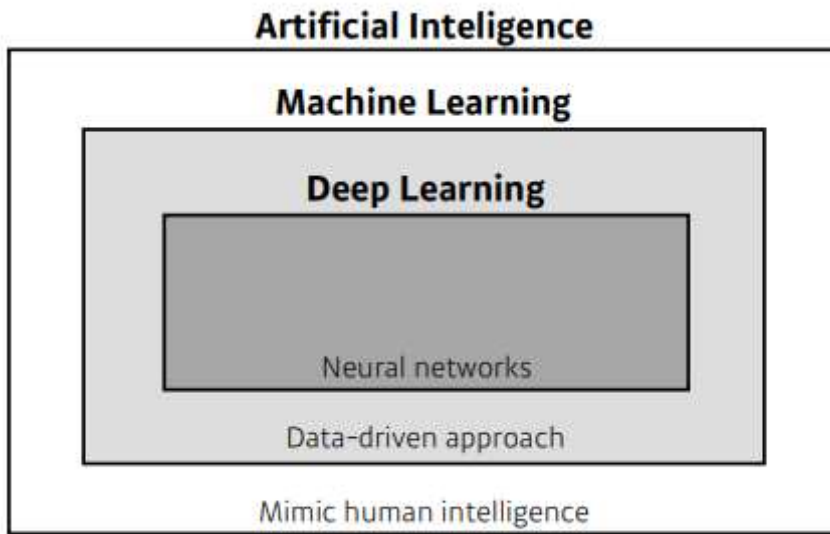


## ▼ (01강) 딥러닝 기본 용어 설명 - Historical Review



- 인공지능 : 인간의 지능 모방
  - 머신러닝 : 데이터를 통해서 학습
    - 딥러닝 : 뉴럴 네트워크 사용

딥러닝 핵심 구성 요소 : 예시

- 학습에 필요한 데이터 : 텍스트, 비디오
- 데이터를 통해 학습한 모델 : CNN, MLP, RNN
- 모델을 평가하기 위한 손실함수 : RMSE, Cross entropy, MLE
- 손실함수를 최소화시키는 알고리즘 : SGD, Adam Optimizer, RAdam Optimizer
  - Dropout
  - Early stopping
  - k-fold validation
  - Weight decay
  - Batch normalization
  - MixUp
  - Ensemble
  - Bayesian Optimization

## ▼ (02강) 뉴럴 네트워크 - MLP (Multi-Layer Perceptron)

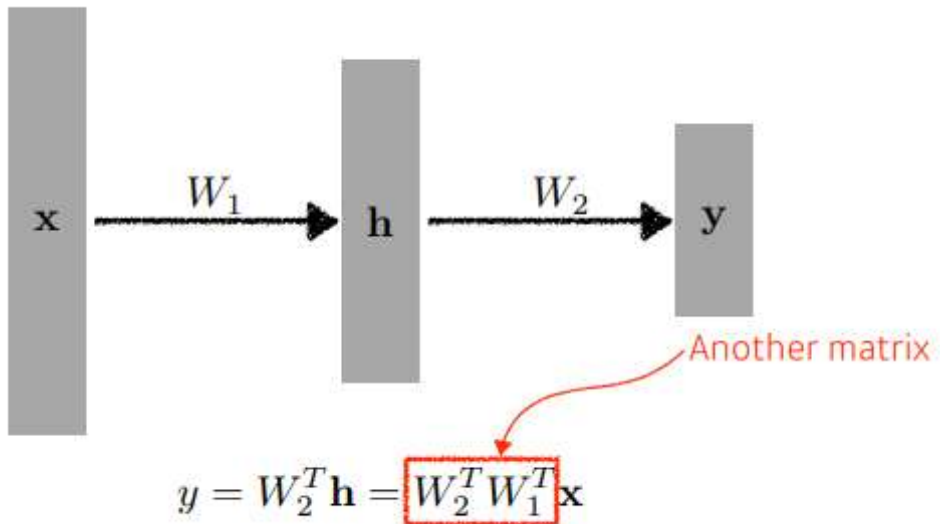
뉴럴 네트워크 정의

- 인간의 신경망 모방 + 알파

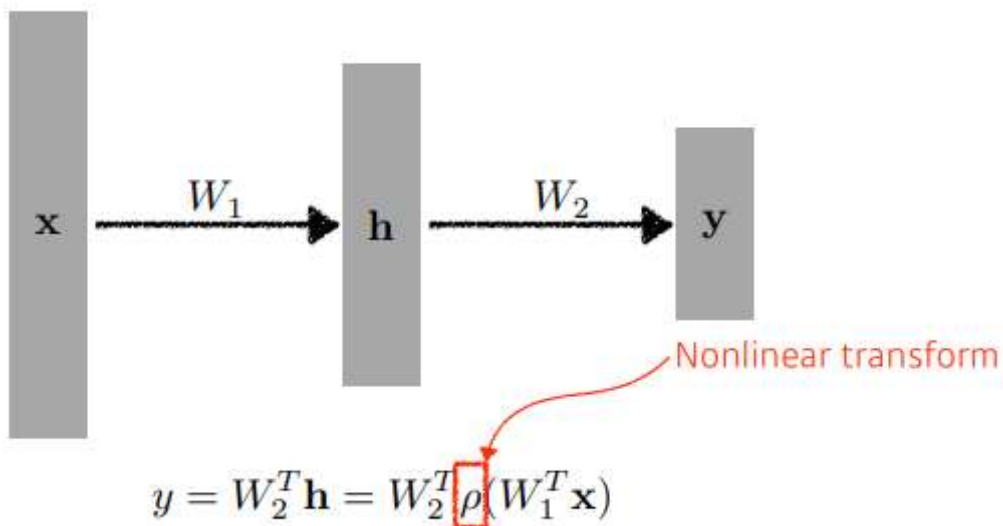
- 행렬곱과 비선형 연산 반복적으로 이루어지는 근사 함수

step size 변경하는 것이 optimizer

## ▼ 선형 결합



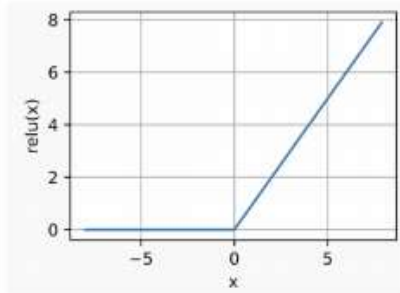
## ▼ 비선형 결합



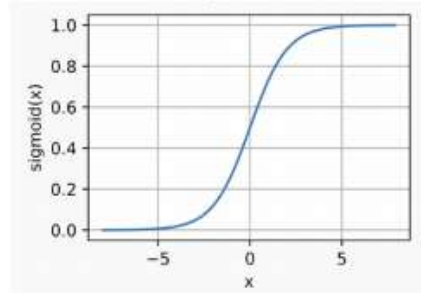
## ▼ 활성화 함수

활성 함수라는 Nonlinear transform 있어야 네트워크를 깊게 쌓여도 작동을 잘 한다.

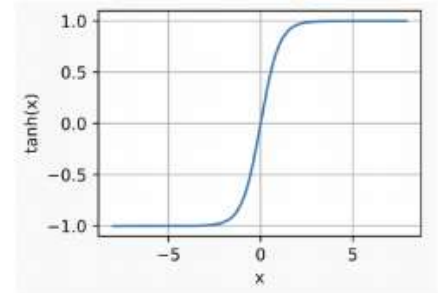
Rectified Linear Unit (ReLU)



Sigmoid



Hyperbolic Tangent



## ▼ Universal Approximation theorem

잘 만든 NN의 표현력이 크다는 것을 증명

### English

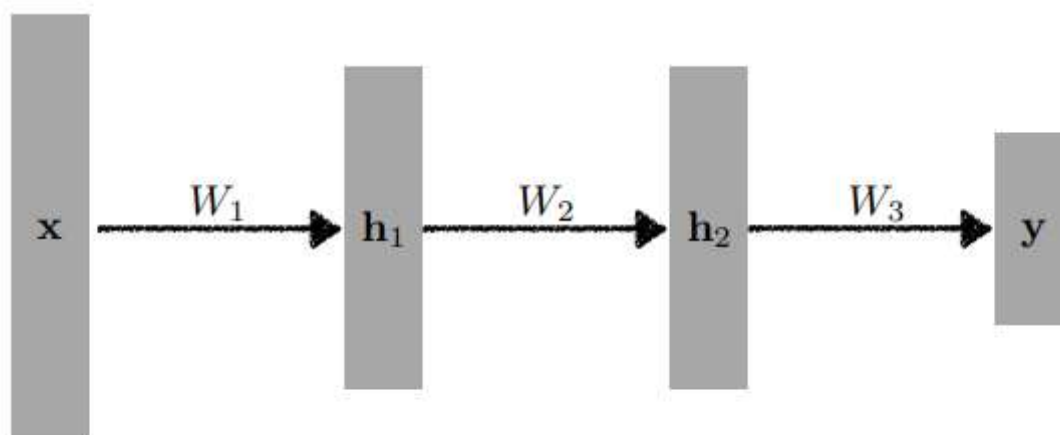
There is a single hidden layer feedforward network that approximates any measurable function to any desired degree of accuracy on some compact set  $K$ .

### Math

For every function  $g$  in  $M^r$  there is a compact subset  $K$  of  $R^r$  and an  $f \in \sum^r(\Psi)$  such that for any  $\epsilon > 0$  we have  $\mu(K) < 1 - \epsilon$  and for every  $X \in K$  we have  $|f(x) - g(x)| < \epsilon$ , regardless of  $\Psi$ ,  $r$ , or  $\mu$ .

## ▼ Multi-Layer Perceptron

NN 구조에서 히든 레이어 1개 이상 있을 때 Multi-Layer Perceptron이라고 한다.



$$y = W_3^T h_2 = W_3^T \rho(W_2^T h_1) = W_3^T \rho(W_2^T \rho(W_1^T x))$$

## ▼ Loss functions

Regression Task

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N \sum_{d=1}^D (\underbrace{y_i^{(d)}}_{\text{True target}} - \underbrace{\hat{y}_i^{(d)}}_{\text{Predicted output}})^2$$

Classification Task

$$\text{CE} = -\frac{1}{N} \sum_{i=1}^N \sum_{d=1}^D y_i^{(d)} \log \hat{y}_i^{(d)}$$

Probabilistic Task

$$\text{MLE} = \frac{1}{N} \sum_{i=1}^N \sum_{d=1}^D \log \mathcal{N}(y_i^{(d)}; \hat{y}_i^{(d)}, 1) \quad (= \text{MSE})$$

$\hat{y}_T : \text{logit}$

## ▼ (03강) Optimization

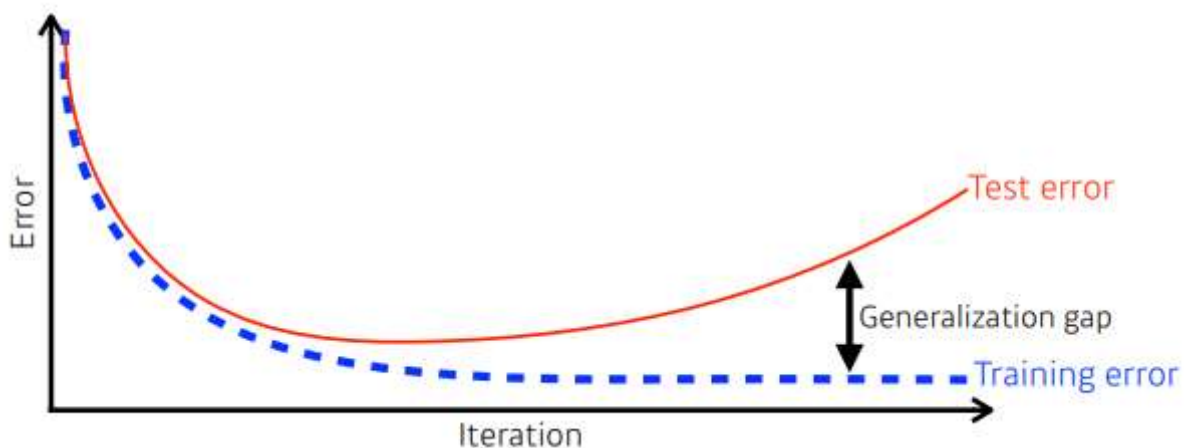
최적화 기법

- Generalization
- Under-fitting vs. over-fitting
- Cross validation
- Bias-variance tradeoff
- Bootstrapping
- Bagging and boosting

## ▼ Concepts in Optimization

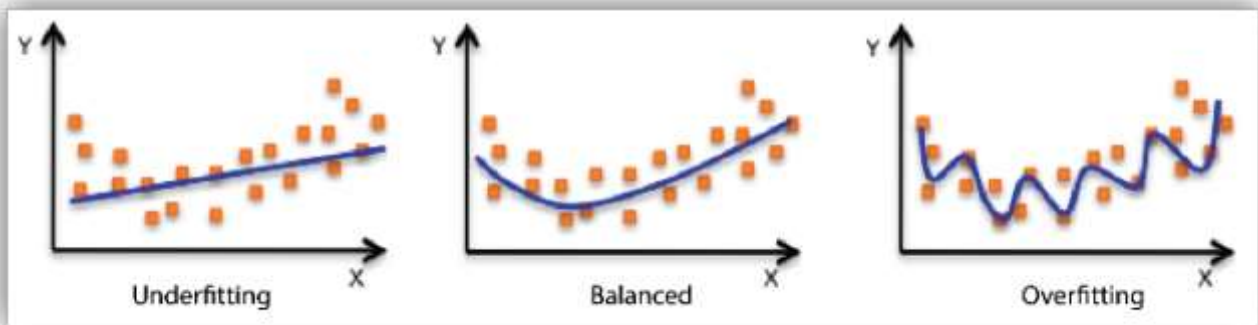
### ▼ Generalization

일반화 : 처음보는 데이터에 대해서도 예측 오차가 적은 것



## ▼ Under-fitting vs. over-fitting

문제에 따라 아래 그림처럼 over-fitting한 근사함수가 목표일 수도 있다.

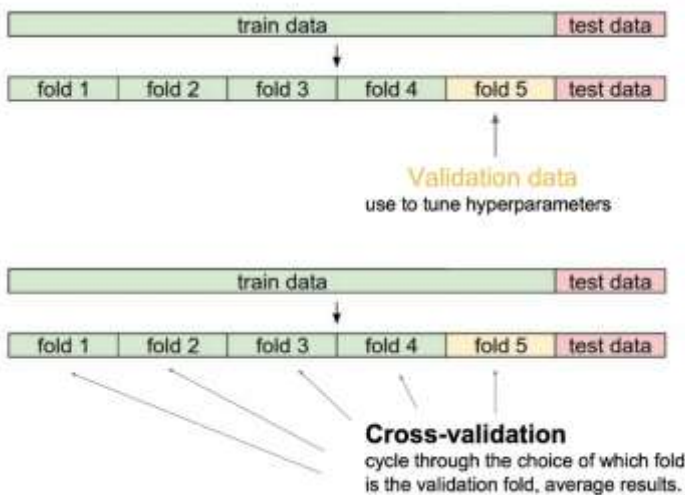


## ▼ Cross validation

파라미터 : mlp의 w와 b, conv 필터

하이퍼 파라미터 : 학습률, 네트워크 크기, 손실함수

- 최적의 하이퍼 파라미터 집합을 찾기 위해 cross validation한다.
- 최적 하이퍼파라미터 집합을 찾으면 모든 train data를 학습하는데 사용한다.
  - 더 많은 데이터로 모델 학습시키기 위한 것
  - 학습에는 train data, validation data만 사용한다.



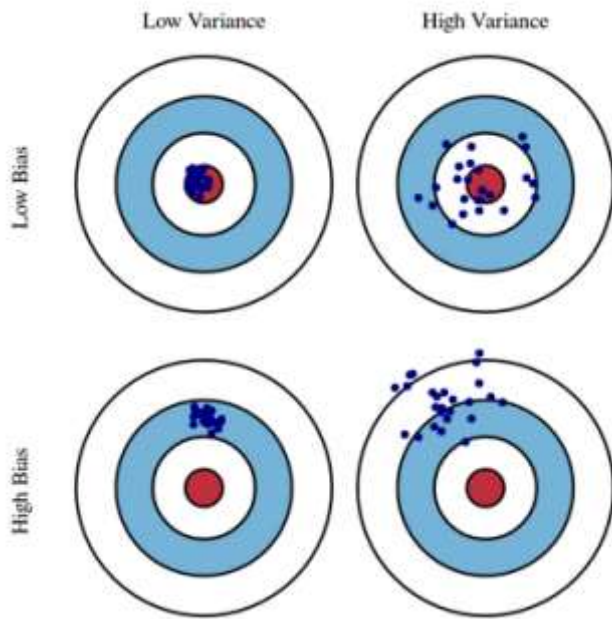
## ▼ Bias-variance tradeoff

Low variance : 비슷한 입력에 관해 출력이 비슷하다.

High variance : 비슷한 입력에 관해 출력이 많이 다르다.

Low bias : 평균적으로 정답에 가깝다.

High bias : 평균적으로 정답과 멀다.



학습 데이터에 노이즈가 있으면 bias와 variance를 둘 다 줄이기 어렵다.

$$\begin{aligned} \mathbb{E} \left[ (t - \hat{f})^2 \right] &= \mathbb{E} \left[ (t - f + f - \hat{f})^2 \right] \\ \text{cost} &= \dots \\ &= \mathbb{E} \left[ (f - \mathbb{E}[\hat{f}])^2 \right] + \mathbb{E} \left[ (\mathbb{E}[\hat{f}] - \hat{f})^2 \right] + \mathbb{E} [\epsilon] \\ &\quad \text{bias}^2 \quad \quad \quad \text{variance} \quad \quad \text{noise} \end{aligned}$$

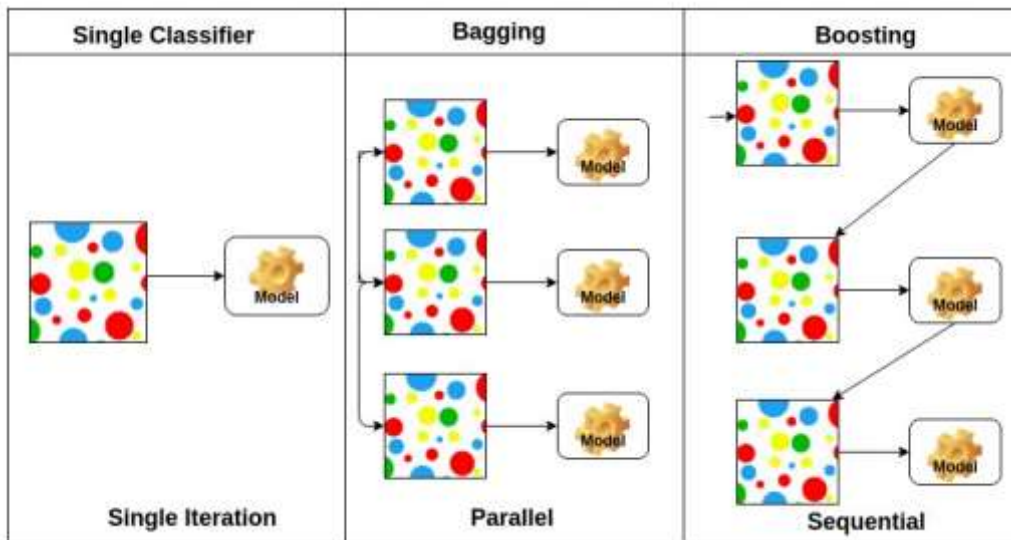
## Bootstrapping

부트스트래핑 : 학습 데이터를 비복원 임의 추출해 모델 만든다.

### ▼ Bagging and boosting

배깅 : 학습 데이터를 비복원 임의 추출해 모델 만들고 이를 반복하고 여러 모델의 평균으로 출력을 냄.

부스팅 : 오답 데이터에 잘 동작하는 모델로 만들고 다음으로 오답 데이터를 잘 맞추는 모델 만들고를 여러 모델(weak learner)을 합침 => weak learner를 합쳐서 하나의 strong learner를 만든다.



## ▼ Gradient Descent Methods

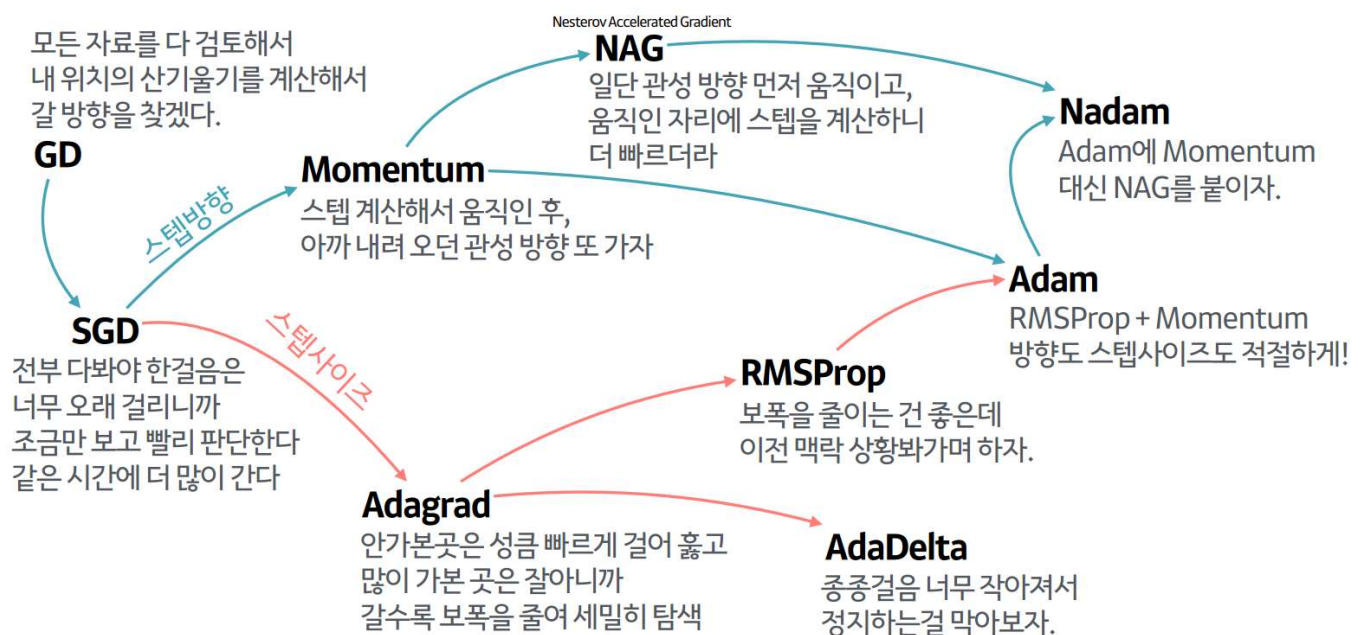
Stochastic GD : 학습 데이터에서 샘플 하나씩 gradient 계산해서 업데이트

Mini-batch GD : 학습 데이터에서 샘플 일부(64, 128, 256개)의 gradient 계산해서 업데이트, 대부분 딥러닝에서 사용

Batch GD : 학습 데이터에서 샘플 전체의 gradient 평균 계산해서 업데이트

참고 : <http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

## 산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



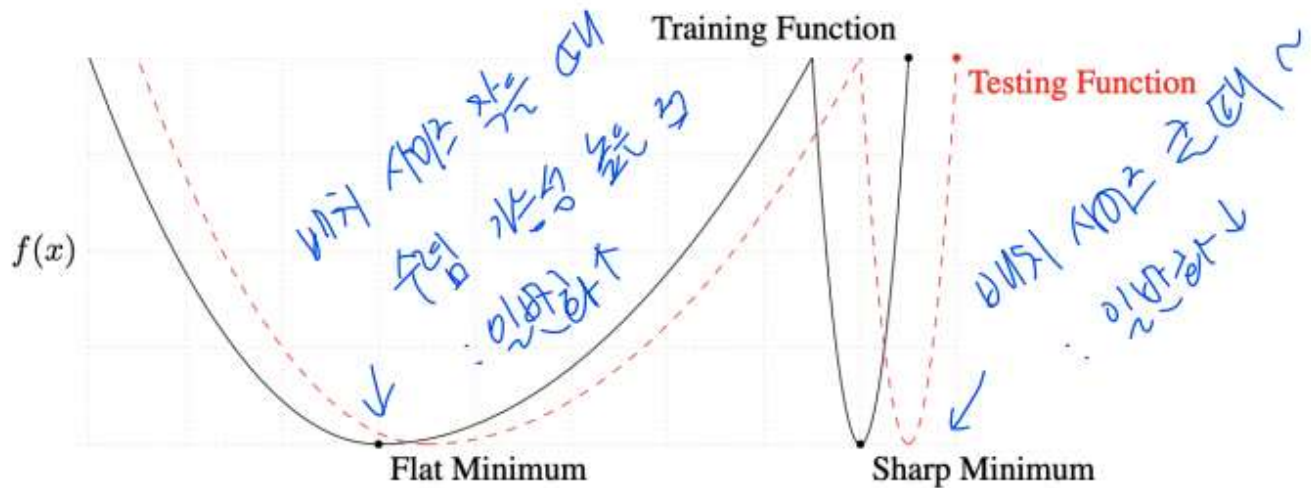


2차 출처 : [https://yngie-c.github.io/deep%20learning/2020/03/19/training\\_techs/](https://yngie-c.github.io/deep%20learning/2020/03/19/training_techs/)

## ▼ Batch-size Matters

배치 사이즈 적을 때 일반화 정도가 더 높다.

논문 : On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima, 2017



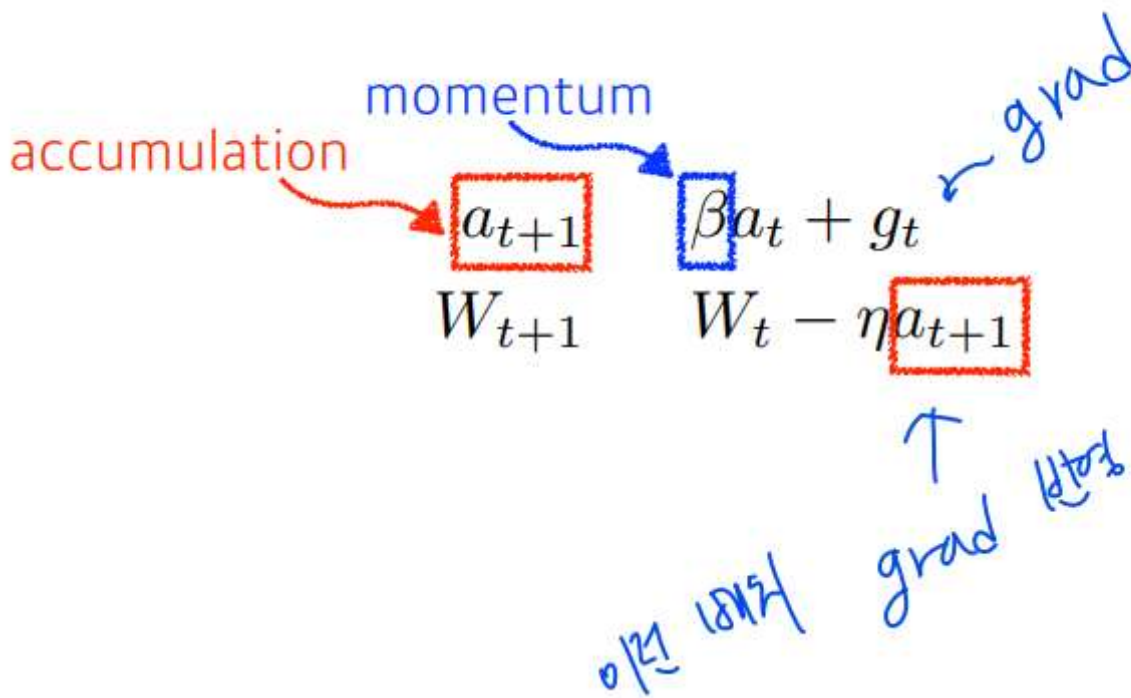
## ▼ Momentum

$$W_{t+1} = W_t - \eta g_t$$

Learning rate = eta

Gradient



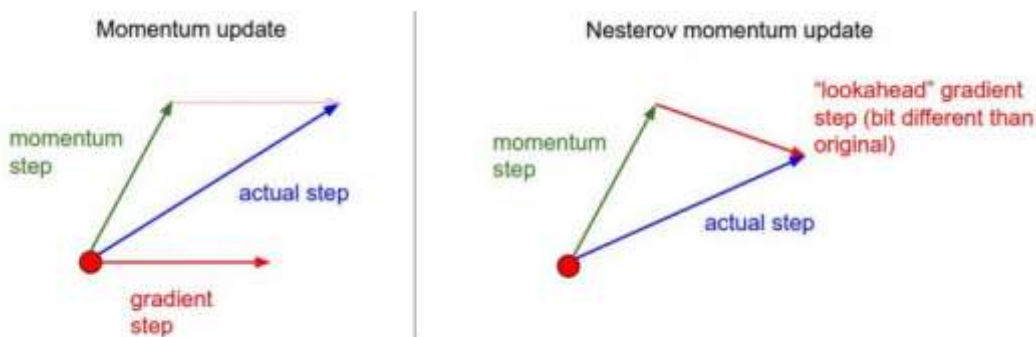


### ▼ Nesterov Accelerated Gradient

$$\begin{aligned}
 a_{t+1} &= \beta a_t + \text{Lookahead gradient} \\
 W_{t+1} &= W_t - \eta a_{t+1}
 \end{aligned}$$

The "Lookahead gradient" is represented by the term  $\nabla \mathcal{L}(W_t - \eta \beta a_t)$  in the equation above.

이동한 다음에 grad 계산해 local minima 수렴이 빠름



### ▼ Adagrad

많이 변한 파라미터 적게 변화, 적게 변한 파라미터 많이 변화시킨다.

$$G_t$$

- 파라미터의 변화 정도를 저장한 값
- G가 부한대로 가면 w가 업데이트가 안 되고 학습이 멈춘다.
  - 이후 등장하는 방법이 이를 해결하기 위해 고안된 것이다.

0으로 나누려는 것 방지

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

for numerical stability

Sum of gradient squares  
↙ 파라미터 변화 정도 저장

## ▼ Adadelata

시간(윈도우 사이즈)에 대한 grad 제공의 변화를 살펴본다.

바꿀 수 있는 하이퍼 파라미터가 적어 사용빈도가 적다.

Exponential moving average == EMA

Ht : lr 없어도 Wt의 변화값을 들고 업데이트

EMA of gradient squares

$$G_t = \gamma G_{t-1} + (1 - \gamma) g_t^2$$

$$W_{t+1} = W_t - \frac{\sqrt{H_{t-1} + \epsilon}}{\sqrt{G_t + \epsilon}} g_t$$

EMA of difference squares

$$H_t = \gamma H_{t-1} + (1 - \gamma) (\Delta W_t)^2$$

There is **no learning rate** in Adadelata.

## ▼ RMSprop

EMA of gradient squares

$$G_t = \gamma G_{t-1} + (1 - \gamma) g_t^2$$

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

stepsize

## ▼ Adam(Adaptive Moment Estimation)

Momentum와 adaptive learning rate 기법을 혼합

The diagram illustrates the Adam optimization algorithm with the following components and handwritten notes:

- Momentum:** Points to the equation  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ . The variable  $m_t$  is boxed in red.
- EMA of gradient squares:** Points to the equation  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ . The variable  $v_t$  is boxed in blue.
- Stepsize:** Points to the learning rate  $\eta$  in the update equation, which is boxed in yellow.
- Update Equation:** 
$$W_{t+1} = W_t - \frac{\eta}{\sqrt{v_t} + \epsilon} \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} m_t$$
  - $\eta$  is boxed in yellow.
  - $\sqrt{v_t} + \epsilon$  is boxed in blue.
  - $\frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t}$  is boxed in pink.
  - $m_t$  is boxed in red.

**Handwritten Notes:**

- 노란색 : 파라미터들 (Yellow: parameters) - points to  $m_t$ .
- 6 조정 가능 항목 (6 adjustable items) - points to  $\eta$  and  $\epsilon$ .
- 분위 조정 위해 사용 (Used for adjusting the atmosphere) - points to the pink box.

실습

Adam이 잘 되는 결과를 보면 adaptive learning rate과 momentum 둘 다 쓰는 게 좋다.

- 특정 파라미터의 학습률을 감소, 증가시켜서 더 빠르게 학습이 된다.

SGD만 하면 iteration이 많아야 전체 데이터가 수렴한다.

- iteration이 엄청 많아지면 SGD가 Adam보다 좋아질 수 있으나 일단 시간 절약을 위해 Adam을 사용하자.

momentum은 이전 grad를 다음 w 업데이트에도 사용

- 미니 배치에서 momentum이 좋은 이유는 이전 grad를 사용하는 것으로 인해 데이터를 많이 보는 효과가 생기기 때문이다.

MSE를 사용하면 많이 틀리는 곳을 오차가 커져서 많이 틀리는 곳을 잘 맞추고 상대적으로 적게 틀리는 곳을 덜 집중하기 때문이다.

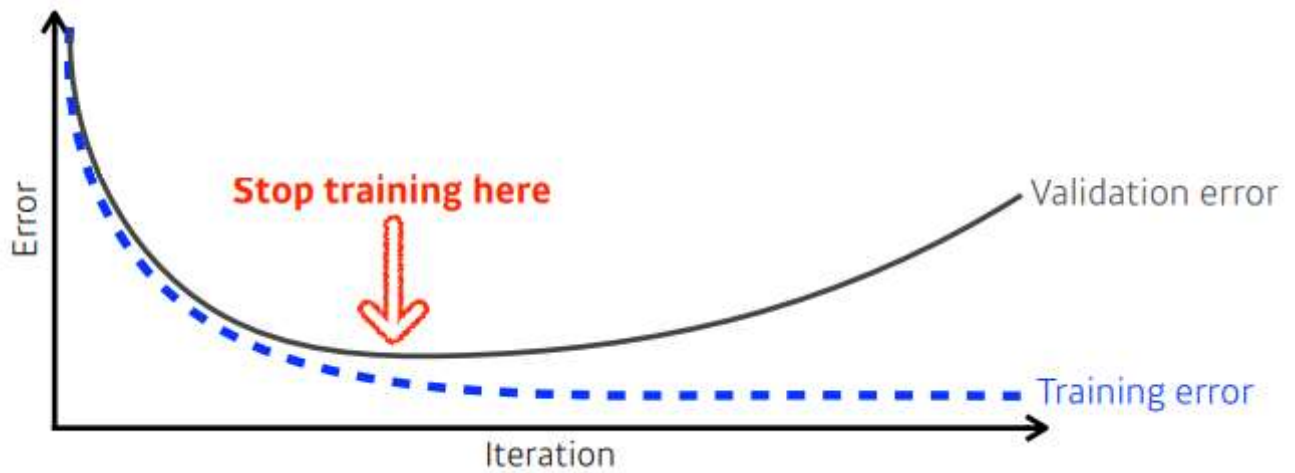
- 따라서 MSE는 이상치에 민감하게 반응한다.

## ▼ Regularization

테스트 데이터에도 모델이 잘 동작하도록 하는 방법

## ▼ Earlystopping

추가적인 검증 데이터 집합 필요



#### ▼ Parameter norm penalty

= weight decay

Parameter Norm Penalty

$$\text{total cost} = \text{loss}(\mathcal{D}; W) + \frac{\alpha}{2} \|W\|_2^2$$

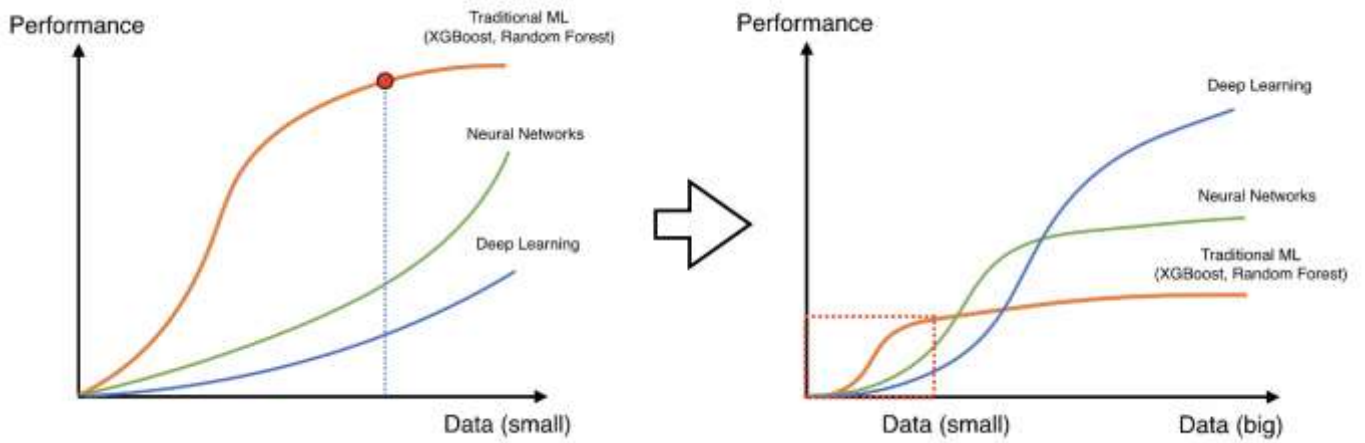
가정 : 부피의 모양의 항이 일반화 높다.

#### ▼ Data augmentation

데이터 셋 적으면 DL보다 ML이 더 성능이 좋다.

이미지와 그 라벨이 변하지 않는 한도 내에서 데이터를 증식시켜서 DL 적용

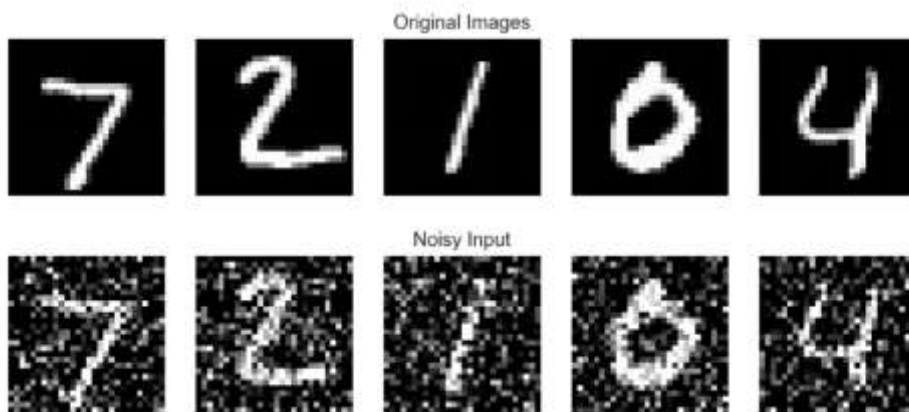
- MNIST는 뒤집는 변환하면 안 돼



## ▼ Noise robustness

입력 데이터와 가중치에 임의의 노이즈 추가

왜 잘되는지 아직 설명이 안됨



## ▼ Label smoothing





Mixup : 학습 데이터의 샘플을 합성해 데이터 증식해 결정 경계 부드럽게 만들어준다.

Cutout : 샘플 중 일부 제거

CutMix : Cutout하고 Mixup함

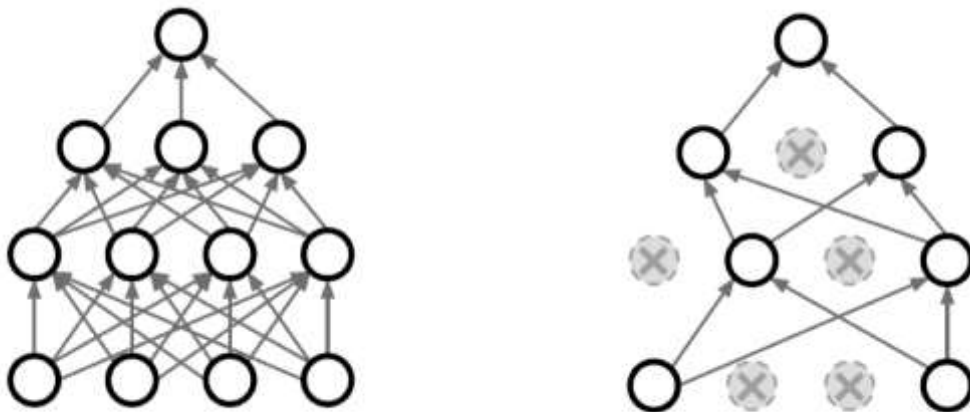
왜 잘되는지 아직 설명이 안됨. 그러나 성능이 좋다.

한정된 데이터 셋에서 분류 문제를 풀 때 사용하자.

	ResNet-50	Mixup [48]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4

## ▼ Dropout

뉴럴 네트워크의 일부 weight를 0으로 만든다. 각각의 뉴런들이 robust해진다. 증명되지는 않음  
 예 : dropout rate == 0.5면 뉴럴 네트워크에서 추론할 때 전체 뉴런 중 50% 뉴런의 가중치를 0으로 변경



## ▼ Batch normalization

레이어의 통계치를 정규화 시킨다. 예 : 천 개의 파라미터의 각 값의 통계치가 평균은 0이고 분산은 1이 되게 만든다.

논란 많은 논문에서 유래 : Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015

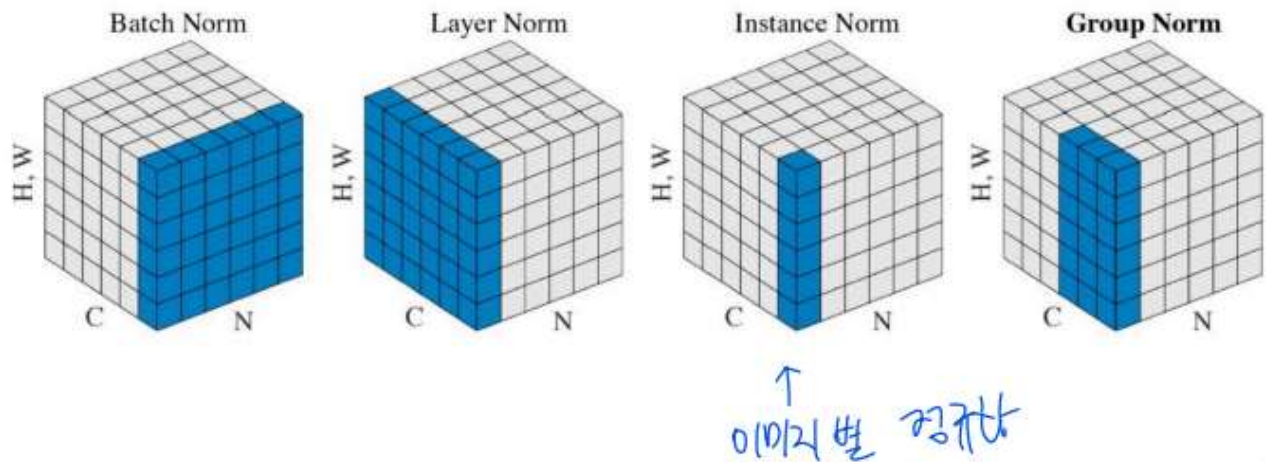
- Internal Covariate : 내부 특성
- 뒤에 나온 논문들은 동의하지 않으나 분류 문제를 풀기 위해 레이어를 깊게 쌓은 경우 배치 정규화를 사용하면 성능이 높아진다.



$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$



## ▼ (04강) Convolution은 무엇인가?

continuous convolution

$$(f * g)(t) = \int f(\tau)g(t - \tau)d\tau = \int f(t - \tau)g(\tau)d\tau$$

discrete convolution

$$(f * g)(t) = \sum_{i=-\infty}^{\infty} f(i)g(t - i) = \sum_{i=-\infty}^{\infty} f(t - i)g(i)$$

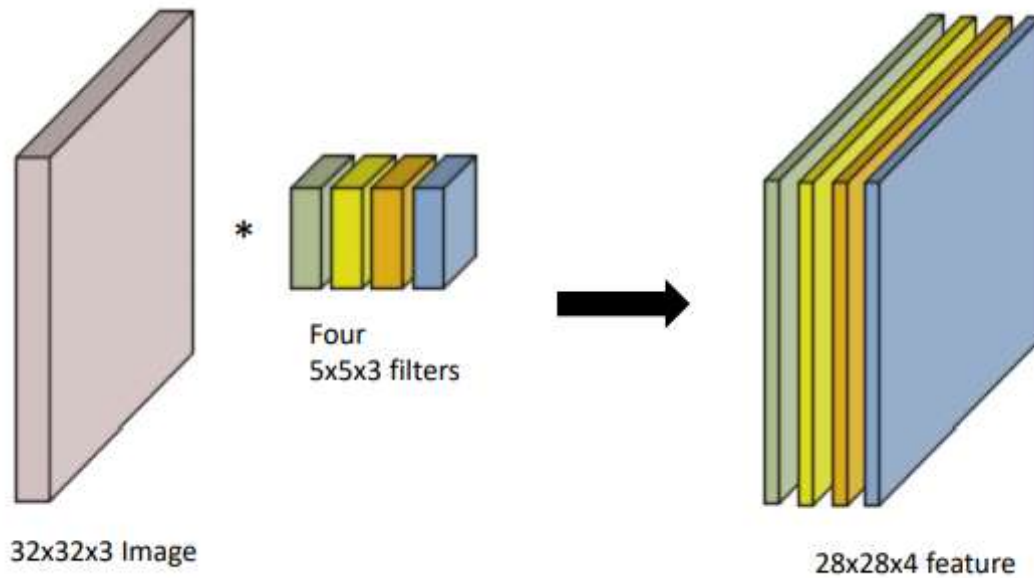
2D 이미지 convolution



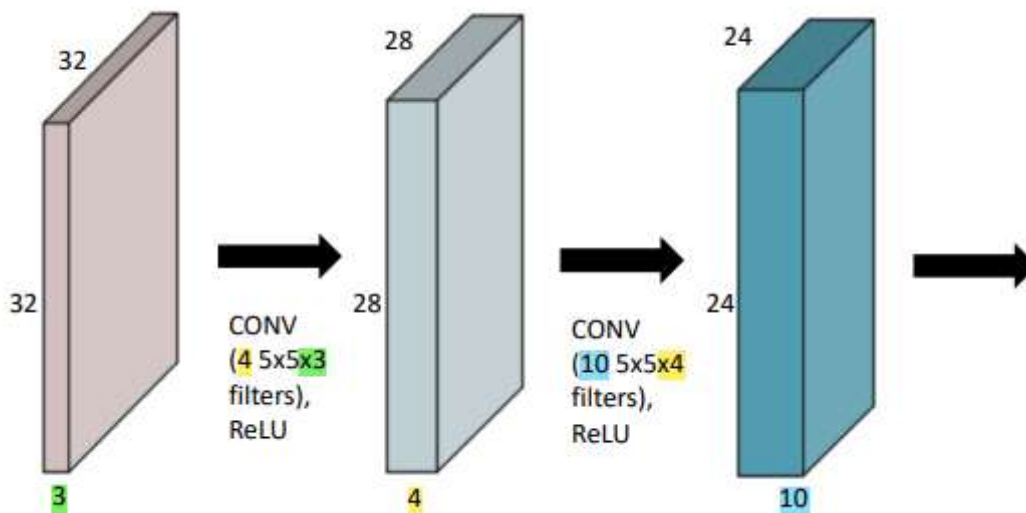
$$(I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n) = \sum_m \sum_n I(i - m, i - n) K(m, n)$$

Handwritten notes in blue ink: "7221", "014", "conv", "7221" with arrows pointing to the indices in the equation.

## RGB Image convolution



## stack of convolution



CNN 기본 구성 : convolution layer, pooling layer, fully connected layer

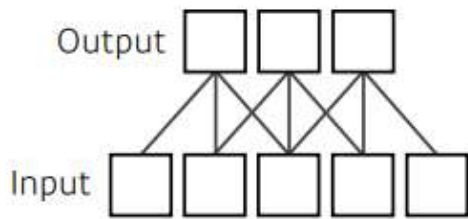
- convolution and pooling layers : feature extraction
- fully connected layer : decision making (e.g., classification)

- 최근에는 fully connected layer는 최소화되고 있음
  - 실험적으로 파라미터 숫자가 늘어날 수록 일반화 성능이 떨어진다. 따라서 conv layer를 많이 쌓고 파라미터 숫자를 줄이기 위해 노력한다.

stride : 커널의 픽셀 보폭

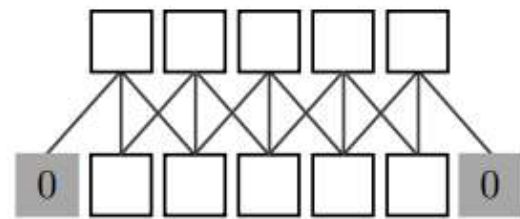
padding : 가장자리 정보를 conv 필터로 추출하기 위해 사용

stride와 padding 파라미터 개수와는 무관하다.



No padding (stride=1)

$$\frac{5-3}{1} + 1 = 3$$



Zero padding (stride=1)

$$\frac{5 + 2 \times 1 - 3}{1} + 1 = 5$$

## 2. 레이어별 출력 데이터 산정

Convloution 레이어와 Pooling 레이어의 출력 데이터 크기를 계산하는 방법을 소개합니다.

### 2.1 Convolution 레이어 출력 데이터 크기 산정

입력 데이터에 대한 필터의 크기와 Stride 크기에 따라서 Feature Map 크기가 결정됩니다. 공식은 다음과 같습니다.

- 입력 데이터 높이:  $H$
- 입력 데이터 폭:  $W$
- 필터 높이:  $FH$
- 필터 폭:  $FW$
- Strid 크기:  $S$
- 패딩 사이즈:  $P$

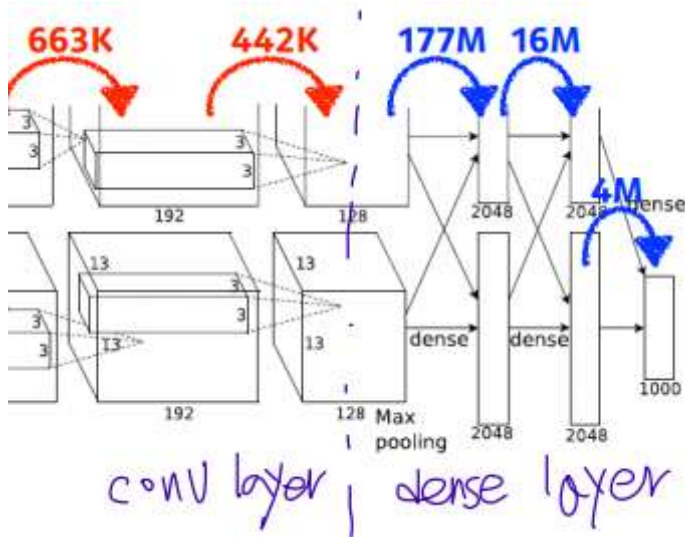
- 식1. 출력 데이터 크기 계산

$$OutputHeight = OH = \frac{(H + 2P - FH)}{S} + 1$$
$$OutputWeight = OW = \frac{(W + 2P - FW)}{S} + 1$$

레이어 출력 크기 구하는 법 출처 : <http://taewan.kim/post/cnn/>

AlexNet의 네트워크 경로 2개로 나눠짐, 당시 사용하는 GPU 메모리가 적어서 한번에 네트워크에 입력되지 않기 때문

Convolution layer에 비해 Dense layer는 파라미터 숫자가 많이 나와서 요새 트렌드는 conv layer를 깊게 쌓고 dense layer를 적게 사용한다.



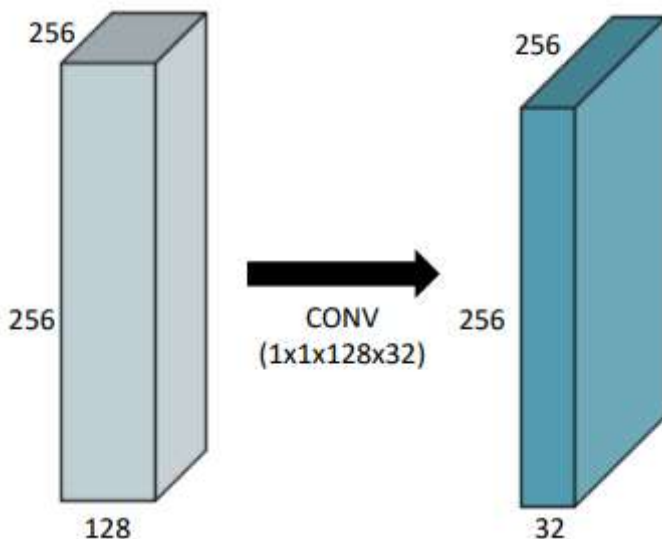
## ▼ 1x1 Convolution

정의 : 이미지에서 한 픽셀만 보고 채널을 줄이는 것

사용 이유

- 채널(차원) 축소
- Conv layer를 깊게 쌓으면서 파라미터 개수를 줄일 수 있다. => 병목 구조

여러 네트워크 구조에서 자주 사용된다.



## ▼ (05강) Modern CNN - 1x1 convolution의 중요성

Classification : 종의 분류

Detection : Bounding Box

Localization : 하나의 물체 찾기

Segmentation : 픽셀별 구분

## AlexNet

ReLU : 기울기 소실 극복 <-> 시그모이드는  $x > 0$ 일 때 기울기가 줄어들어 기울기가 소실되는 단점을 가지고 있다.

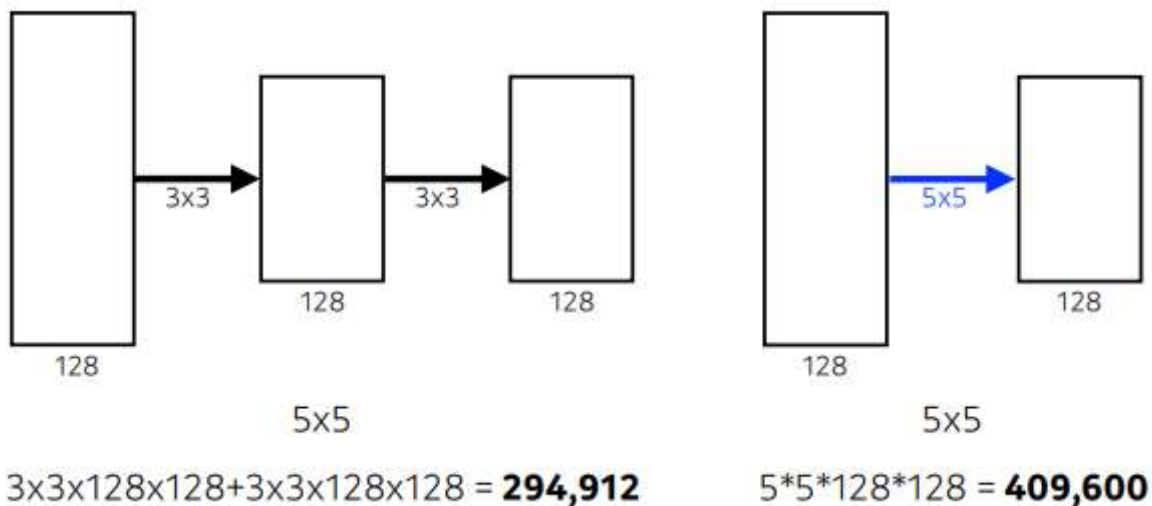
- 활성화함수는 Non-linear해야 한다.

데이터 증식

드롭 아웃

## ▼ VGGNet

3x3 Conv 필터를 사용한 것이 주목할 점

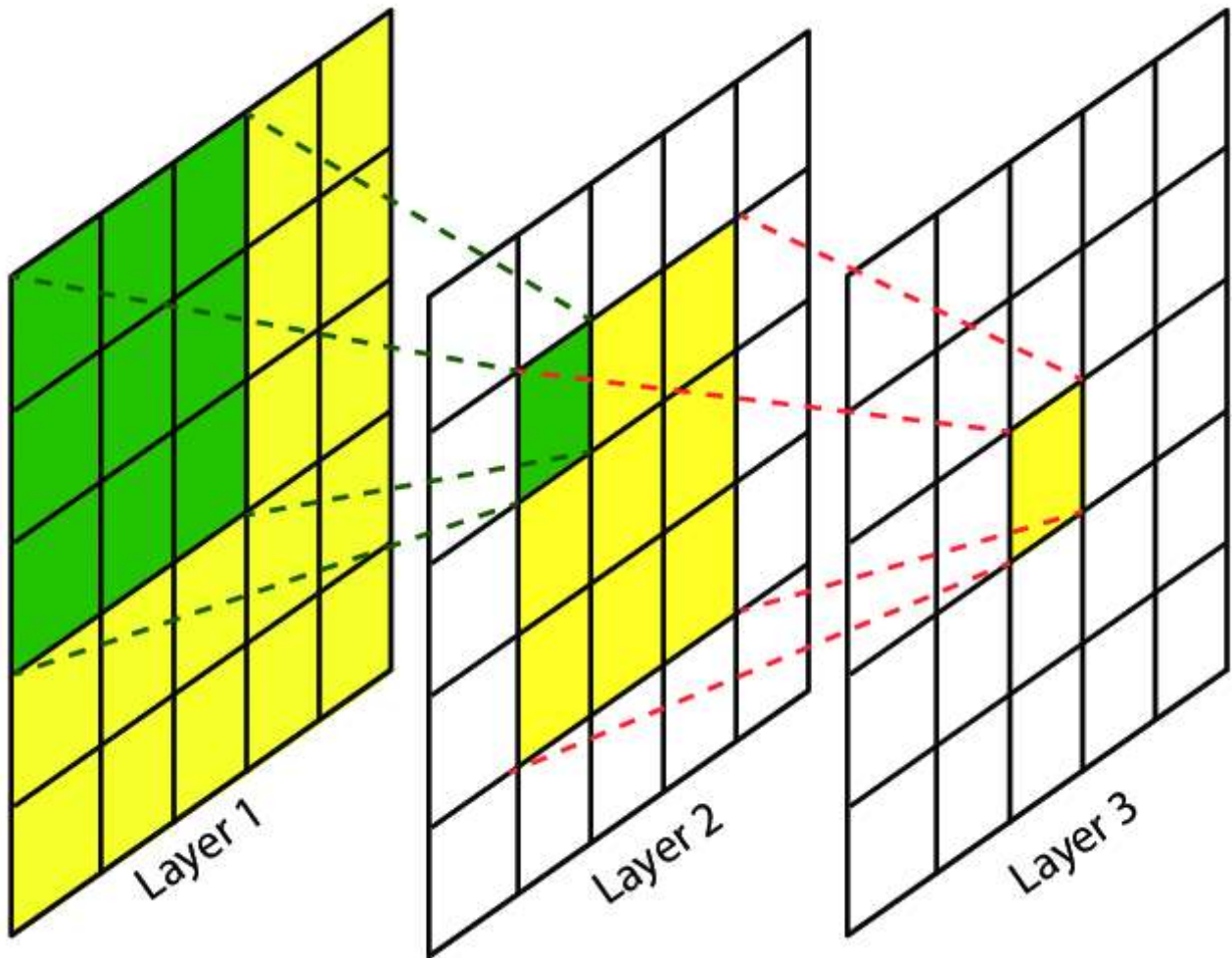


필터  $w \times h \times c$  개수

- $3 \times 3 \times 2$ (개수) == 18
- $5 \times 5 \times 1$ (개수) == 25

3 x 3 필터를 2개 사용하는 게 더 파라미터 개수 줄일 수 있음

그래서 최신 논문들의 필터 최대 크기는 7x7 or 5x5 라고 한다.



## ▼ GoogLeNet

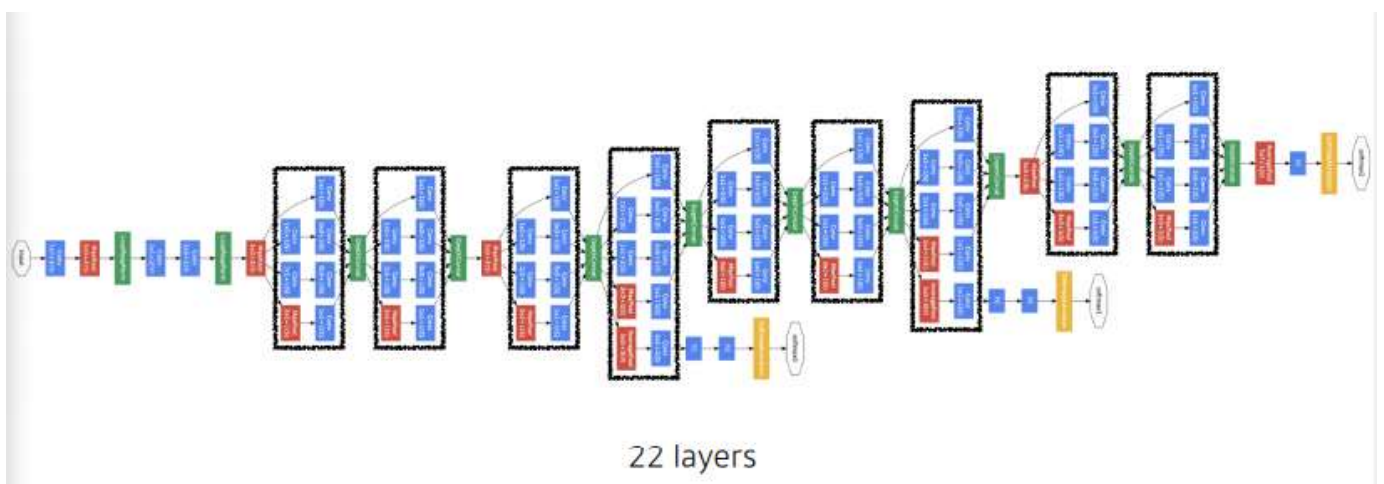
1x1 필터를 사용해서 파라미터를 많이 줄일 수 있다.

NiN 구조와 inception blocks 결합

이전에 비해 Dense layer를 적게 사용

network-in-network(NiN)

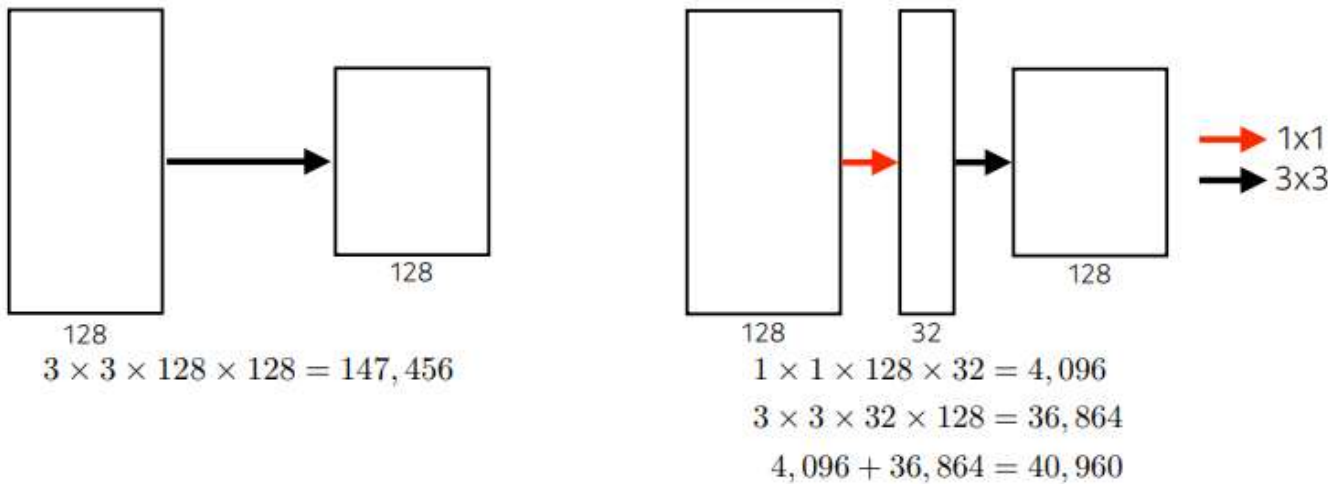
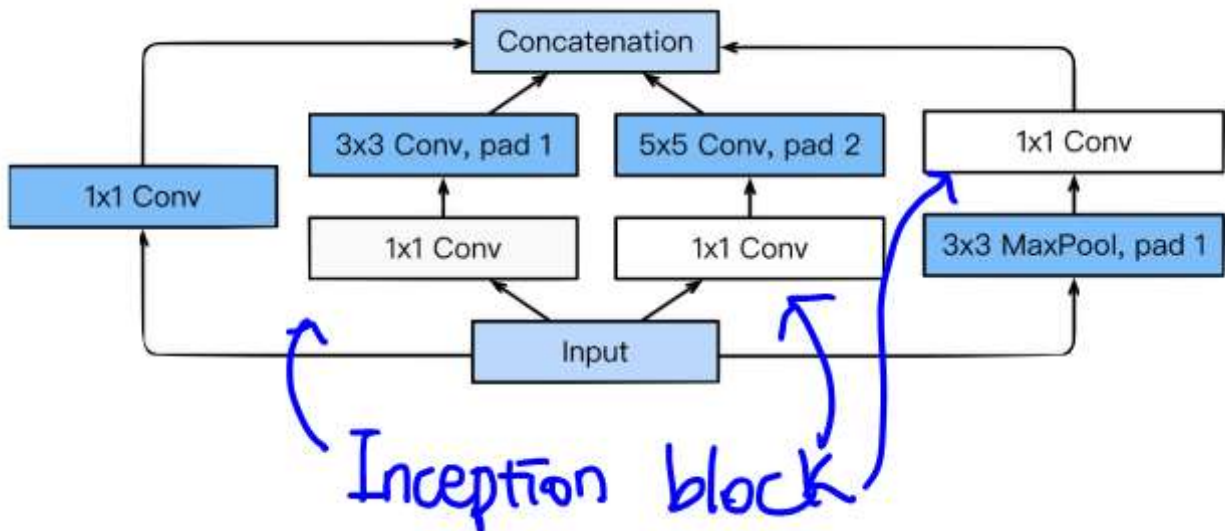
비슷한 레이어가 여러번 반복 => 네트워크 안에 네트워크



## ▼ Inception blocks

1x1 convolution을 통해 파라미터 개수를 30% 정도 줄일 수 있었다.

parameter가 많으면 오히려 연산이 느리고 overfitting이 일어나서 1x1 convolution을 사용해서 param 수를 줄일 수 있다.



## ResNet

보통 파라미터 많으면 오버피팅이 발생하거나 학습이 잘 안됨

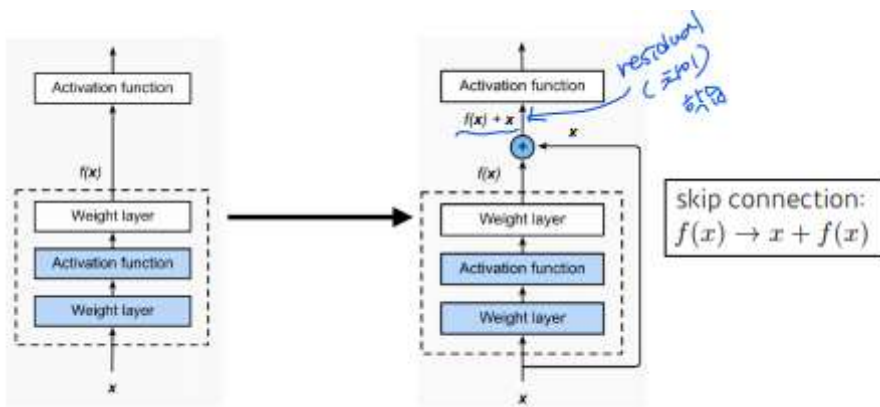
- 오버피팅 : 학습 오류 감소하나 테스트 오류 증가 == 일반화 악화

ResNet은 identity map을 사용

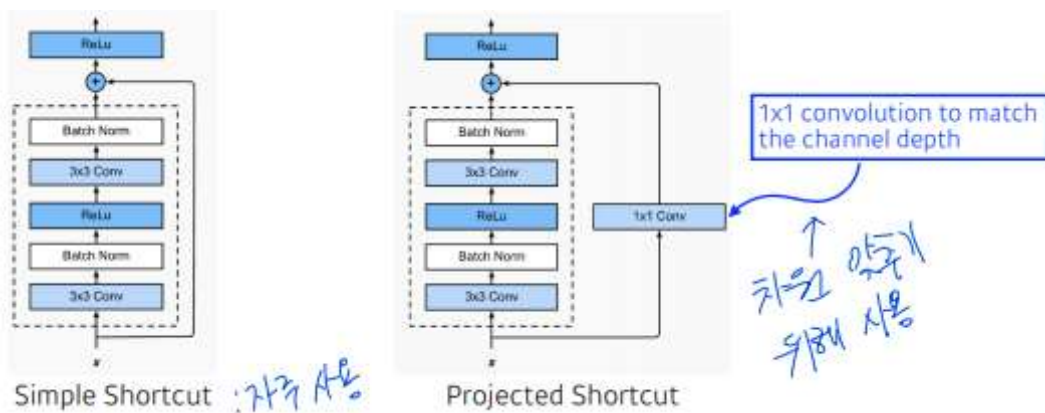
- 네트워크 깊게 쌓을 수 있는 토대

## ▼ Identity map(Skip connection)

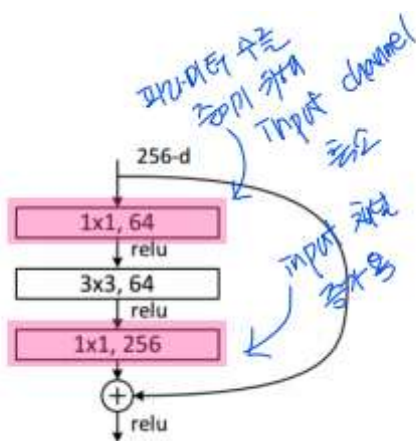




▶ 비선형 활성화함수 사용 전에 Identity map 및 배치정규화 사용



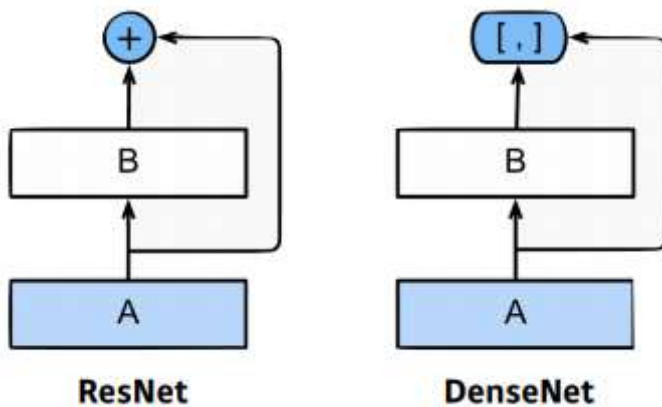
▶ Bottleneck architecture



Depth(receptive field)를 늘려서 파라미터 수를 줄일 수 있고 성능을 높이는 것이 목표

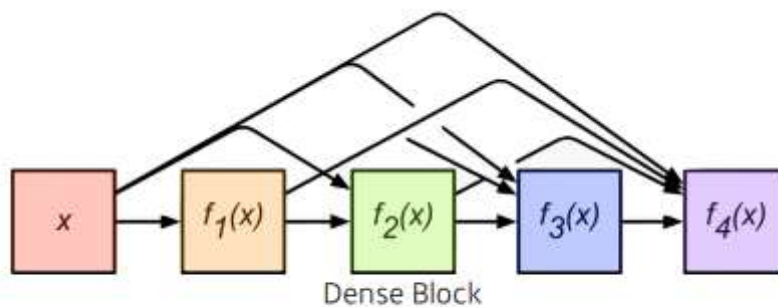
▶ DenseNet

Conv.한 결과를 더하지 말고 결합



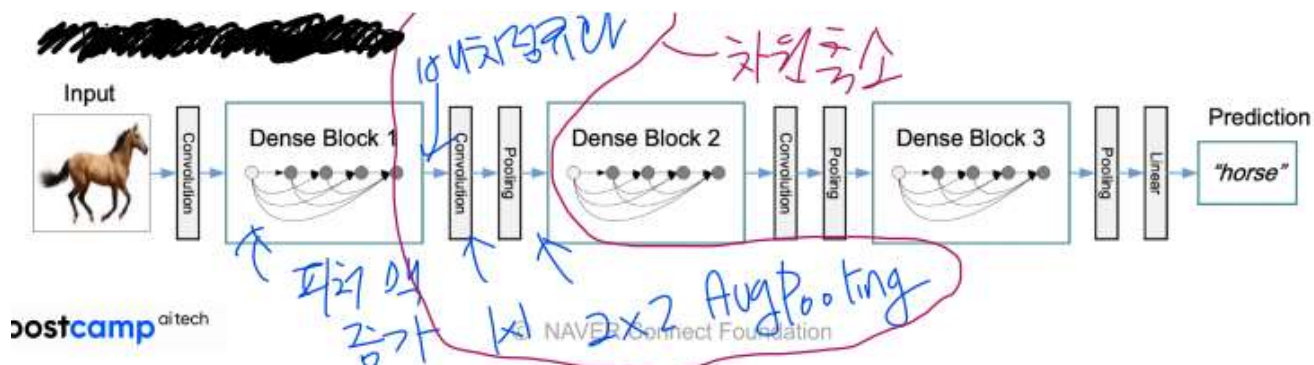
단, concatenate하면 점점 채널이 커진다.

- 1배, 2배, 3배 ... 채널이 기하급수적으로 커진다.



$$x \mapsto [x, f_1(x), f_2(x, f_1(x)), f_3(x, f_1(x), f_2(x, f_1(x))), f_4(x, f_1(x), f_2(x, f_1(x)), f_3(x, f_1(x), f_2(x, f_1(x))))]$$

그래서 중간에 한 번씩 채널을 축소 ==> 1x1 convolution을 한다.



ResNet, DenseNet 구조 성능 잘 나와

요약

VGG : receptive field 늘리는데 반복되는 3x3 필터가 좋다.

GoogLeNet : 1x1 convolution으로 채널 수를 줄여 파라미터를 줄였다.

ResNet : skip-connection으로 feature map을 더해서 네트워크를 깊게 쌓았다.

DenseNet : concatenation으로 feature map을 더하는 대신에 쌓아서 더 좋은 성능을 낼 수 있었다.

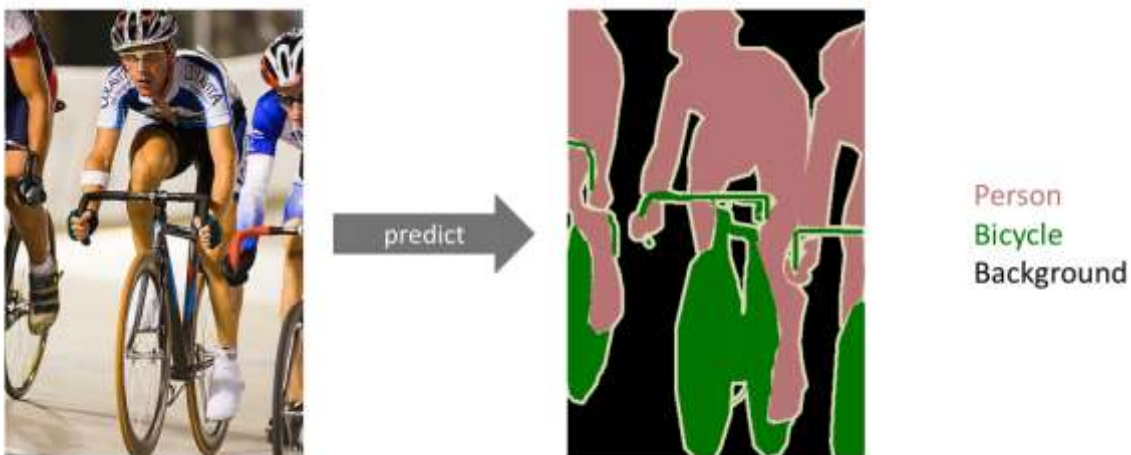
## ▼ (06강) Computer Vision Applications

### ▼ Semantic segmentation

이미지의 모든 픽셀이 어떤 라벨이 속하는 지를 분류한다.

- 응용 분야 : 자율주행(사람? 신호등? 차량? 인도?)

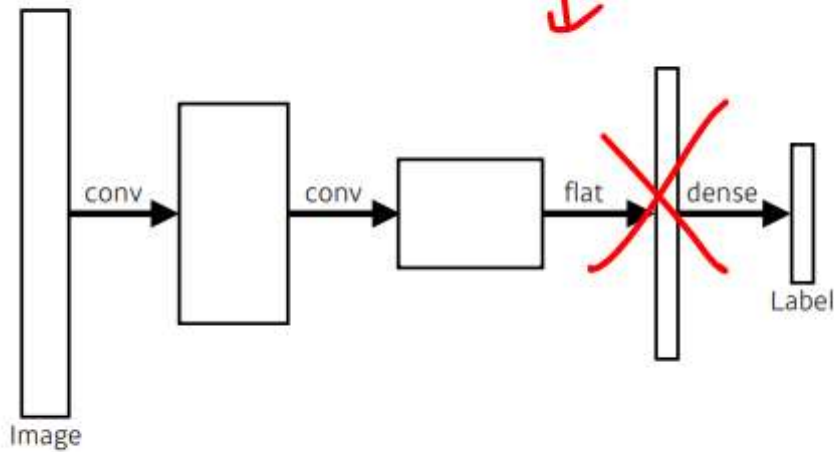
Semantic segmentation == Dense classification == Per pixel classification



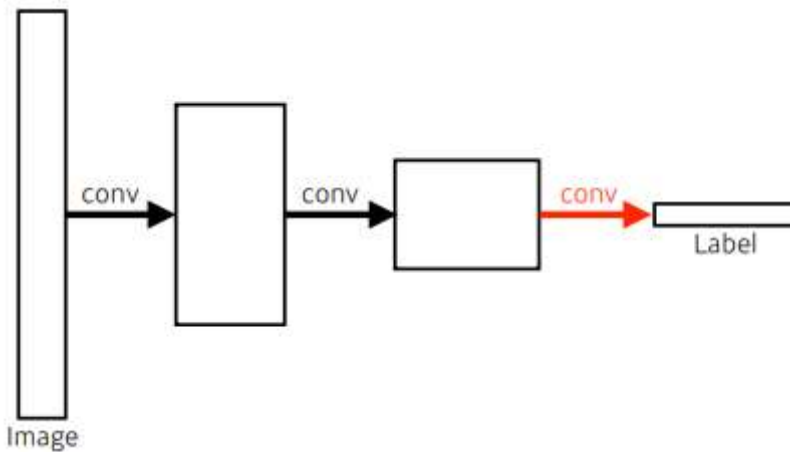
### ▼ Fully Convolutional Network

보통의 CNN 구조에서 Dense layer를 없애면 Fully Convolutional Network가 된다.

convolutionalization  
↓



보통의 CNN 구조



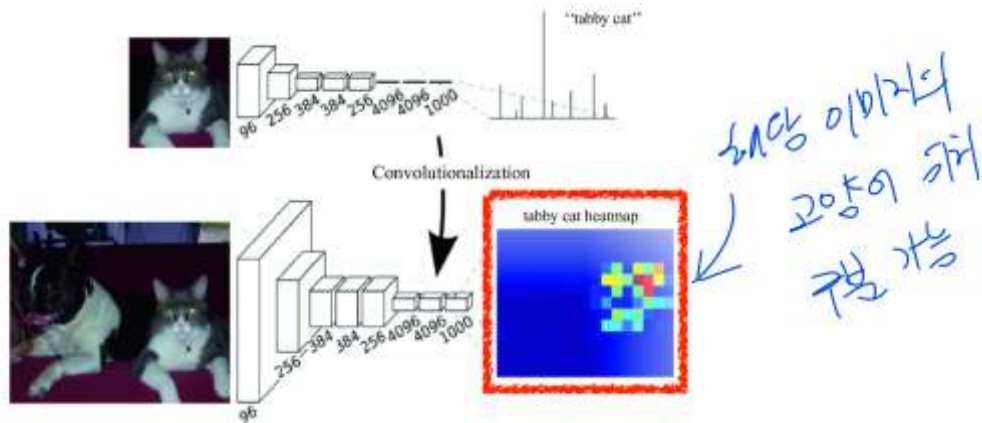
파라미터 개수 동일

$$4 \times 4 \times 16 \times 10 = 4 \times 4 \times 16 \times 10$$

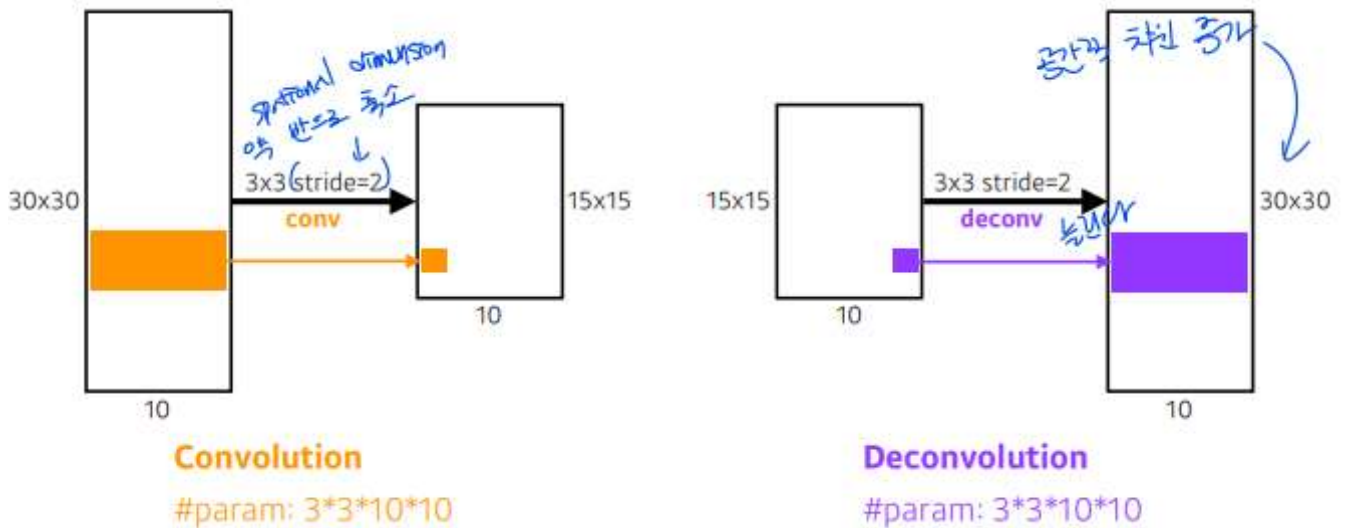


파라미터 개수 동일한 데 왜 Fully Convolutional network 사용하냐?

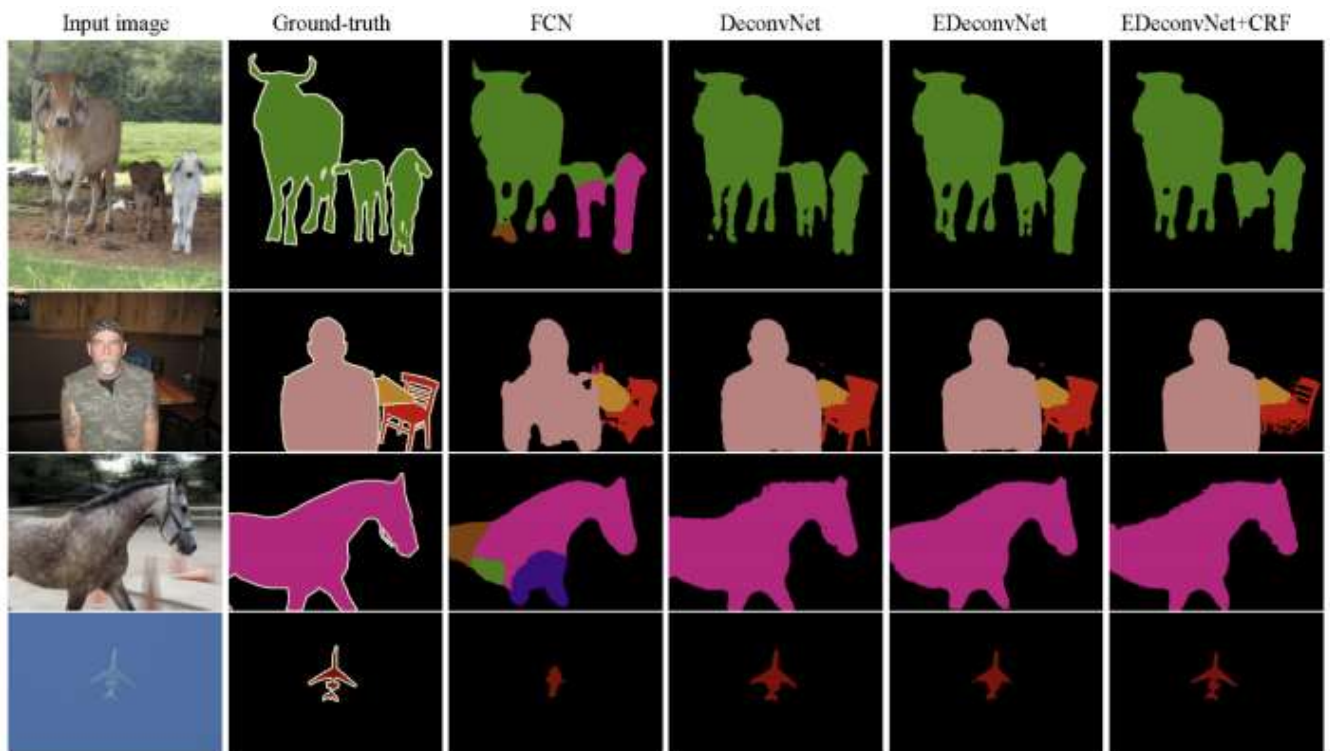
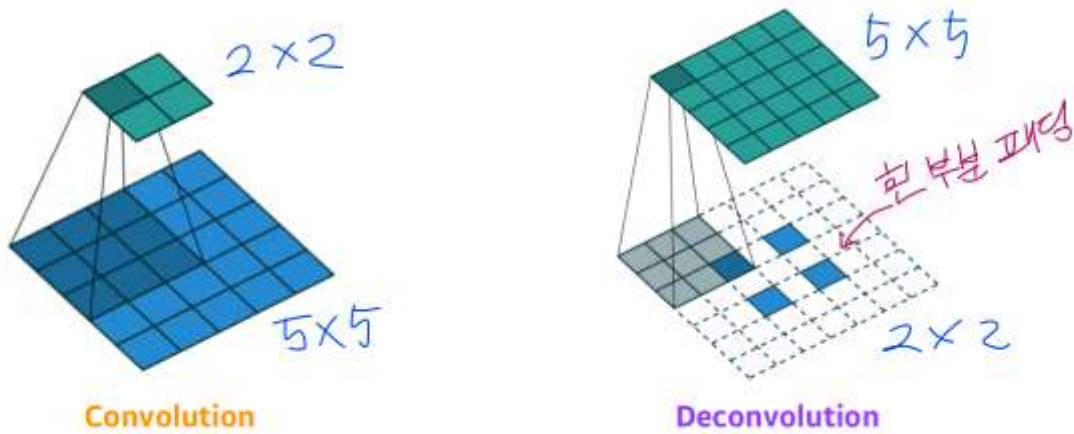
- 입력 이미지에 상관없이 작동한다. 따라서 입력 이미지 크기에 비례해서 뒷단의 네트워크 (spatial dimension)가 커진다.
  - 왜? convolution이 가지는 shared parameter 성질 때문이다. 입력 이미지 크기에 상관 없이 동일한 컨볼루션 필터가 동일하게 찍기 때문에 spatial dimension만 커지고 동작한다. 그 동작이 마치 히트 맵과 유사하다.



#### ▼ Deconvolution (conv. transpose)







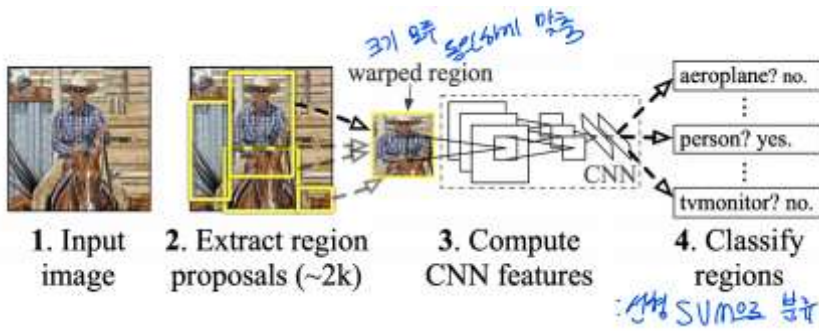
## ▼ Object detection

Per pixel이 아닌 Bounding box로 물체를 찾는다.

## ▼ R-CNN

아래 단계 + bounding box regression(더 좋은 box 위치 조절을 위해)으로 구성됨

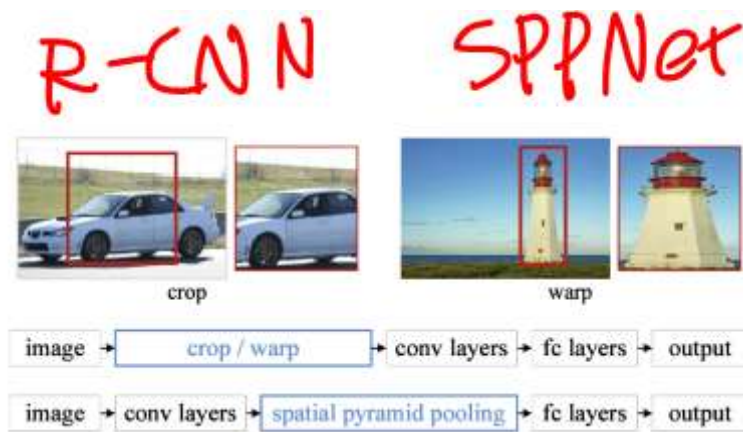
단점 : 추출한 바운딩 박스 2천개를 모두 CNN에 통과시켜야 한다. (== 2000번 CNN을 한다.) CPU에서 하나의 이미지 처리하는데 1분이 걸린다.



## ▼ SPPNet

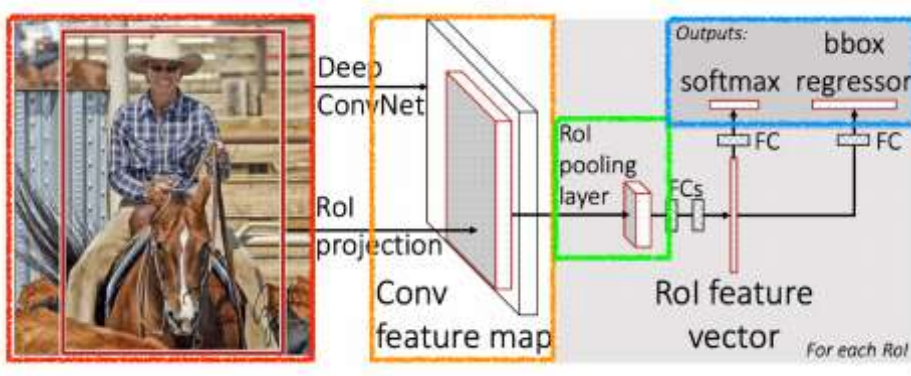
처리 시간을 줄이기 위해 이미지 안에서 CNN을 한 번만 돌리자!

이미지 안에서 바운딩 박스를 뽑고 이미지 전체에 대해서 컨볼루션 피쳐 맵을 만든 다음에 뽑힌 바운딩 박스 위치에 해당하는 컨볼루션 피쳐 맵의 텐서만 가져오자.



## ▼ Fast R-CNN

1. 이미지에서 바운딩 박스 추출
2. conv. feature map 생성
3. 각각의 region에 대해서 ROI Pooling으로 fixed length feature를 추출
4. 뉴럴 네트워크를 통해서 바운딩 박스 regression하고 바운딩 박스의 라벨 찾음





## ▼ Faster R-CNN

Faster R-CNN = Region Proposal Network(바운딩 박스를 뽑는 것도 네트워크로 학습) + Fast R-CNN

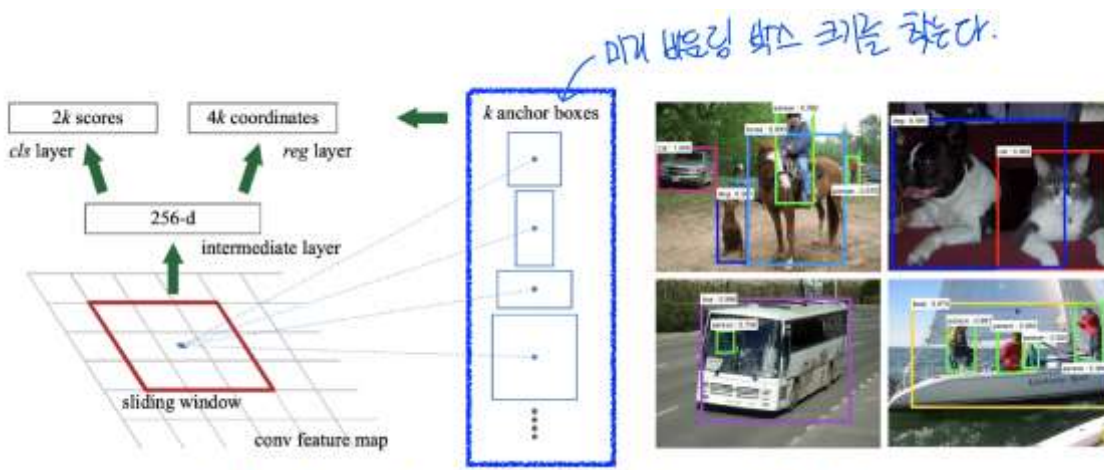
작은 물체도 잘 감지해냄



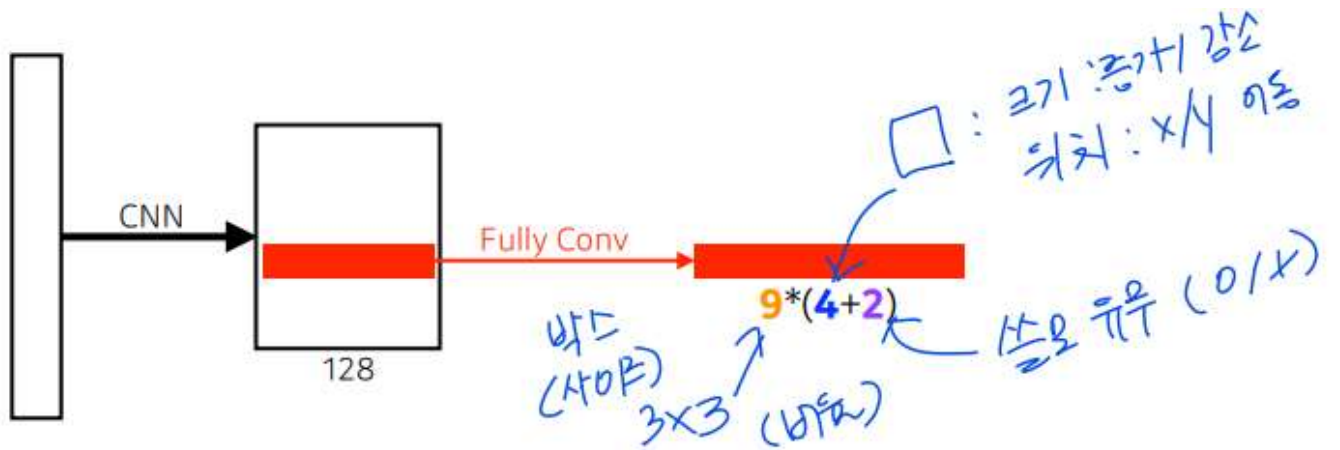
## ▼ RPN (Region Proposal Network)

이미지에서 특정 영역이 바운딩 박스로 의미가 있을 지 없을지 탐색

- 무엇인지는 구별하지 않아



## ▼ 파라미터 개수

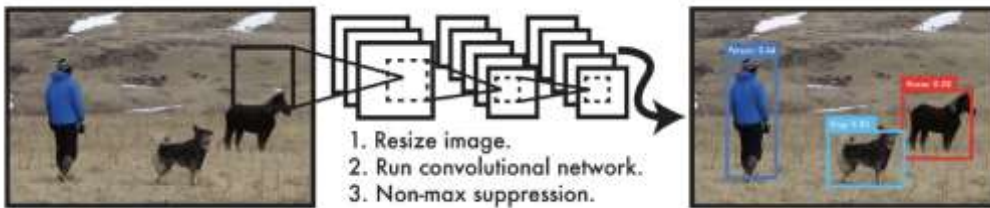


## ▼ YOLO v1

그냥 이미지에서 한번 찍어서 아웃풋이 나온다 == 동시에 많은 바운딩 박스를 만들고 클래스의 확률을 예측한다 == bbox 찾는 것과 클래스 찾는 것 동시에 실행 == You only look once

- 바운딩 박스를 따로 뽑는 과정(RPN)이 없다

이전 방법에 비해 속도가 굉장히 빠르다.





최근 Object detection 방법들은 네트워크가 땀으로 bbox를 예측하는 것보다 미리 바운딩 박스 크기 고려한 다음에 크기를 조절하는 방식으로 접근하고 있다.

## ▼ (07강) Sequential Models - RNN

### Sequential Model

#### Naive sequence model

Sequential data는 입력의 차원이 고정되지 않아서 CNN을 적용하기 힘들다.

- 예 : 발화가 어디서 끝날 지 모름.

Sequential Models은 이전 데이터로 다음 데이터를 예측한다.

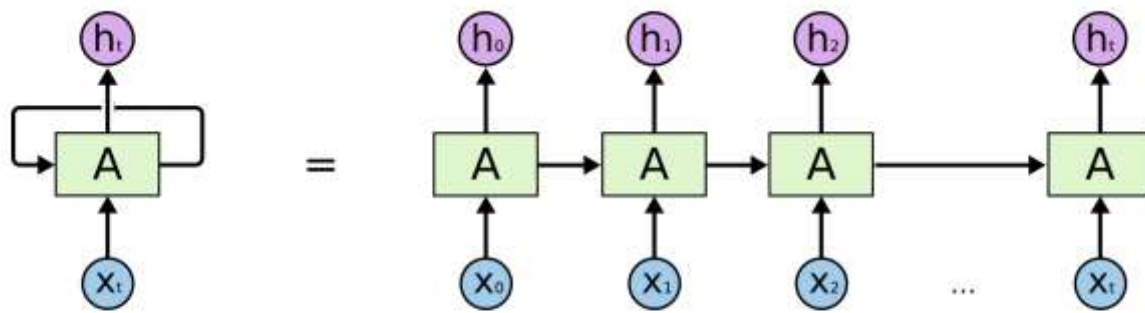
### ▶ PPT 필기

↳ 숨겨진 셀 7개

## ▼ Recurrent Neural Network

RNN의 구조

- 자기 자신으로 돌아오는 구조가 있음

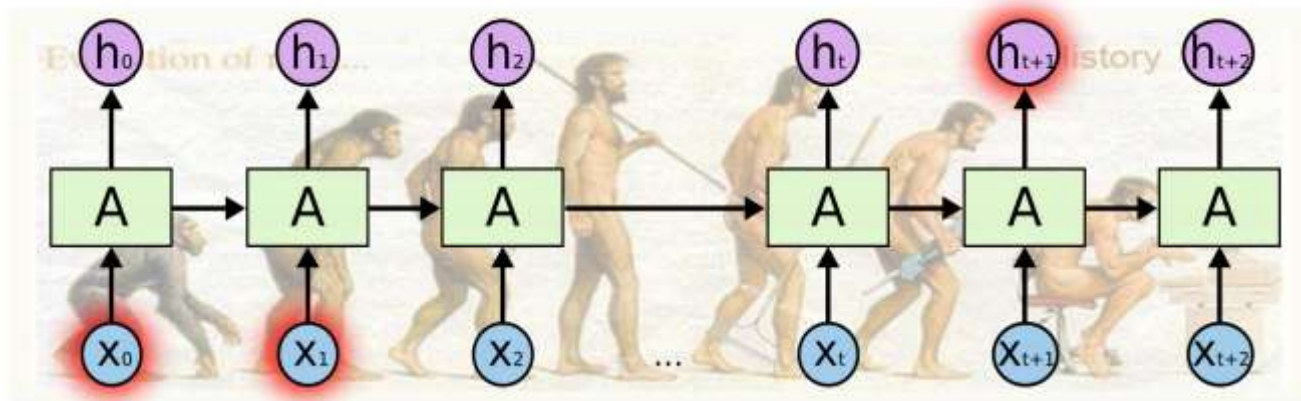


자기 자신으로  
돌아오는 구조

시간 도약 구조  
== Fully Connected

## RNN의 단점

- Short-term dependencies : 먼 과거의 정보가 미래까지 살아남기 힘들다.



## ▶ PPT 필기

↳ 숨겨진 셀 8개

## Gated Recurrent Unit

LSTM보다 네트워크 파라미터가 적는데 더 좋은 성능을 보인다 == 일반화 성능이 높다

## ▶ PPT 필기

↳ 숨겨진 셀 2개

## ▼ (08강) Sequential Models - Transformer

참고 : <https://nlpinkorean.github.io/illustrated-transformer/>

### Transformer

- 재귀 없고 attention 구조 활용
- 번역, 이미지 분류, 탐지에도 사용

#### ▶ PPT 필기

↳ 숨겨진 셀 21개

## ▼ (09강) Generative Models 1

#### ▶ PPT 필기

↳ 숨겨진 셀 7개

## ▼ (10강) Generative Models 2

#### ▶ PPT 필기

↳ 숨겨진 셀 11개

