

②

## 머신러닝 : 데이터가 둘어왔을 때

이미 정해진 알고리즘 (이미지 필터, 엣지 검출, HOG)으로 데이터 특징을 추출하고,  
이를 기반으로 분류모델을 만들어 예측한다.

딥러닝 : '인공신경망'을 사용하여 데이터 특징을 스스로 학습하고, 결과를 예측한다.

## 활성화 함수

|  | gradients 소실   | zero-centered             | 계산 비용                       | 역전파 학습   |
|--|--|---------------------------|-----------------------------|--|
| Sigmoid<br>$\sigma(x) = \frac{1}{1+e^{-x}}$                                | 출력이 0이나 1에 가까워질수록 발생   | X                         | 지수함수 쓰기 때문<br>(But, 큰 문제 x) |  |
| tanh<br>$\tanh(x)$   | 출력이 -1이나 1에 가까워질수록 발생  | O                         |                             |  |
| ReLU<br>$\max(0, x)$   | 음수면 0.<br>양수면 입력값<br>그대로 출력<br>gradients 소실 X<br>↓<br>음 → dead ReLU  | X                         | 단순 연산이라<br>계산 효율 뛰어남        |  |
| Leaky ReLU<br>$\max(0.1x, x)$  | 음의 영역에도 0 X<br>(= dead ReLU X)   | X                         | 단순 연산이라<br>계산 효율 뛰어남        | Parametric Rectifier (PReLU)<br>$f(x) = \max(\alpha x, x)$<br>backprop into 'alpha'<br>(parameter)<br><br>$\alpha \rightarrow$ 역전파 통해<br>학습됨 |
| Maxout<br>$\max(w_1^T x + b_1, w_2^T x + b_2)$                             | <ul style="list-style-type: none"> <li>이 활성화 함수에서는 내적의 기본적인 형태를 미리 정의하지 않는다.</li> <li>대신에, <math>\max(w_1^T x + b_1, w_2^T x + b_2)</math>를 사용한다.</li> <li>maxout은 둘 중에 최대값을 취한다.           <ul style="list-style-type: none"> <li>이는 saturation되지 않고, 그레이디언트를 잘 계산할 수 있다.</li> </ul> </li> <li>하지만, 파라미터의 수가 두배가 되는 문제점이 있다.</li> </ul> |                           |                             |  |
| ELU<br>$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$ | 음의 영역에도 0 X<br>(= dead ReLU X)<br><br>그러나 값이 너무 ↓<br>일정 값에 수렴  | zero-centered<br>특성에 가까워짐 | exp() 연산이 필요                | $\alpha \rightarrow$ 역전파 통해<br>학습됨   |

### Summary

- Default는 ReLU를 쓴다. (하지만 학습률을 잘 따져본다.)
- Leaky ReLU/Maxout/ELU/SELU 등을 시도해본다.
- sigmoid/tanh는 쓰지 마라!

1. Deep neural networks can generate non-linear responses because of activation functions such as ReLU, sigmoid, and tanh.) Answer the following questions. [15pt]

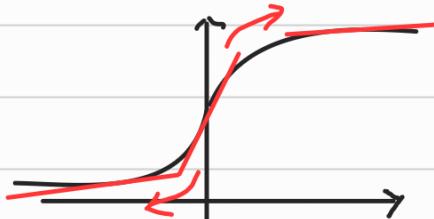
- a) Divide the sigmoid, tanh, and ReLU activation functions into those that 1) cause the vanishing gradient problem and those that 2) do not cause the vanishing gradient problem. [3pt]

↳ 기울기 소실 문제

1) 기울기 소실 문제를 일으키는 함수 : sigmoid, tanh

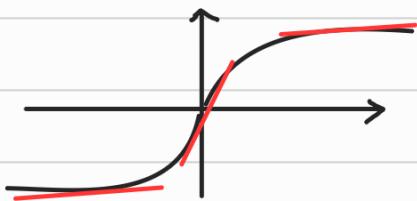
2) 기울기 소실 문제를 일으키지 X 함수 : ReLU

- b) Explain why each of the three activation functions does or does not cause the vanishing gradient problem. [12pt]



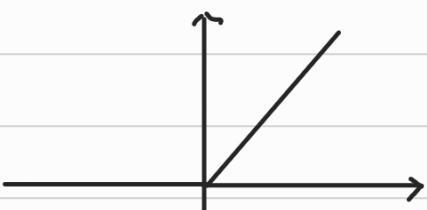
Sigmoid

Sigmoid 함수는 출력이  $(0, 1)$  사이의 값으로 업데이트되는데, 출력이 0이나 1에 가까워сь로 기울기는 0에 가까워진다. 이로 인해 기울기 소실 문제가 일어난다.



tanh

tanh 함수는 출력이  $(-1, 1)$  사이의 값으로 업데이트되는데, 출력이 -1이나 1에 가까워сь로 기울기는 0에 가까워진다. 이로 인해 기울기 소실 문제가 일어난다.



ReLU

ReLU는 입력이 양수일 때 기울기가 1로 일정하게 유지되며 음수일 때 기울기는 0이다. 이로 인해 양수 영역에서는 기울기가 소실되지 않는다.

(단, dying ReLU (모든 음수 입력에 대해 바탈성화되는 문제) 문제를 가질 수 있다.)

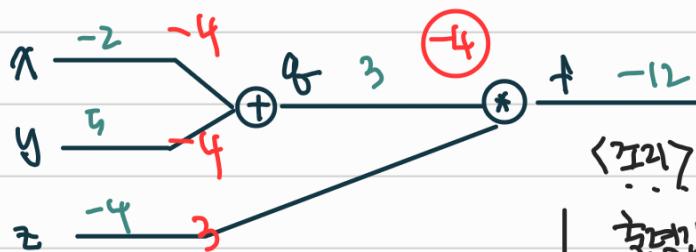
3

**Forward Propagation** : input layer부터 output layer까지 순서대로 범주를 계산하는 것

**단점** : 연산량이 기하급수적으로 늘어나 비효율적

**Back Propagation** : 출력값에 대한 입력값의 기울기를 계산하여 거꾸로 전파.  
(chain rule)

**문제 1**



<조각>

출력값에 대한 입력값의 기울기

$$\begin{aligned} \text{〈해설〉 } f(x) &= (x+y)z \\ &= fz \end{aligned}$$

$$\begin{array}{l|l} f = x+y & f = fz \\ \frac{\partial f}{\partial x} = 1 & \frac{\partial f}{\partial f} = z = -4 \\ \frac{\partial f}{\partial y} = 1 & \frac{\partial f}{\partial z} = f = 3 \end{array}$$

$$\frac{\partial f}{\partial x} = z = -4 \quad \text{chain}$$

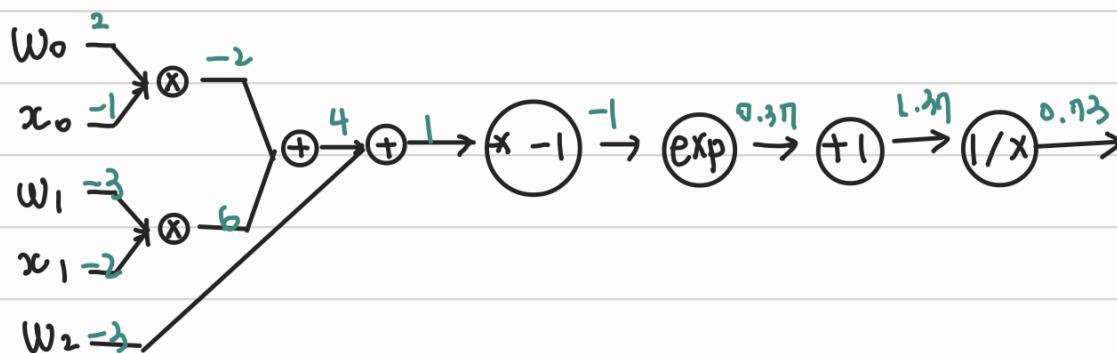
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial f} \times \frac{\partial f}{\partial y} = -4 \cdot 1 = -4$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial f} \times \frac{\partial f}{\partial z} = -4 \cdot 1 = -4$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial z} = 3$$

**문제 2**

$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$

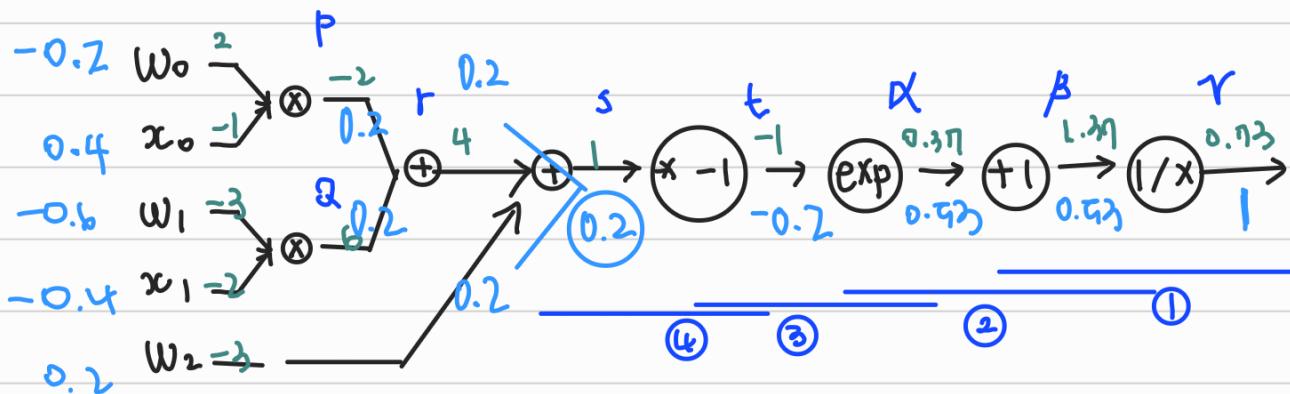


$$f(x) = e^x \rightarrow \frac{\partial f}{\partial x} = e^x$$

$$f(x) = \frac{1}{x} \rightarrow \frac{\partial f}{\partial x} = -\frac{1}{x^2}$$

$$f(x) = ax \rightarrow \frac{\partial f}{\partial x} = a$$

$$f(x) = 1+x \rightarrow \frac{\partial f}{\partial x} = 1$$



$$\textcircled{1} \quad Y = \frac{1}{x}$$

$$Y = \frac{1}{\beta},$$

$$\frac{\partial r}{\partial \beta} = -\frac{1}{\beta^2} = -\frac{1}{(1.37)^2} = -0.73,$$

What I want:

$$\frac{\partial r}{\partial \beta} = -0.73$$

$$\textcircled{2} \quad Y = \alpha + 1$$

$$\beta = \alpha + 1,$$

$$\frac{\partial \beta}{\partial \alpha} = 1$$

$$\frac{\partial r}{\partial \alpha} = \frac{\partial r}{\partial \beta} \cdot \frac{\partial \beta}{\partial \alpha} = (-0.73) \cdot 1$$

$$\textcircled{3} \quad Y = \exp(x)$$

$$\alpha = e^t,$$

$$\frac{\partial \alpha}{\partial t} = e^t = e^{-1} = 0.36$$

$$\frac{\partial r}{\partial t} = \frac{\partial r}{\partial \alpha} \cdot \frac{\partial \alpha}{\partial t} = -0.2$$

$$\textcircled{4} \quad Y = -x$$

$$t = -s \rightarrow \frac{\partial t}{\partial s} = -1$$

$$\frac{\partial r}{\partial s} = \frac{\partial r}{\partial t} \cdot \frac{\partial t}{\partial s} = 0.2$$

$$x \nearrow \oplus \leftarrow x$$

$$x+y - \bigcirc \nearrow \begin{matrix} x \\ y \end{matrix}$$

$$x \nearrow \otimes \leftarrow x$$

$$(z) \quad x \nearrow \max \leftarrow x$$

## Data preprocessing

Mean subtraction : 각 특성에서 평균을 뺠 데이터를 중심화하여 모델학습에 용이

Normalization 하는 이유? 모든 차원이 동일한 범위를 가지게 함으로써 전부 동등히 기여할 수 있음.

PCA (Principle Component Analysis) : 주요 패턴을 캡처하면서 데이터 주성분 추출, 차원을 낮춰어끔  
주성분 사이 decorrelated한 data를 제거한다.  
(Whitening) 상관관계  $\downarrow \rightarrow$  독립성  $\uparrow \rightarrow$  데이터 파악  $\uparrow$

## Weight Initialization // 가중치 초기화

$W = 0$  // 학습에 아무 도움  $\otimes$

$W = \text{random}$  // 작은 신경망에는 괜찮나. 깊어질수록 문제  $\uparrow$

$\therefore W = @\text{random}$  // 이정도 단계일 상수가 필요하다.

$0.01 \rightarrow$   $W=0$ 에 가까워므로 학습에 도움  $\otimes$  ↘ 차역개념  
 $0.05 \rightarrow$  오히려 너무 크며, 그레이언트 0이 되어 학습이 흥료되는데  
우리는 가중치 초기화하는데다가 학습이 멈춰갈... 때

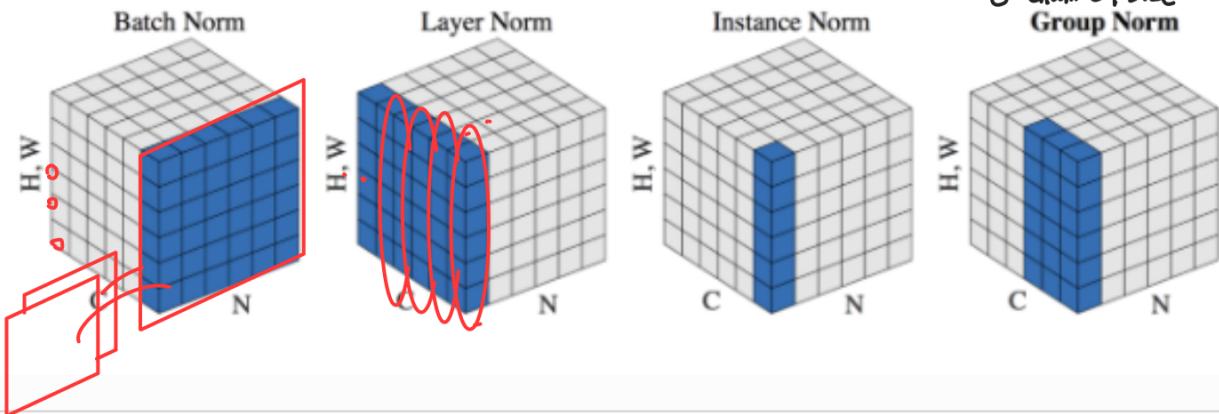
$\therefore \alpha = \frac{1}{\sqrt{\text{입력} + \text{출력}}}$ 로 가중치를 초기화한다. ... Xavier [제이비어]

그러나 tanh()에만 잘 적용되고 ReLU x ... Kaiming의 MSRA 초기화

$\alpha = \frac{1}{\sqrt{2(\text{입력} + \text{출력})}}$ 로 가중치를 초기화한다. ... Kaiming의 He 초기화

★★★ 가보

Batch Normalization과 Group Normalization을 비교하세요



N: Batch size

H, W: Spatial resolution

C: Channel size  
Group Norm

**Batch Normalization:** 배치의 단위로 정규화하는 것을 말한다.  
(평균, 분산)

ex)  $R \rightarrow 64개 이미지$  평/분

$G \rightarrow 64개 이미지$  평/분  $\rightarrow$  batchsize  $\uparrow \rightarrow$  성능  $\downarrow$

$B \rightarrow 64개 이미지$  평/분

**Layer Normalization:** 하나의 레이어 포인트에 대해 정규화하는 것  
(평균, 분산)

1번 이미지  $\rightarrow R, G, B$  평/분

2번 이미지  $\rightarrow R, G, B$  평/분

:

64번 이미지  $\rightarrow R, G, B$  평/분

장점: 이미지 특성을 가짐

$\rightarrow$  단점: 각 채널별 특성 X

**Instance Normalization:** 각 훈련 이미지의 각 채널에 대해 정규화한 것

(1번 이미지) R 1번 이미지 G 1번 이미지 B  
평/분 평/분 평/분

:

장점: 각 채널별 특성 ↑

단점: 색상 ↓, 신뢰성 ↓

**Group Normalization:** Instance와 Layer의 사이로 채널을 그룹 지어 정규화한 것 (성능 ↑)

## Learning Rate Scheduling

학습률은 초기에 그다지 점진적으로 작아야 한다. → 이유는?

초기 학습률 크게 하여 최적화 과정을 빠르게 하게 풀고,

점진적으로 감소하여 최적점에 안정적으로 수렴할 수 있게 때문이다.

배치 사이즈 ↑ → learning rate ↑

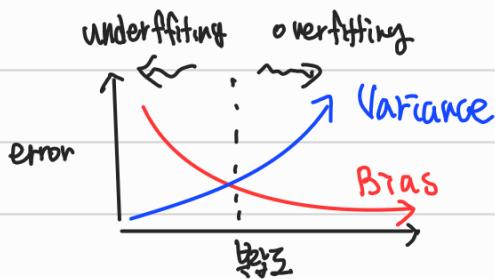
(장시간 학습 → 학습률)

## Bias - Variance Training

차이↑ 차이↑  
학습능력↓ 예측능력↓

- Bias ↓ Variance ↓ : 목표
- ⋮ Bias ↑ Variance ↓ : 고지적합 상태로, 모델의 복잡도 늘려라.
- ⋮ Bias ↓ Variance ↑ : 과적합, 모델의 복잡도 줄여라.
- ⋮ Bias ↑ Variance ↑ : 성능 ×. 모델 복잡度过

## Overttting 이런?



복잡도가 커지면 훈련 파라미터 수가 커져  
Bias는 줄고, Variance는 커지게 되는데,  
새로운 데이터가 올을 때 예측하지 못한다.

## 언제 발생하는가

< 학습데이터셋이 충분하지 않을 때  
모델의 파라미터 개수보다 적은 수의 샘플을 사용하는 경우

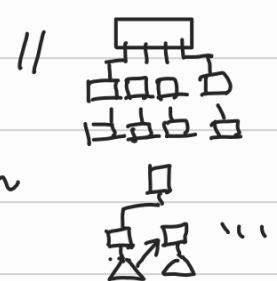
## 해결책

① Early Stopping : test data가 들어갈 때 저하되기 시작하면, 초기에 학습률 멈추게 함

② Model Ensembles : 여러 모델을 조합하여,

단일 모델이 갖는 한계를 극복하고 보다 안정적이고 정확한 예측을 목표로 함.

bagging : 복원추출을 사용하여 다수의 훈련데이터셋을 생성하고,  
각각의 데이터셋에 대해 **독립적인** 모델을 학습함



boosting : 악한 학습기를 **노자적으로** 학습시켜 강력한 학습기로 ~

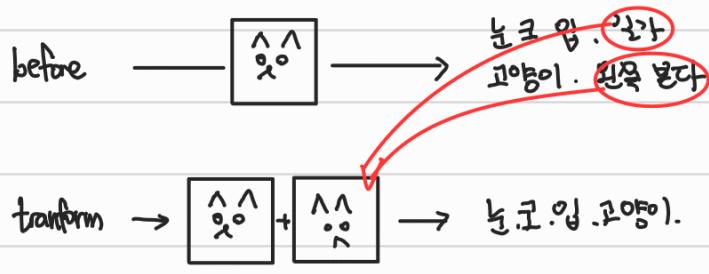


### ③ Regularization: Overfitting ↓ 위해 딘을 모델 성능을 높이는 방법

- L1: 가중치 절대값 합
  - L2: 가중치 제곱 합
- 잘 사용 X

+ Dropout: 임의의 노드를 드롭해서 학습에 참여 ⊗

+ Data Augmentation: 데이터의 양을 늘려 불필요한 특징들을 모델이 학습하는 것을 방지하고자 함



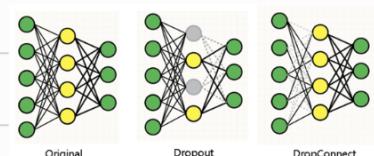
① Horizontal Flips ↔

② Random Crops and scales

③ Color Jitter (색 변화)

④ Random mix / combination : Translation, Rotation, Scaling, Stretch, Shearing

- Drop Connect: 노드가 아닌 weight들을 제거하는 방법 ↗



- Cutout: 이미지 영역을 무작위로 0으로 하는 방법 ↗



- Mixup: 이미지를 무작위로 섞은 것



- Cutmix: Cutout + Mixup

| Image | ResNet-50 | Mixup              | Cutout  | CutMix             |
|-------|-----------|--------------------|---------|--------------------|
| Label | Dog 1.0   | Dog 0.5<br>Cat 0.5 | Dog 1.0 | Dog 0.6<br>Cat 0.4 |

**Hyperparameter** : 모델의 학습 과정에서 사전에 설정하는 것

Hyperparameter 선택 과정

① 초기 상태에서의 loss 감소 확인

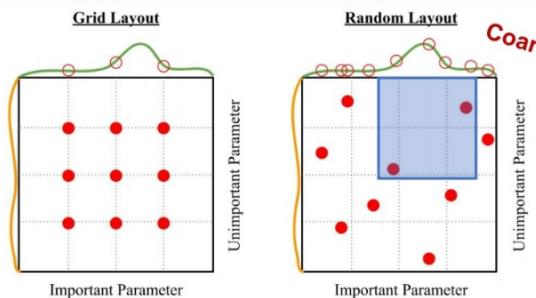
② 작은 데이터 셋을 사용하여 overfitting 시킴.

③ loss 각각 하는 적절한 학습률을 찾는다.

④ **Coarse grid for 1~5 epoch** : 1~5 epoch 동안 모델을 훈련하고,

결과적으로 얻어진 손실(loss) 값과 비교하여 적절한 범위를 꿰침.

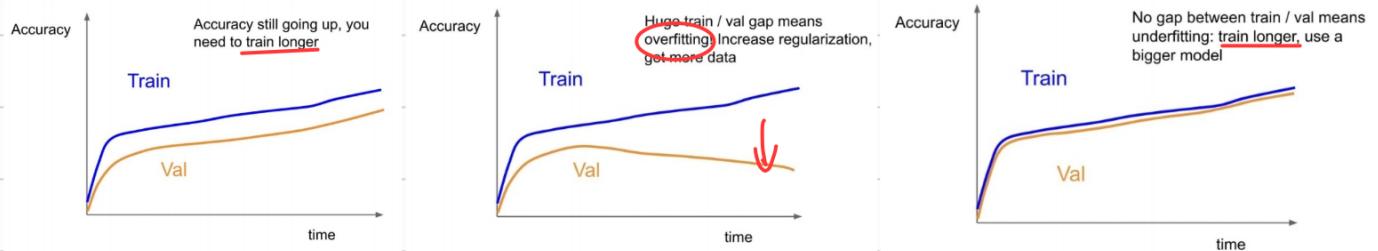
⑤ **Refine grid, train longer** : 더 세밀하게 조정하는 과정



방법 ( Grid search : 하이퍼파라미터의 모든 가능한 조합을 시도하여 최적의 값을 찾는 방법  
Random search : 하이퍼파라미터의 랜덤 조합 선택하는 방법 )

⑥ Look at loss curves

overfitting → ① 정교화 ② 데이터 추가



⑦ GOTO step 5

**Transfer learning** (전이학습) : ImageNet과 같이 아주 큰 데이터넷 (fully tuning)에서 훈련된 모델의 가중치를 가지고 와서 우리가 해결하고자 하는 과제에 맞게 재보정(fine-tuning)

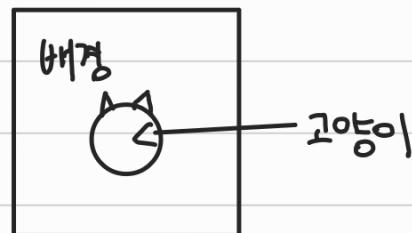
6

## Fully connected Layer의 장단점

장점: 모든 것을 다 학습할 수 있다

단점: 데일리 형상을 무시한다

= 중요한 것과 중요 × 부분 구분하기 ⊖



Convolution 필터 여러개 사용하는 이유는?

각 필터마다 서로 다른 특성을 추출하기 위해



stride: 'filter가 얼마나 움직이는가'

- 장점: 불필요한 특성 제거하는 효과 → 연산속도 ↑

- 단점:  $S \uparrow \rightarrow$  공간 feature 특성 손실

zero-padding을 사용하는 이유는?

1x1 convolution layer를 통과하게 되면 input 이미지 ↓. padding 이용하면 그대로 유지할 수 있다.

$N \times N \times C \dots K \text{ } F \times F \text{ filter}$

stride S, padding P  $\longrightarrow$

$$\text{output size} = \left( \frac{N+2P-F}{S} \right) + 1$$

output volume size = output size  $\times$  K

number of parameter (each) =  $F^2 C + \text{bias}$

number of parameters = number of parameter  $\times$  K

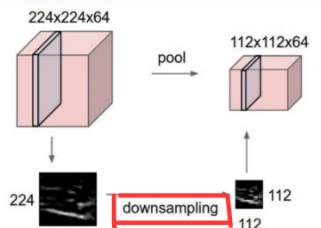
pooling 이런,

Conv layer와 activation 결과값에 technique를 적용한 것.

(ex) Max pooling



receptive field에서 큰 값



// depth는 유지되면서  
downsampling

공식  $\text{output size} = \frac{N+2P-F}{S} + 1$ , number of parameters : 0

output volume size : output size  $\times$  C

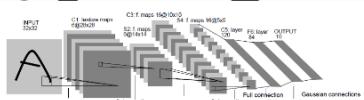
## CNN의 역사



CNN에 이전 위치 fully connected layer에 넣어서 핵심의 행으로 축약화해서 인식했다.

이는 레이어 흐름이 꼬시되고, 상관관계를 잊기 되어 전체를 고려하지 못하는 문제점이 있음.

즉. 순서서 A 만날 시, 겸하검하 ...> 'A' 전체 인식 ✗



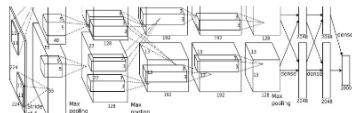
LeNet은

(딥러닝 X, 지금 CNN 구조와 바꾸기 예상)

3개 Convolution layer, 2개의 sub-sampling layer, 1개 fully connected layer  
이러한 fully connected NN의 아키태처에 대해 잘 대응하는 것으로 확인되었다.

LeNet-5 ↗

AlexNet



large dataset ↗

: smaller compute, still memory heavy. lower accuracy

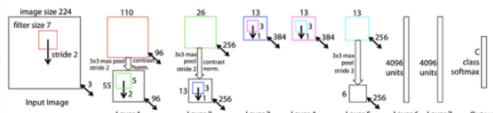
- 5개 convolution layer, 3개 fully connected layer

- 2개의 GPU 사용을 위한 병렬구조

- ReLU → 왜 써? gradient 편평 우려 X

- max pooling

- overfitting 문제(해결하기) 위해 dropout 사용



ZFNet

새로운 CNN 모델 구조를 제시했다가 보다는 visualizing 기법을 통해 CNN이 왜 좋은가를 증명

2개 GPU 대신 dense connection을 사용해(교류: 훈련하기 하는 것) 성능↑

VGGNet : most parameters. most operation

- AlexNet (8개 layer) → VGGNet (16-19 layer) : layer을 많이 쌓는다. CNN 성능 ↑

★ 3x3 Conv만 사용했다

그리고 그동안이나 7x7에서 receptive field 사용하는 것보다

7개의 3x3 Conv → receptive field가 1개 7x7를 만드는 것과 같다.

$$(3 \times 3 \times C^2) \times 3 \text{ vs } (7 \times 7 \times C^2)$$

$$= 27C^2 \quad = 49C^2$$

① 파라미터 개수 ↓

② non-linearity가 증가함 (비선형)  $\text{ReLU} \rightarrow \text{복잡} = \text{표현} \uparrow$

7x7 < 2개 3x3

7x7 < 7개 3x3.

INPUT: [224x224x3] memory: 224\*224\*3=150K params: 0 (not counting biases)  
 CONV3-1: [224x224x64] memory: 224\*224\*64=3.2M params: (3\*3\*3)\*64 = 1,728  
 CONV3-2: [224x224x64] memory: 224\*224\*64=3.2M params: (3\*3\*64)\*64 = 36,864  
 POOL2: [112x112x64] memory: 112\*112\*64=800K params: 0  
 CONV3-1B: [112x112x128] memory: 112\*112\*128=1.6M params: (3\*3\*64)\*128 = 73,728  
 CONV3-1B: [112x112x128] memory: 112\*112\*128=1.6M params: (3\*3\*128)\*128 = 147,456  
 POOL2: [56x56x128] memory: 56\*56\*128=400K params: 0  
 CONV3-2B: [56x56x256] memory: 56\*56\*256=500K params: (3\*3\*256)\*256 = 294,912  
 POOL2: [28x28x256] memory: 28\*28\*256=200K params: 0  
 CONV3-5: [28x28x5] 2] memory: 28\*28\*512=400K params: (3\*3\*256)\*512 = 1,179,648  
 CONV3-5: [28x28x5] 2] memory: 28\*28\*512=400K params: (3\*3\*512)\*512 = 2,359,296  
 CONV3-5: [28x28x5] 2] memory: 28\*28\*512=400K params: (3\*3\*512)\*512 = 2,359,296  
 CONV3-5: [28x28x5] 2] memory: 28\*28\*512=400K params: (3\*3\*512)\*512 = 2,359,296  
 POOL2: [7x7x512] memory: 7\*7\*512=2048K params: 0  
 FC: [1x1x4096] memory: 4096 params: 7\*512\*4096 = 102,760,448  
 FC: [1x1x4096] memory: 4096 params: 4096\*4096 = 16,777,216  
 FC: [1x1x1000] memory: 1000 params: 4096\*1000 = 4,096,000

TOTAL memory: 24M \* 4 bytes ~ 96MB / image (only forward ~2 for bwd)  
 TOTAL params: 138M parameters

Note: ① ②  
Most memory is in early CONV

Most params are in late FC

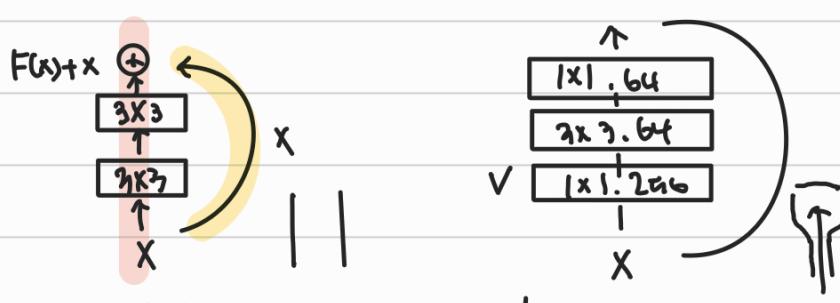
## Google Net : most efficient

- 224 layer
- Inception module (network in network) → ensemble (앙상블); 동시에
- $|X|$  convolution 역학

GoogleNet은 깊은 conv, 덥석 conv 앞쪽에  $|X|$  conv를 두어 연산 ↓, 비용 ↑

## ResNet (작자) 모드의 폰 → 적당한 폰, 높은 정확성

layer 쌓으니까 학습이 어려워 오히려 training error ↑ (파라미터 ↑) → 성능 ↓



왜? identity mapping

기존에는  $f(x)$ 는  $f_2(x) + x_1 = f_3(x) + x_2 + x_1$  와 같아 경사도 문제 발생하여 기울기  $x$ 를 더해준다 문제를 해결하기 때문에 덕분에

→ 계산이 깊어질수록 성능 ↑

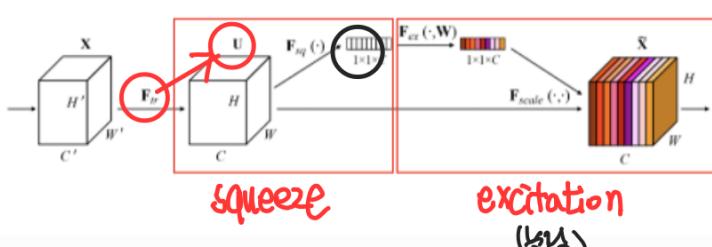
$3 \times 3$  con 연산량 = 1x1 con  $\times 9$  times

∴  $|X|$  bottleneck block

대부 강하고 연산 속도 ↑

## SENet

→ how does it work?



기본 문제점은

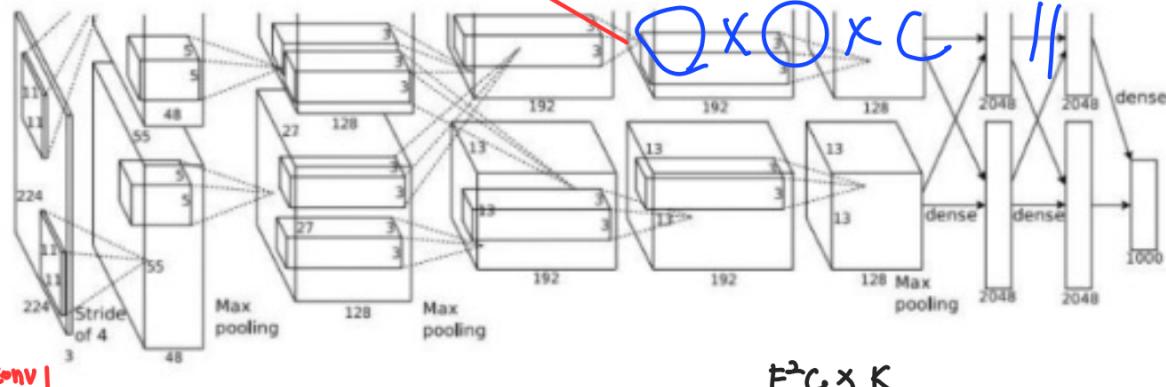
squeeze하면 전체 막아 x, 정보 활용 제한적 ...  $\rightarrow |X| \times C$ 로 압축 ★  
 기본의 주제로 조작해온 것이다.

CNN의 첫 번째 뉴런

AlexNet ~ ReLU 사용

$$\frac{N+2P-F}{S} + 1$$

$$O \times O \times K \quad || \quad F^2 C \times K$$



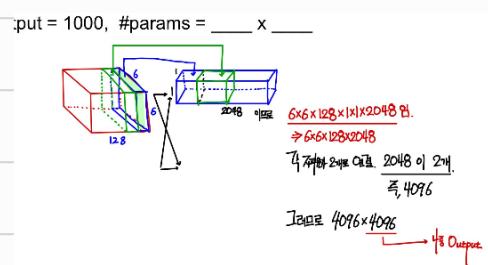
|           |
|-----------|
| CONV1     |
| MAX POOL1 |
| NORM1     |
| CONV2     |
| MAX POOL2 |
| NORM2     |
| CONV3     |
| CONV4     |
| CONV5     |
| Max POOL3 |
| FC6       |
| FC7       |
| FC8       |

|           | output volume size  | numbers of parameters             |
|-----------|---|-----------------------------------|
| CONV1     | $227 \times 227 \times 3 \rightarrow 55 \times 55 \times 48$ filter, stride 4<br>$\frac{227-11}{4} + 1 = \frac{216}{4} + 1 = 54 + 1 = 55$<br>$55 \times 55 \times 48$ | $11 \times 11 \times 3 \times 48$ |
| Max Pool1 | $55 \times 55 \times 48 \rightarrow 27 \times 27 \times 128$ , stride 2.<br>$\frac{55-3}{2} + 1 = \frac{52}{2} + 1 = 26 + 1 = 27$<br>$27 \times 27 \times 128$        | 0                                 |
| NORM1     |   |                                   |
| CONV2     | $27 \times 27 \times 128 \rightarrow 13 \times 13 \times 128$ , padding 2, $K=128$<br>$\frac{27+4-11}{1} + 1 = 27$<br>$13 \times 13 \times 128$                       | $5 \times 5 \times 48 \times 128$ |
| MaxPool2  | $27 \times 27 \times 128 \rightarrow 13 \times 128$ with stride 2<br>$\frac{27-3}{2} + 1 = 13$<br>$13 \times 13 \times 128$   | 0                                 |

: CONV3  
 $(3 \times 3 \times 128 \rightarrow 3 \times 3 \times 128)$ , stride 2

$$\frac{13-3}{2} + 1 = \frac{10}{2} + 1 = 6$$

$$6 \times 6 \times 128 \downarrow \times 2048$$



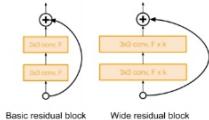
## Wide ResNet

→ 뭐가 다른가?

채널을 커우면 그만큼 풍부하고 다양한 학습을 할 수 있다.

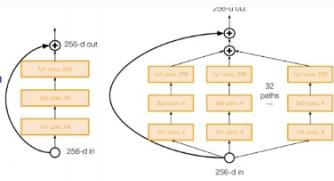
그러나 채널이 2배 커지면 파라미터 개수는 제곱 배 많아진다.

또한 연산 부도는 꼬집만 커진다 (gpu에서 유통하기에 힘기 때문)



## ResNext (= Inception Module)

Group Normalization.



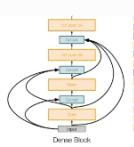
concatenate

## Dense Net

| identity mapping X → concat 문제를 해결했다.

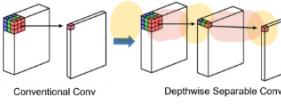
Vanishing gradient 문제

[더하는게 X, 갈다붙임(똑같은걸 copy해서)]



## MobileNet

Depthwise Convolution → 공간 정보 학습 (채널 상관X) (경과)



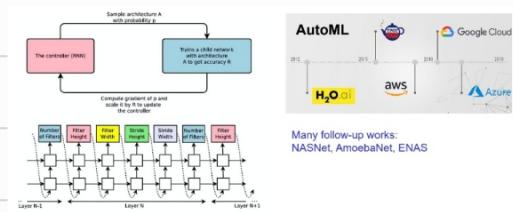
pointwise convolution을 따로 수행하여 채널간 상관성 학습 높은 성능

group의 개수 = channel

## NasNet

AI가 AI를 낳은 자식 느낌으로

최적은 신경망 아키텍처를 선택하는 혁신적인 접근방법



Many follow-up works:  
NASNet, AmoebaNet, ENAS

## EfficientNet

compound scaling 방법으로 깊이, 너비, 해상도를 조정할 때 확장함으로써

효율성 & 성능을 극대화한 딥러닝 모델

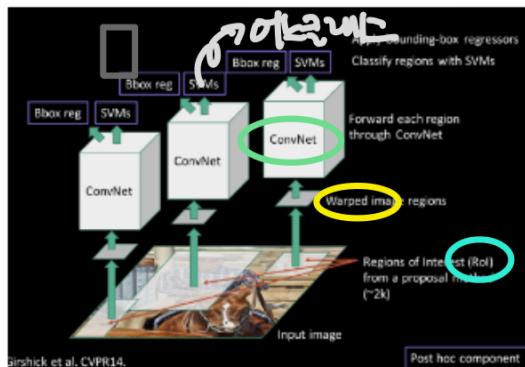
↑

지정하면 자동적으로

$$\begin{aligned} \text{depth: } d &= \alpha^d \\ \text{width: } w &= \beta^d \\ \text{resolution: } r &= \gamma^d \end{aligned}$$

s.t.  $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$   
 $\alpha \geq 1, \beta \geq 1, \gamma \geq 1$

## 01. R-CNN



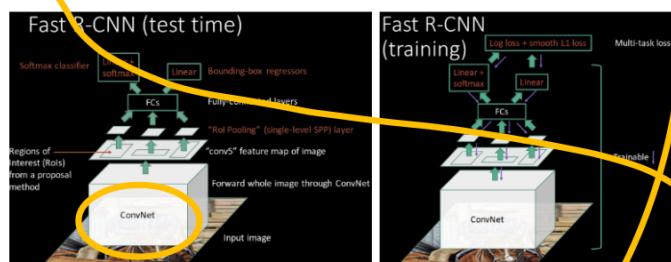
### R-CNN의 작동원리

- ① Input 이미지를 가지고 CPU상에서 selective search를 진행하여 물체가 존재할 것 같은 지역(ROI)을 2000개 정도 뽑아낸다.
- ② 이렇게 뽑아낸 ROI(Refions Of Interest)들은 각각 다른 크기와 위치를 가진다. 때문에 이를 crop & warp하여 CNN에 들어가기 좋은 정사각 형태로 만든다.
- ③ 각각의 ROI들을 ConvNet으로 둘러 feature vector를 추출한다.
- ④ 추출된 feature vector들을 SVMs를 이용하여 어떤 클래스에 해당되는지, 그리고 Regressor를 이용하여 정확한 물체의 위치가 어디인지를 bounding box를 조절한다.

문제점

- ① training & test 과정에 많은 시간.
- ② SVM + regression 분리되어 있어 정확도↓.
- ③ 여러 단계로 training pipeline 복잡

## 02. Fast R-CNN

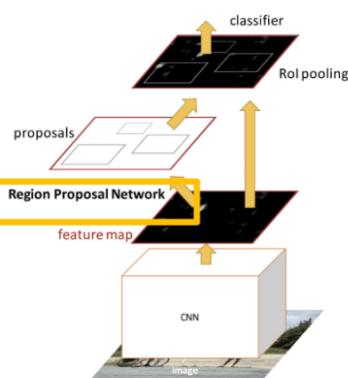


R-CNN의 한계점을 보완하기 위해 Fast R-CNN이 등장하게 되었다. Fast R-CNN의 기본적인 아이디어는 CNN을 돌리는 것과 region을 추출하는 순서를 바꾸자!는 것이다. 이렇게 순서를 바꾸게 되면, 고해상도의 Conv feature map이 생성되는 효과를 얻을 수 있다.

- ① input 이미지에 대して ConvNet을 돌린다.
- ② 생성된 feature map을 가지고 region proposal method를 통해 ROI를 추출한다.
- ③ 추출 된 ROI를 ROI pooling 기법을 이용하여 각각의 region들에 대해 feature에 대한 정보를 추출한다.
- ④ fully connected layer로 넘어간다.
- ⑤ 마지막으로 classification과 regression을 진행한다. R-CNN에서는 여러 개의 클래스가 존재하는 상황에서 각각의 클래스에 대해 binary SVMs를 사용했지만, Fast R-CNN은 기본적인 CNN network를 이용해서 Softmax layer를 거쳐 각 클래스에 대한 확률 값을 구한다는 점에서 차이가 있다.

end-to-end

## 03. Faster R-CNN



R-CNN과 Fast R-CNN 모두 CPU에서 region proposal을 진행하기 때문에 속도가 매우 느리다(기본적으로 selective search가 CPU상에서 돌아가도록 라이브러리가 설계되어있기 때문). Faster R-CNN은 이러한 점을 보완하기 위해 region proposal을 위한 모든 연산을 GPU상에서 진행하는 RPN(Region Proposal Network)을 제안한다 (Faster R-CNN의 핵심 아이디어). RPN은 region proposals를 진행하는 network로, feature map을 보고 어느 곳에 물체가 있을법한지 예측하며, selective search의 시간적인 단점을 해결하는 방법이다. 학습이 이루어 진 다음 GPU 상에서 한번의 forwarding만 수행하면 바로 어느곳에 물체가 있을법한지 예측할 수 있기 때문에 훨씬 빠르게 작동한다. 그다음 Fast R-CNN에서 사용하는 detection network의 구조를 그대로 사용해서 각 물체와 그 물체가 존재할법한 위치에 대해 classification과 regression을 진행한다. 쉽게 말해, Faster R-CNN은 기존의 Fast R-CNN에서의 selective search를 RPN으로 바꿔서 모든 과정을 end-to-end로 학습하는 것이라고 생각하면 된다.