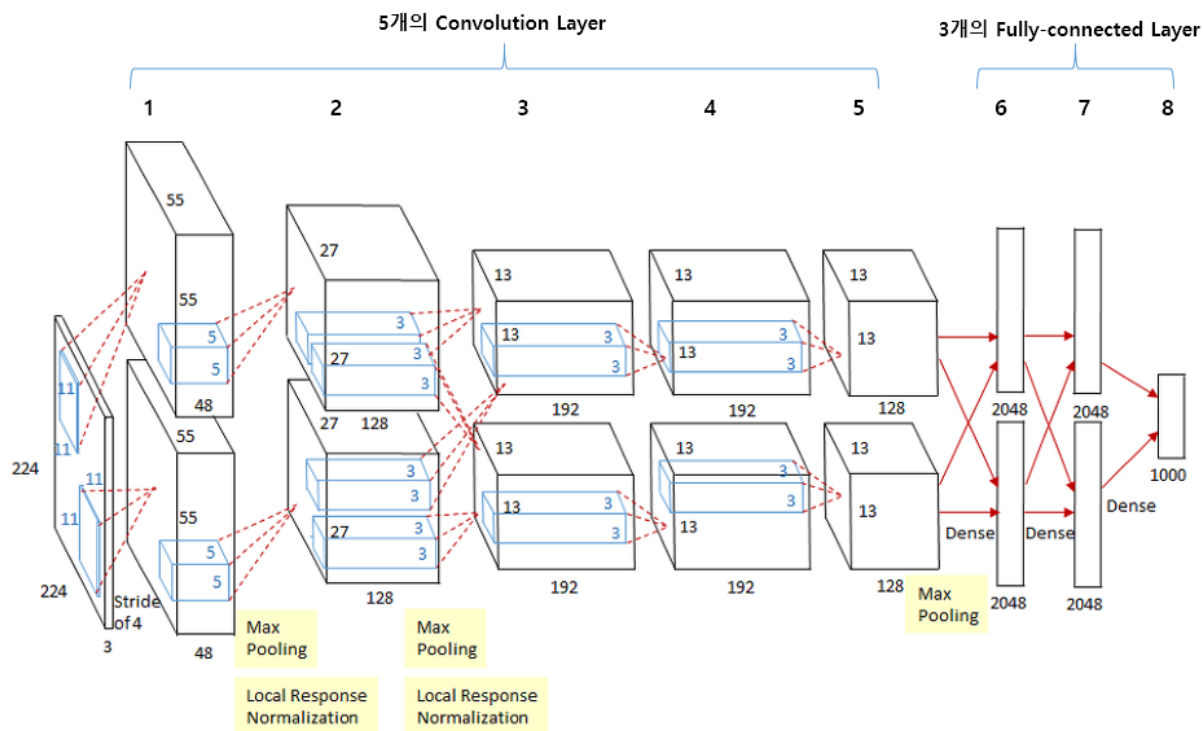
# 3

## 발전된 CNN 모델

### AlexNet

### AlexNet

2012년 ILSVRC에서 1등 기록한 8개 층의 CNN 모델



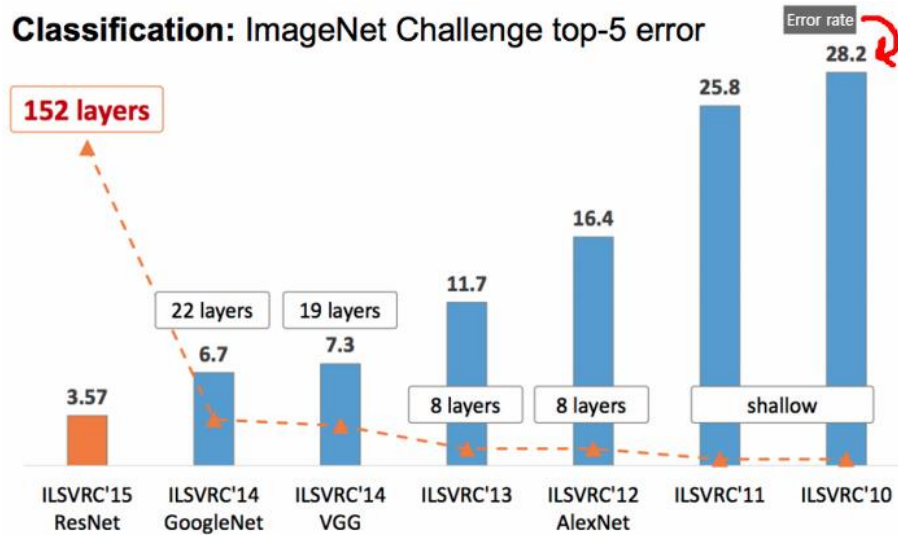
# 3

## 발전된 CNN 모델

AlexNet

AlexNet

2012년 ILSVRC에서 1등 기록한 8개 층의 CNN 모델



AlexNet을 기점으로 많은 CNN 관련 모델들이 탄생!

조심니다!



## AlexNet - 문제 정의

☒ 학습최적화

대회에서 제공되는 데이터와  
상용화된 이미지 데이터 부족



모델의 최적화 어려움

☒ 과적합

파라미터 수에 비해  
데이터 양이 적음



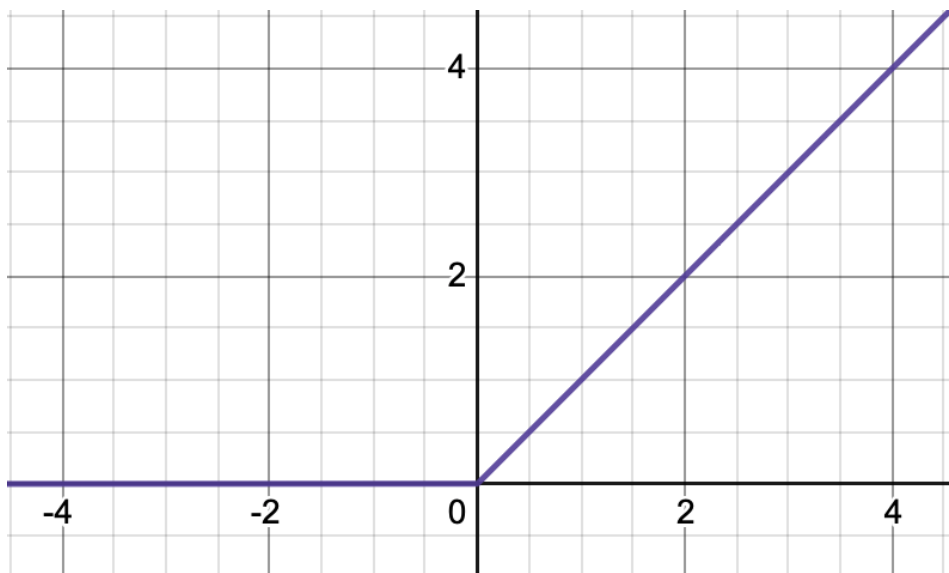
Overfitting



## AlexNet - 학습최적화 문제 해결

## ReLU 활성화 함수

이전에 사용하던 Tanh보다 학습속도가 6배나 빠르고 비선형성 역시 두드러짐



$$\text{ReLU}(x) = \max(0, x)$$

ReLU가 주목받기 시작한 계기!

## AlexNet - 학습최적화 문제 해결

데이터 증강 *Data Augmentation*

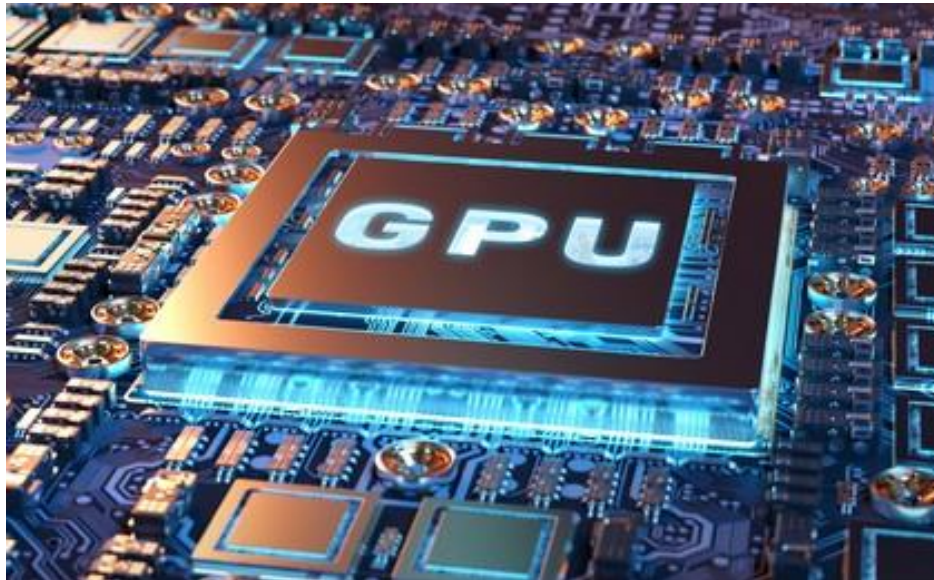
회전, 밝기 조절, 노이즈 추가 등 데이터 증강기법을 활용해 **부족한 데이터 수 보완**



## AlexNet - 학습최적화 문제 해결

### 병렬 GPU

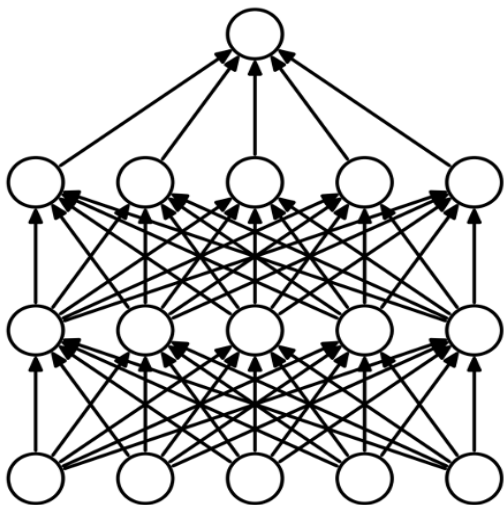
GPU가 CUDA를 탑재하지 않던 시기라 병렬 GPU로 연산 진행



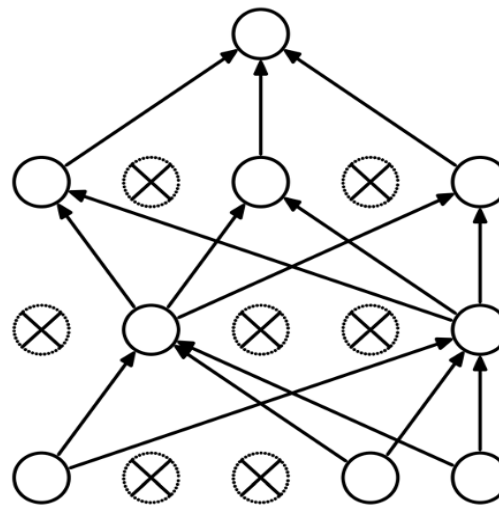
## AlexNet - Overfitting 문제 해결

## Dropout

Dropout 기법으로 과도한 연산량에 의해 **특정 가중치에 치중되는 현상 방지**



(a) Standard Neural Net



(b) After applying dropout.



## AlexNet - Overfitting 문제 해결

Dropout

Dropout 기법의 학습단계에서 Drop된 노드들이

테스트 단계에서도 계속 Drop되어 있는 것은 아님!



(a) Standard Neural Net



(b) After applying dropout.

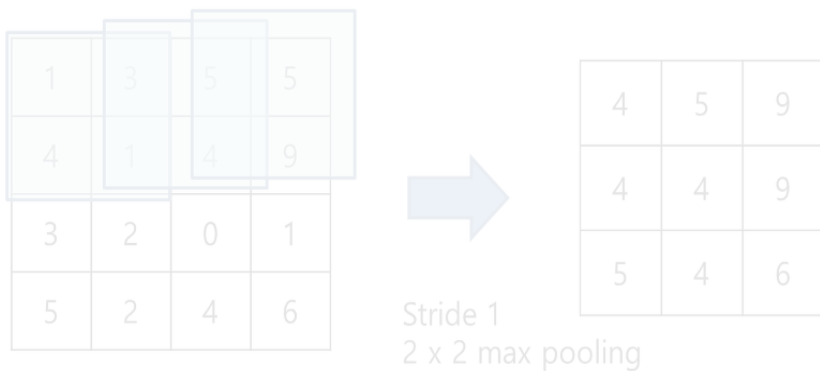
테스트 단계에서는 Drop-rate로  
가중치를 조절해 모든 노드를 활용!

## AlexNet - Overfitting 문제 해결

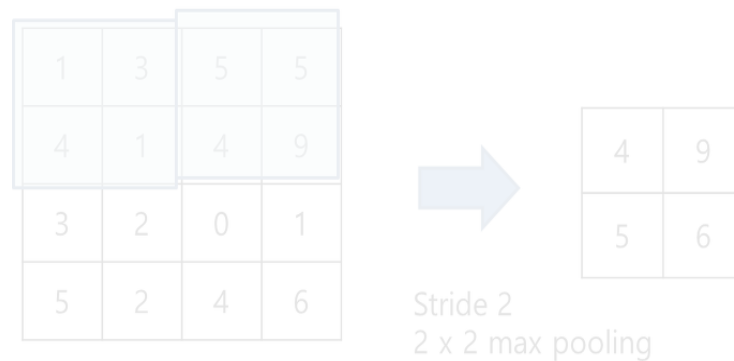
## Non-Overlapping Max pooling

파라미터의 수를 줄여 Overfitting 방지 및 학습 최적화

Overlapping Max Pooling



Non-Overlapping Max Pooling

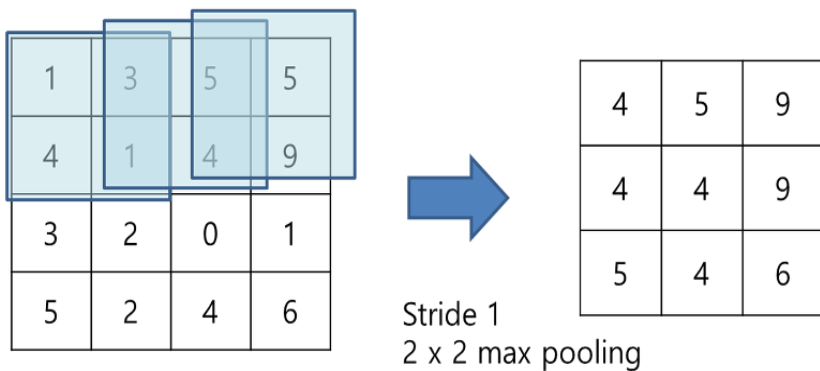


## AlexNet - Overfitting 문제 해결

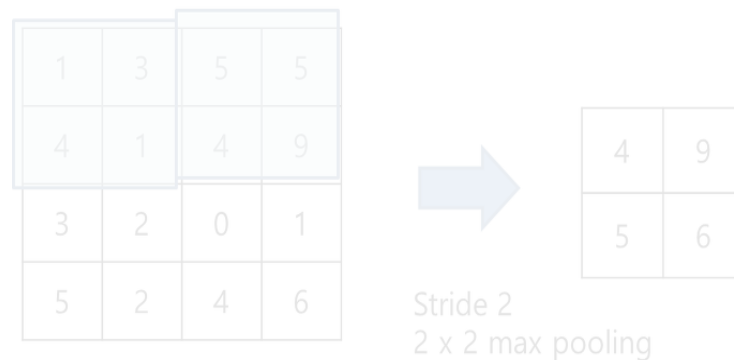
## Non-Overlapping Max pooling

파라미터의 수를 줄여 Overfitting 방지 및 학습 최적화

Overlapping Max Pooling



Non-Overlapping Max Pooling

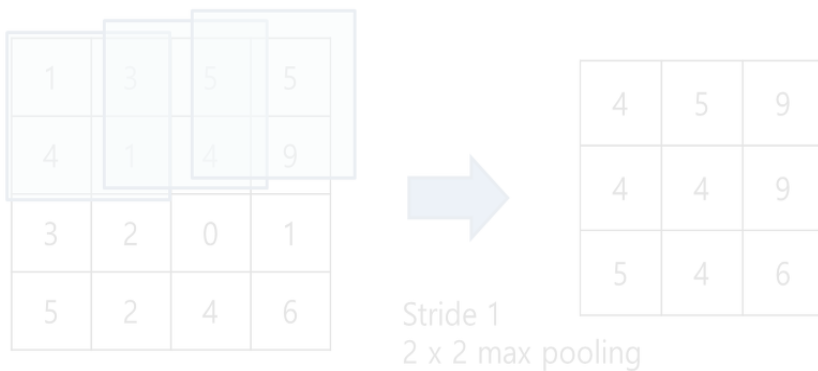


## AlexNet - Overfitting 문제 해결

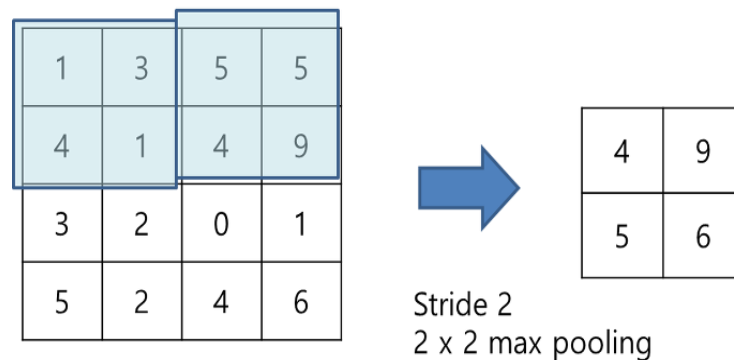
## Non-Overlapping Max pooling

파라미터의 수를 줄여 Overfitting 방지 및 학습 최적화

Overlapping Max Pooling



Non-Overlapping Max Pooling

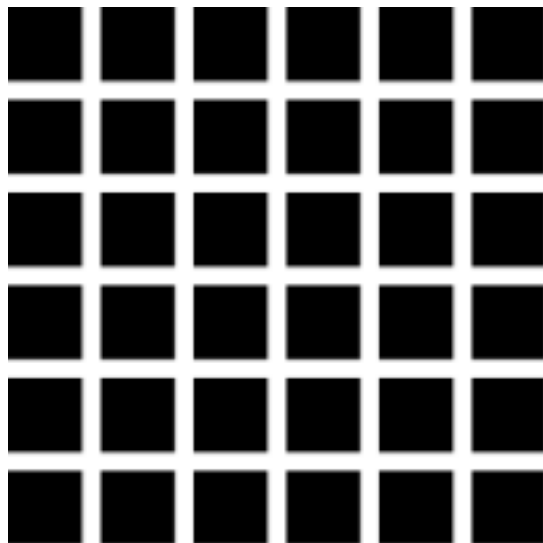




## AlexNet - Overfitting 문제 해결

### 측면 억제 *Lateral Inhibition*

강한 뉴런의 활성화가 근처 다른 뉴런의 활동을 억제시키는 현상



한 모서리를 보았을 때 주변 모서리들은 회색으로 보이게 됨

나는 바보농



## AlexNet - Overfitting 문제 해결

### 측면 억제 *Lateral Inhibition*

ReLU는 입력값이 양수보다 클 경우 그 값을 그대로 이용하기 때문에  
Conv Layer나 Pooling Layer에서 큰 픽셀 값이 다른 픽셀에 영향을 미침

## AlexNet - Overfitting 문제 해결

### Local Response Normalization

측면 억제(Lateral Inhibition) 현상을 구현한 Layer로  
feature map의 같은 위치에 있는 픽셀끼리 정규화

$b_{x,y}^i$  는 LRN 후 feature map,  $a_{x,y}^i$  는 입력 feature map  
단순하게 분모를 정규화 용도로 사용해 **scaling** 했다고 받아들일 수 있음!

$$b_{x,y}^i = a_{x,y}^i / (k + \alpha \sum_{j=\max(0,i-n/2)}^{j=\min(N-1,i+n/2)} a_{x,y}^j{}^2)^\beta$$

## AlexNet - 의의

### AlexNet의 문제 정의

어려운 최적화와 과도한 파라미터로 인한 Overfitting

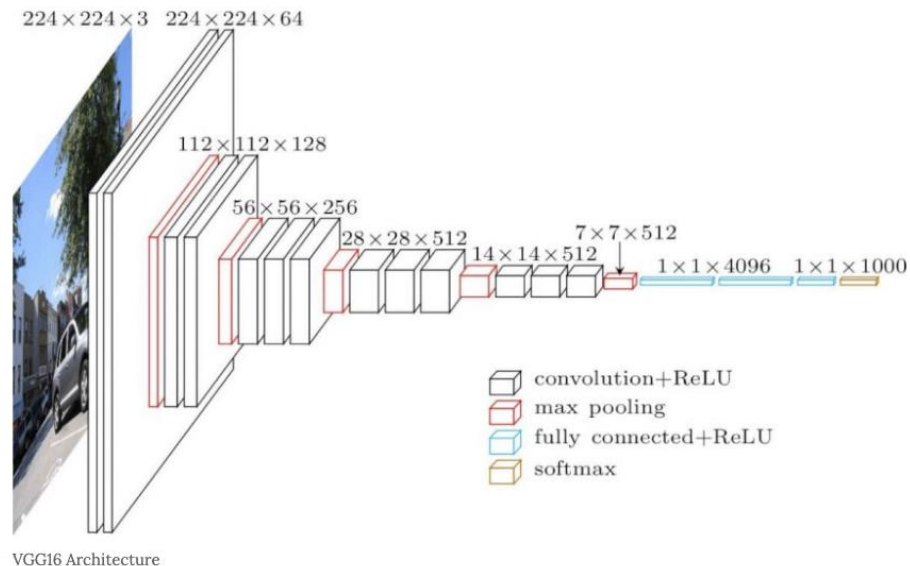


**ReLU, Max pooling, LRN**을 사용하여  
학습 최적화와 Overfitting 문제를 해결하고 CNN의 시대를 열었음!

## VGGNet

## Visual Geometry Group Net

이미지 특징 탐색에서 자주 사용되는 2014년 ILSVRC 2등 모델



2등을 기록했음에도 주목받는 이유는 **구조의 단순함!**

## VGGNet - 문제 정의

Layer를 깊게 쌓는 것이 성능을 향상시킬까?

단순히 Layer를 깊게 쌓는다면 파라미터 수가 증가하고

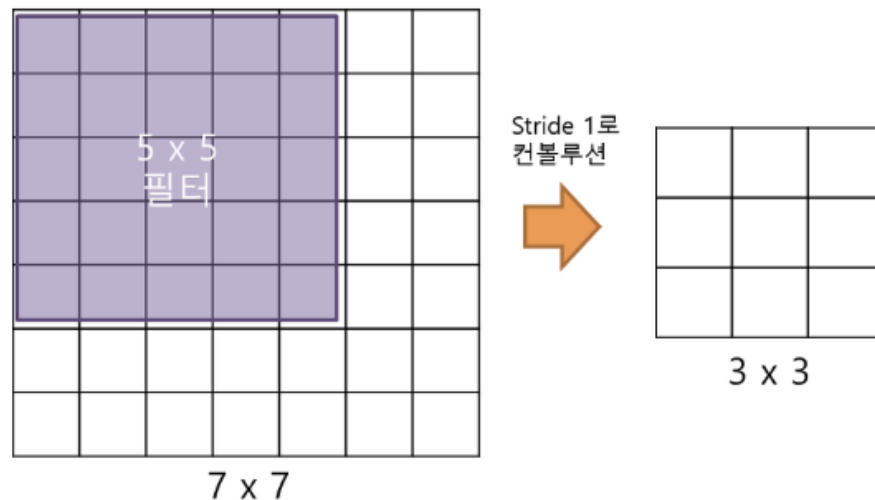
Overfitting과 연산량 문제 발생!



# 3

## 발전된 CNN 모델

### VGGNet - 커널의 크기

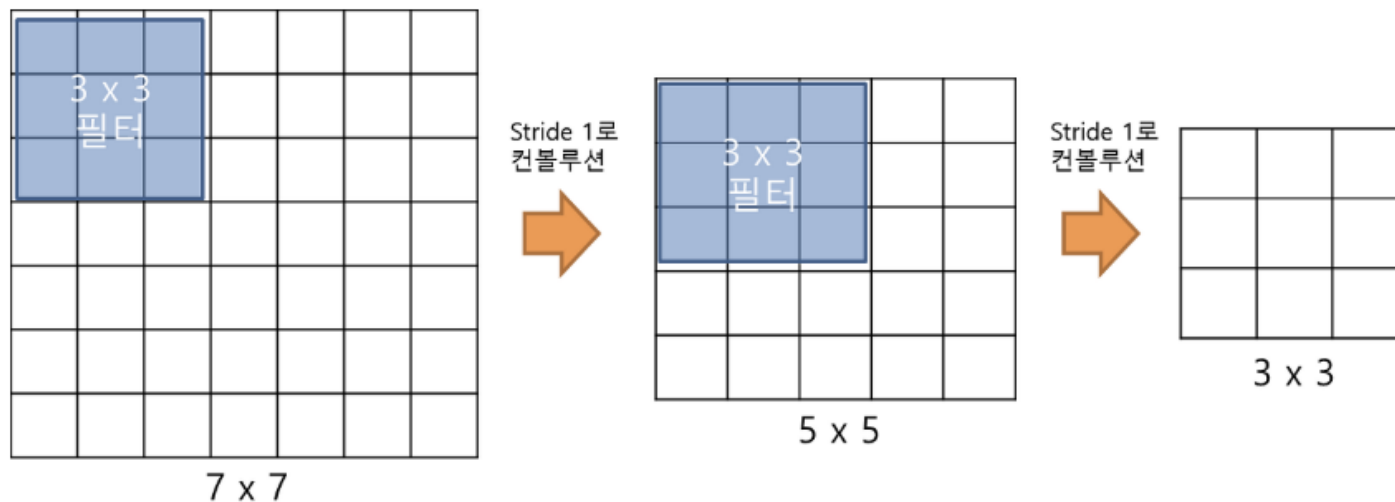


7x7 input을 3x3 size로 출력해주려면 5x5 필터로 합성곱 연산 가능

파라미터 수 :  $5 \times 5 = 25$

### 3 발전된 CNN 모델

#### VGGNet - 커널의 크기



3x3 필터를 두 번 통과시킴으로써 같은 결과를 얻을 수 있음

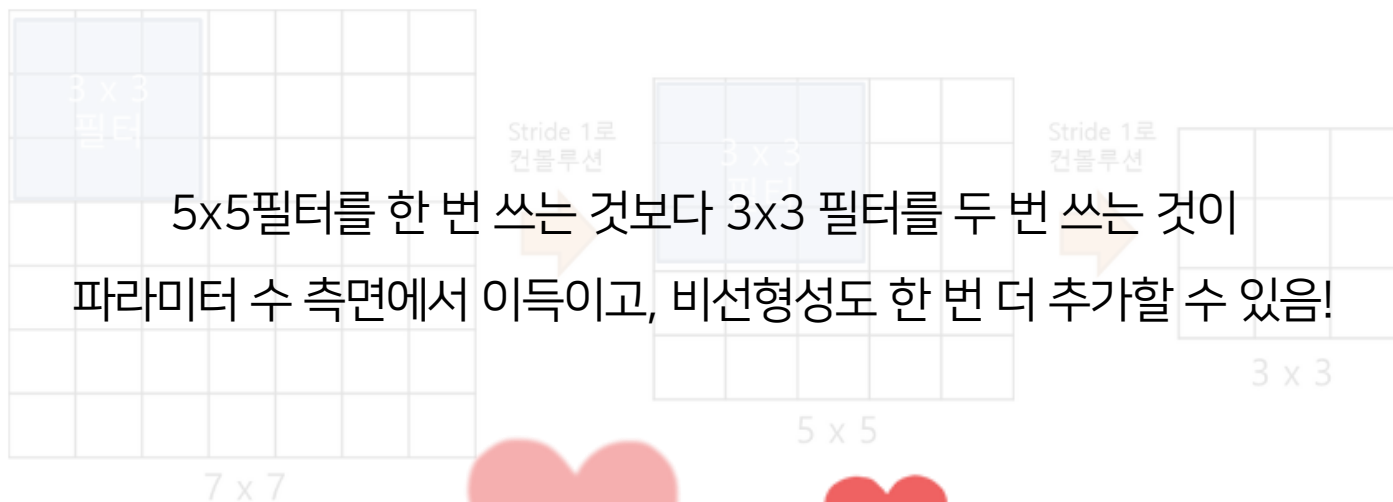
$$\text{파라미터 수} : 3 \times 3 + 3 \times 3 = 18$$



# 3

## 발전된 CNN 모델

### VGGNet - 커널의 크기



## VGGNet - 의의

### VGGNet의 문제 정의

Layer를 깊게 쌓기만 하면 파라미터의 수와 과적합 위험성이 증가



큰 사이즈의 커널 대신 **작은 사이즈 커널을 연속적으로 사용**하여  
파라미터 수를 줄이고 비선형성도 더 추가할 수 있었음

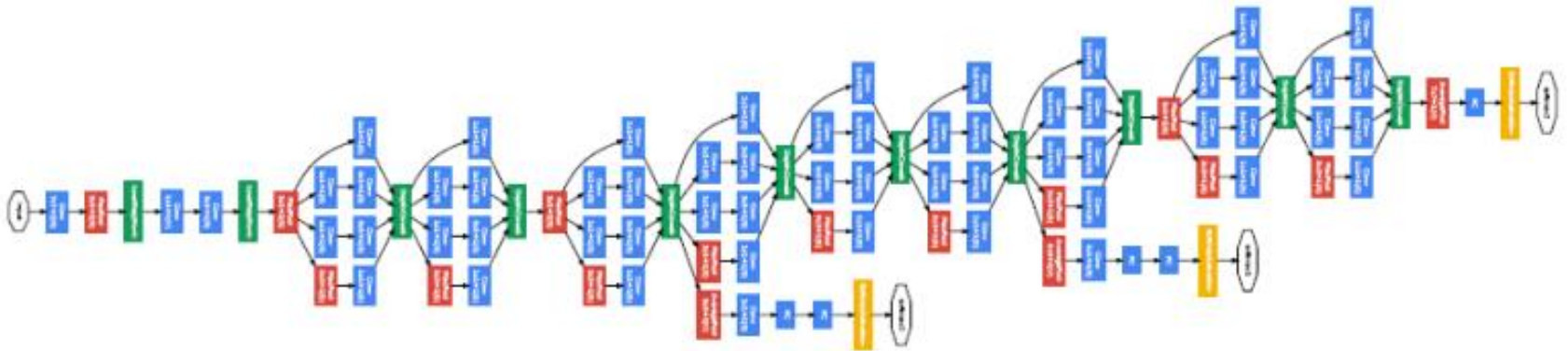
# 3

## 발전된 CNN 모델

### GoogLeNet

#### GoogLeNet

모듈을 설계하고 중첩하는 **Network in Network** 구조의 2014년 ILSVRC 우승 모델



## GoogLeNet - 문제 정의

딥러닝 네트워크 성능 향상을 위해서는 **네트워크 크기를 늘려야 함**

## GoogLeNet - 문제 정의

딥러닝 네트워크 성능 향상을 위해서는 **네트워크 크기를 늘려야 함**



그러나 기존 Dense 구조의 크기를 키우면  
**파라미터 수의 증가**로 Overfitting 및 연산량 증가

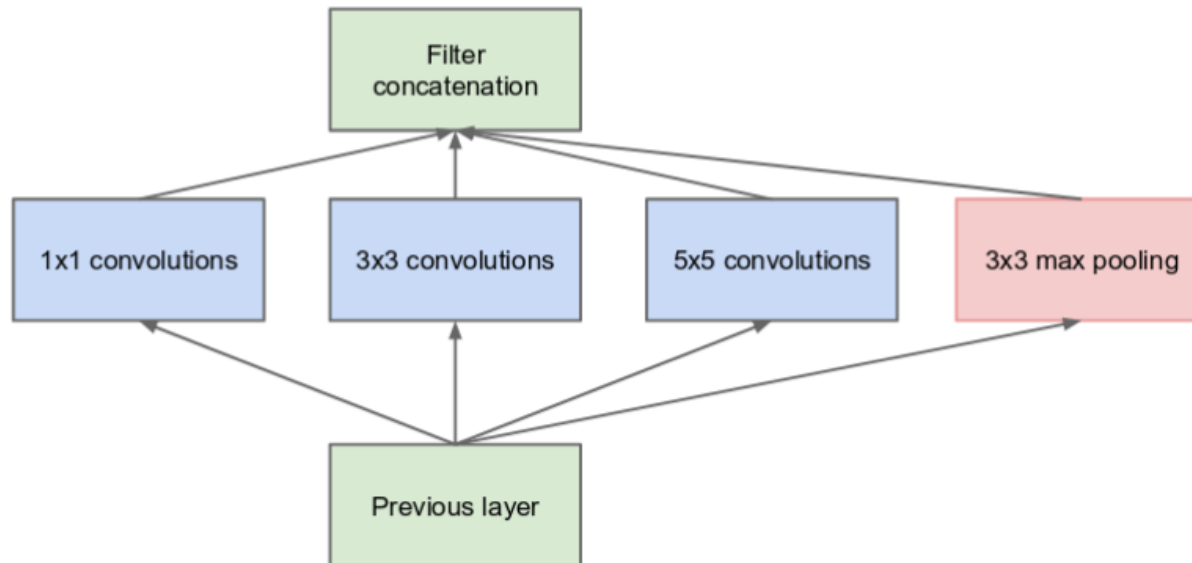


# 3

## 발전된 CNN 모델

### GoogLeNet - 문제 정의

Layer를 깊게 쌓으며 파라미터를 줄이기 위해 **Sparse Architecture** 채택





## GoogLeNet - 문제 정의

그러나, 행렬 연산은 **dense matrix에 최적화되어 있음**

Layer를 깊게 쌓으며 파라미터를 줄이기 위해 **Sparse Architecture** 채택

Sparse matrix는 대부분 0의 값을 가지고 있어

정보량은 적고 **연산량은 증가**



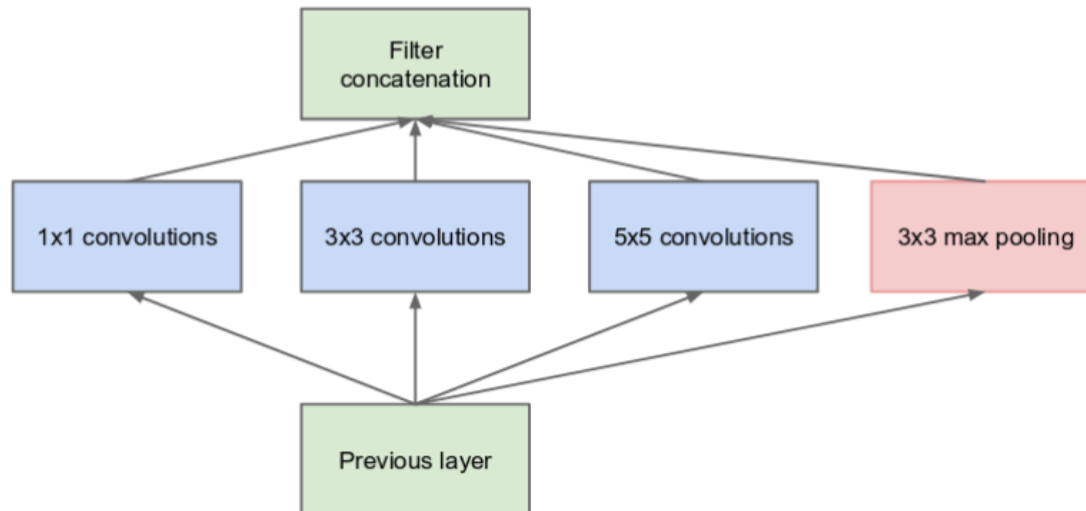
**구조는 Sparse하게, 연산은 Dense하게!**

## GoogLeNet - 문제 정의

### Inception Module

Sparse한 구조를 위해 고안된 것이 **Inception 모듈**

GoogLeNet은 이를 **중첩시키는** 구조를 가짐

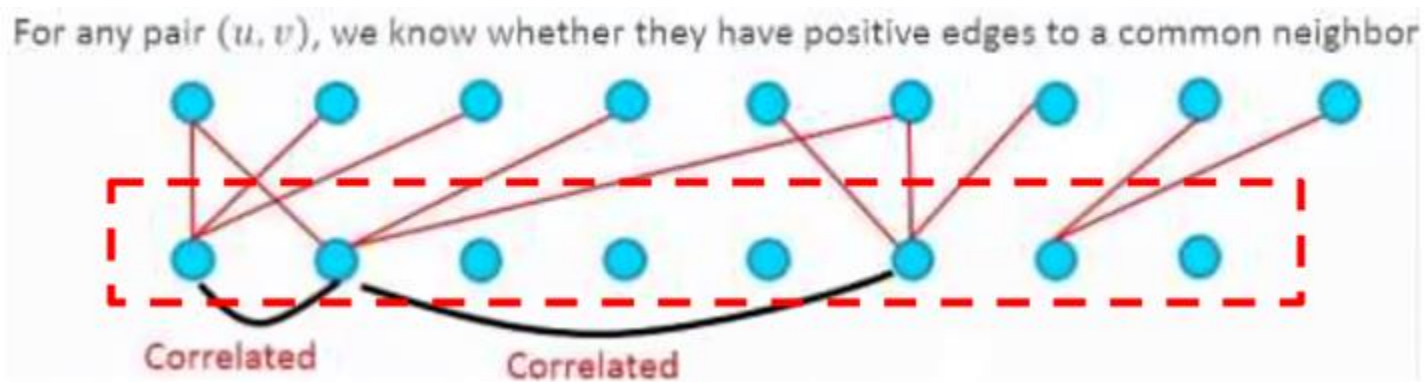




## GoogLeNet - 문제 해결

### Sparse Architecture

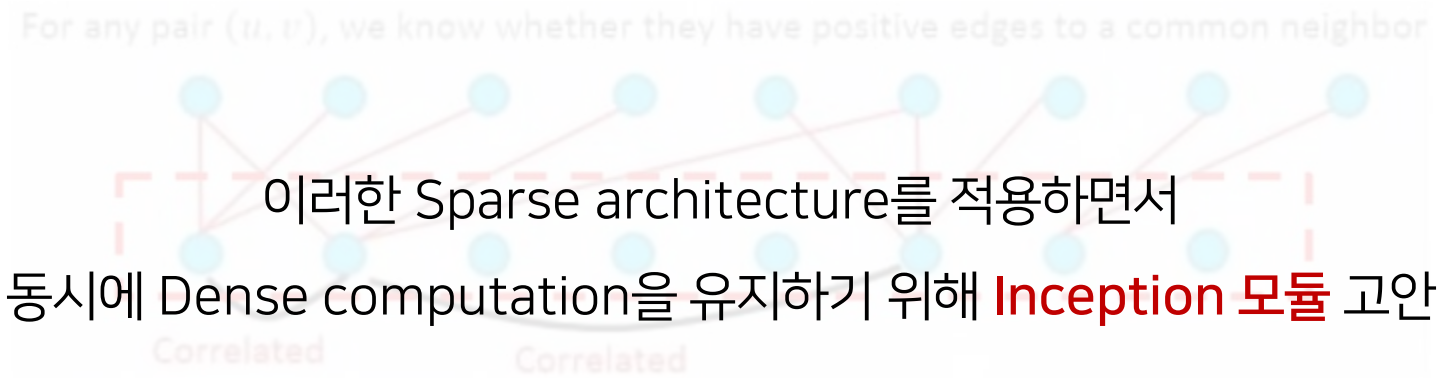
기존의 input node 사이의 상관성이 높으면 연결, 나머지는 연결하지 않는 구조



## GoogLeNet - 문제 해결

### Sparse Architecture

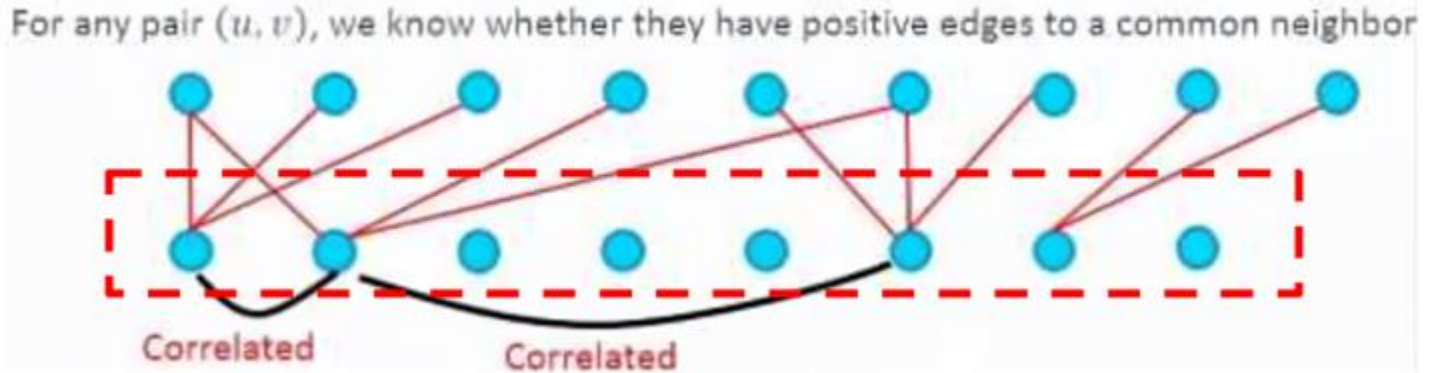
기존의 input node 사이의 상관성이 높으면 연결, 나머지는 연결하지 않는 구조



## GoogLeNet - 문제 해결

### Network Topology Construction Alogrithm

Vision Network를 함축하는 Sparse Matrix를 Approximate 하는 알고리즘



## GoogLeNet - 문제 해결

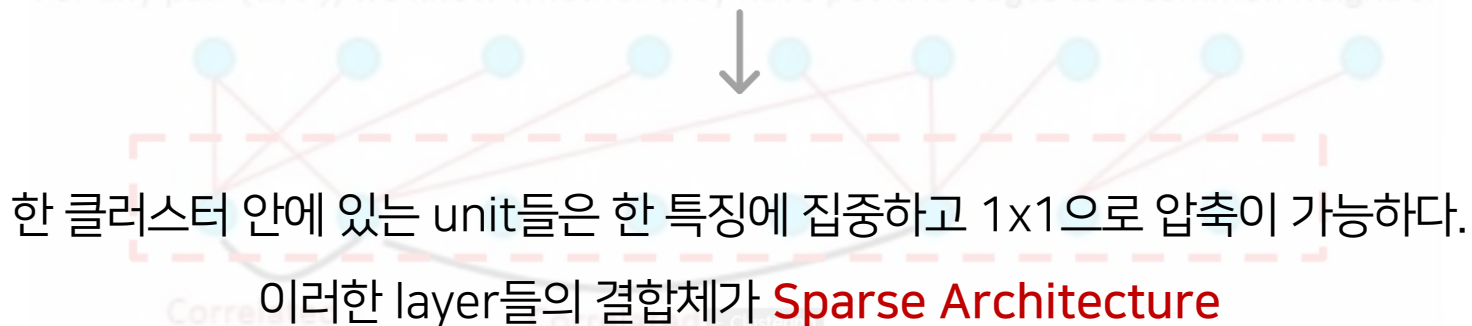
### Network Topology Construction Alogrithm

Vision Network를 함축하는 Sparse Matrix를 Approximate 하는 알고리즘

Unit들 간의 상관관계가 높으면 클러스터로 묶는데,

이 클러스터는 input image에서 **특정한 local region에 집중**한다.

For any pair  $(u, v)$ , we know whether they have positive edges to a common neighbor



## GoogLeNet - 문제 해결

문제 해결에 적용된 두가지 개념

### ✓ Concatenate

3가지 Conv와 한가지 pooling을  
적용한 후 concatenate



**Dense Matrix Computation** 유지

### ✓ Backpropagation

Optimal local construction를  
찾는 오차 역전파 반복

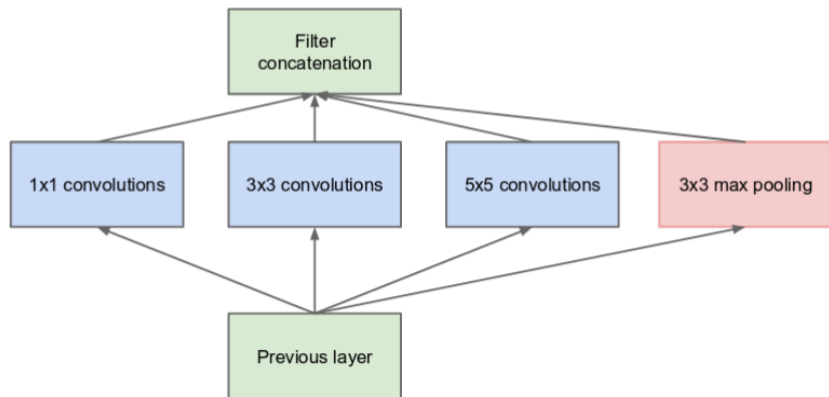


**Sparse Architecture** 찾음

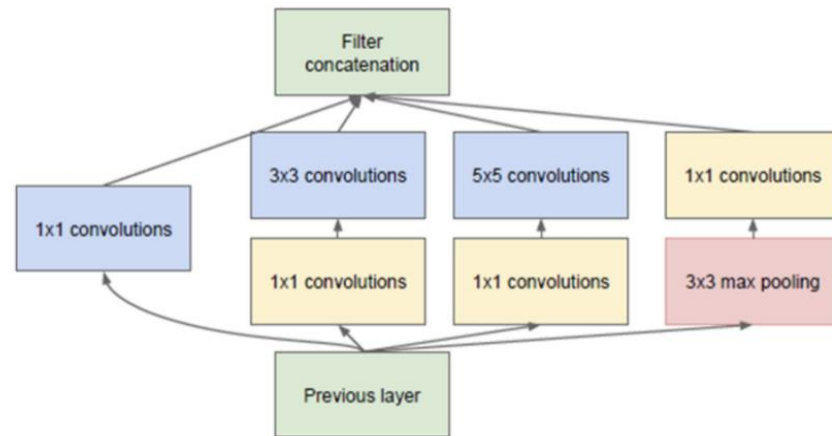
## GoogLeNet

## Inception Module

1x1, 3x3, 5x5의 합성곱 연산과 3x3 max pooling으로 얻은 feature map들을 합침



Naïve version



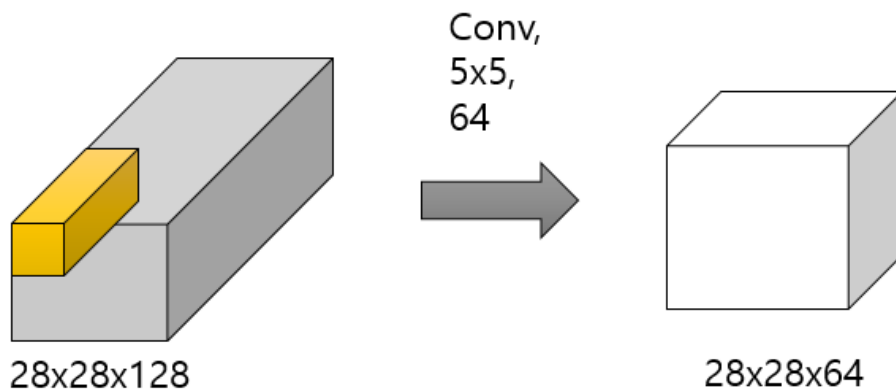
Dimension reduction version

## GoogLeNet

## 1x1 Convolution

모듈 내 여러 연산과 concatenate로 과도했던 파라미터 수를

**1x1 합성곱 연산**을 통해 감소시킴



$$\#params = 28 \times 28 \times 64 \times 5 \times 5 \times 128 = 160M$$

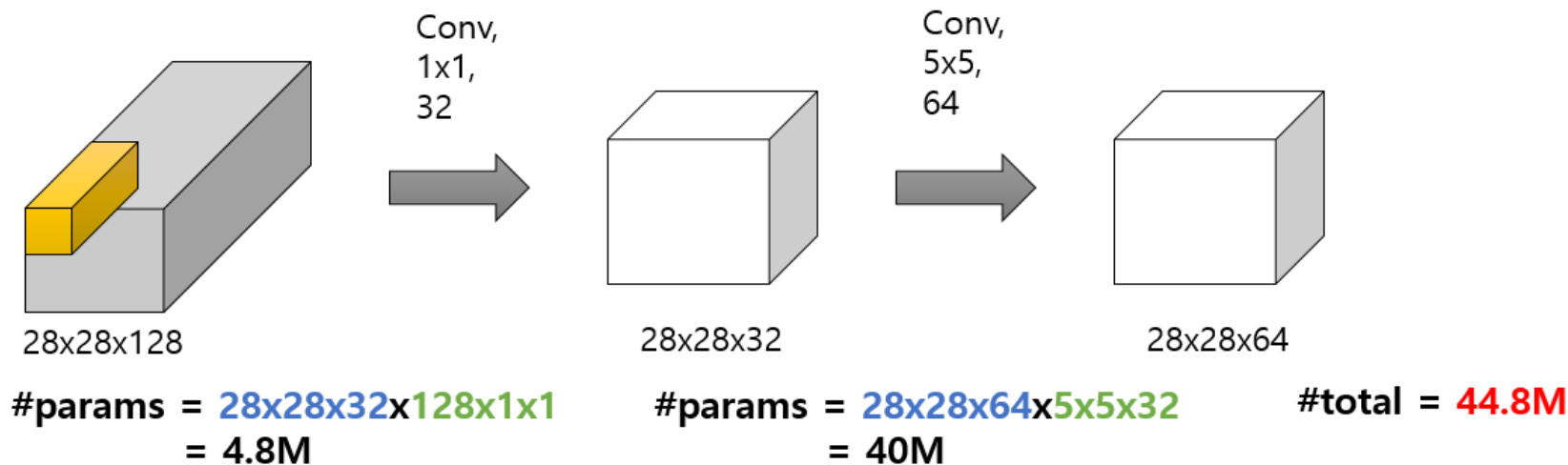
## 3

## 발전된 CNN 모델

## GoogLeNet

## 1x1 Convolution

즉 1x1 합성곱으로 본 연산 이전에 feature를 줄여 **차원 축소의 효과** 및 **비선형성** 추가!



같은 결과지만 파라미터 수가 150M에서 44.8M로 감소!





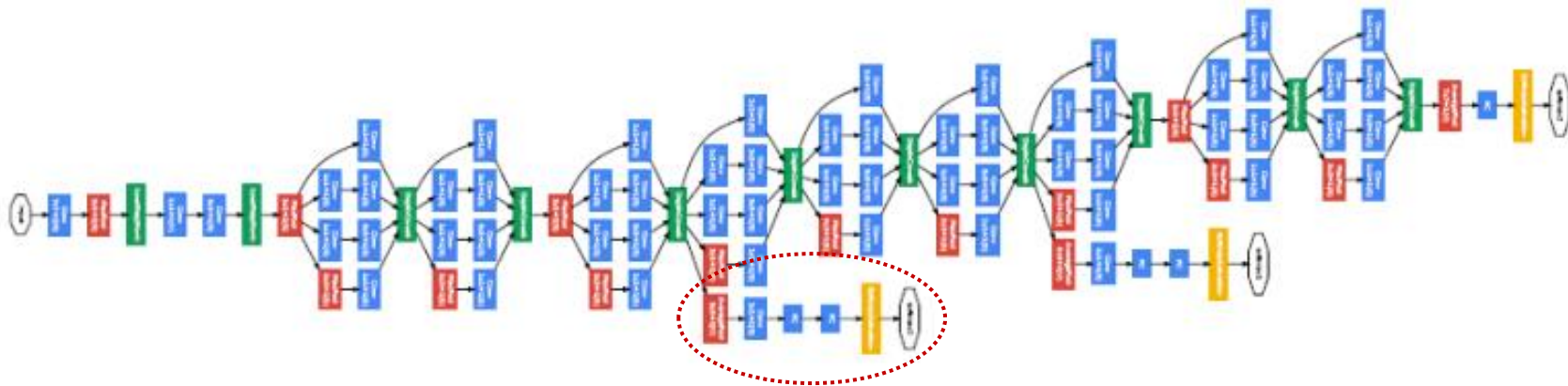
# 3

## 발전된 CNN 모델

### GoogLeNet

#### 보조분류기 *Auxiliary Classifier*

Vanishing Gradient를 방지하기 위해 **추가적인 오차 역전파**를 진행하는 분류기



Auxiliary Classifier

## GoogLeNet - 의의

### GoogLeNet의 문제 정의

Dense Network의 네트워크 크기 증가는 Overfitting과 과도한 연산량이 문제가 됨



Inception Module, 1x1 Conv 연산, 보조분류기를 사용하여  
파라미터의 수를 줄이고 Vanishing Gradient 문제 방지

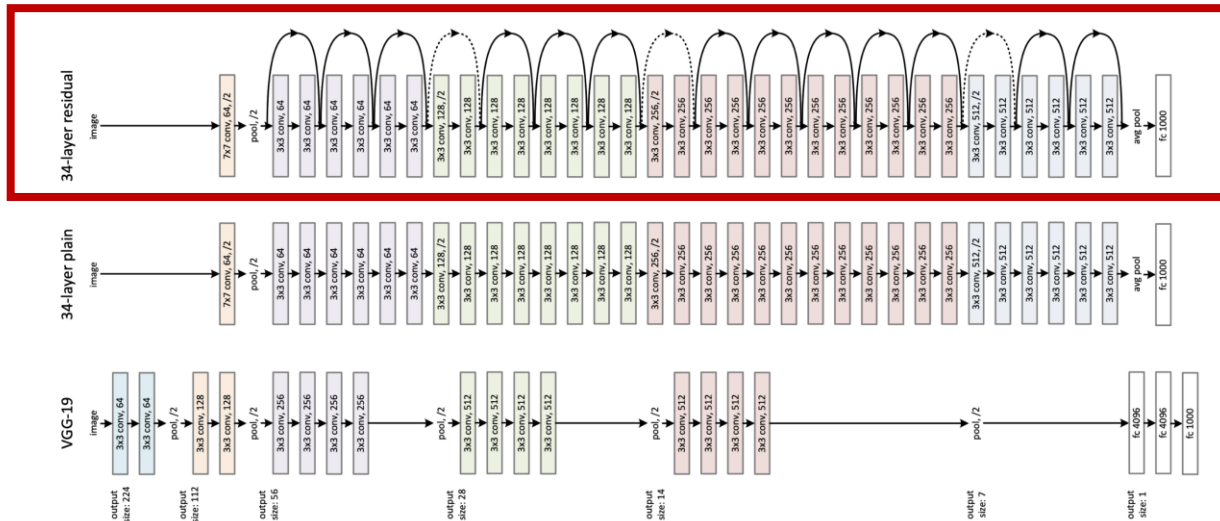
# 3

## 발전된 CNN 모델

ResNet

ResNet

2015년 ILSVRC에서 획기적인 성능으로 우승 차지



가장 위가 ResNet!



## ResNet - 문제 정의

## Degradation Problem

Layer 수가 더 많을 때 train error과 test error 모두 높음

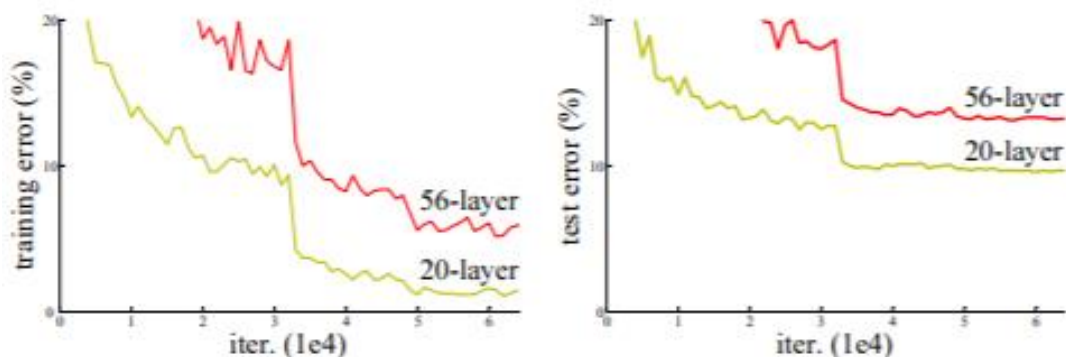


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

## ResNet - 문제 정의

## Degradation Problem

Layer 수가 더 많을 때 train error과 test error 모두 높음

Layer가 더 깊어질 수록 모델이 복잡해지고  
학습할 방향을 잡지 못해 **최적화가 복잡해지는 일종의 부작용!**

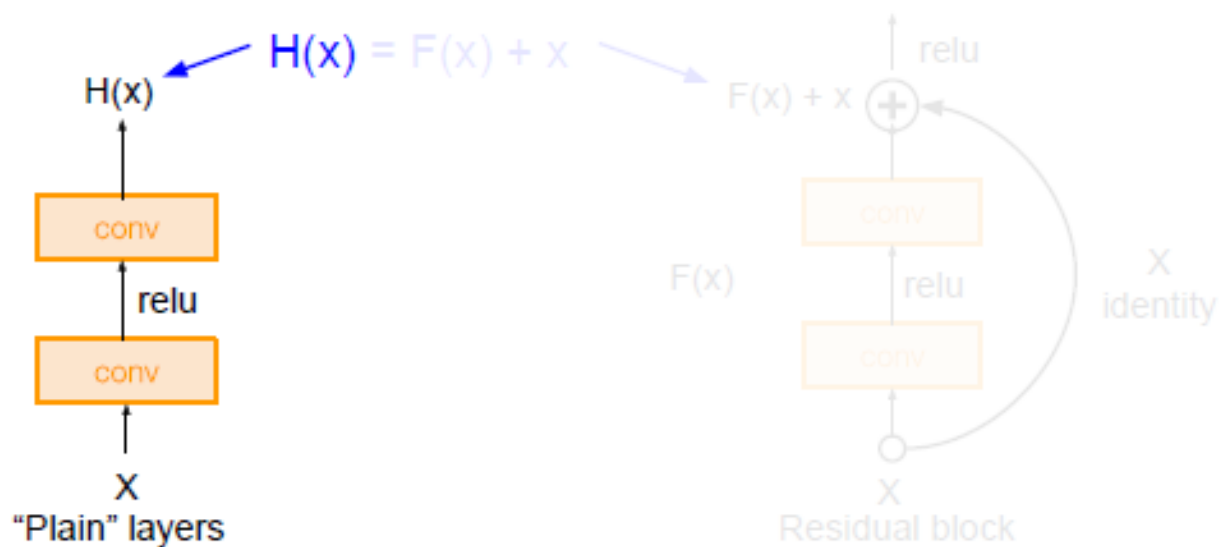


with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

## ResNet - Residual Block

### Skip connection

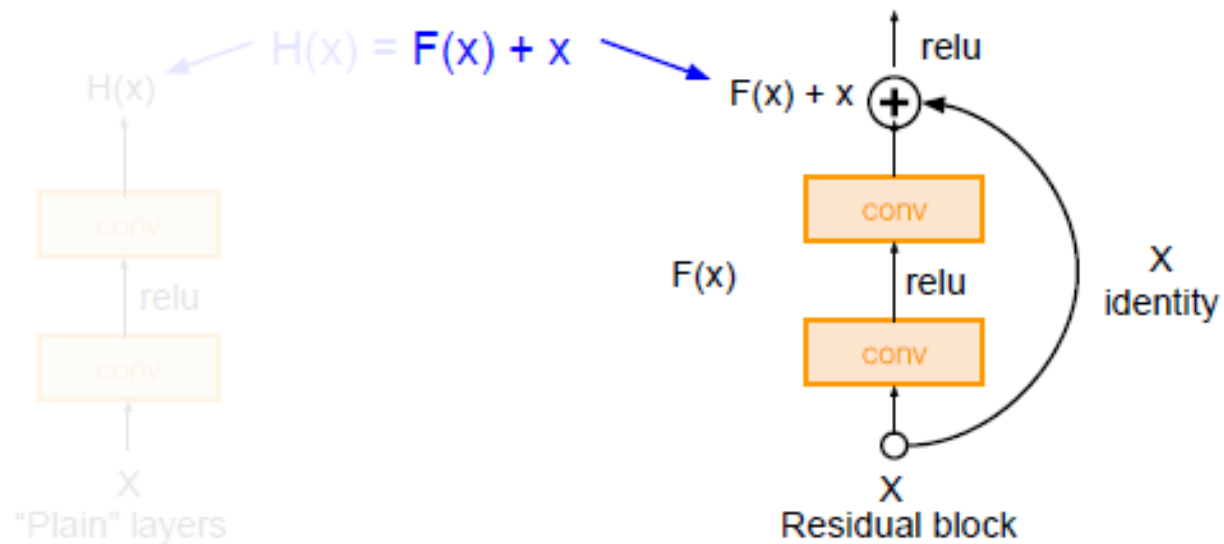
기존 방식은 input  $x$ 가 보존되지 않고 완전히 변형되어 **새로운 정보  $H(x)$** 로 바뀜!



## ResNet - Residual Block

### Skip connection

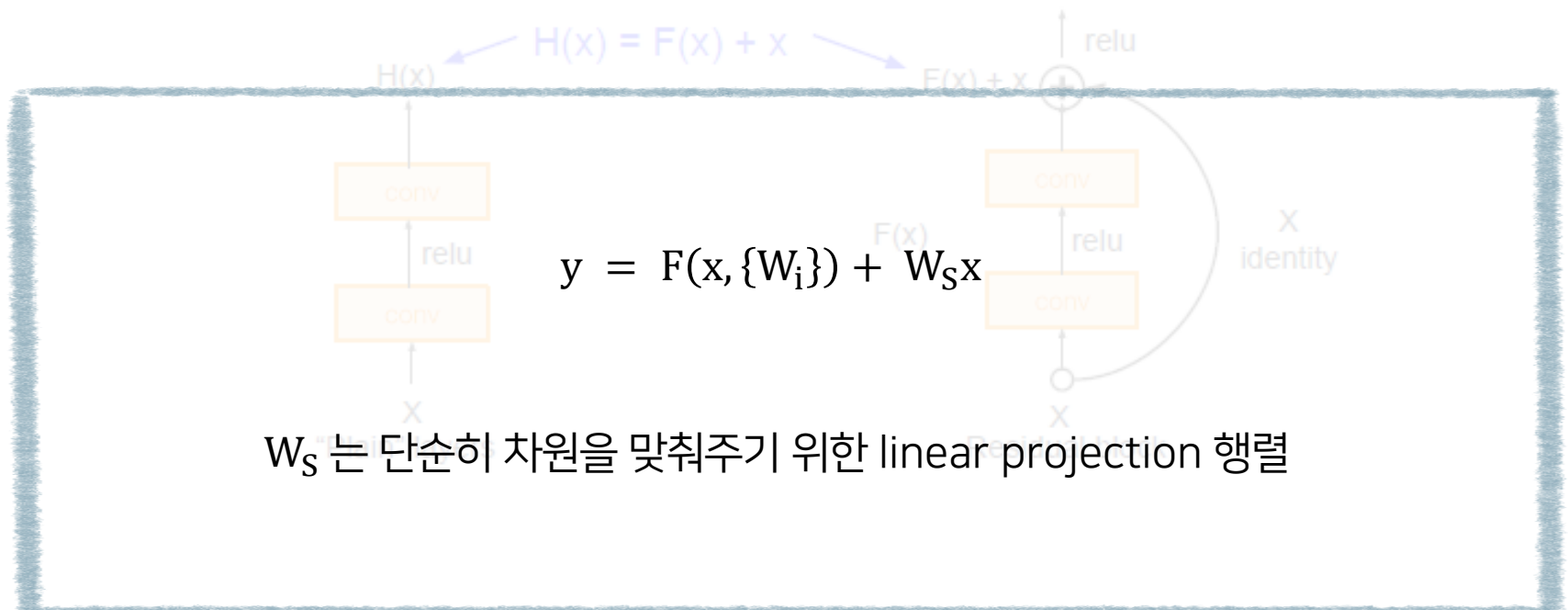
Identity  $x$ 를 output으로 전달하여  
각 블록이 항상 **추가적으로** 학습해야 할 정보만 학습함



## ResNet - Residual Block

## Skip connection

실제로는 아래의 식을 이용해 Skip Connection 진행





## ResNet - 의의

### ResNet의 문제 정의

Degradation Problem으로 인해 깊게 쌓아도 학습 성능이 저하됨



Residual Block에 **Skip Connection**을 적용하여  
각 블록들이 독립된 Mapping으로써  
추가적으로 학습해야 할 정보만 학습하여 훨씬 좋은 성능을 보임

# 4

객체 탐지

## 객체 탐지

객체 탐지 *Object Detection*

이미지 내에서 특징을 탐지해내는 과정



단순히 이미지를 분류하는 **Image classification**

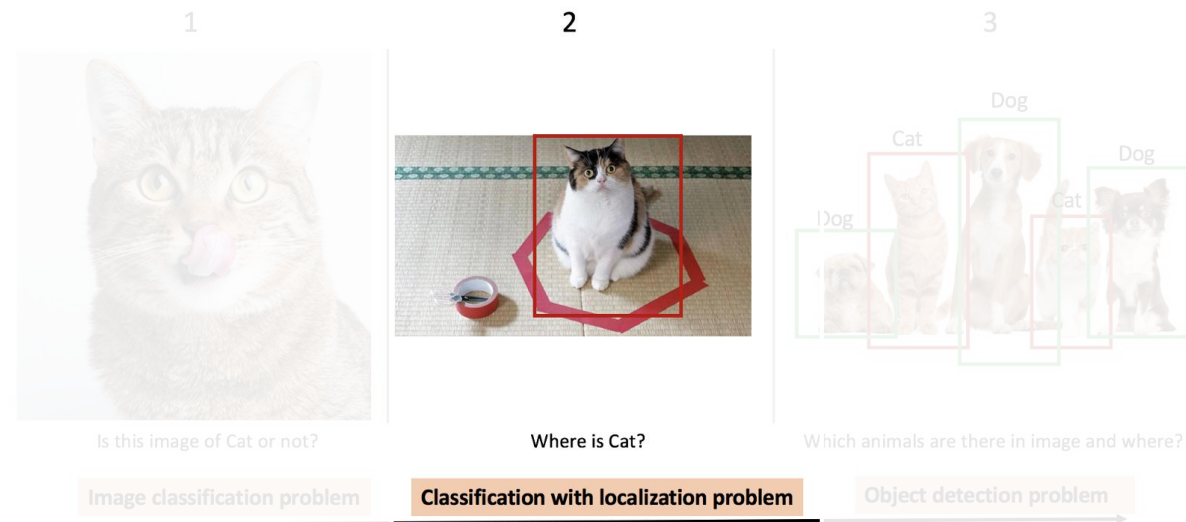
# 4

## 객체 탐지

### 객체 탐지

#### 객체 탐지 *Object Detection*

이미지 내에서 특징을 탐지해내는 과정

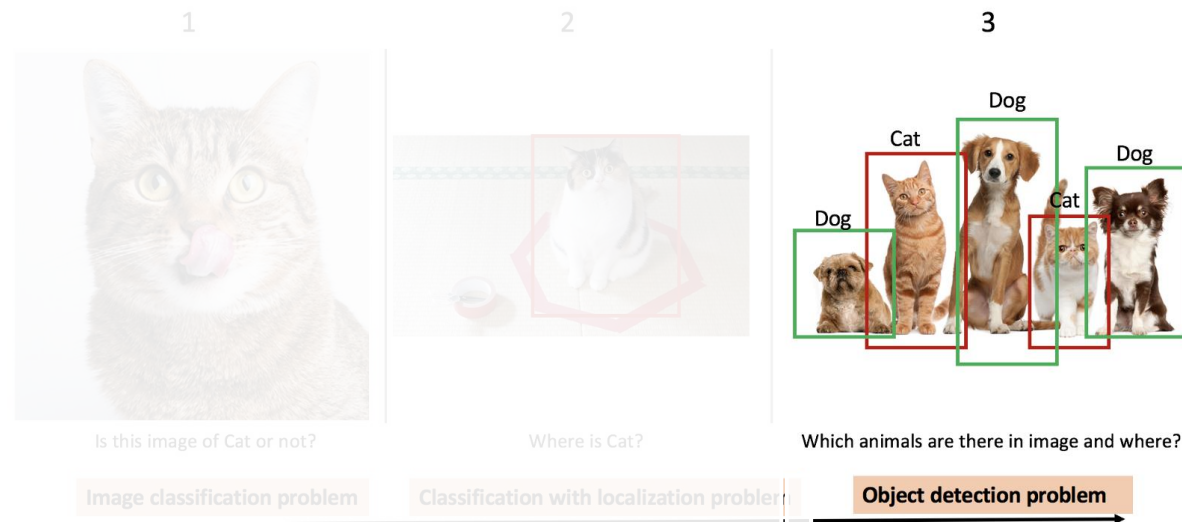


물체가 어디에 있는지를 찾아내는 **localization**

## 객체 탐지

객체 탐지 *Object Detection*

이미지 내에서 특징을 탐지해내는 과정



Classification과 Localization을 이미지 내의 여러 물체에 대해 수행하는 과정이 **Object Detection**

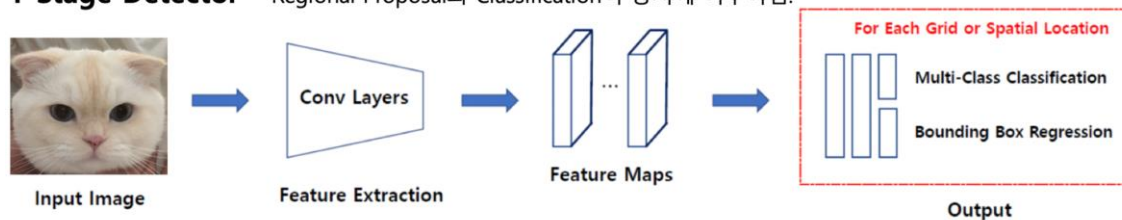
## 객체 탐지

**2-Stage Detector** - Regional Proposal와 Classification이 순차적으로 이루어짐.



Ex) R-CNN 계열 (R-CNN, Fast R-CNN, Faster R-CNN, R-FCN, Mask R-CNN ...)

**1-Stage Detector** - Regional Proposal와 Classification이 동시에 이루어짐.

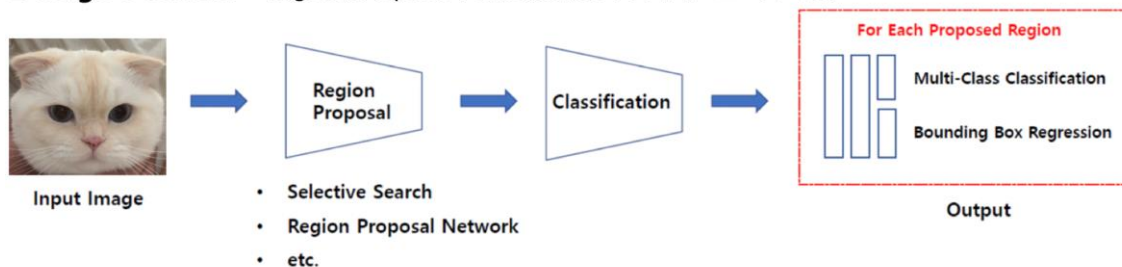


Ex) YOLO 계열 (YOLO v1, v2, v3), SSD 계열 (SSD, DSSD, DSOD, RetinaNet, RefineDet ...)

영역 지정과 분류를 한 번에 다루는 **1-Stage Detector**

## 객체 탐지

**2-Stage Detector** - Regional Proposal와 Classification이 순차적으로 이루어짐.



Ex) **R-CNN 계열** (R-CNN, Fast R-CNN, Faster R-CNN, R-FCN, Mask R-CNN ... )

**1-Stage Detector** - Regional Proposal와 Classification이 동시에 이루어짐.



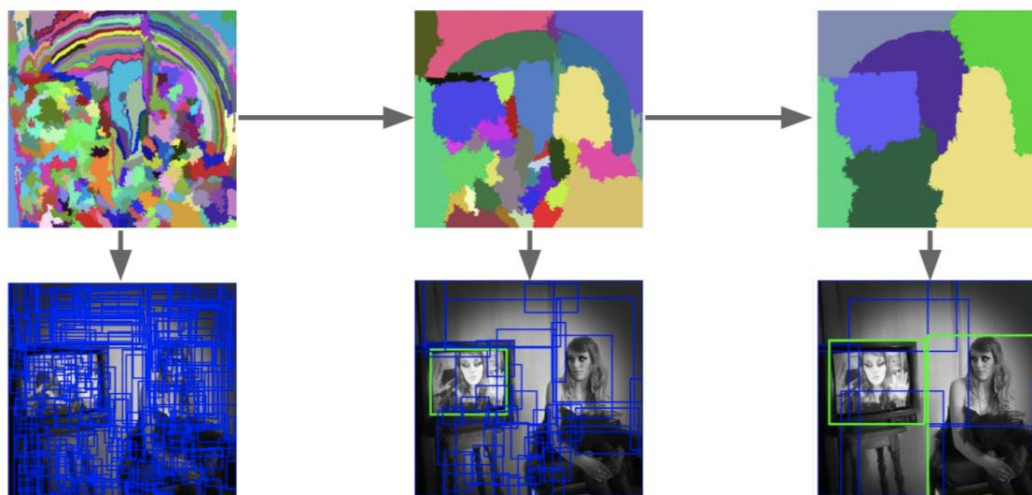
Ex) **YOLO 계열** (YOLO v1, v2, v3) , **SSD 계열** (SSD, DSSD, DSOD, RetinaNet, RefineDet ... )

영역 지정과 분류를 따로 다루는 **2-Stage Detector**

## 2-Stage Detector

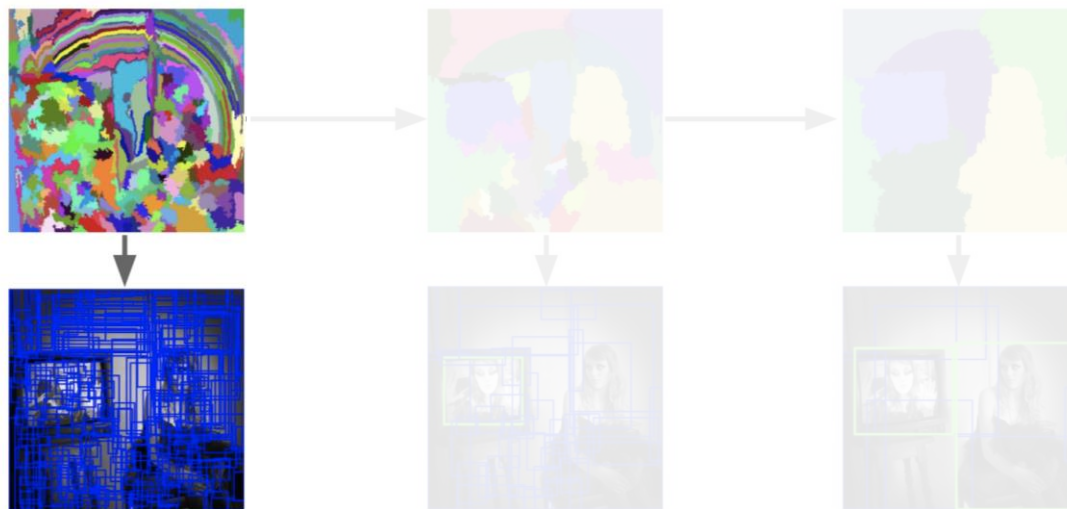
### *Selective Search*

Region Proposal(영역 지정)의 대표적인 방법





## 2-Stage Detector - ① Selective Search

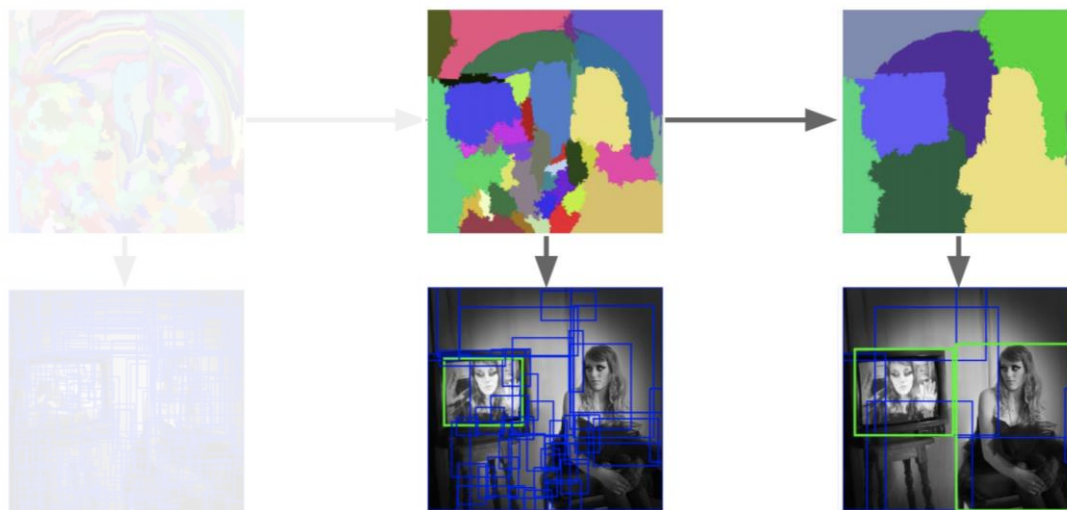


초기 특징들을 대상으로 **많은 후보군을 선정**하여, 물체가 있을 법한 위치를 체크

## 4

## 객체 탐지

## 2-Stage Detector - ① Selective Search



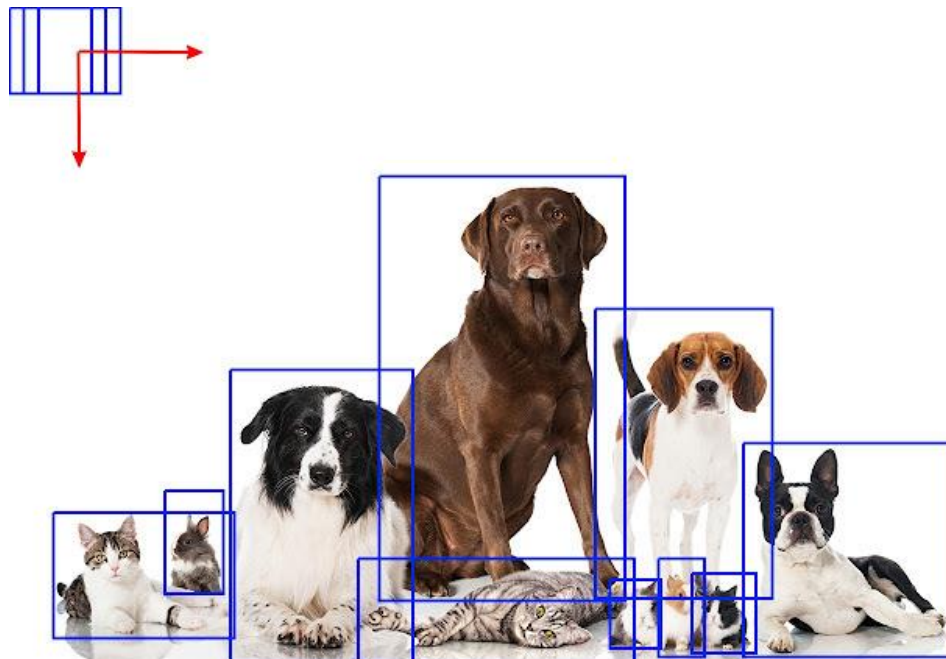
물체가 있을 것 같은 위치를  
**boundary box**라고 부름

후보군들을 합쳐 나가며 **최종 위치를 지정**한 후, **Classification** 진행

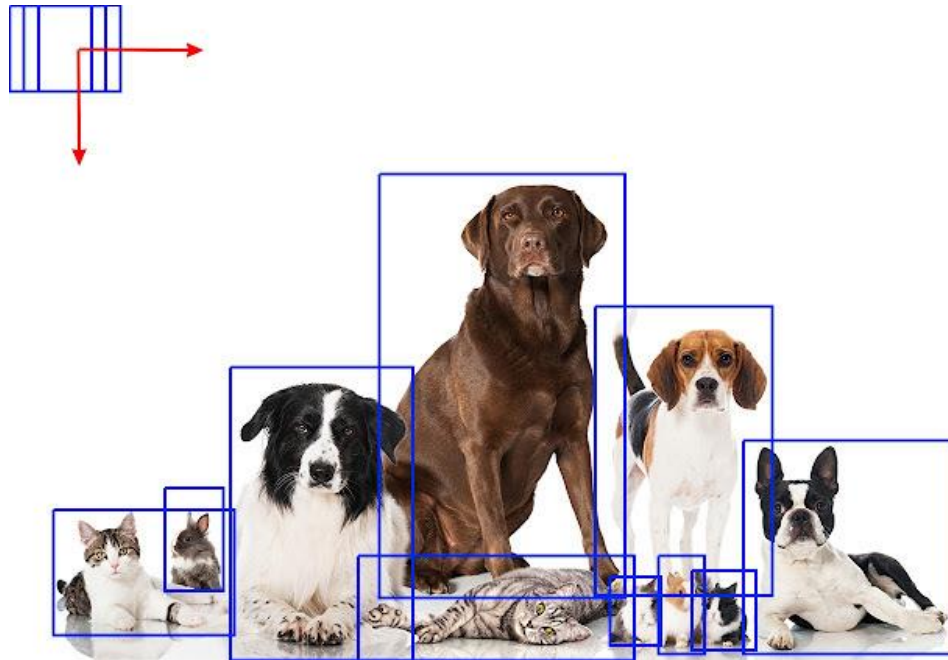
## 2-Stage Detector

### Sliding Window

정해진 크기의 박스가 이미지를 돌아다니면서 물체가 있다고 생각되는 박스를 추출



## 2-Stage Detector - ② Sliding Window



여러 크기의 박스를 전체적으로 탐색해야 하기 때문에 **속도가 느림**

## 2-Stage Detector의 장단점

### ✓ 장점

각각의 과정을 나누어 해당 과정에 맞게  
확실히 **최적화**시킬 수 있음



**성능이 높음 !**

### ✓ 단점

영역 지정과 Classification이  
따로 진행되기 때문에  
**시간이 오래** 걸림

## 1-Stage Detector

Region Proposal과 Classification이 **동시에** 이루어지는 Detector

**End-to-End의 학습**이 가능



구조 단순 + 빠른 추론

## 1-Stage Detector



Region Proposal과 Classification이 동시에 이루어지는 Detector

Q. **End-to-End** 학습이란?

A. 중간 처리 단계 없이 입력 데이터에서 출력 데이터로  
**하나의 Loss function을 사용하는** 학습 과정




구조 단순 + 빠른 추론

## Object Detection 평가지표 - ① IOU

IOU *Intersection of Union*

탐지의 정확도를 평가하는 지표

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$




교집합  
합집합

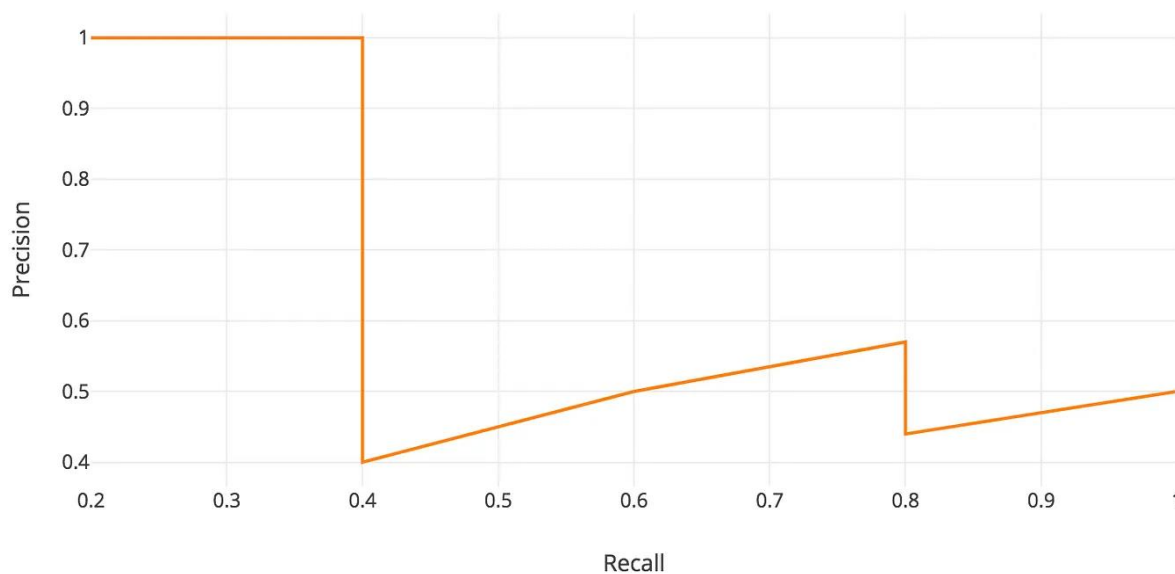
으로 이해!!



## Object Detection 평가지표 - ② AP

AP *Average Precision*

정밀도( $\frac{TP}{TP+FP}$ )의 평균값으로 계산



$$AP = \int_0^1 p(r) dr$$

$$mAP = \frac{1}{N} \sum_{i=1}^n AP_i$$

적분을 통해 구한  
AP값의 평균

## Object Detection 평가지표 - ③ mAP@IOU

$$mAP = \frac{1}{N} \sum_{i=1}^n AP_i$$

IOU에 따라 Positive로 판단되어 계산된 **Precision**를 **평균**낸 값



**IOU Threshold**에 따라 값이 달라짐

## Object Detection 평가지표 - ③ mAP@IOU

AP에 따른 mAP 표기		IOU
AP@IoU=0.50	AP50	0.50
AP@IoU=0.55	AP55	0.55
AP@IoU=0.60	AP60	0.60
AP@IoU=0.65	AP65	0.65
AP@IoU=0.70	AP70	0.70
AP@IoU=0.75	AP75	0.75
AP@IoU=0.80	AP80	0.80
AP@IoU=0.85	AP85	0.85
AP@IoU=0.90	AP90	0.90
AP@IoU=0.95	AP95	0.95

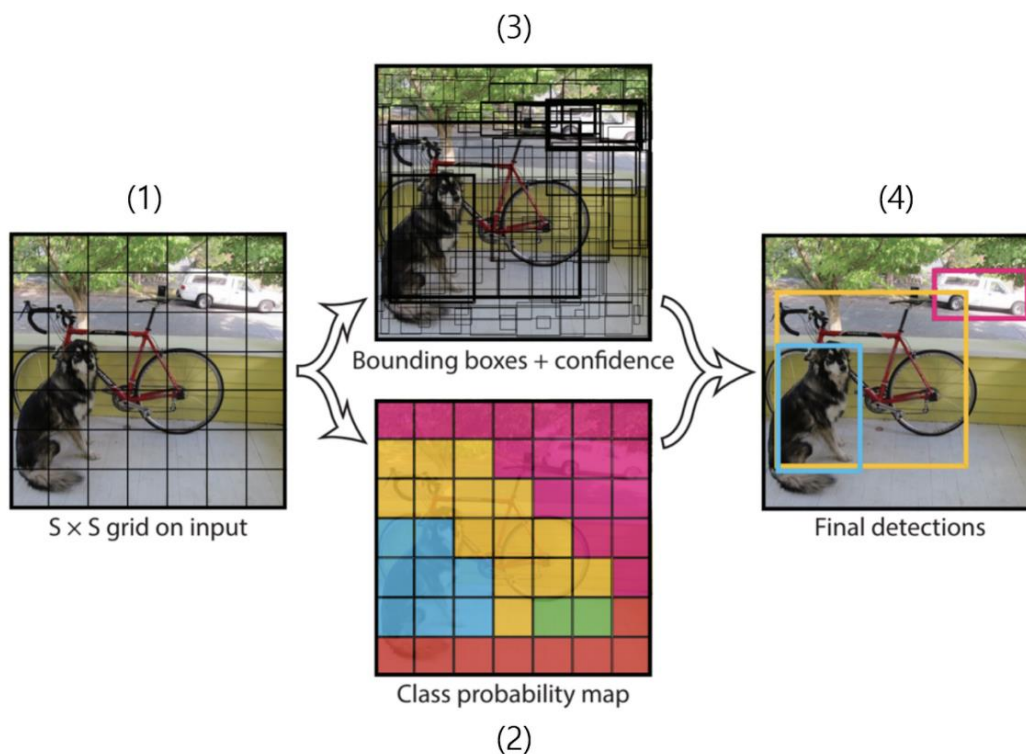
waytoliah.com

실제로는 mAP@IOU들을 가중평균 낸 **AP@[.5:.05:.95]**를 **AP**라고 지칭

## YOLO 모델

YOLO *You Only Look Once*

Object Detection 중 가장 뛰어난 속도와 1-Stage Detector 중 만족할 만한 성능을 보임



## YOLO 모델

*YOLO You Only Look Once*

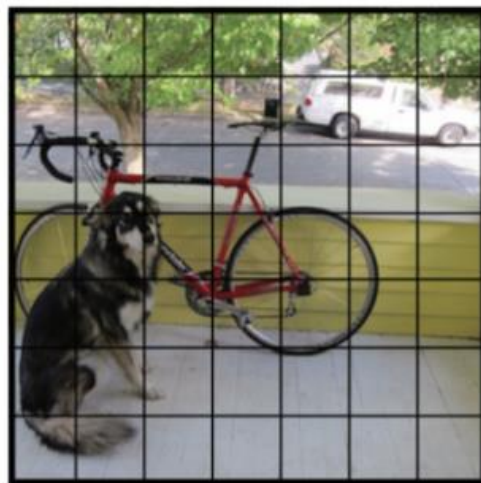
Object Detection 중 **가장 뛰어난 속도**와 1-Stage Detector 중 **만족할 만한 성능**을 보임



# 4

## 객체 탐지

### YOLO 모델



$S \times S$  grid on input



7 X 7 예시

$S \times S$  의 그리드로 이미지를 나눔

## YOLO 모델



YOLOv1에서는 그리드마다 **두 개의 bounding box**를 가짐

## YOLO 모델

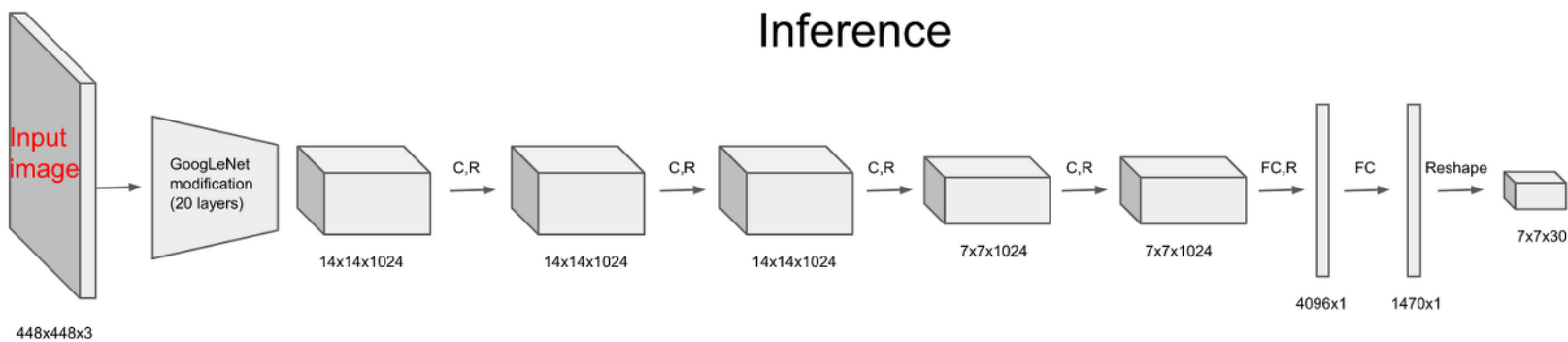


예시에서는 각 grid는  $(5 \times 2) + 20$ 의 채널 공간을 가짐

각각의 grid는  $(5 \times 2) + \text{\# of classes}$ 의 채널 공간을 가짐



## YOLO 모델

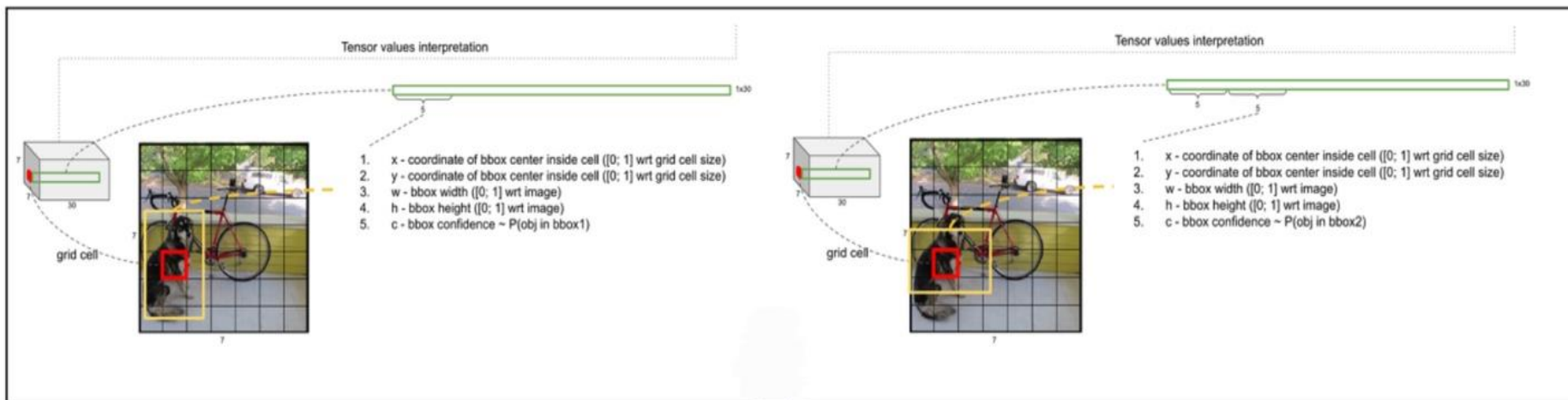


앞의 예시에서는 총 7 X 7 X 30의 차원의 grid cell을 **잘 예측**하는 것이 목표 !!

일반화하면  $S \times S \times \{(5 \times 2) + \# \text{ of classes}\}$  차원의 grid cell

## YOLO 모델

## Bounding box의 구성 요소



$x, y$ 는 bounding box의 **중심 좌표**



해당 그리드에 대한 상대 좌표로 표시

$w, h$ 는 bounding box의 **너비와 높이**



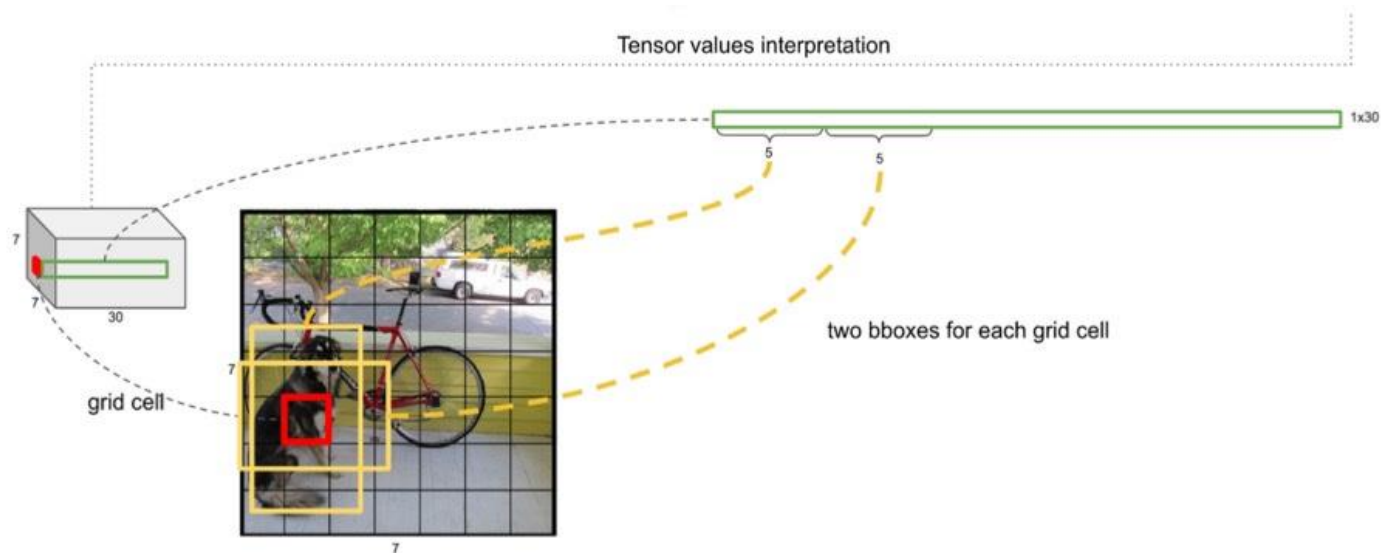
그리드 크기 결정

$c$ 는 **box confidence**



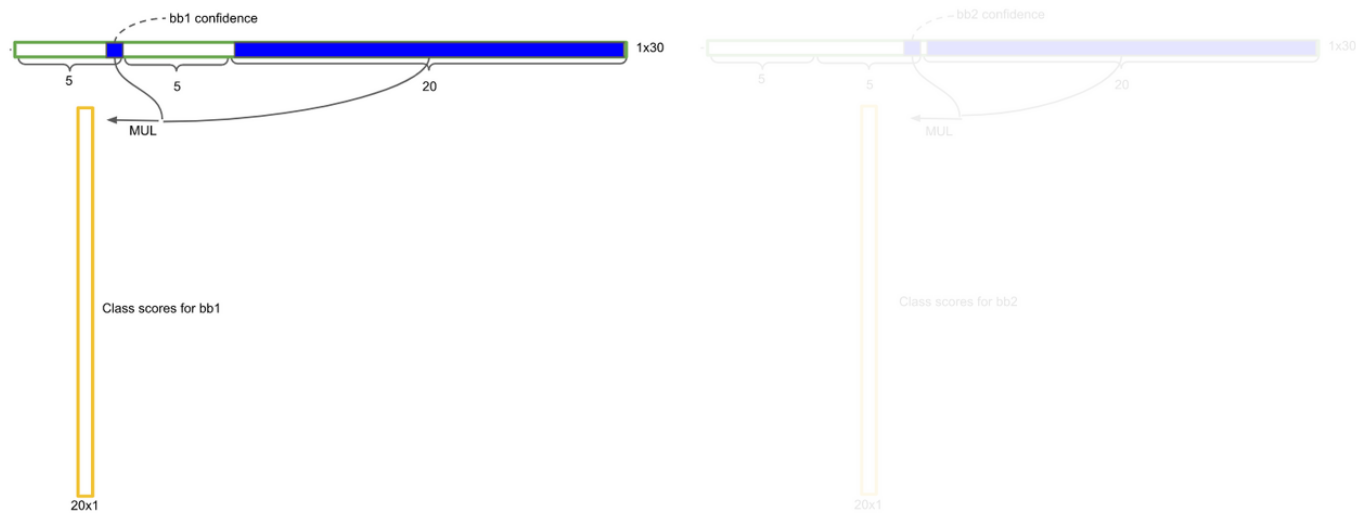
해당 영역에 물체가 존재할 확률

## YOLO 모델



Grid cell이 학습을 진행하며 **bounding box의 위치와 너비,**  
**그리드에 해당하는 클래스의 확률**까지 예측 !

## YOLO 모델

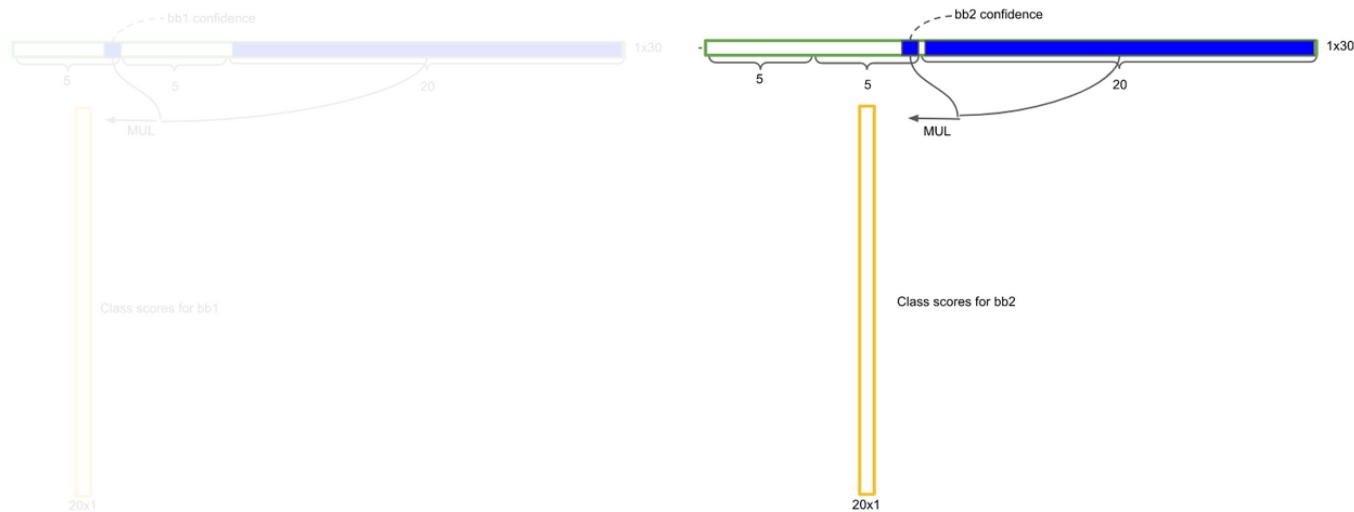


예측된 grid cell에서 **클래스 분류 점수**를 도출 -> 총 7 X 7 X 2의 98개

## 4

## 객체 탐지

## YOLO 모델



클래스 분류 점수가 **0.2 이상**인 grid cell의 bounding box만 남게 됨

## YOLO 모델



Q. 각 grid 마다 bounding box가 두 개씩 나타나면  
너무 많은 bounding box가 생기지 않을까?

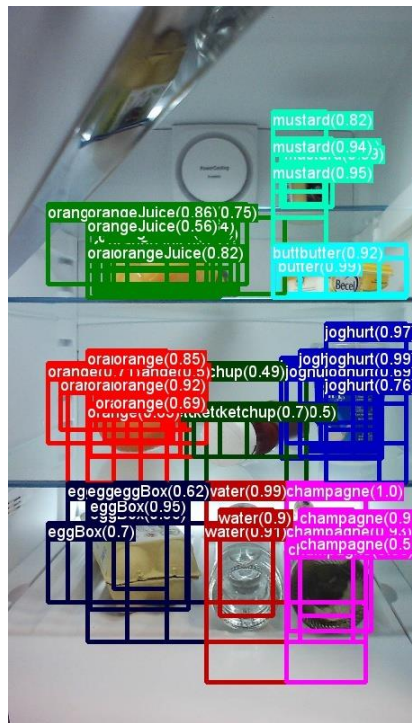
A. NMS를 통해 간단히 해결 가능 !

Confidence가 0.2보다 작으면 0으로 채우고 나머지만 살림

# 4

## 객체 탐지

### YOLO 모델 - NMS step



특정 % 이상으로 겹치는 bounding box에 대해 **최댓값만 남기고 나머지는 지움**

## YOLO 모델 학습

## YOLO의 Loss Function

$$\begin{aligned}
 & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
 & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{\omega_i} - \sqrt{\hat{\omega}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(C_i - \hat{C}_i)^2] \\
 & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} [(C_i - \hat{C}_i)^2] \\
 & + \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned}$$

$S$  : grid 크기,  $B$  : bounding box 개수,  $\lambda_{coord}$  : coordinate  $(x, y, w, h)$ 에 대한 loss의 균형을 위한 값,  $\lambda_{noobj}$  : Object가 있는 box와 없는 box의 균형을 위한 값

$1_{ij}^{obj}$  : Object가 존재하는 grid cell의  $i$ 의 bounding box,  $1_{ij}^{noobj}$  : Object가 존재하지 않는 grid cell의 bounding box,  $1_i^{obj}$  : Object가 존재하는 grid cell  $i$ ,



## YOLO 모델 학습

## YOLO의 Loss Function

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

Object가 존재하는 grid cell i와 bbox j에 대해

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} \text{ x와 y좌표의 loss 계산}$$

→ **Localization Loss**

$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} [(C_i - \hat{C}_i)^2]$$

$$+ \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

$S$  : grid 크기,  $B$  : bounding box 개수,  $\lambda_{coord}$  : coordinate  $(x, y, w, h)$ 에 대한 loss의 균형을 위한 값,  $\lambda_{noobj}$  : Object가 있는 box와 없는 box의 균형을 위한 값

$1_{ij}^{obj}$  : Object가 존재하는 grid cell의  $i$ 의 bounding box,  $1_{ij}^{noobj}$  : Object가 존재하지 않는 grid cell의 bounding box,  $1_i^{obj}$  : Object가 존재하는 grid cell  $i$ ,

## YOLO 모델 학습

## YOLO의 Loss Function

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

Object가 존재하는 grid cell i와 bbox j에 대해

w와 h 좌표의 loss 계산

$$\rightarrow \text{Localization Loss}$$

$$+ \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

$S$  : grid 크기,  $B$  : bounding box 개수,  $\lambda_{coord}$  : coordinate  $(x, y, w, h)$ 에 대한 loss의 균형을 위한 값,  $\lambda_{noobj}$  : Object가 있는 box와 없는 box의 균형을 위한 값

$1_{ij}^{obj}$  : Object가 존재하는 grid cell의  $i$ 의 bounding box,  $1_{ij}^{noobj}$  : Object가 존재하지 않는 grid cell의 bounding box,  $1_i^{obj}$  : Object가 존재하는 grid cell  $i$ ,

## YOLO 모델 학습

## YOLO의 Loss Function

Object가 존재하는 grid cell  $i$ 에 대해  
confidence score의 loss 계산

$$\rightarrow \text{Confidence Loss} + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(C_i - \hat{C}_i)^2]$$

$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} [(C_i - \hat{C}_i)^2]$$

$$+ \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

$S$  : grid 크기,  $B$  : bounding box 개수,  $\lambda_{coord}$  : coordinate  $(x, y, w, h)$ 에 대한 loss의 균형을 위한 값,  $\lambda_{noobj}$  : Object가 있는 box와 없는 box의 균형을 위한 값

$1_{ij}^{obj}$  : Object가 존재하는 grid cell의  $i$ 의 bounding box,  $1_{ij}^{noobj}$  : Object가 존재하지 않는 grid cell의 bounding box,  $1_i^{obj}$  : Object가 존재하는 grid cell  $i$ ,

## YOLO 모델 학습

## YOLO의 Loss Function

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

Object가 존재하지 않는 grid cell i에 대해  
confidence score의 loss 계산

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} [C_i - \hat{C}_i]^2 \rightarrow \text{Confidence Loss}$$

$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} [(C_i - \hat{C}_i)^2]$$

$$+ \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

$S$  : grid 크기,  $B$  : bounding box 개수,  $\lambda_{coord}$  : coordinate  $(x, y, w, h)$ 에 대한 loss의 균형을 위한 값,  $\lambda_{noobj}$  : Object가 있는 box와 없는 box의 균형을 위한 값

$1_{ij}^{obj}$  : Object가 존재하는 grid cell의  $i$ 의 bounding box,  $1_{ij}^{noobj}$  : Object가 존재하지 않는 grid cell의 bounding box,  $1_i^{obj}$  : Object가 존재하는 grid cell  $i$ ,

## YOLO 모델 학습

## YOLO의 Loss Function

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{\omega_i} - \sqrt{\hat{\omega}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

Object가 존재하는 grid cell i에 대해  
classification score의 loss 계산

$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{noobj} [(c_i - \hat{c}_i)^2]$$

→ **Classification Loss**

$$+ \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

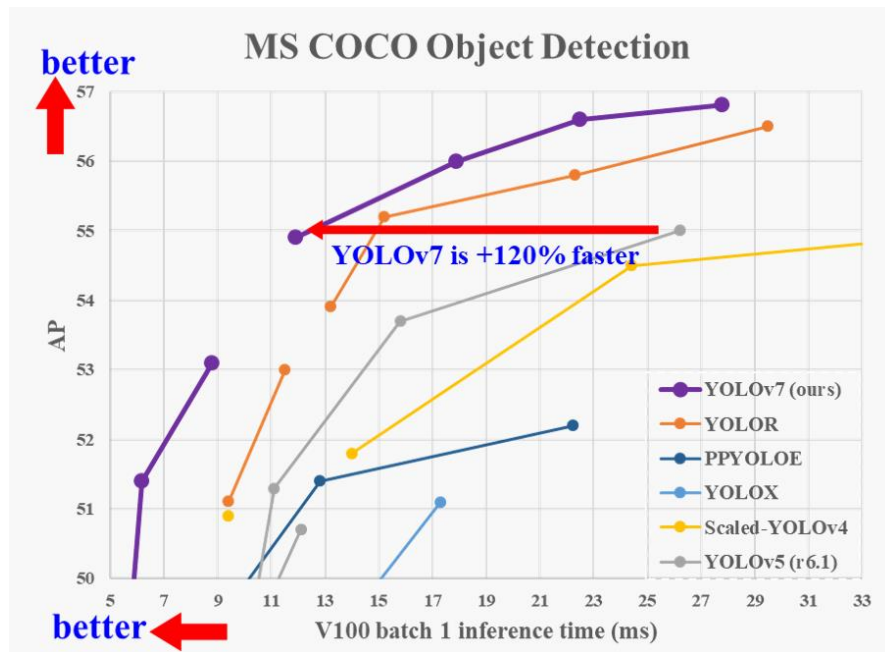
$S$  : grid 크기,  $B$  : bounding box 개수,  $\lambda_{coord}$  : coordinate  $(x, y, w, h)$ 에 대한 loss의 균형을 위한 값,  $\lambda_{noobj}$  : Object가 있는 box와 없는 box의 균형을 위한 값

$1_{ij}^{obj}$  : Object가 존재하는 grid cell의  $i$ 의 bounding box,  $1_{ij}^{noobj}$  : Object가 존재하지 않는 grid cell의 bounding box,  $1_i^{obj}$  : Object가 존재하는 grid cell  $i$ ,

## 4

## 객체 탐지

## YOLO 모델



YOLOv7은 13ms당 한 번씩 Detection하는 경우,  
AP가 58이 넘어가는 속도 대비 놀라운 성능을 보여줌

# 다음주 예고

---

1. Neural Style Transfer
2. VAE
3. GAN
4. Conditional GAN
5. Image-to-Image Translation



THANK YOU

