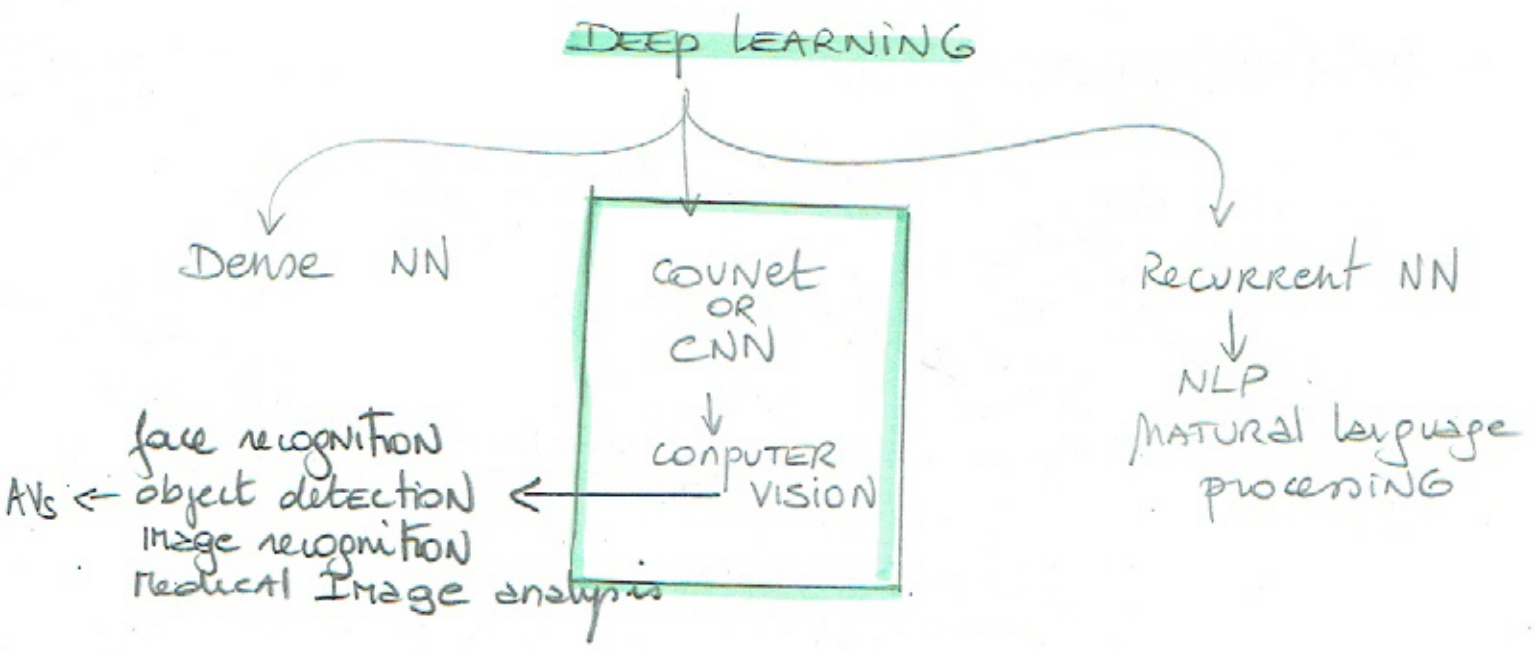
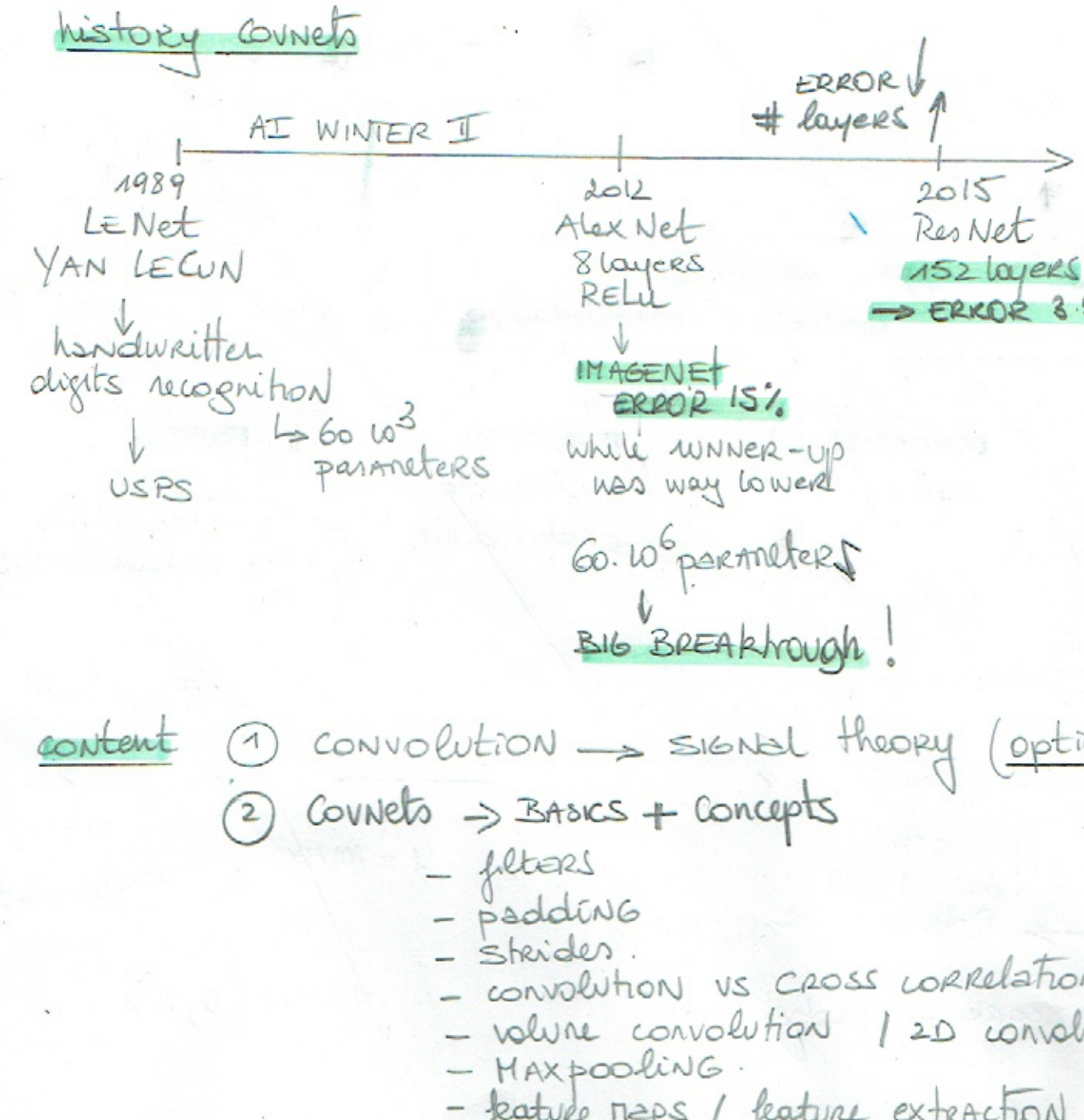
# LESSON 16 : Convolutional Neural Networks : CovNets ①




## history CovNets




**content**

① CONVOLUTION → SIGNAL theory (optional)

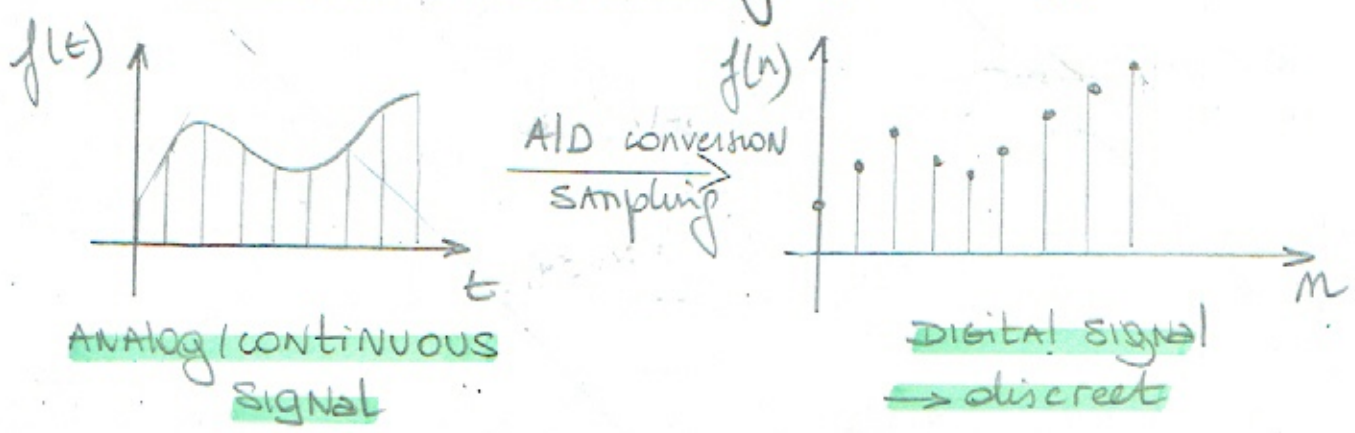
② CovNets → BASICS + Concepts

- filters
- padding
- strides
- convolution vs CROSS CORRELATION
- volume convolution / 2D convolution
- MAXpooling
- feature maps / feature extraction

→ convnets for small datasets
- transfer learning
- fine tuning
- data augmentation

→ coding (KERAS) → CAT DOG picture recognition

## ① Convolution → signal theory (optional)

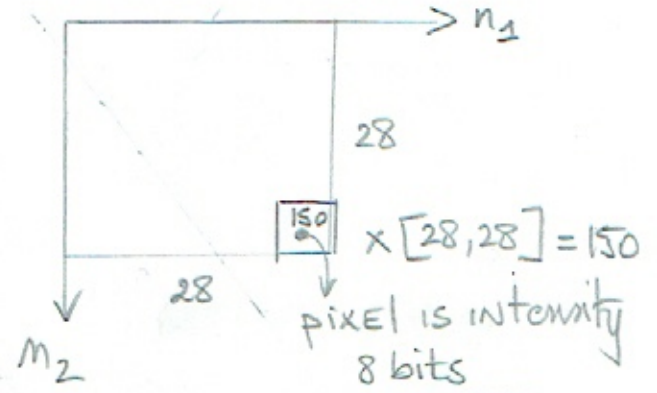



Analog / continuous signal

Digital signal → discreet

### Image representation

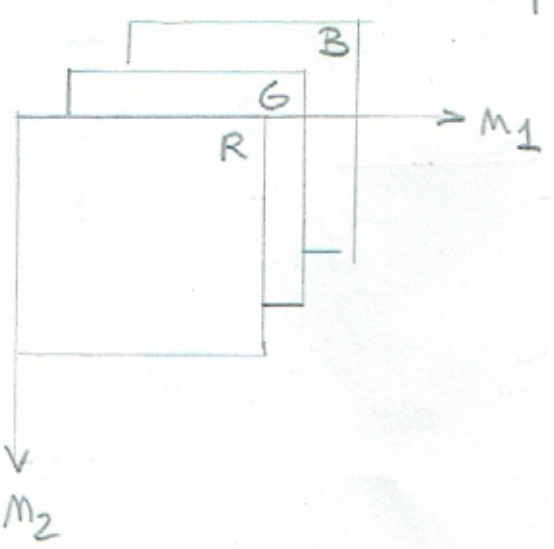
grayscale image → SCALAR 2D discreet signal




color image → VECTOR 2D Discreet signal.

$$X(n_1,n_2) = [X_R(n_1,n_2), X_G(n_1,n_2), X_B(n_1,n_2)]$$

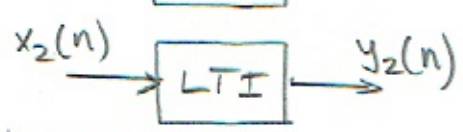
$X_R, X_G, X_B$ represent pixel intensity along their AXIS

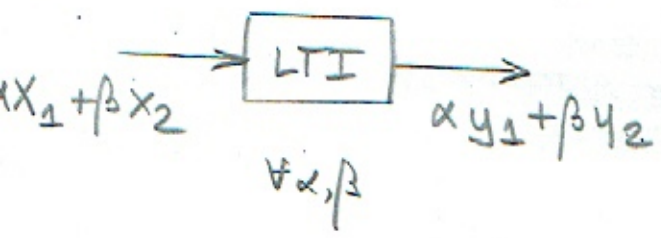


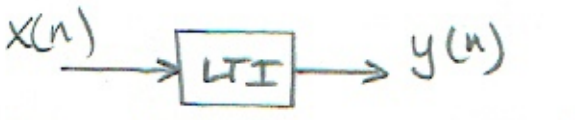
# CONVOLUTION of 2 SIGNALS

↳ LINEAR, TIME-INVARIANT Systems (LTI)

superposition
scaling



shift IN INPUT → SAME shift in OUTPUT.

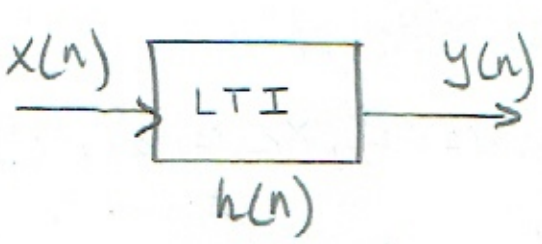
Now what ? when system is LTI

→ decompose AN arbitrary signal INTO A SET of BASIC SIGNALS

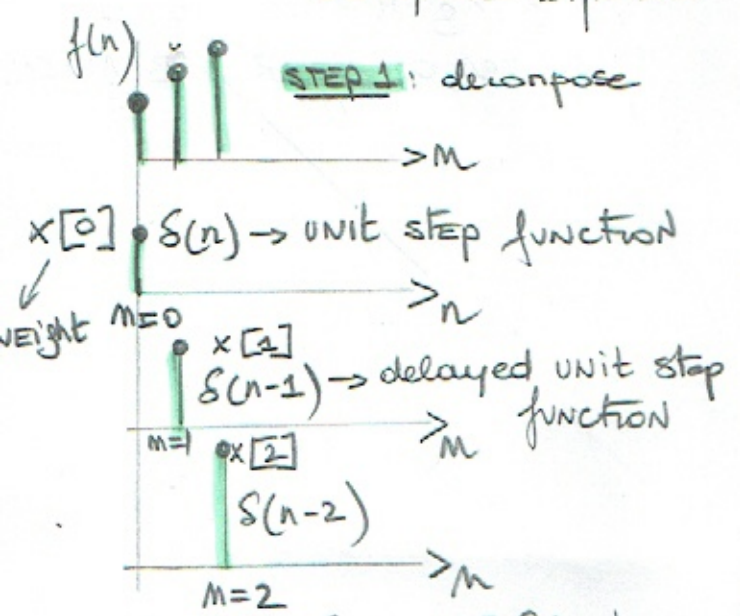
h(n)
↳ delayed impulse responSES (convolution)

↓

GOAL: determine y(n) based ON x(n) AND h(n)

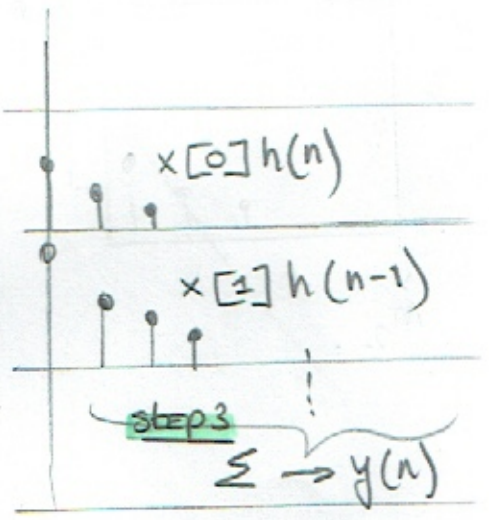



↳ impulse response




UNIT STEP response

STEP 2




→ $x[n] = x[0]\delta(n) + x[1]\delta(n-1) + x[2]\delta(n-2) + \dots \rightarrow \sum_{k=-\infty}^{+\infty} x[k]\delta(n-k)$ (1)

(1) → we decomposed arbitrary sequense to a linear combination of delayed unit step functions

↓ LINEARITY

response to that linear combination is a linear combination of the responses $h_k(n)$

$$\rightarrow y(n) = \sum_{-\infty}^{+\infty} x(k)\, h_k(n)$$

↓ time-invariant (response to impulse at time k = respone to an impulse at time 0 shifted over to time k)

$$\rightarrow h_k(n) = h_0(n-k)$$

$$\rightarrow \boxed{y(n) = \sum_{-\infty}^{+\infty} x_k\, h(n-k)}$$

↳ CONVOLUTION SUM

$$\boxed{y(n) = X * h(n)}$$

SUMMARY:

$\delta(n) \rightarrow$ LTI $\rightarrow h(n)$ ⌐→ impulse response

$\delta$: unit step function

↳ DIRAC DELTA FUNCTION

$\delta(n) = +\infty \quad n = 0$

$0 \quad n \neq 0$

(AREA = 1)

$\delta(n-k)$ (shifted unit step) → LTI → $h(n-k)$ time-invr

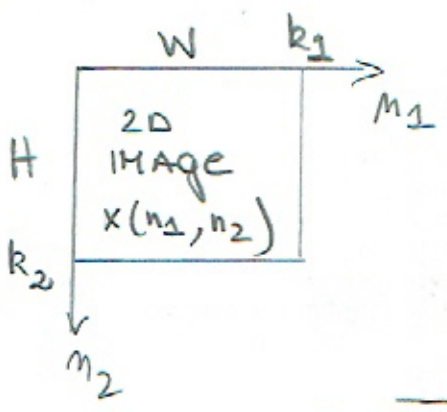
$x(k)\,\delta(n-k)$ → LTI → $x(k)\,h(n-k)$ Linearity

$x(k)\,\delta(n-k)$ ↙ weighted × shifted unit step

CONCLUSION for LTI ($h(n)$: response to $\delta(n)$)

If you know its response to a unit step signal at n=0 you can generate the response to an arbitrary sequence $y(n)$ through the convolution sum

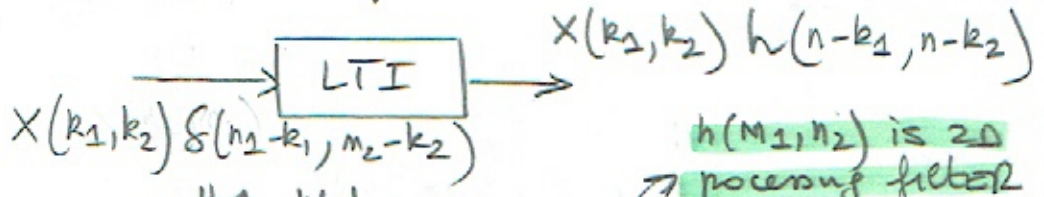
# IMAGE processing → 2D convolution




2D unit step function

$\delta(n_1, n_2)$ — $+\infty$ if $n_1 = n_2 = 0$
— 0 elsewhere

1D analogy



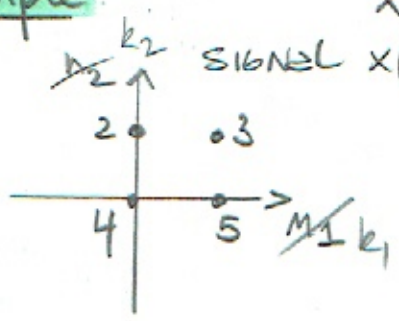

$h(n_1, n_2)$ is 2D processing filter

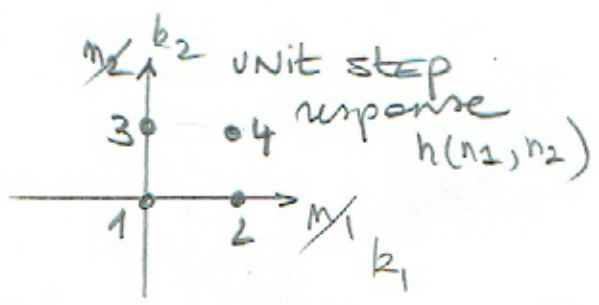
$$y(n_1, n_2) = \sum_{k_2=0}^{H-1} \sum_{k_1=0}^{W-1} x(k_1, k_2)\, h(n_1 - k_1, n_2 - k_2)$$

convolution sum

$k_1, k_2$ pixels
$x(k_1, k_2)$ → intensity values

**example:**



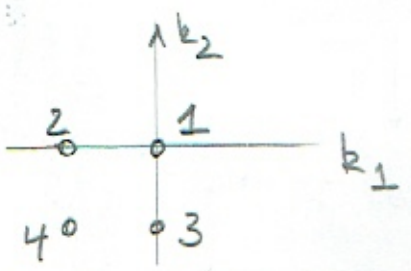




$$x(n_1, n_2) \begin{cases} = 5 & n_1 = 1 \quad n_2 = 0 \\ = 4 & n_1 = n_2 = 0 \\ = 2 & n_1 = 0 \quad n_2 = 1 \\ = 3 & n_1 = n_2 = 1 \\ = 0 & \text{elsewhere} \end{cases}$$

$$h(n_1, n_2) \begin{cases} = 2 & n_1 = 1 \quad n_2 = 0 \\ = 1 & n_1 = n_2 = 0 \\ = 3 & n_1 = 0 \quad n_2 = 1 \\ = 4 & n_1 = n_2 = 1 \\ = 0 & \text{elsewhere} \end{cases}$$

**Step 1**: $n_1, n_2$ axes → $k_1, k_2$ axes

**Step 2**: leave $x(k_1, k_2)$ as is
reverse impulse response $h(k_1, k_2) \rightarrow h(-k_1, -k_2)$

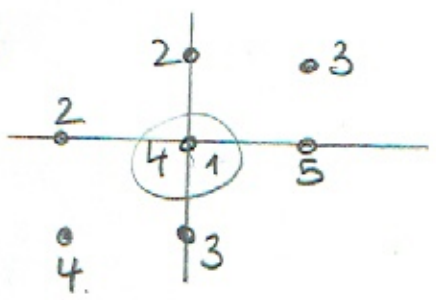



**Step 3**: → place $h(-k_1, -k_2)$ at diffent offset values for $n_1, n_2$
→ $h(n_1 - k_1, n_2 - k_2)$

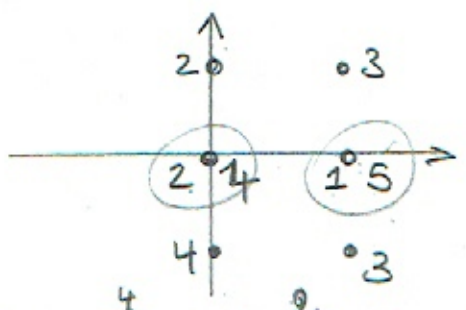
$y(0,0) = x(0,0) \cdot h(0,0) = 4$ (x(0,0) = 4, h(0,0) = 1)

We look for overlap.
Otherwise conv. sum = 0!

**SITUATION 1**




**SITUATION 2**

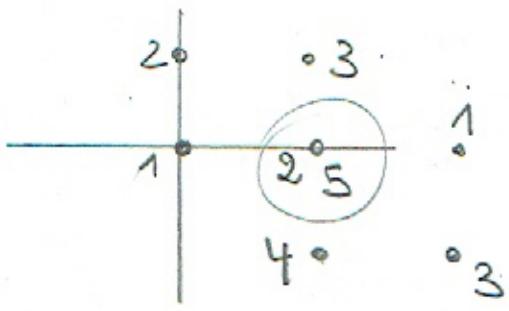



$y(1,0) = x(0,0) \cdot h(0,0) \rightarrow 8$ (x(0,0) = 4, h(0,0) = 2)

$+ x[1,0] \cdot h(1,0) \rightarrow 5$ (x[1,0] = 5, h(1,0) = 1)

$= 13$

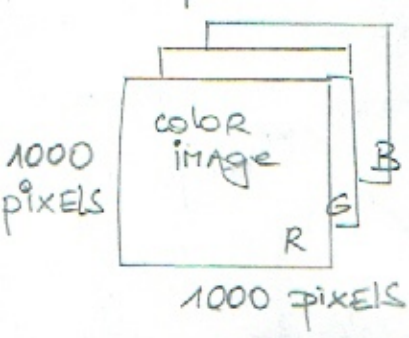
**SITUATION 3**




$y(2,0) = 10$

## CONVNETS

2 issues at hand with Dense, fully connected NNs with respect to computer vision




$\rightarrow$ 1000 × 1000 × 3

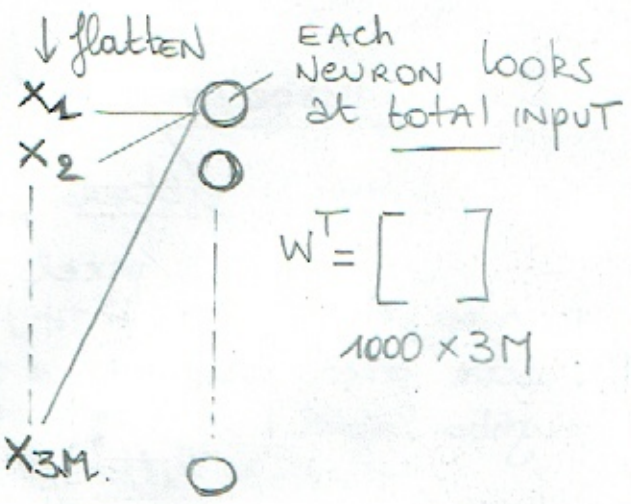
$\rightarrow$ 3 MEGA pixel

$\downarrow$

3M features

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$ 3M × 1.

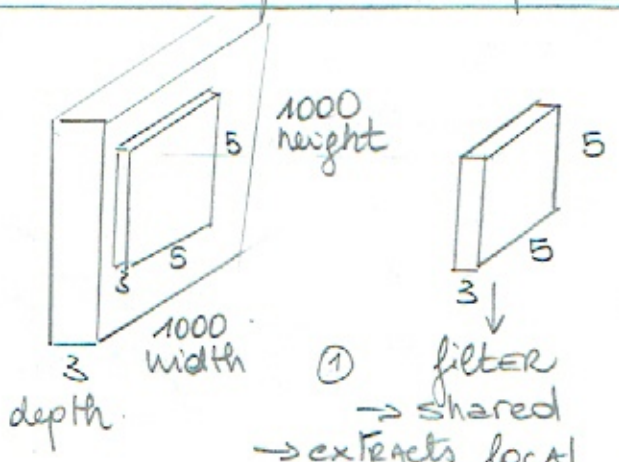
3 Million rows




② → flatten image → spacial info is lost

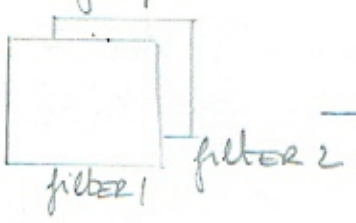
① → Dimensions quickly spin out of control

## We want to preserve spacial structure of picture

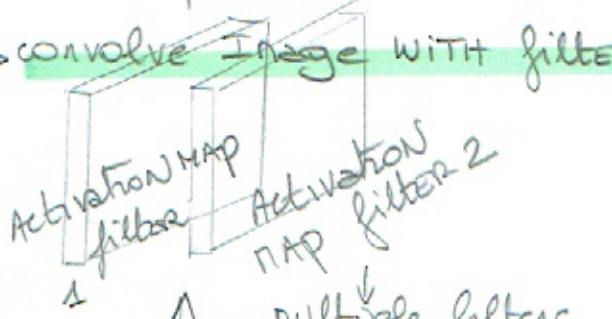



① filter
→ shared
→ extracts local features

② you can apply ≠ filters to extract ≠ types of features.

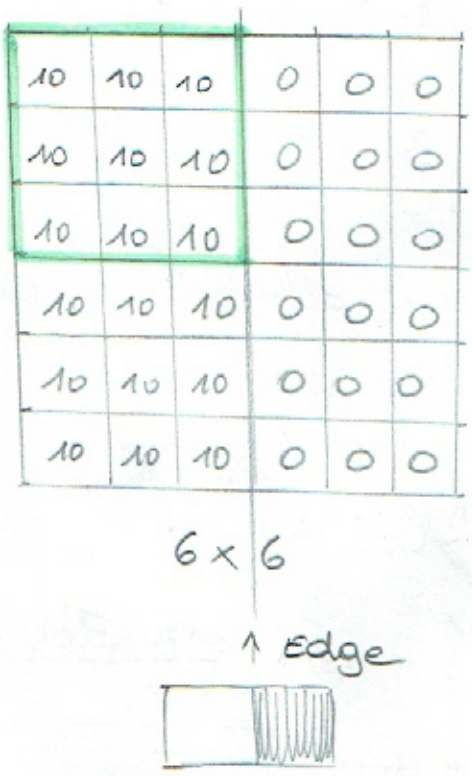



→ convolve Image with filter

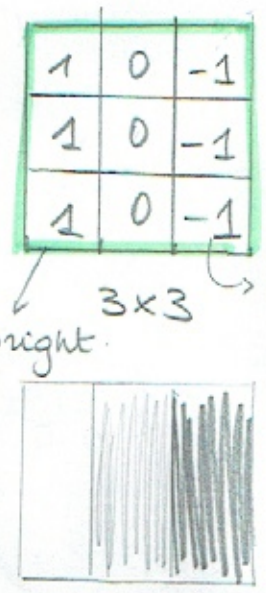


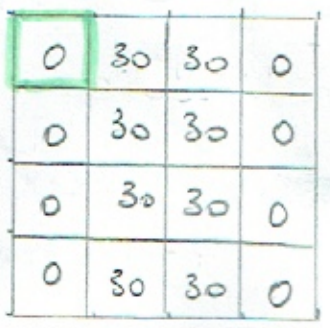

multiple filters generate multiple Activation maps

## How does filter work ?



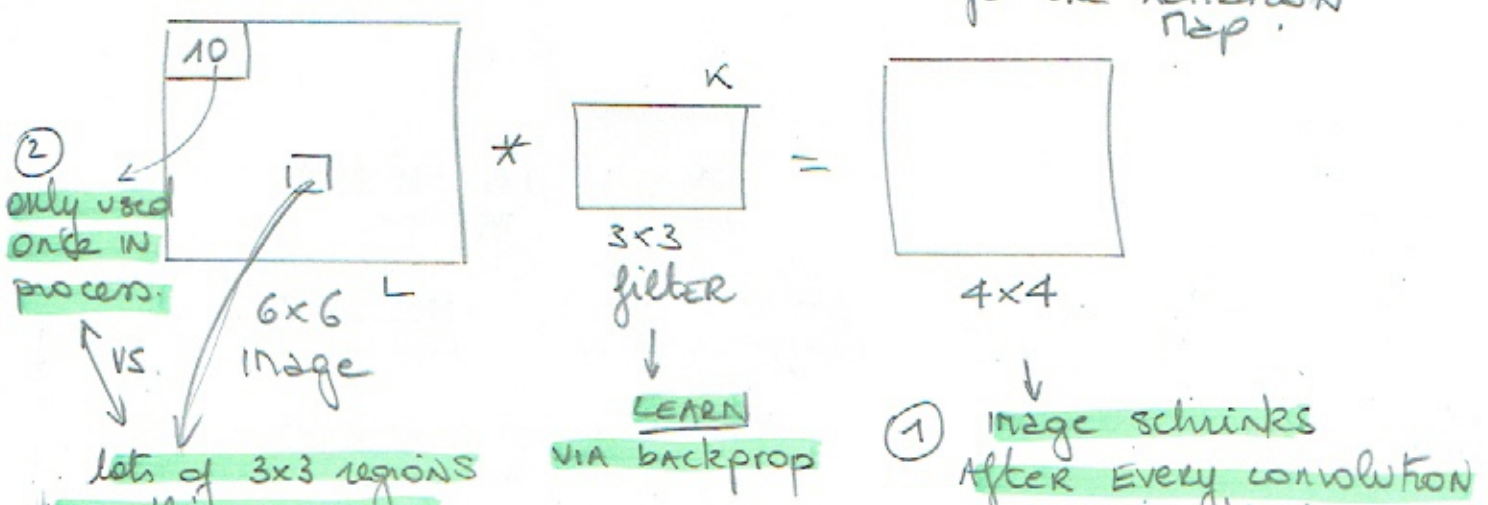

VERTICAL EDGES filter



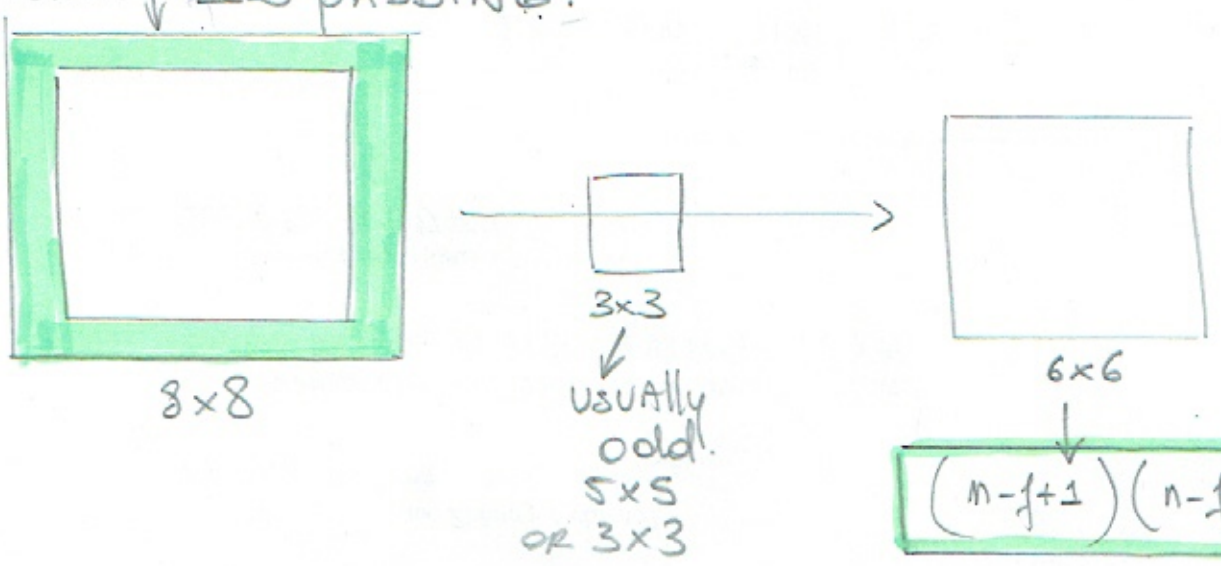




↑ VERTICAL EDGE
→ brighter region

## ZERO PADDING




throwing info ZERO PADDING.




**STRIDING** → default is 1

↳ number of pixels to move in each spatial direction

**OUTPUT image size** (convolved)

L → length of input image
K → length filter / kernel
P → # of zeros padded
S → Stride of the convolution

$$L' = \frac{L - K + 2p}{S} + 1$$

$L'$ → length of convolved output image

# Architecture of a convNet


IMAGE
64 x 64 x 3
↑
color

CONV layer 1

CONV layer
↓ filters
generates output feature maps.
↙
# maps = # filters
↙
4 hyper parameters
# filters
size filters
stride
padding.

RELU
1) Adds non-linearity
2) provides non saturating gradient for pos net inputs

Pooling
↓
downsamples Activation maps (except for depth)

Fully connected layer
↓
NO weight sharing !

OUTPUT UNIT
→ sigmoid

Pooling → downsample ! → no parameters just hyperparameters

| | | | |
|---|---|---|---|
| 1 | 3 | 2 | 1 |
| 6 | 2 | 0 | 5 |
| 9 | 0 | 8 | 0 |
| 0 | 8 | 0 | 7 |

4x2

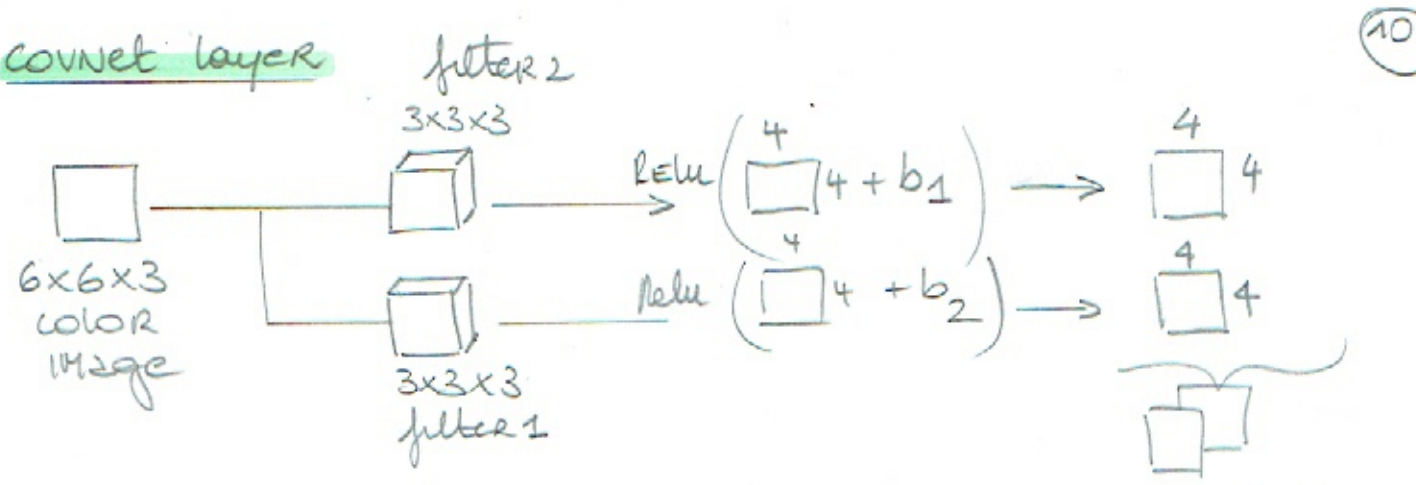
take max of region
→ MAXpooling
2x2
↳ stride 2

| | |
|---|---|
| 6 | 5 |
| 9 | 8 |

2x2

↓
you can also do AVERAGE pooling

## CONVNet layer




If 10 filters 3×3×3

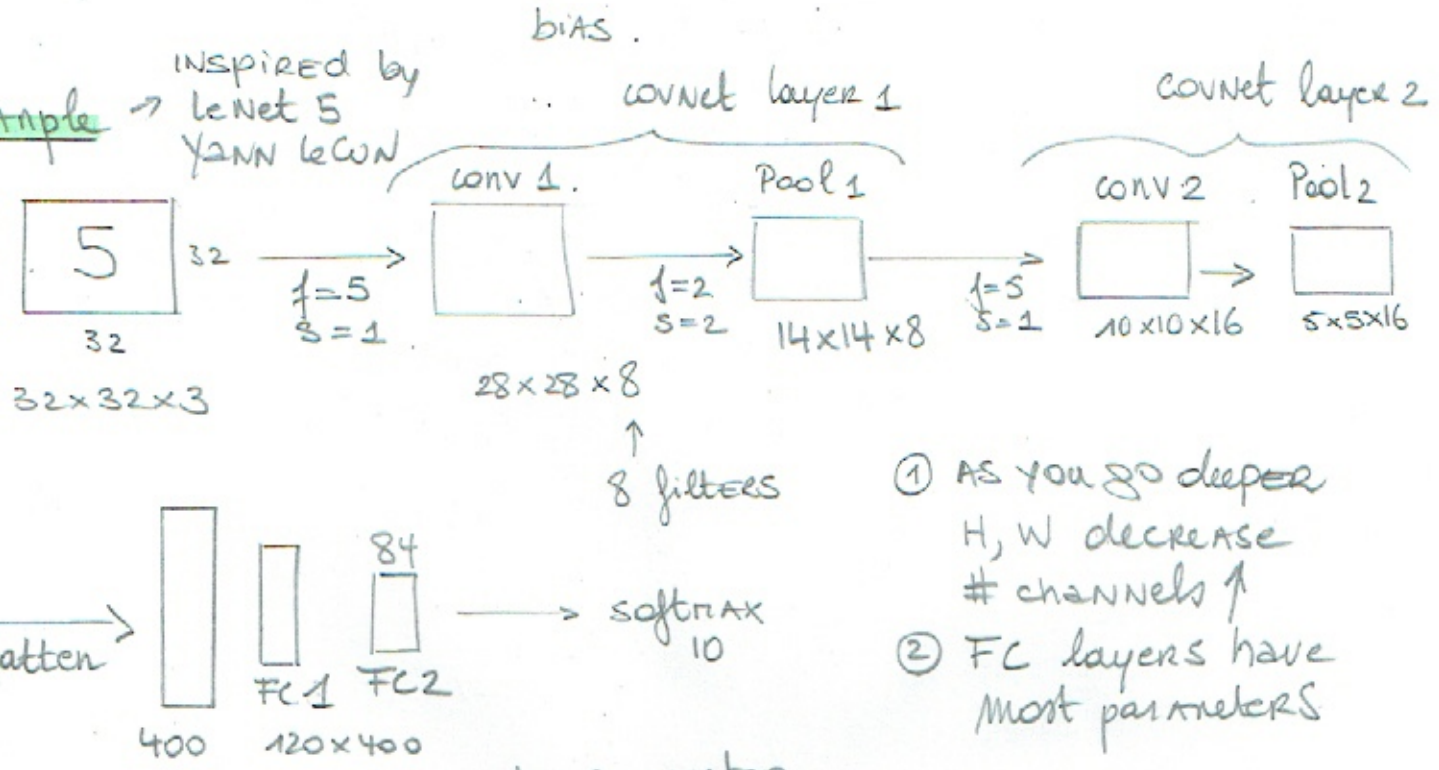
→ (27 + 1) × 10 → 280 parameters

would be 10

↓

bias.

## Example → INSPIRED by LeNet 5 YANN LeCUN




① As you go deeper H, W decrease # channels ↑

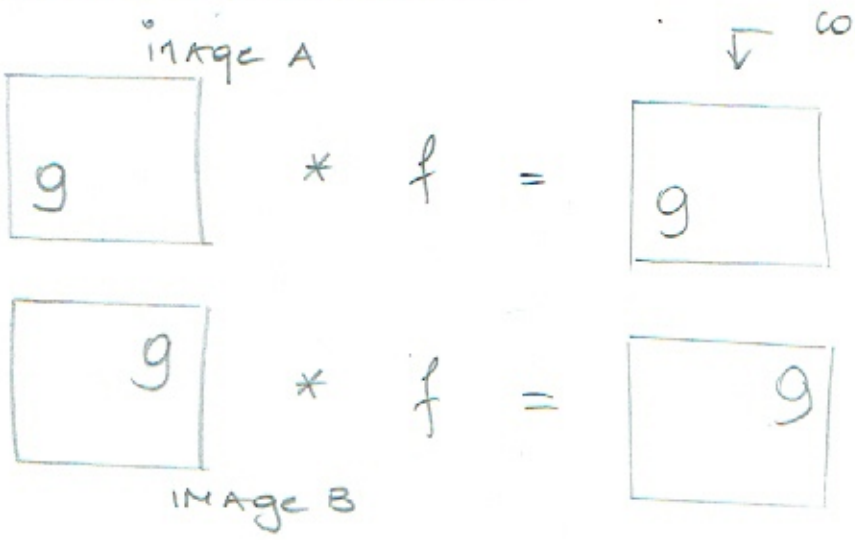
② FC layers have most parameters

hyperparameter.

| | Activation shape | Activation Size | # parameters |
|---|---|---|---|
| INPUT | 32×32×3 | 3072 | 0 |
| CONV 1 | 28×28×8 | 6272 | f=5 → 5×5×3 = 75 (+1) ×8 → 608 |
| Pool 1 | 14×14×8 | 1568 | 0 |
| conv 2 | 10×10×16 | 1600 | f=5 → 1216 |
| pool 2 | 5×5×16 | 400 | 0 |
| FC1 | 120×1 | 120 | 48000 + 1 |
| FC2 | 84×1 | 84 | 10080 + 1 |
| Softmax | 10×1 | 10 | 840 + 1 |

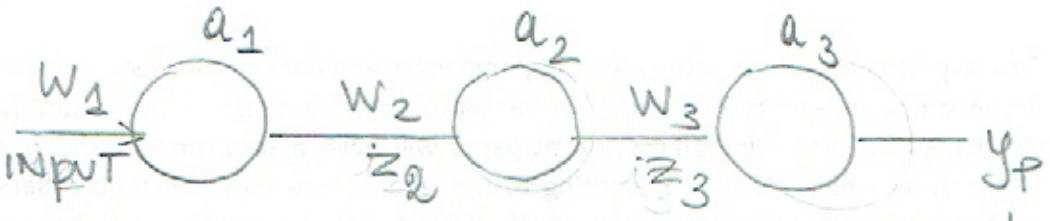
× 60.10³ parameters

## Translation Equivariance




↳ same feature has been detected in both cases despite the fact that 9s are in different locations

# VANISHING GRADIENT

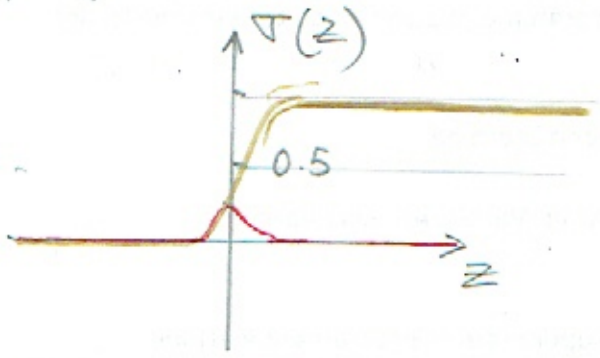



GRADient descent
update RULE

$$C \sim y_p - y$$

$$W_1^{new} = W_1^{old} - \alpha \frac{\partial C}{\partial W_1}^{old}$$

before 2000 → SIGMOID ACTIVATION : $\sigma(z) = \frac{1}{1+e^{-z}}$




$$\sigma'(z) = \sigma(z)\left(1-\sigma(z)\right)$$

↳ between
0 - 0.25

$$\frac{\partial C}{\partial W_1} = \frac{\partial C}{\partial a_3} \cdot \frac{\partial a_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial a_1} \frac{\partial a_1}{\partial W_1}$$

↓ Chain RULE.

multiplication of number between 0 AND 0.25

→ Deeper → SMALLER NUMBER

→ VANISHING GRADIENT !

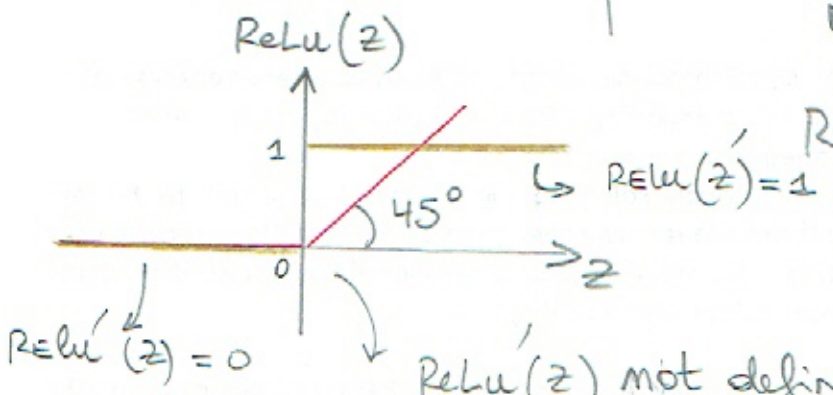
because $W_1^{new} = W_1^{old} - \alpha$ ↑ small NUMBER

↓

$$W_1^{new} \approx W_1^{old}$$

→ algorithm stops

## ReLu → alternative for ∇




MAX(0, z)

$$Relu(z) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases}$$

$Relu'(z) = 1$

$$\frac{\partial C}{\partial w_1}^{old} = \frac{\partial C}{\partial a_3} \frac{\partial a_3}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial w_1}^{old}$$

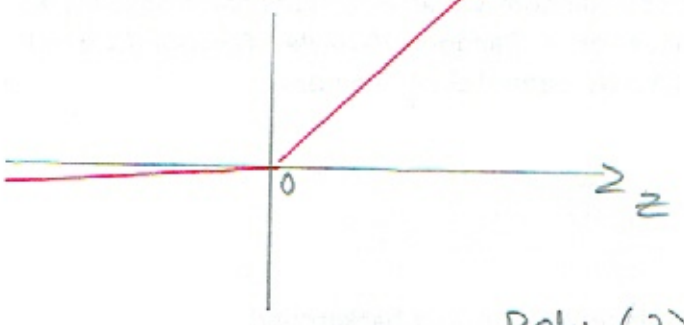
$1 \times 1 \times 1$ for $z > 0$

issue if $Relu'(z)$ is 0 or undefined

$\times 0 \rightarrow 0$

↓

DEAD NEURON !

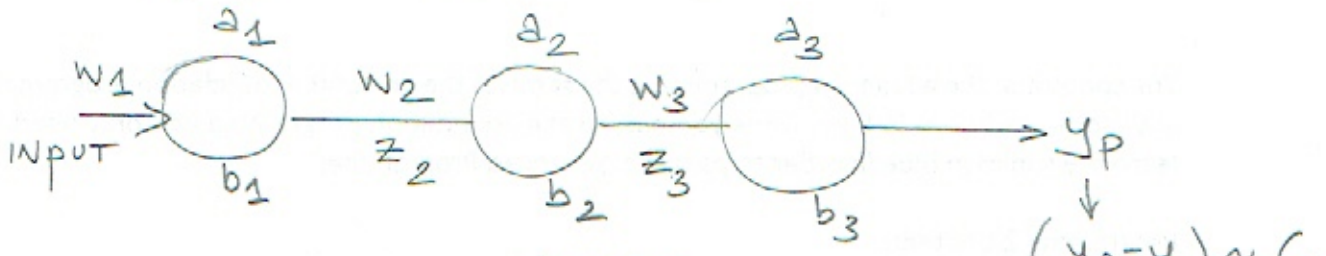



solution

leaky Relu

$$Relu(z) = \begin{cases} z & z > 0 \\ 0.01 z & z < 0 \end{cases}$$

# exploding gradients




$$\frac{\partial C}{\partial w_1}^{old} = \frac{\partial C}{\partial a_3} \boxed{\frac{\partial a_3}{\partial a_2}} \frac{\partial a_2}{\partial a_1} \frac{\partial a_1}{\partial w_1}^{old}.$$

$w_3 a_2 + b_3$ is output $z_3$

$$\rightarrow \frac{\partial a_3}{\partial a_2} = \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial a_2} \rightarrow w_3$$

chain rule

$0 \rightarrow 0.25$

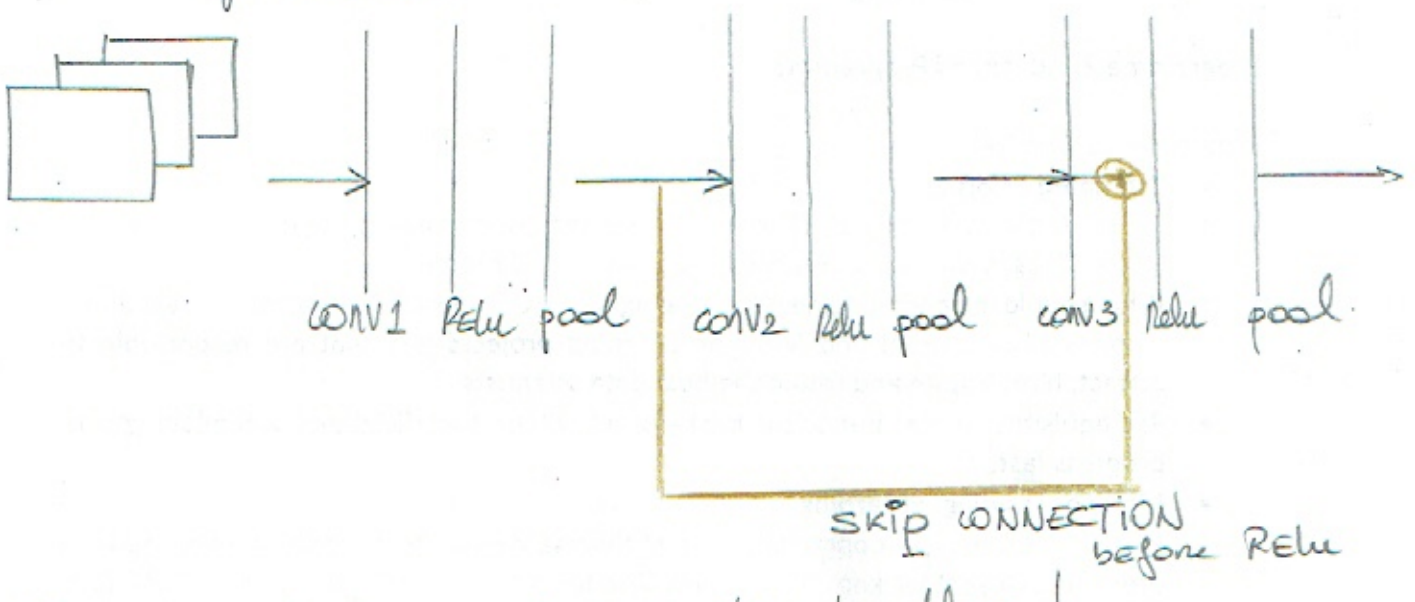
↳ initializeren is important to avoid exploding gradient.

ex: $w_3 = 500$

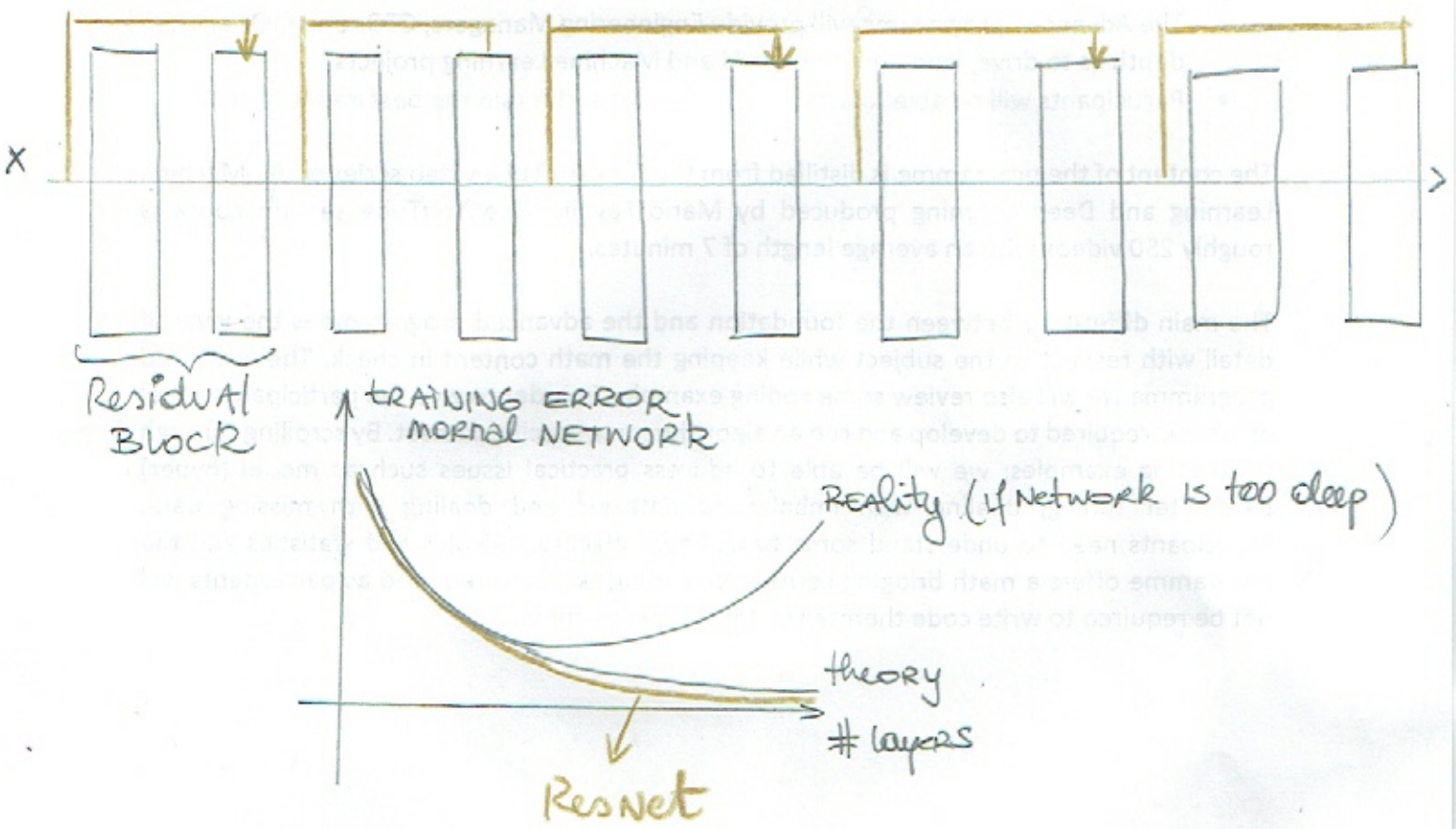
deeper → wil grow

ResNet : 2015 (He et all)
Microsoft
152 layers
↓
Residual Block

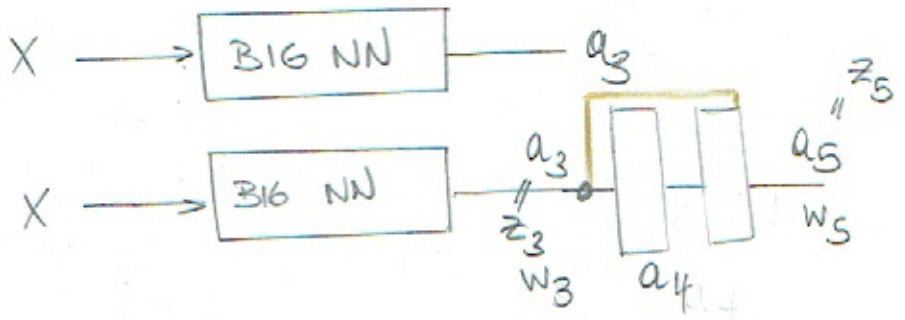
→ ResNet allows you to train deeper networks while avoiding vanishing or exploding gradients




ResNet : → stacks residual blocks together !



## Why do ResNets work well ?




We assume that dimensions are same

$$a_5 = \text{Relu}(z_5 + z_3)$$

$$= \text{Relu}(W_4 a_4 + b_4 + z_3)$$

Make small using weight decay

$$\rightarrow \text{Relu}(\approx 0 + z_3) = z_3$$

$$\rightarrow a_5 \approx a_3$$

↳ identity function is easy to learn for residual block

add $W_s$ MATRIX

if dimensions are different

→ zero padding