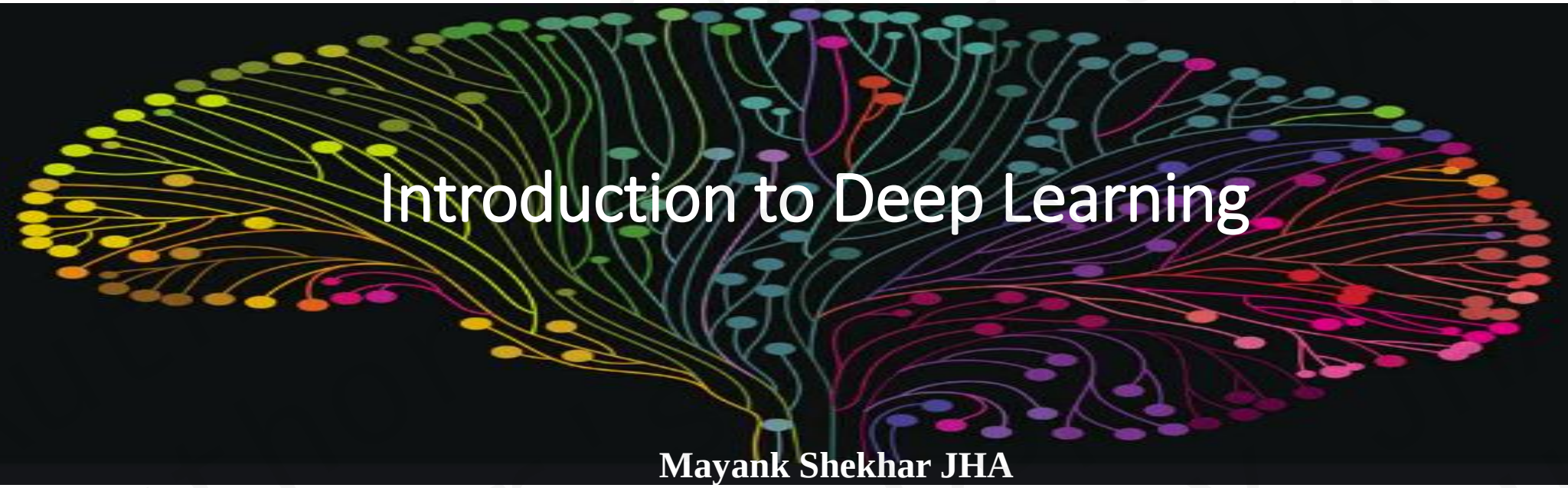


Introduction to Deep Learning

Mayank Shekhar JHA
Associate Professor
(Maitre de Conférences)
Automatic Control, Reliability of Systems
mayank-shekhar.jha{at}univ-lorraine.fr

Research
Centre de Recherche en Automatique de Nancy
(CRAN) UMR 7039,
Faculté des Sciences et Technologies
Boulevard des Aiguillettes - BP 70239 - Bât.
1er cycle 54506 Vandoeuvre-lès-Nancy
Cedex France

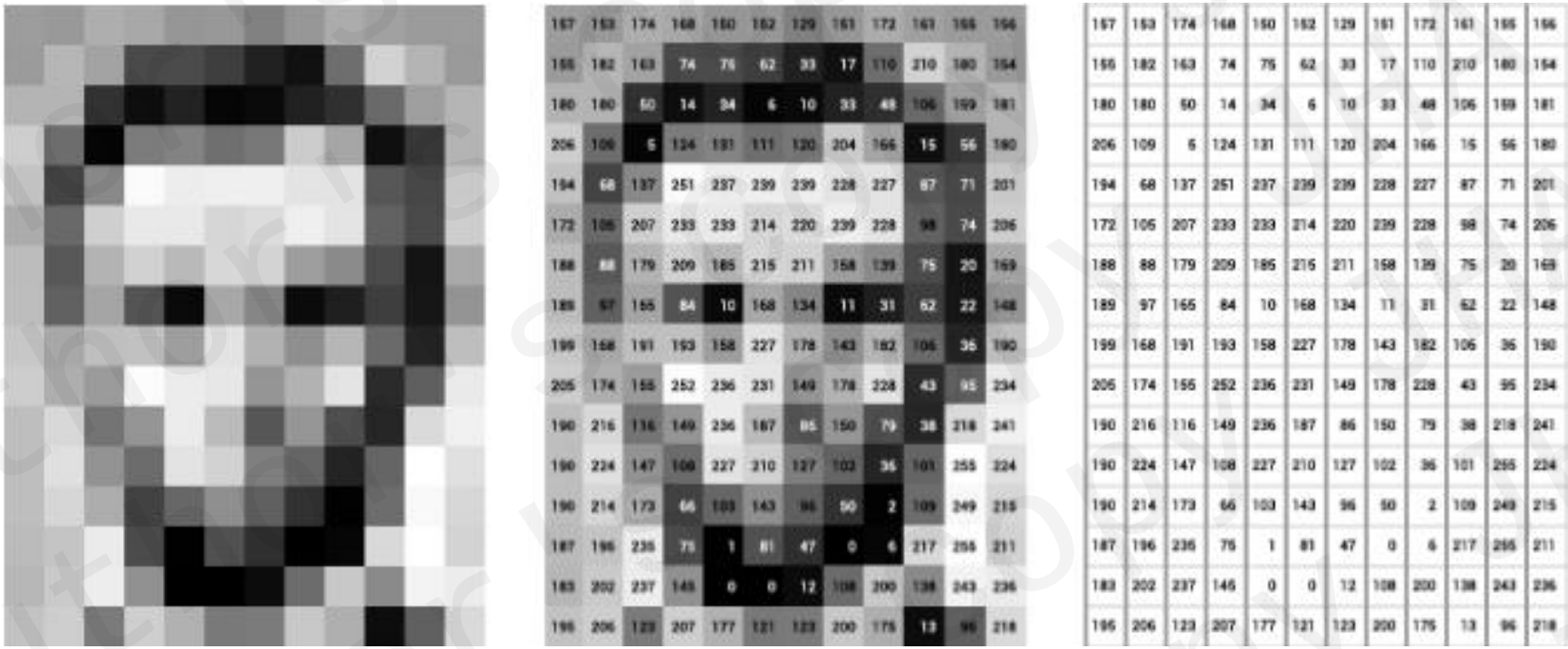
Teaching
Polytech Nancy (ESSTIN),
2 Rue Jean Lamour 54509
Vandoeuvre-lès-Nancy,
Cedex France

Email: mayank-shekhar.jha [at] univ-lorraine.fr

Convolutional Neural Networks

POLYTECH NANCY
UNIVERSITÉ DE LORRAINE
CRAN
JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Images are just numbers for computer!



Source: OpenFrames

Images are matrix of numbers.

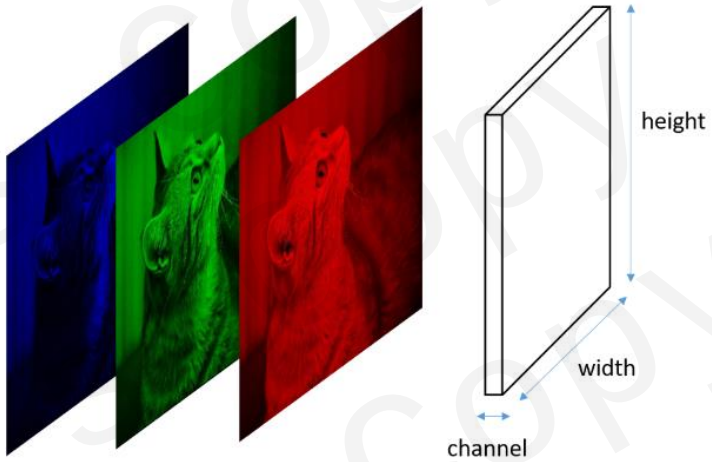
Gray Scale Images ➔ One channel Grey Scale ➔ 2D matrix of numbers (pixel values).

Each pixel =[0,255],

No of pixels proportional to image size → No of rows and columns.

Color images

Source: Broher



Source: Medium

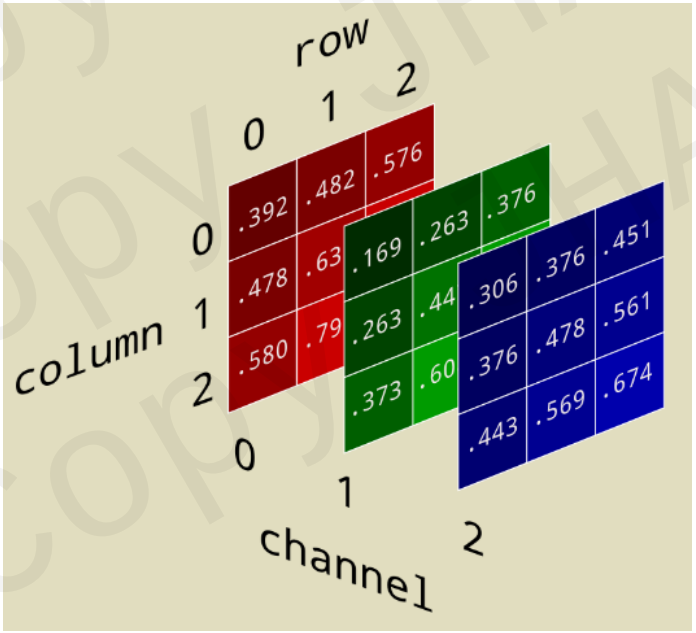
Colored Images: 3 channels of colors: 3D array

RGB channels ➔ 3D Arrays

Red: 2-D matrix

Green: 2-D matrix

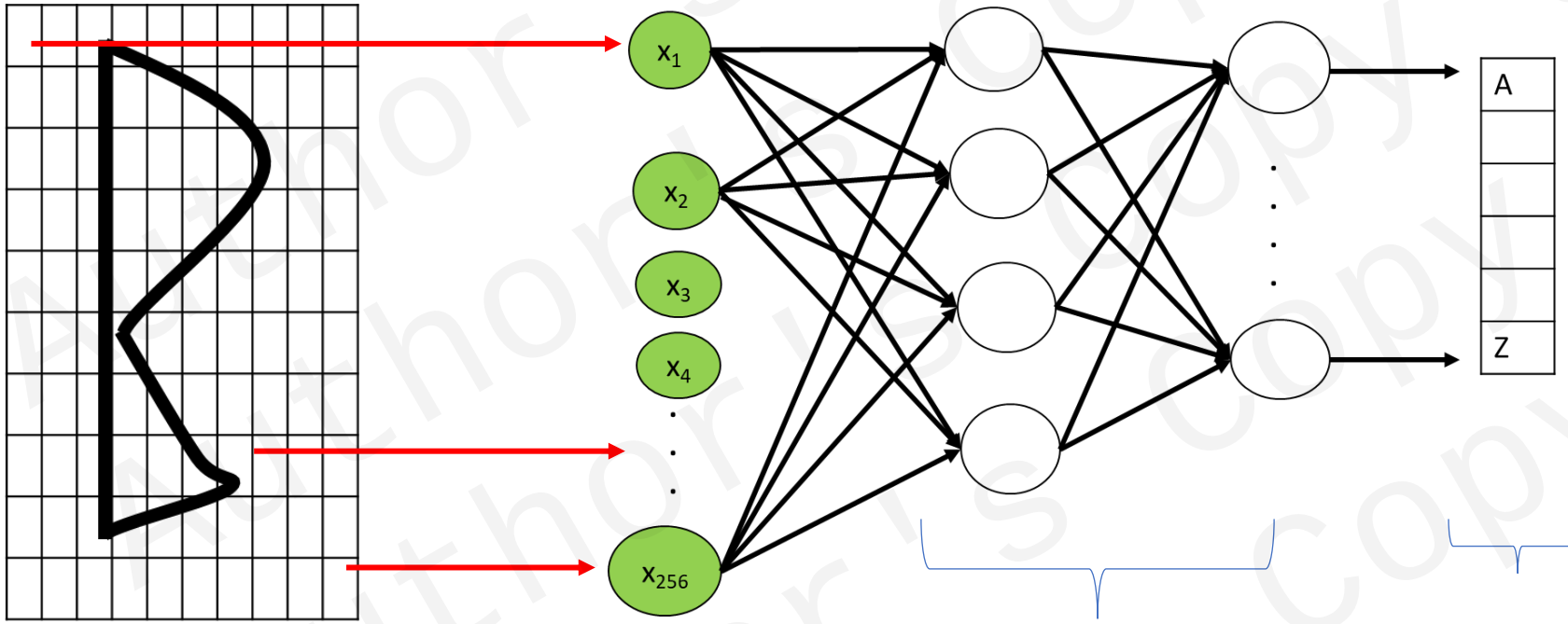
Blue: 2-D matrix



Drawbacks

Drawbacks

Explosion of training parameters



16 X 16 Image GreyScale

Deep Hidden layers

Multi class output

Colored Images: 3 channels of colors: 3D array

RGB channels ➔ 3D Arrays

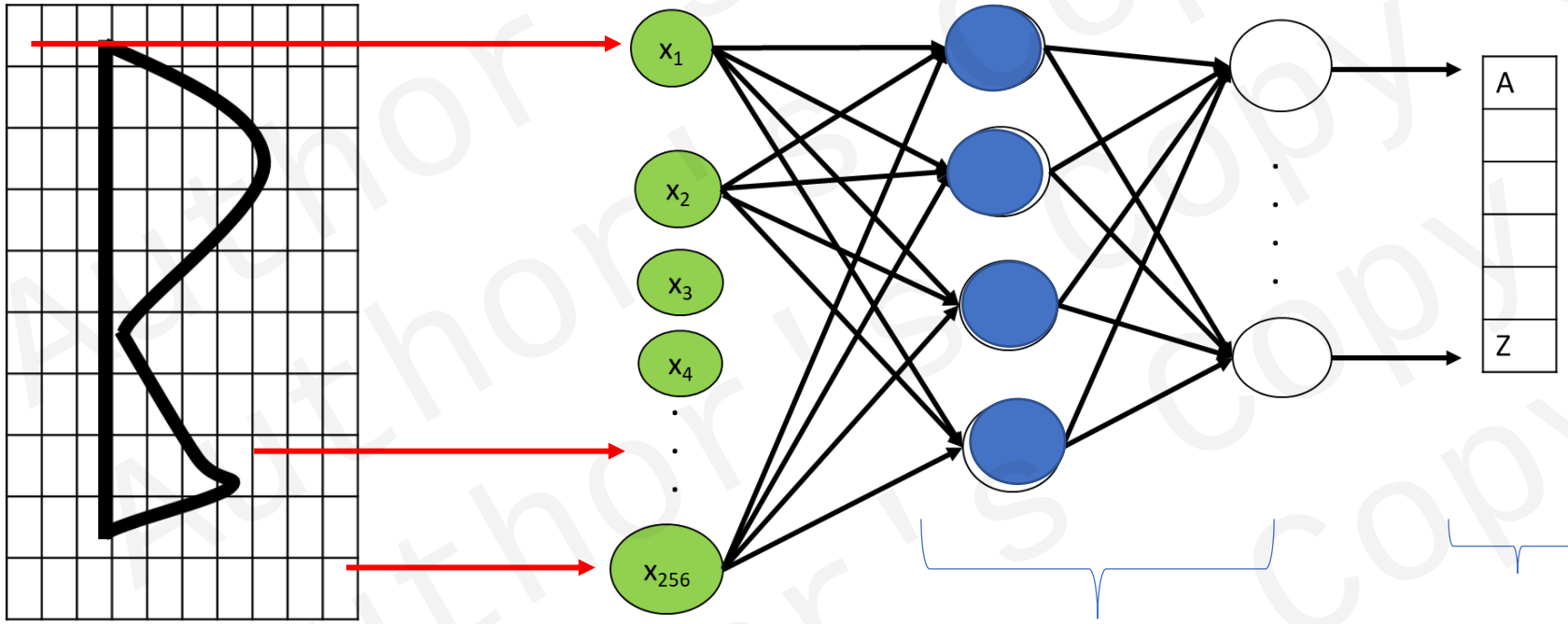
Red: 2-D matrix

Green: 2-D matrix

Blue: 2-D matrix

Drawbacks

Explosion of training parameters



16 X 16 Image GreyScale

Deep Hidden layers

Multi class output

Colored Images: 3 channels of colors: 3D array

RGB channels ➔ 3D Arrays

Red: 2-D matrix

Green: 2-D matrix

Blue: 2-D matrix

Simple calculation for 1 layer , 100 hidden units:

256 inputs → 256 weights

100 hidden units → 256 x 100=25600 input weights

Bias → 100 bias

26 Outputs (A-Z)→ 26 X 100 output weights

Biases → 26

Total: 25600 + 100+ 2600 + 26 = 28326

That is just with one layer !!

Drawbacks: Trainable parameter explosion

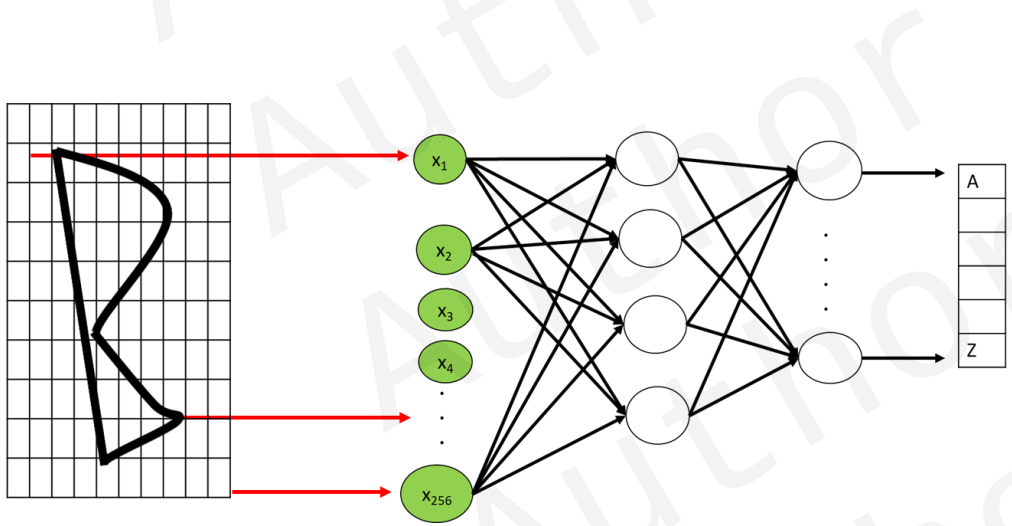
- Most images → high resolution (1MB or more) → several thousands of pixels ➔ several thousands inputs.
- Several hundreds of hidden layers with several hundreds of units.
- Total parameters to train → Extremely large → Computation intractable !!
- Strong regularization needed → difficult and little reproducibility.

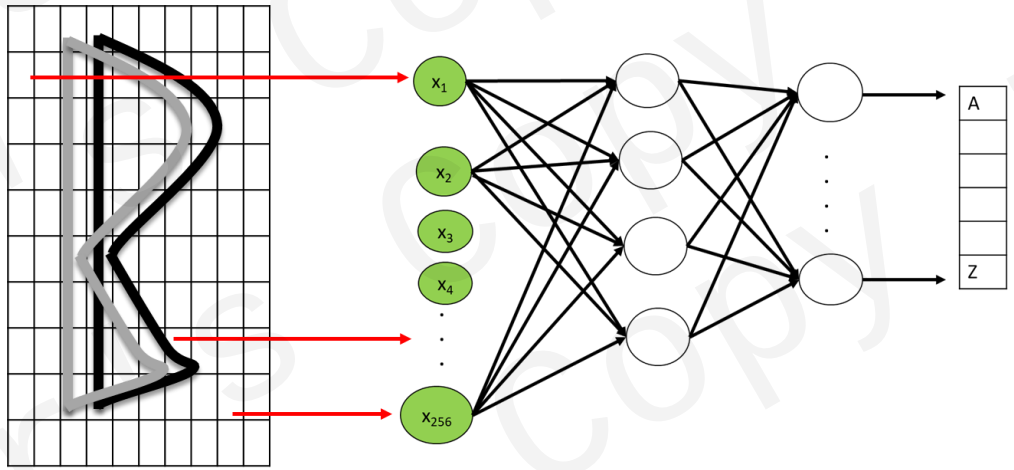
Drawbacks: Variance to distortions

- The orientation / location of object within an image should have little influence over it getting detected.

  This is not true with previous NNs (MLPs, ANNs).

- Variance to scaling, shifting and other distortions, influence of surroundings (global context).
- The topology of the data is ignored.
- Inherent distributions are not learnt well.





- Must avoid parameter explosion in face of large inputs.
- Identification of object should be invariant to scaling, shifting and different orientation.

- The object should be identifiable in any location / orientation ➔ placement of object in an image should not influence the outcome, only local information about the object should be sufficient.

Convolutional neural networks

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Motivation

Convolutional layer: Motivation

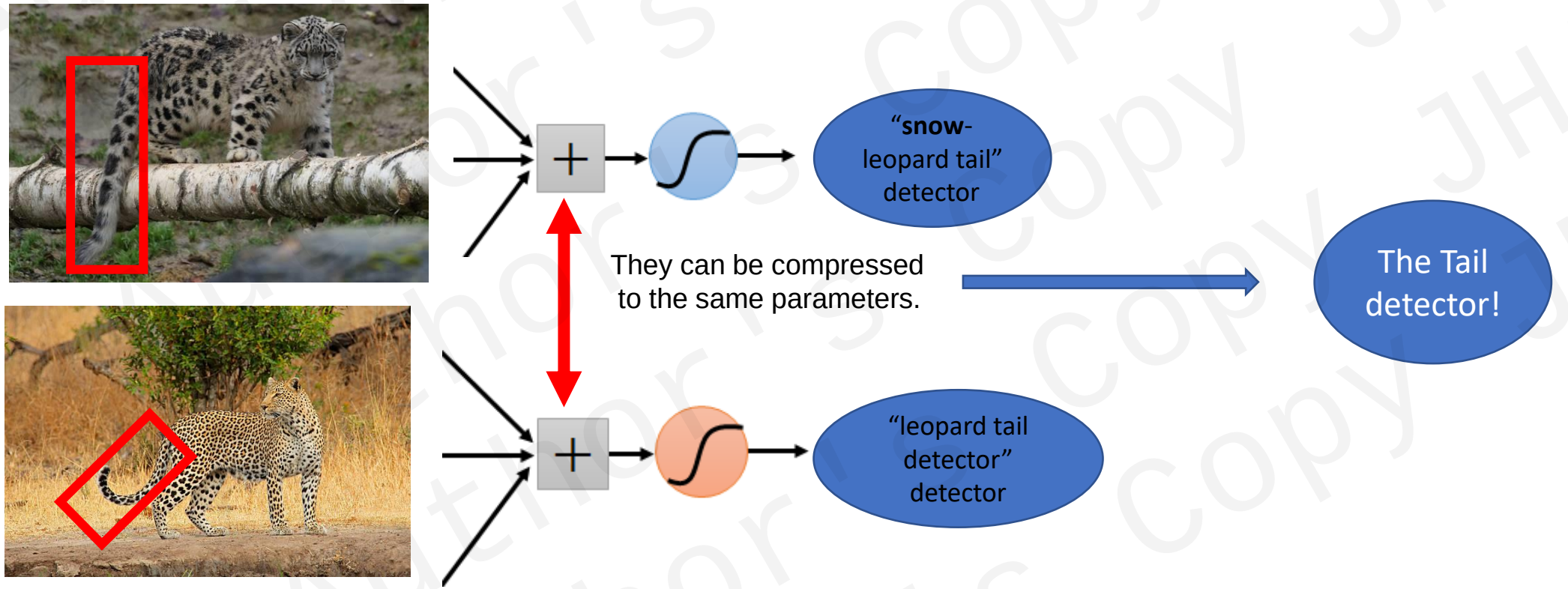
Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.

Convolutional layer: Motivation

Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.



Convolutional layer: Motivation

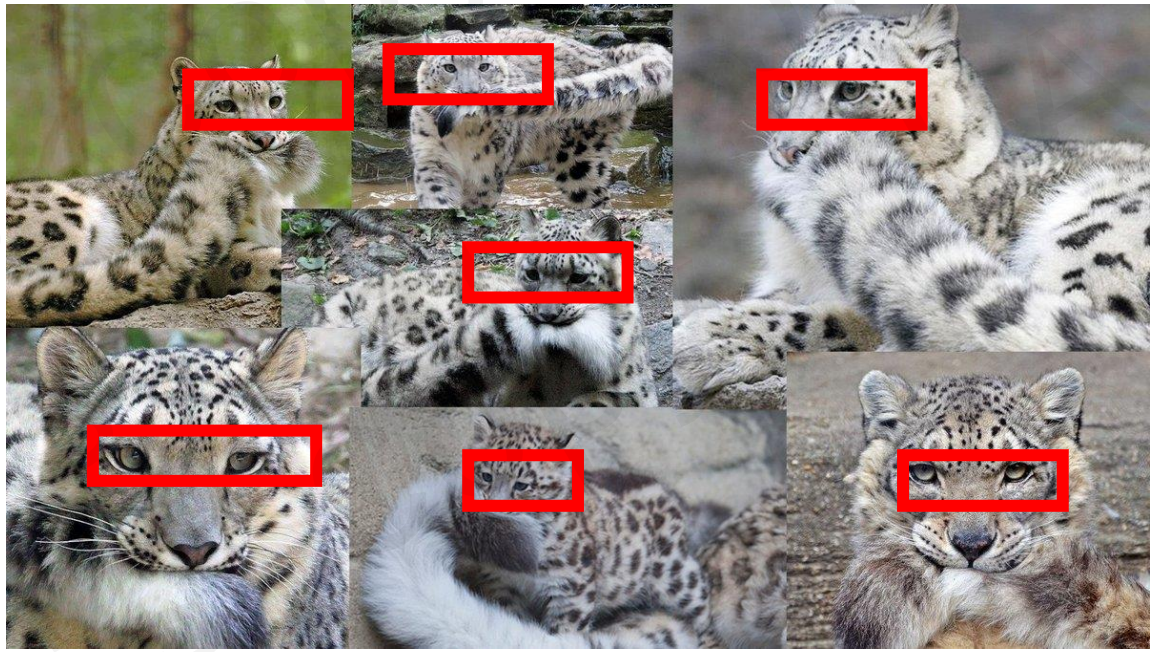
Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.

Intuition:

- Understanding the inherent data distribution.
- Using local information to extract topological properties from image.
- Implicitly extract relevant features.

Understanding the new data using learnt attributes.



Convolutional layer: Motivation

Proposed by Yann LeCun and Yoshua Bengio in 1995.

- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- Inspired by neuro-biology: brain's mechanism of understanding different attributes of an object
  - Attributes : shape, size, orientation and color.

Intuition:

- Understanding the inherent data distribution.
- Using local information to extract topological properties from image.
- Implicitly extract relevant features.

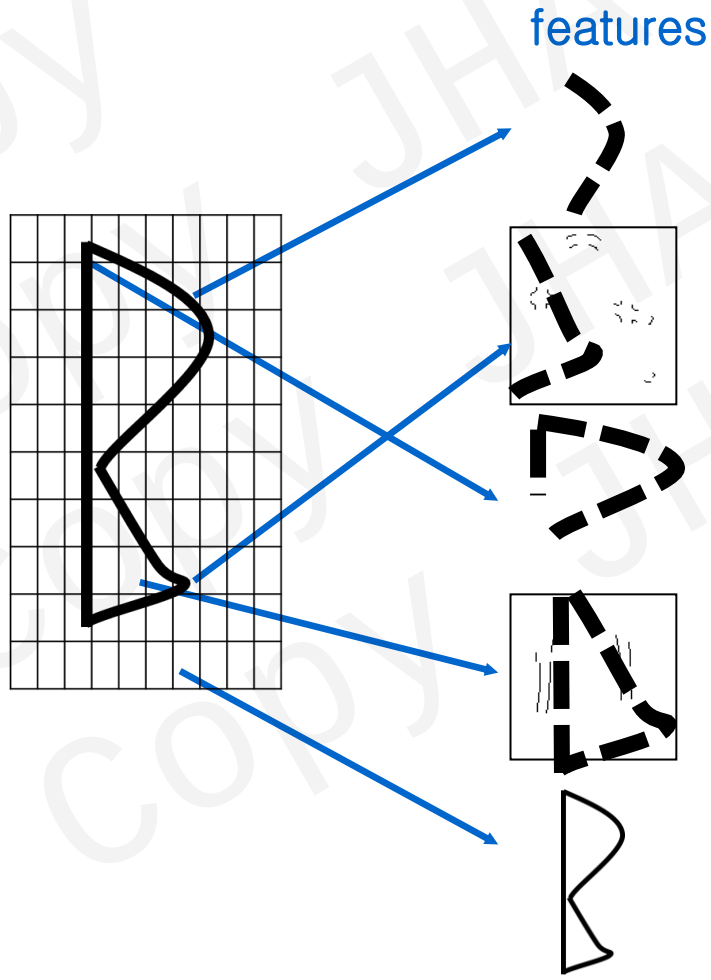
Understanding the new data using learnt attributes.

Example:

A door is always rectangular in shape,

A ship has a characteristic shape,

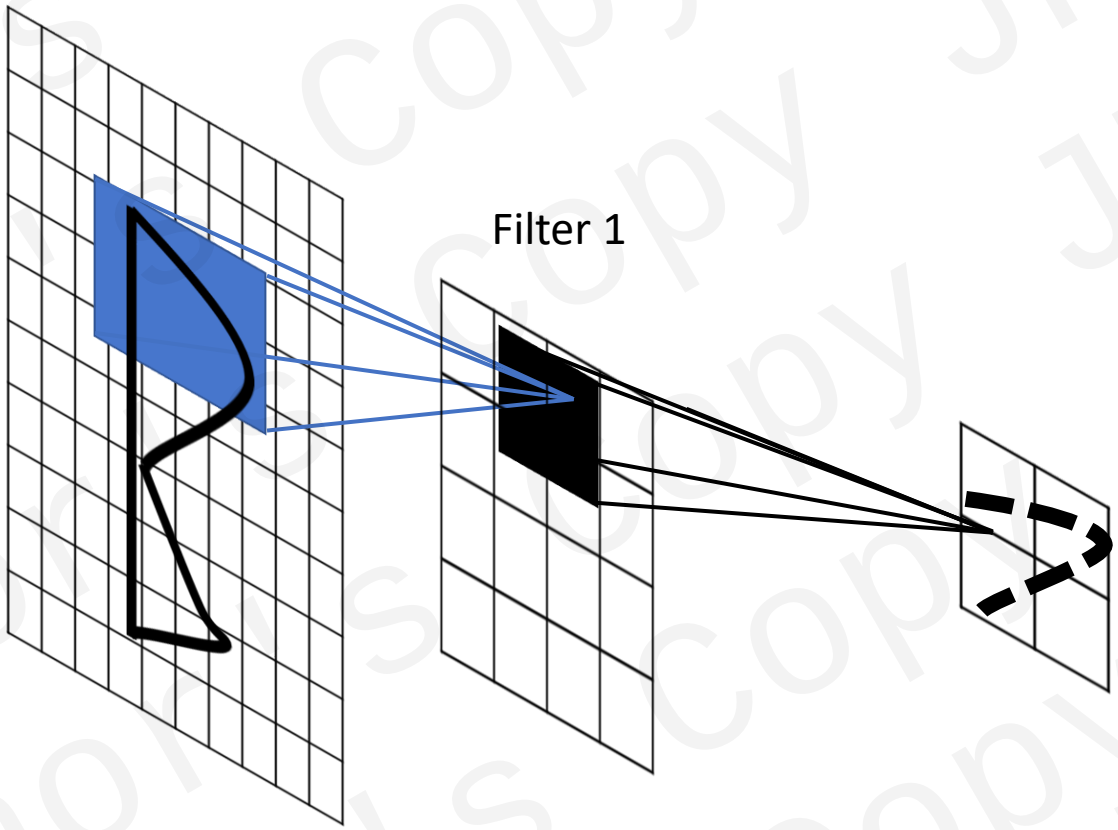
a car of any brand shall have a typical shape….



Intuition

Intuition

Shape 1 / feature 1



Intuition:

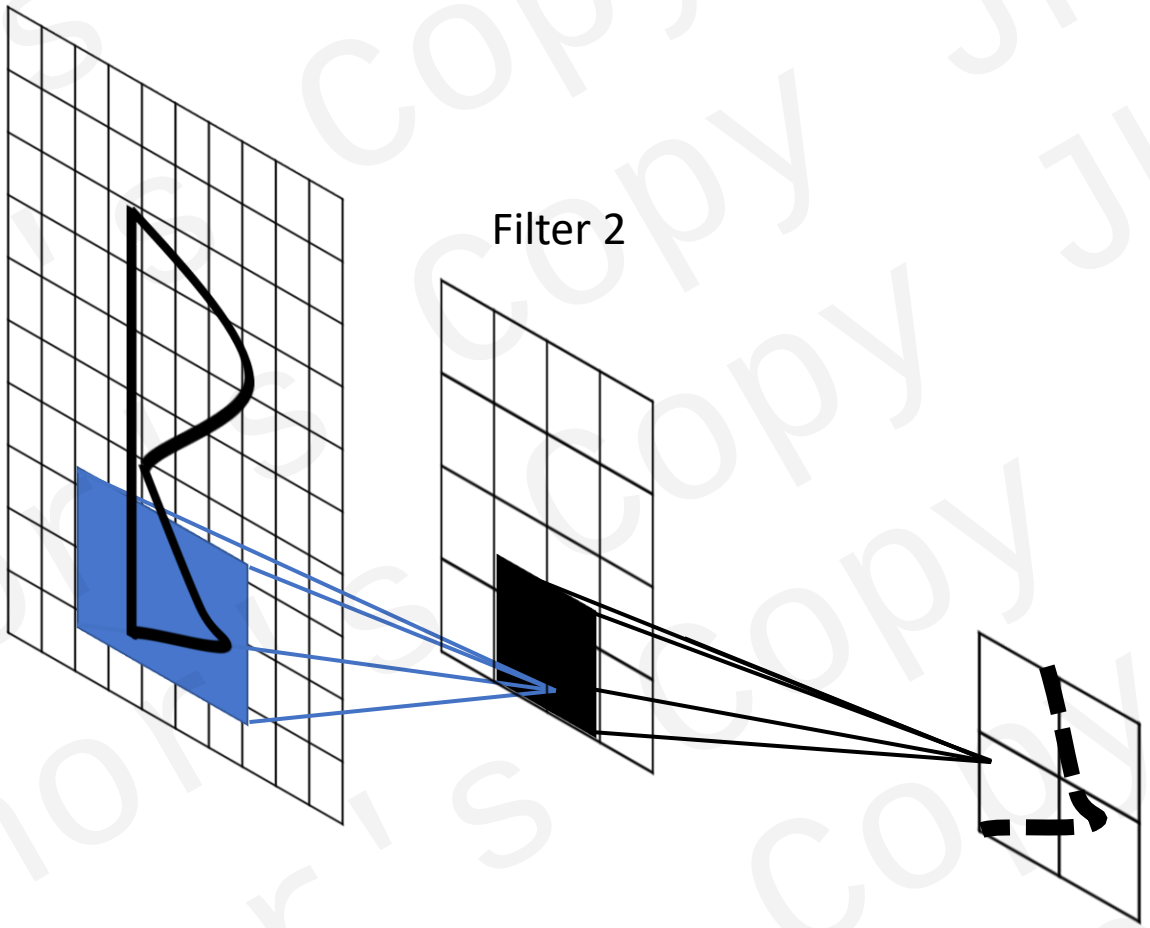
Use an image kernel to extract relevant features from the image.

Image kernel = image matrix.

Learn an appropriate filter weights through successive training (BP).

Intuition

Shape 2 / feature 2



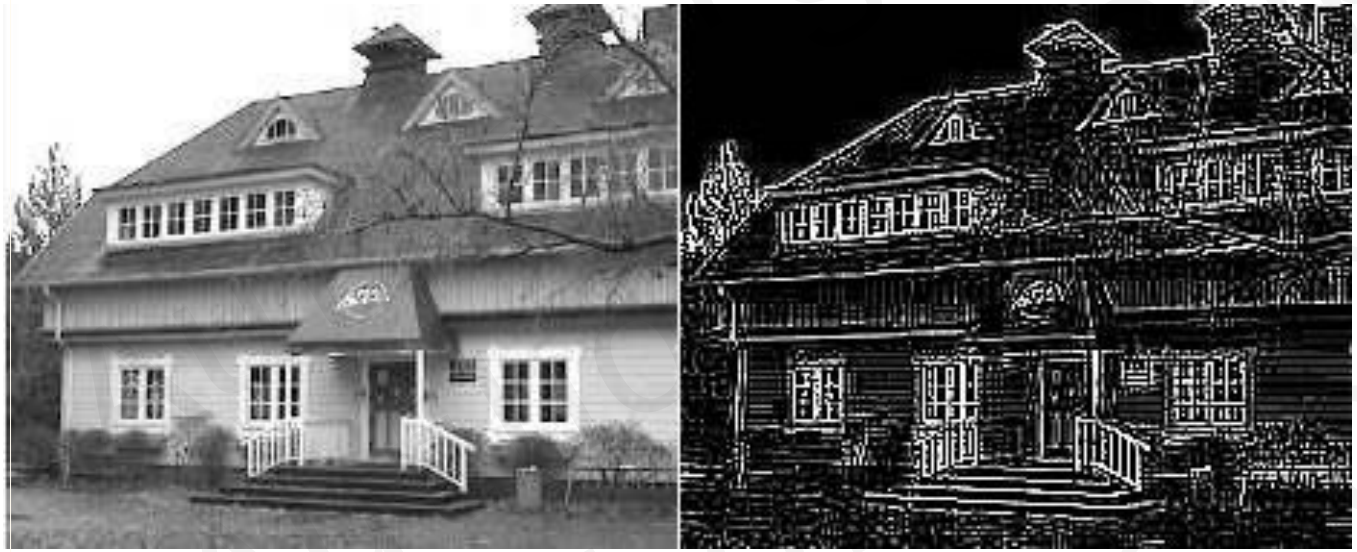
Intuition:

Use an image kernel to extract relevant features from the image.

Image kernel = image matrix.

Learn an appropriate filter weights through successive training (BP).

Intuition

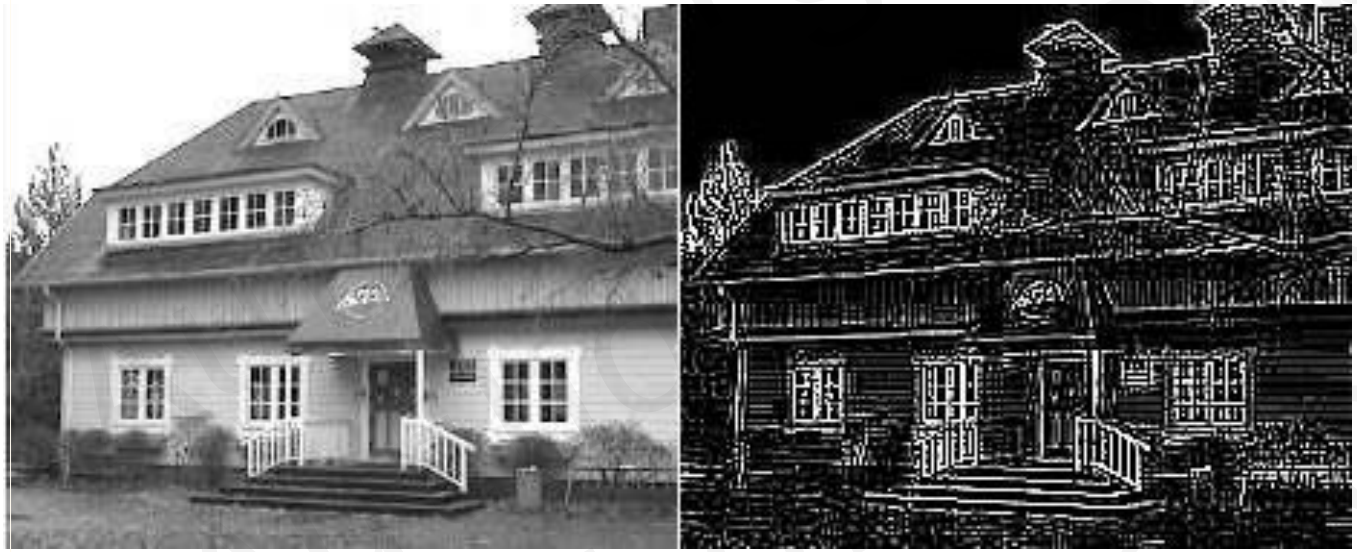


Source :Blog AndrewSzot

Intuition:

Edge Detection: An image kernel for edge detection.

Intuition



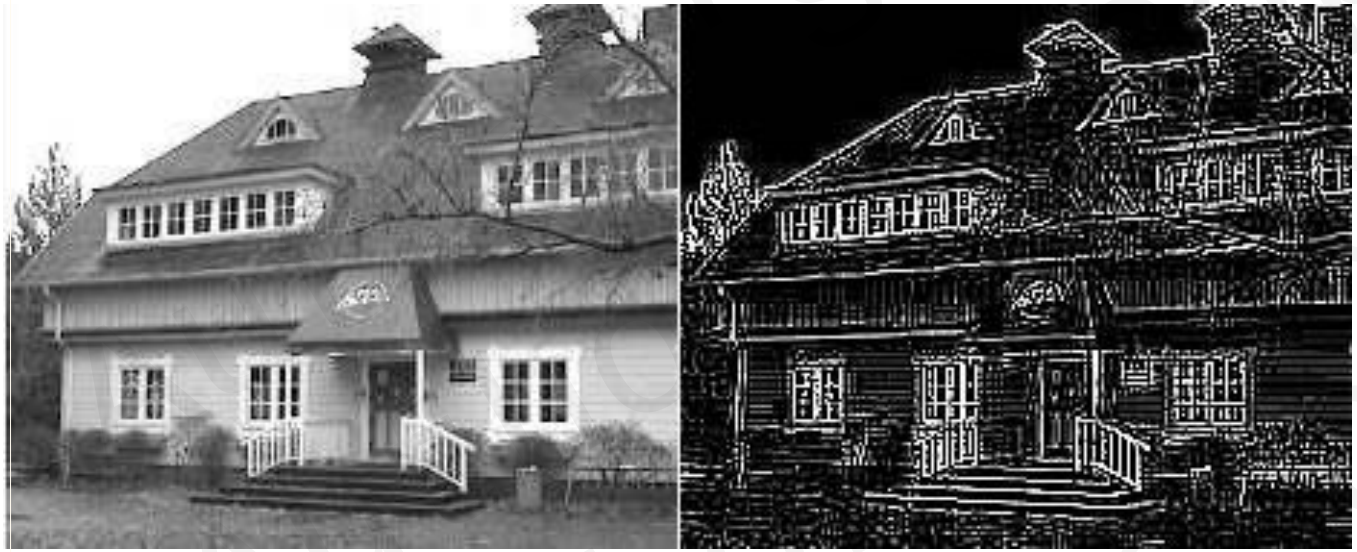
Source :Blog AndrewSzot

Intuition:

Edge Detection: image kernel for edge detection.

Different image kernels to extract different features.

Intuition



Source :Blog AndrewSzot

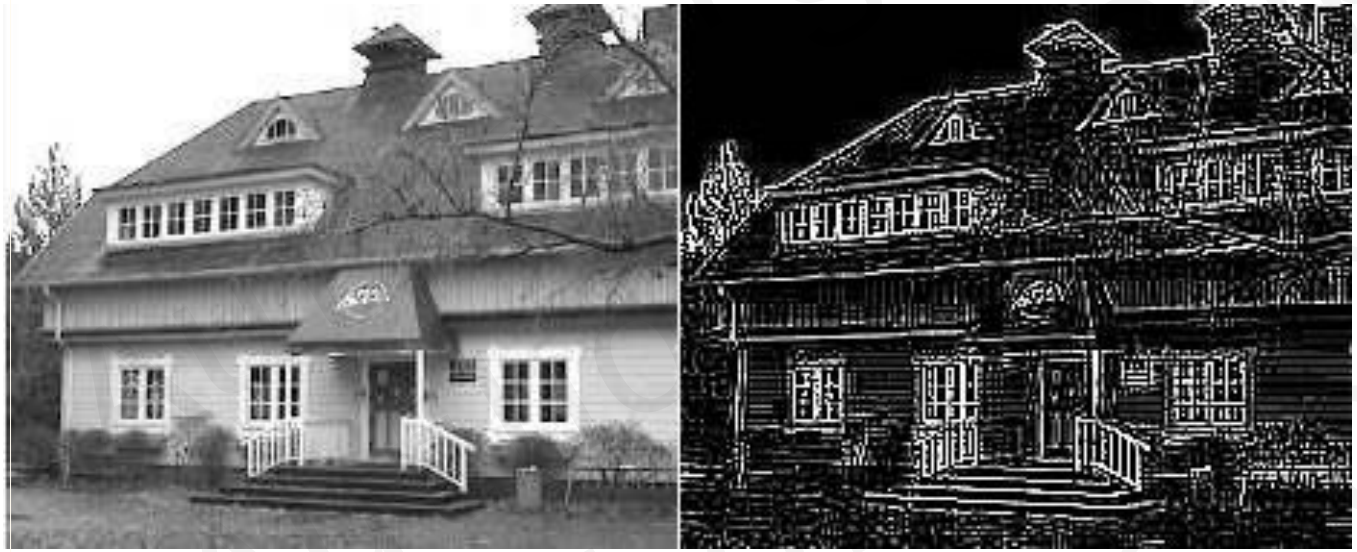
Intuition:

Edge Detection: image kernel for edge detection.

Multiple image kernels to extract different features.

Why not multiple kernels to extract set of features expected from object / required for the objective.

Intuition



Source :Blog AndrewSzot

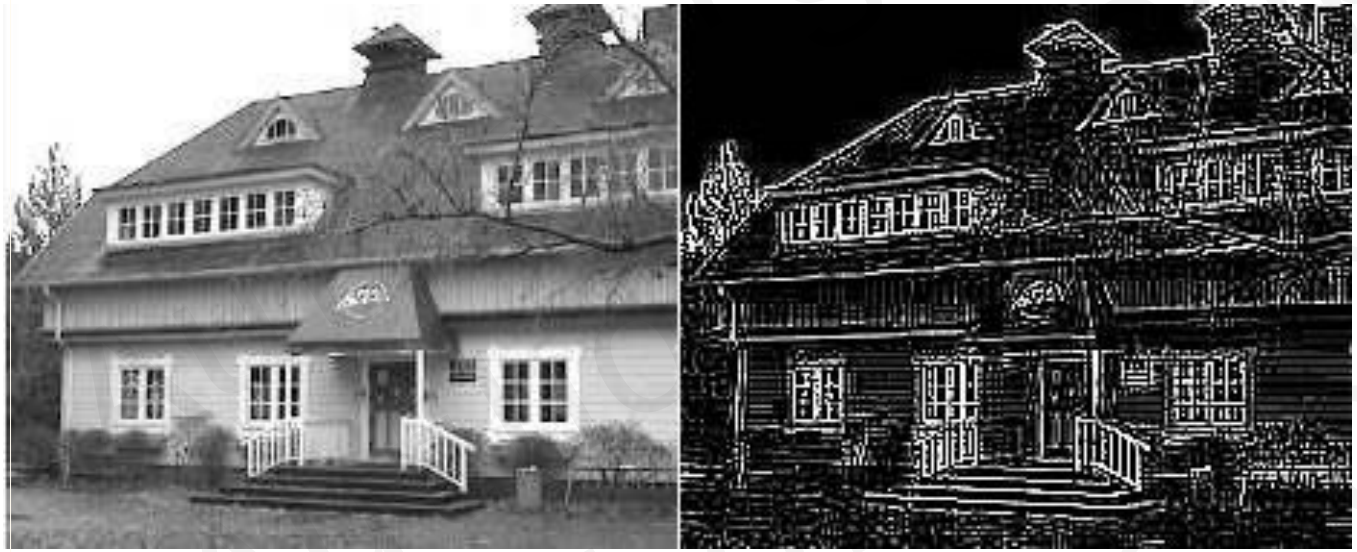
Intuition:

How to construct these filters? Edge detection is straight foreword. Not obvious in general.

Essence of CNN :

- learn the values (weights) of these filters (BP).
- stack multiple layers of feature detectors (kernels) on top of each other for abstracted levels of feature detection.

Intuition



Source :Blog AndrewSzot

Intuition:

How to construct these filters? Edge detection is straight foreword. Not obvious in general.

Essence of CNN :

- learn the values (weights) of these filters (BP).
- stack multiple layers of feature detectors (kernels) on top of each other for abstracted levels of feature detection.
- extract relevant features: Convolution operation . What and How?

Convolution Operator

Convolution Operator

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

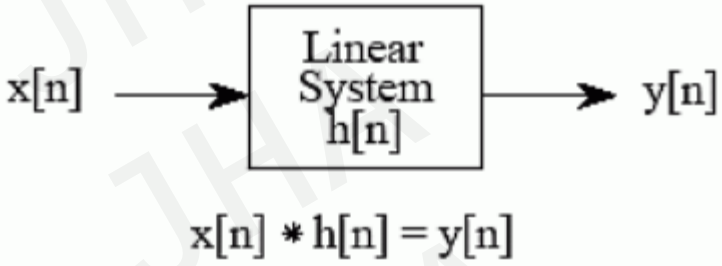
Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- In signal processing, input signal convolution with impulse response of the system → output response

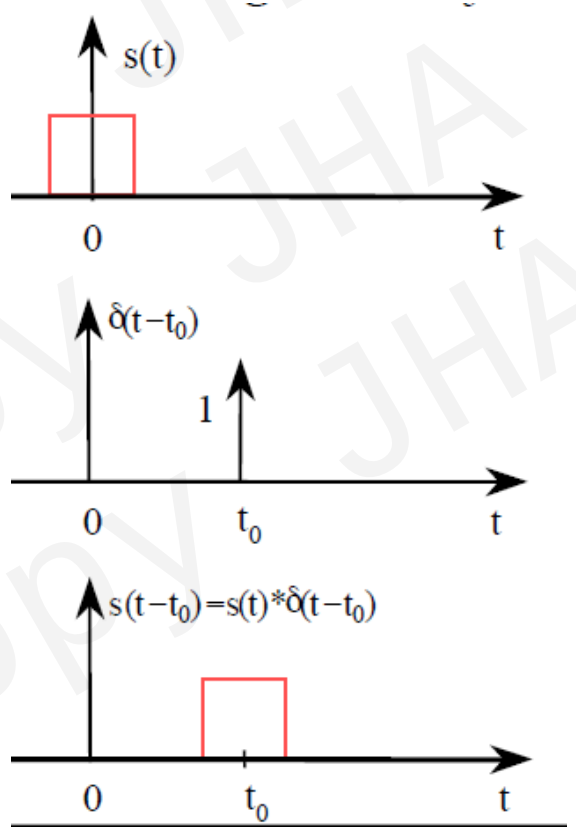
Convolution Operator

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

$$s(t) * \delta(t-t_0) = s(t-t_0)$$



Source: DSP Guide book



Source: DSP Course by Prof. Garnier, Polytech Nancy

Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- input signal * impulse response of the system ➔ output response.

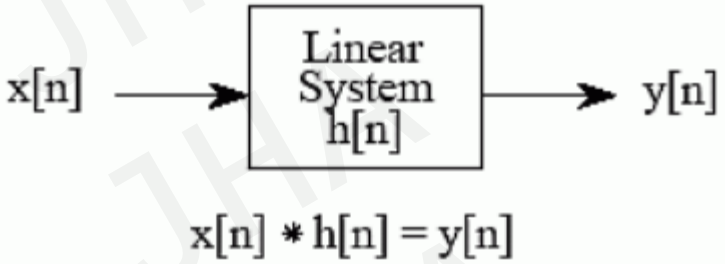
Convolution of a signal by Dirac impulse positioned at t_0 ➔ signal shift to t_0.

Convolution Operator

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

$$s(t) * \delta(t - t_0) = s(t - t_0)$$

$$s(t) * \delta_{Te}(t) = \sum_{k=-\infty}^{+\infty} s(t - kTe)$$

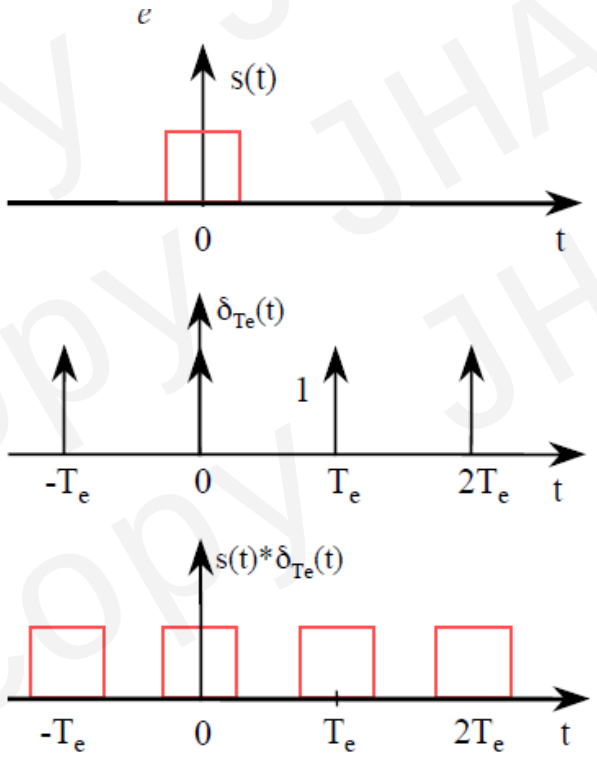


Source: DSP Guide book

Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- input signal * impulse response of the system ➔ output response.

Convolution of a signal by Dirac train → periodic signal with period T_e



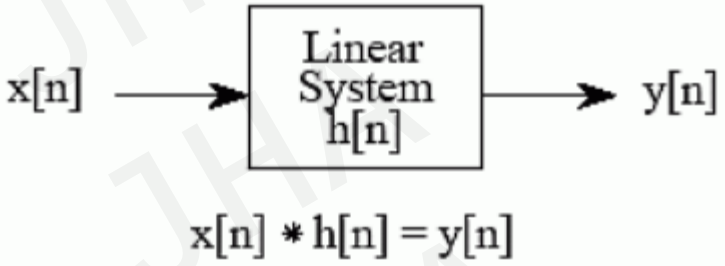
Source: DSP Course by Prof. Garnier, Polytech Nancy

Convolution Operator

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

$$s(t) * \delta(t - t_0) = s(t - t_0)$$

$$s(t) * \delta_{Te}(t) = \sum_{k=-\infty}^{+\infty} s(t - kTe)$$



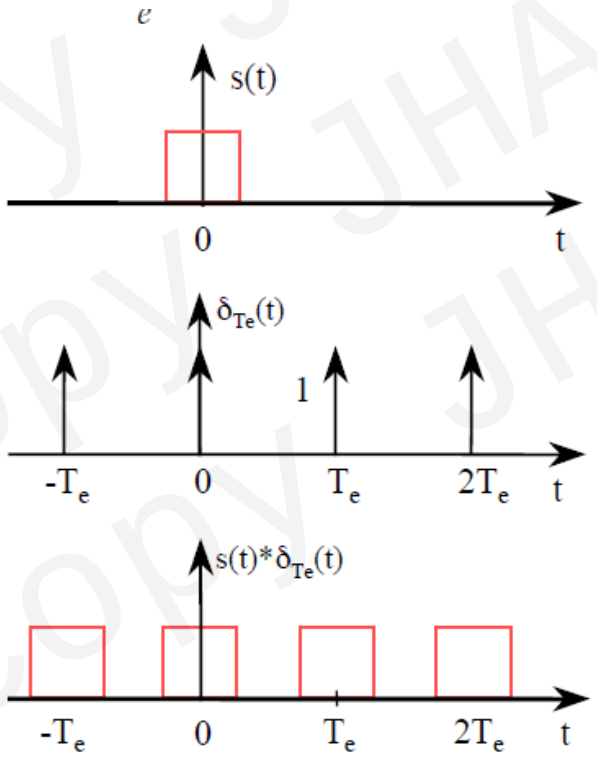
Source: DSP Guide book

Reminders:

- origins in Signal Processing.
- convolution of two signals produces a third signal
- input signal * impulse response of the system ➔ output response.

Convolution of a signal by Dirac train → periodic signal with period T_e

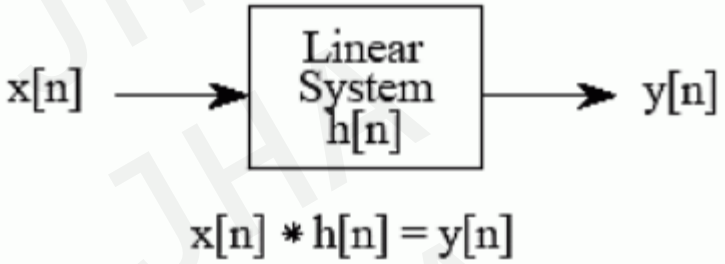
- Convolution operation constructs a system response signal.
- Convolution operation fundamental in assessing the similarity between two signals.



Source: DSP Course by Prof. Garnier, Polytech Nancy

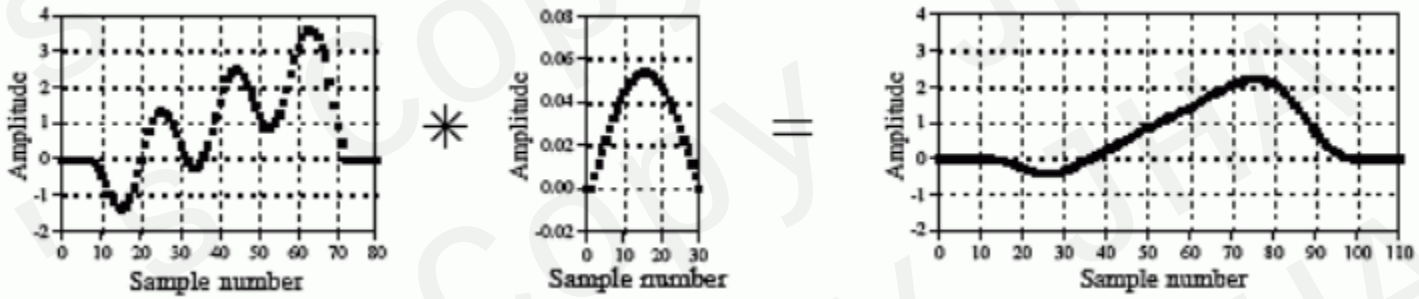
Convolution Operator

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



$$S(t) = (x * w) \times (t) = \sum_{a=-\infty}^{\infty} w(a)x(t-a)$$

Source: DSP Guide book



Reminders:

Low pass filtering:

Input: three cycles of sine wave plus a slow increasing ramp.

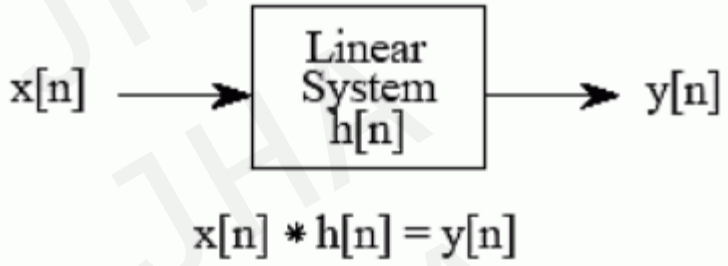
Low pass filter impulse response (or Convolution kernel / filter kernel)

Output = slow component ramp.

Convolution operation → extracts the weighted feature.

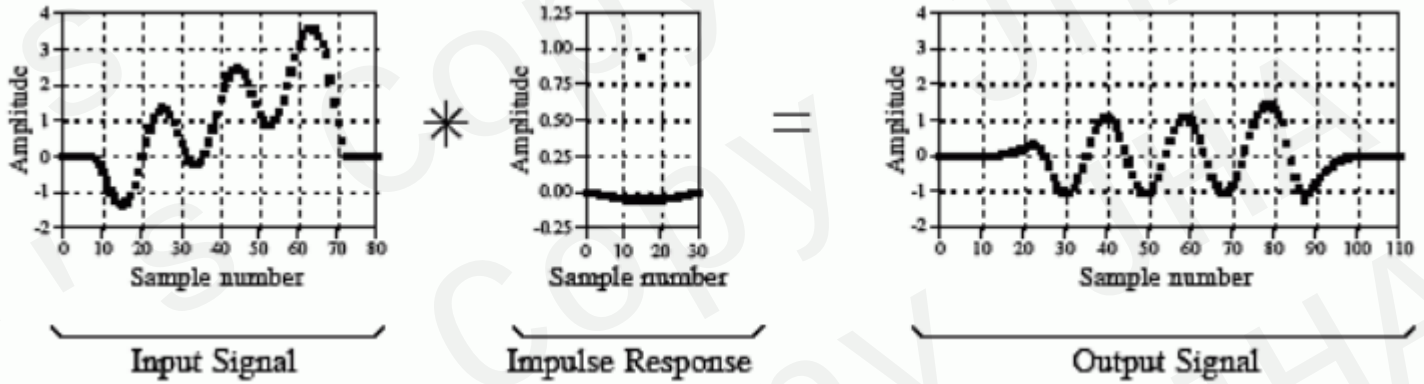
Convolution Operator

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



$$S(t) = (x * w) \times (t) = \sum_{a=-\infty}^{\infty} w(a)x(t-a)$$

Source: DSP Guide book



Reminders:

High pass filtering:

Input: three cycles of sine wave plus a slow increasing ramp.

High pass filter impulse response (or Convolution kernel / filter kernel)

Output = Fast component ramp.

Convolution operation → extracts the weighted feature.

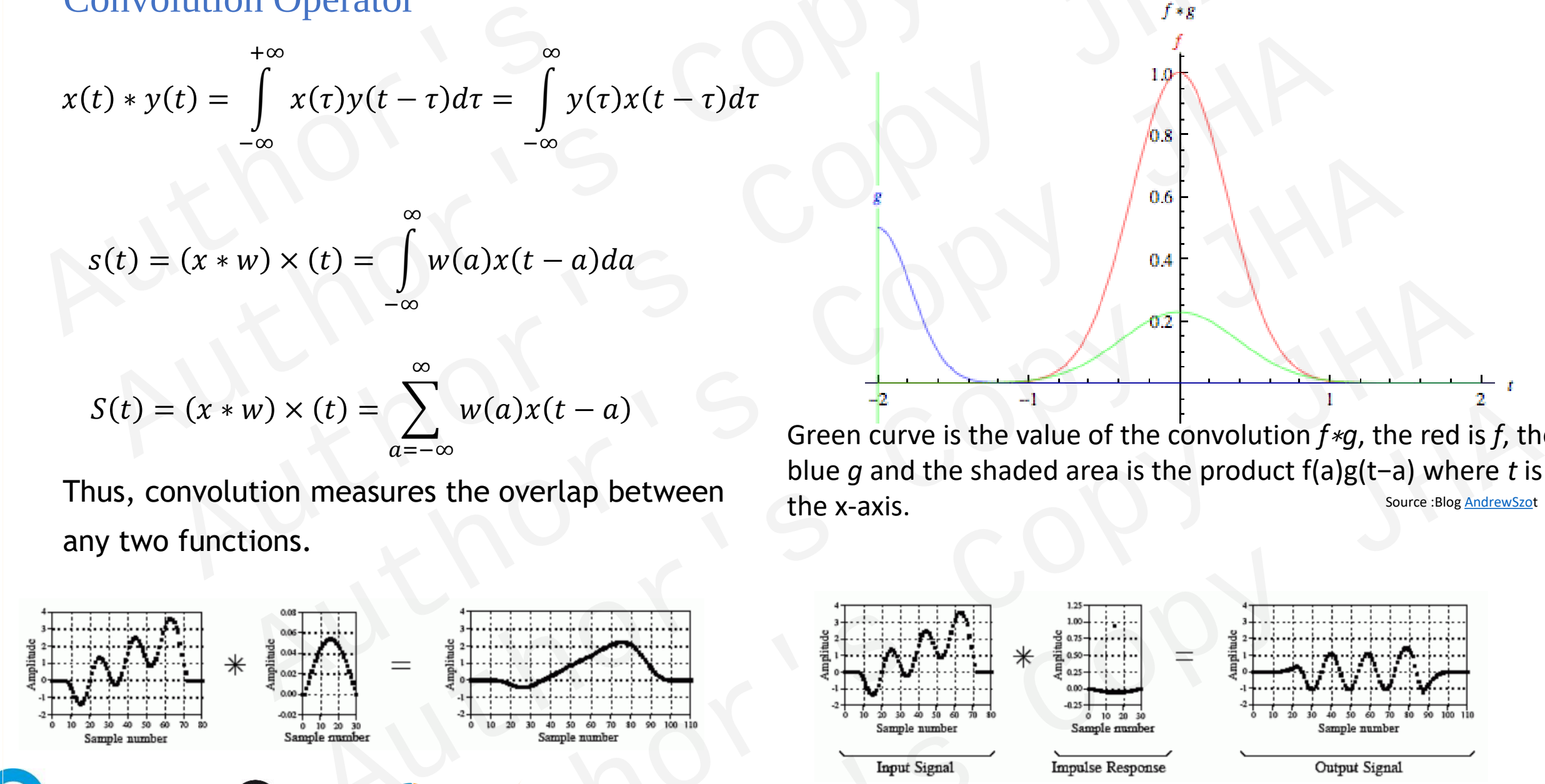
Convolution Operator

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

$$s(t) = (x * w) \times (t) = \int_{-\infty}^{\infty} w(a)x(t-a)da$$

$$S(t) = (x * w) \times (t) = \sum_{a=-\infty}^{\infty} w(a)x(t-a)$$

Thus, convolution measures the overlap between any two functions.

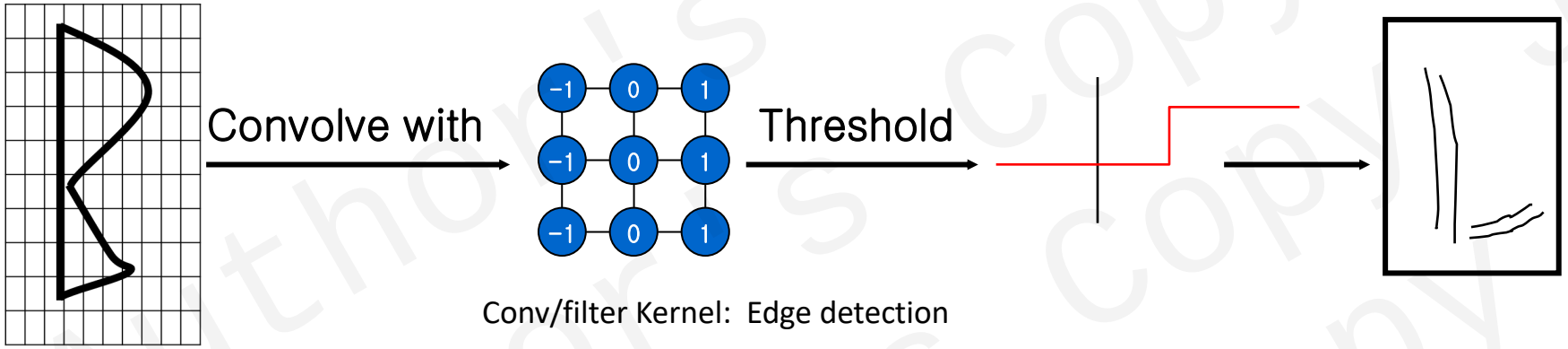


Green curve is the value of the convolution $f*g$, the red is f, the blue g and the shaded area is the product f(a)g(t−a) where t is the x-axis.

Source :Blog AndrewSzot

Convolution

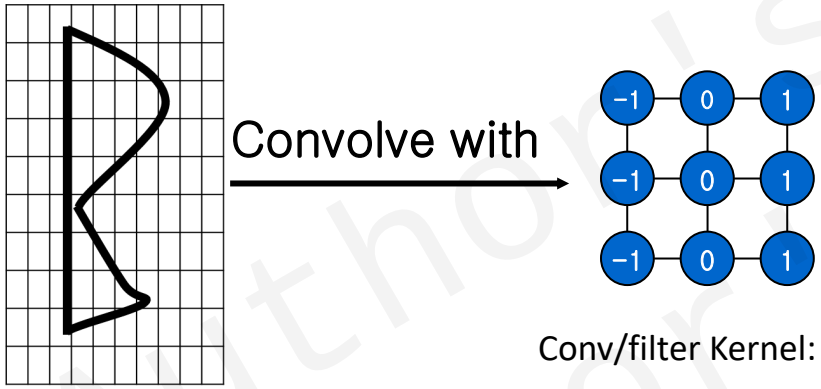
Convolution: CNN context.



Conv/filter Kernel: Edge detection

Back to CNNs:

Convolution Operator : CNN context.



Conv/filter Kernel: Edge detection

Back to CNNs:

Images can be represented as 2D array.

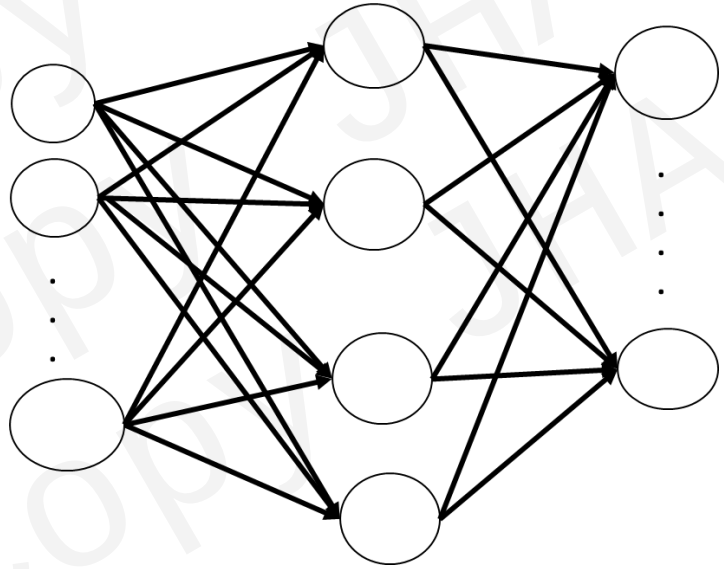
Consider (i,j) → any position in an image.

Consider Hidden layers as 2-D array ,

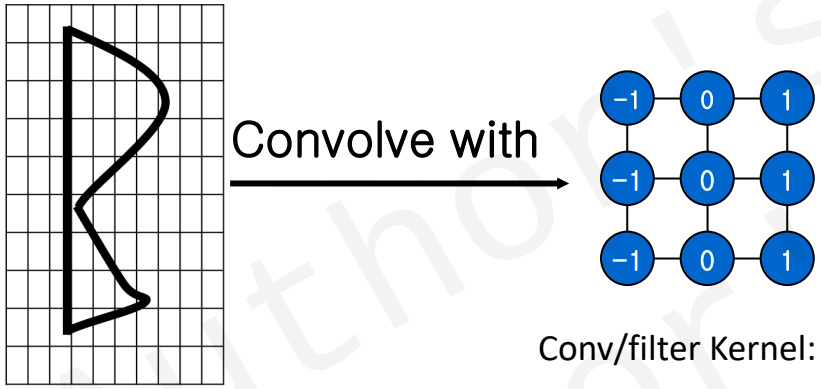
Then, dense layers → 4D tensors (Weights in a hidden layer X no of layers)

Weights matrices become weight tensors

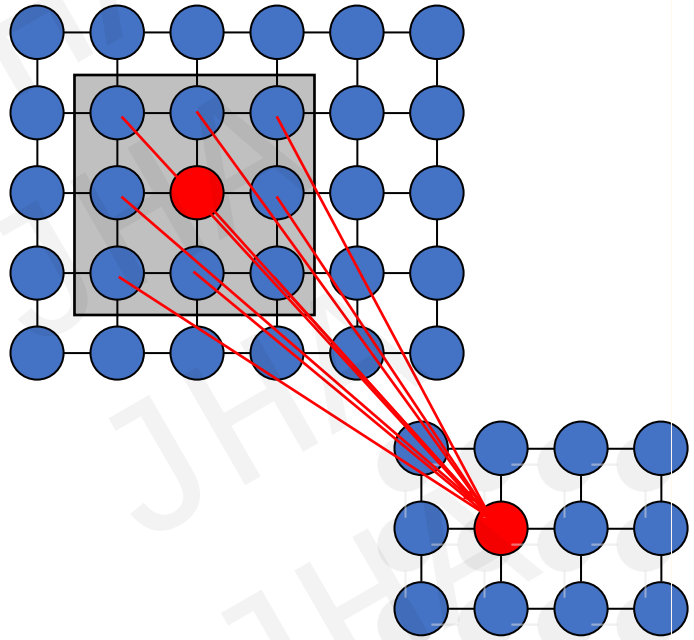
$$h[i,j] = \sum_{a,b} W(i,j,a,b) \cdot x(i+a, j+b)$$



Convolution Operator : CNN context.



Conv/filter Kernel: Edge detection

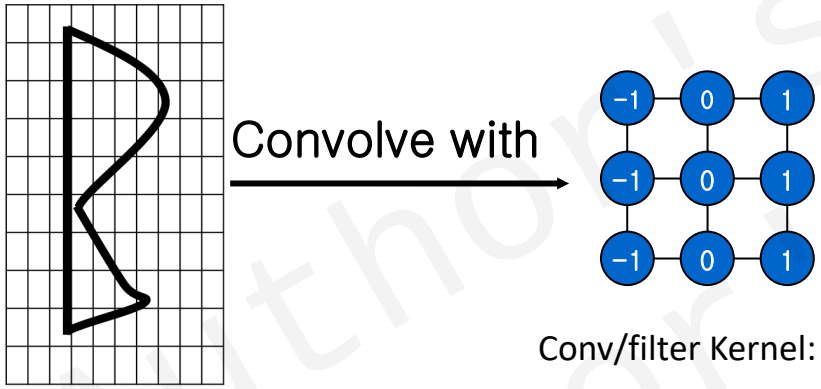


For any location, (i,j), consider an activation value in hidden layer $h[i,j]$

$h[i,j]$ is computed by summing over pixels in x and centered around (i,j).

$$h[i,j] = \sum_{a,b} W(i,j,a,b) \cdot x(i+a, j+b)$$

Convolution Operator : CNN context



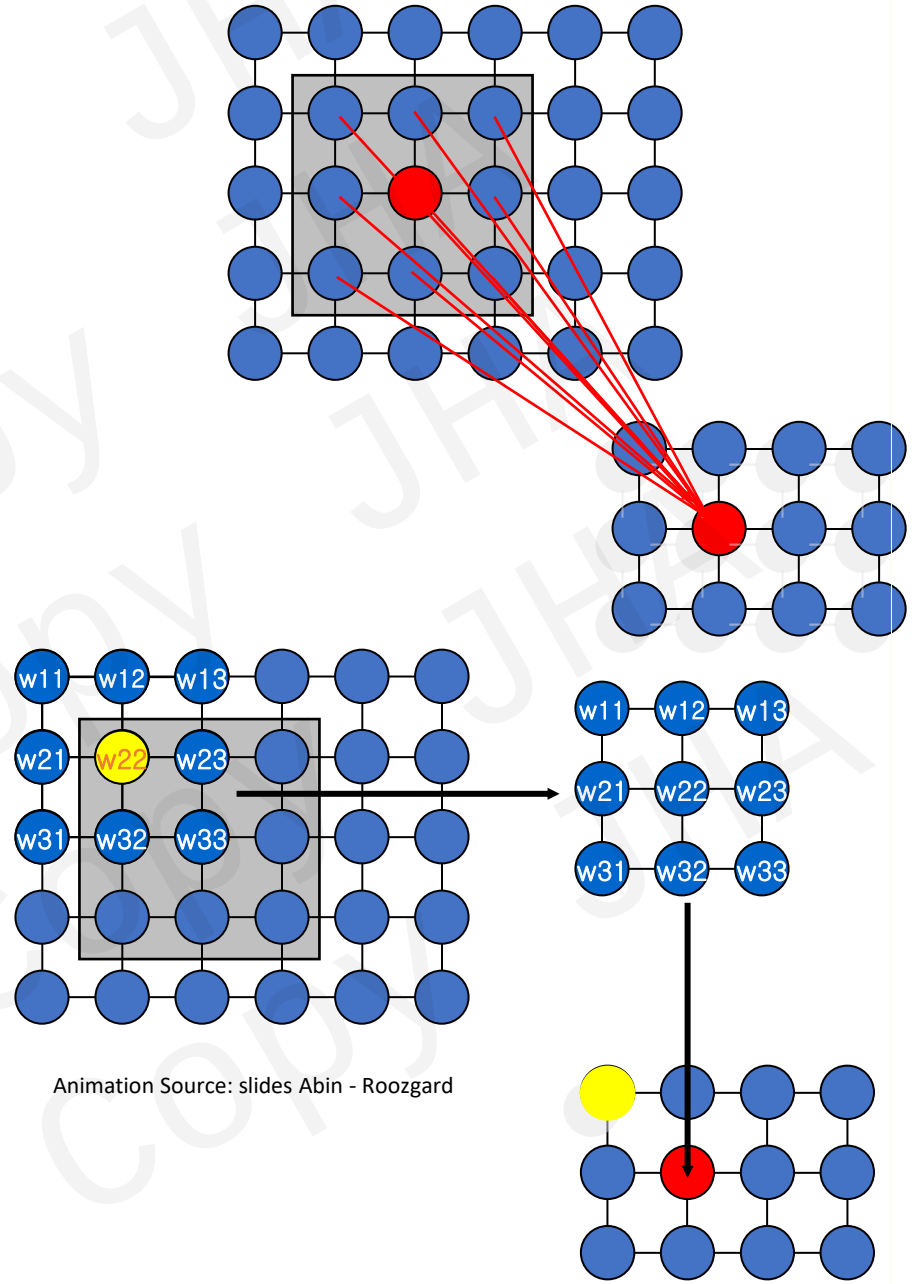
Conv/filter Kernel: Edge detection

For any location, (i,j), consider an activation value in hidden layer $h[i,j]$

$h[i,j]$ is computed by summing over pixels in x and centered around (i,j).

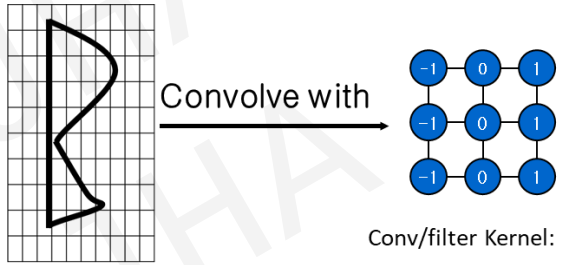
Run the image kernel (filter kernel, convolution) over entire a and b.

$$h[i,j] = \sum_{a,b} W(i,j,a,b) \cdot x(i+a, j+b)$$



Animation Source: slides Abin - Roozgard

Convolution Operator



Conv/filter Kernel: Edge detection

Invoke *Translation invariance:*

Now, activation *h should only change with* shift in inputs *x* .

Or, filter kernel (weights) should be same for all *(i,j)(*pixel positions)

➔This means same feature is searched over whole image.

➔In this way all neurons detect the same feature at different positions in the input image.

$$W[i,j,a,b] = V[a,b]$$

$$h[i,j] = \sum_{a,b} V[a,b].\cdot x[i+a, j+b]$$

Source :Blog AndrewSzot

Convolution Operator

Invoke *Locality:*

The feature should be recognized using local aspects, look in proximity and not very far.

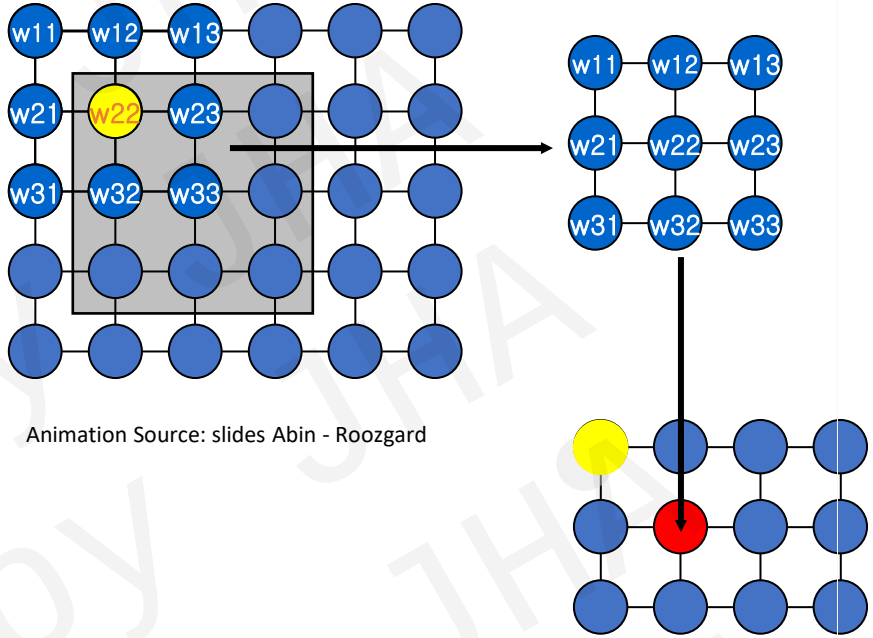
i.e. constrain the size of the kernel filter.

$$W[i,j,a,b] = V[a,b]$$

$$h[i,j] = \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$$

$$for\ |a|,|b| > \Delta$$
$$put\ V[a,b] = 0$$

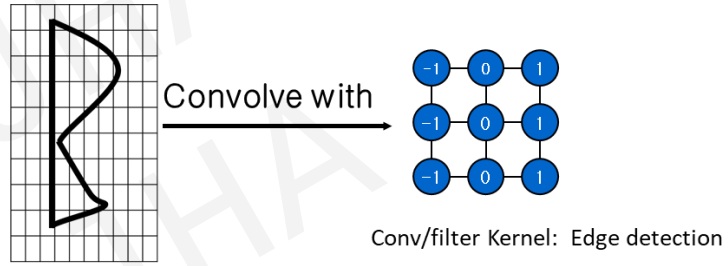
$$h[i,j] = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V[a,b] \cdot x[i+a, j+b]$$



Animation Source: slides Abin - Roozgard

Source :Blog AndrewSzot

Example:



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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

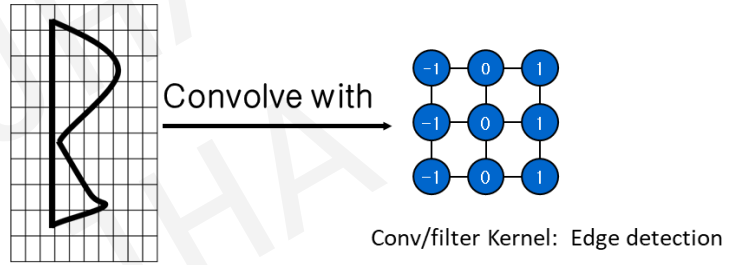
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

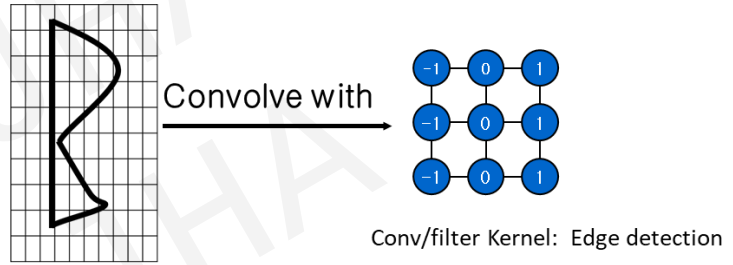
3			

⋮ ⋮

Each filter detects a small feature (3 x 3).

POLYTECH NANCY · UNIVERSITÉ DE LORRAINE · CRAN

Example:



stride=1

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1		

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

stride=1

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

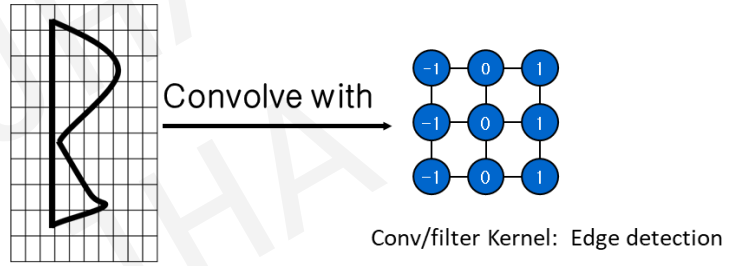
6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	

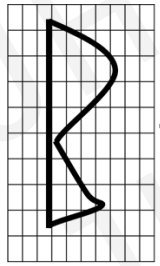


Conv/filter Kernel: Edge detection

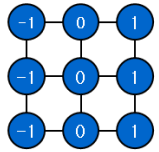
⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



Convolve with



Conv/filter Kernel: Edge detection

stride=1

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	-1

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

stride=1

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

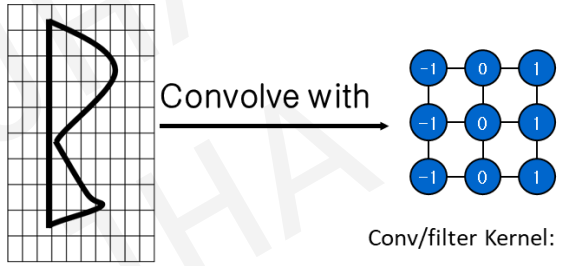
6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	-1
-3			



Conv/filter Kernel: Edge detection

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

stride=1

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

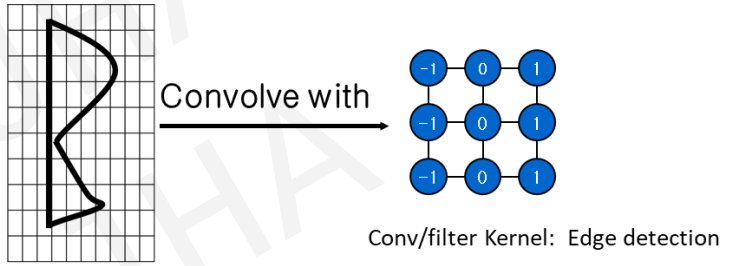
6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	-1
-3	1	0	



Conv/filter Kernel: Edge detection

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

stride=1

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

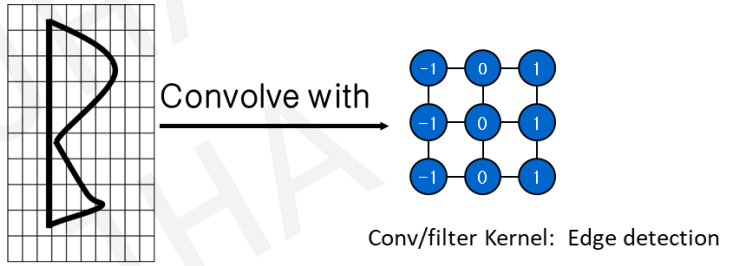
6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	-1
-3	1	0	



Conv/filter Kernel: Edge detection

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

stride=1

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

6 x 6 image

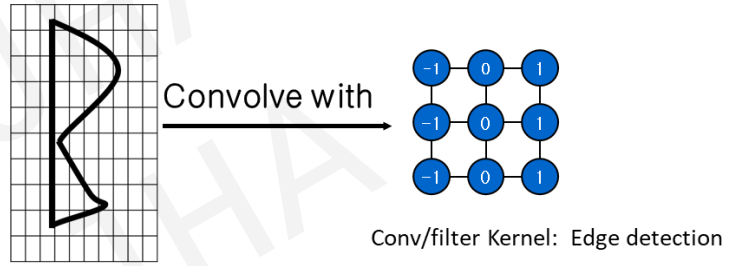
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

convolve (slide) over all spatial locations

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	1

Feature Map

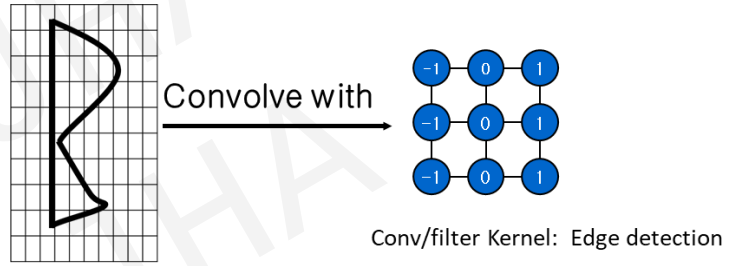


Conv/filter Kernel: Edge detection

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



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

6 x 6 image

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	1

Feature Map

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:

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

6 x 6 image

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

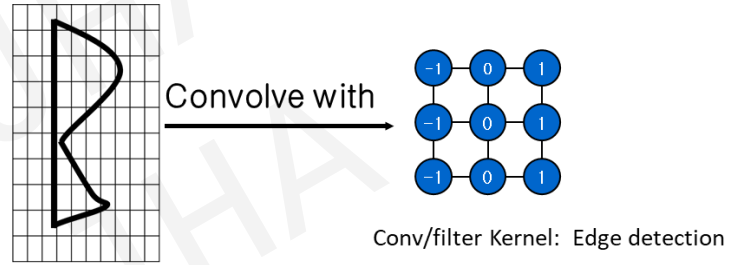
-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	4	3

Feature Maps

⋮ ⋮

Each filter detects a small feature (3 x 3).

Example:



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

6 x 6 image

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	4	3

Feature Maps

⋮ ⋮

2 images of 4 x 4 matrix is produced.
This procedure is repeated for each filter

Example: Edge detection

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

H x W image

1	-1

Edge detector kernel

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

Feature Map

Kernel: if
horizontally elements are same , output is 0.
Else, non-zero.

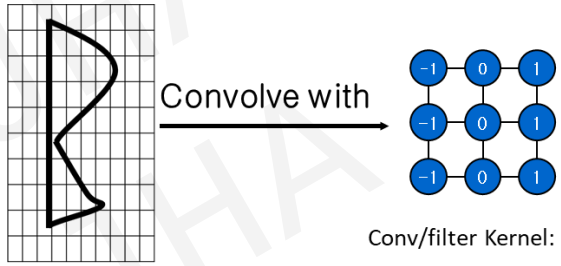
Detected:
1 for edge from white to **black**
-1 for edge from **black** to white

Difficult to handcraft such filters.
Thus, filter kernel weights must be learnt !!

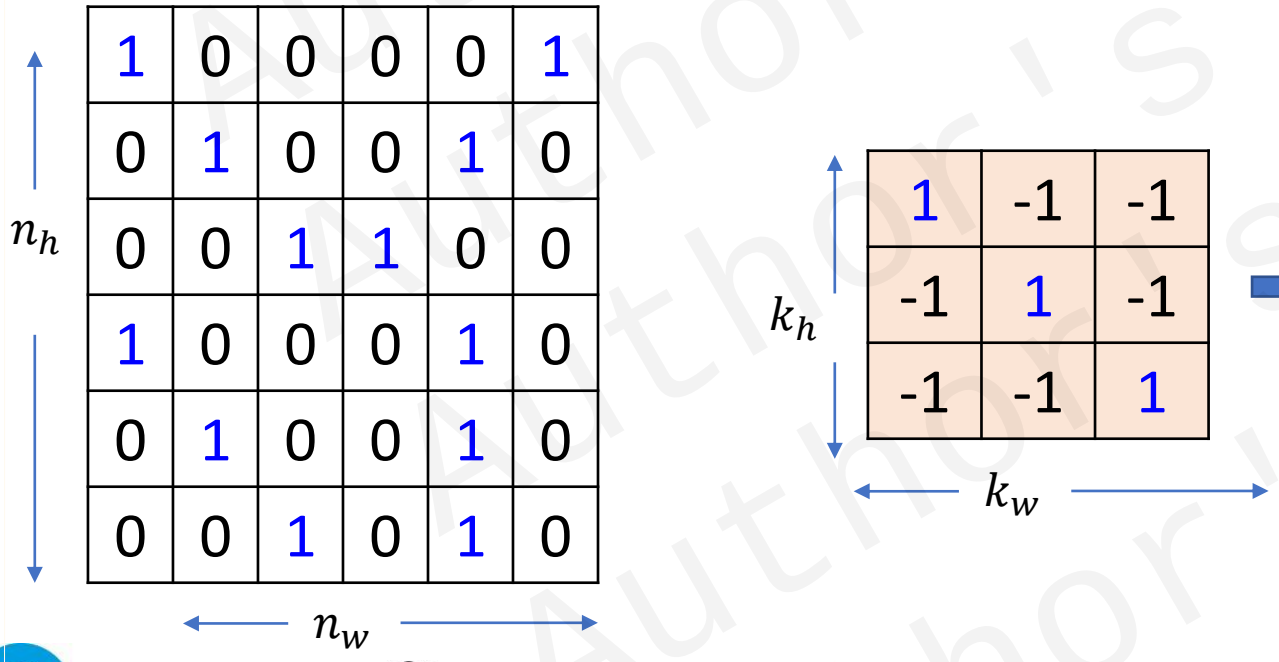
Remarks:

Output shape determined by shape of input and convolutional kernel window.

Small convolution with filter kernels → “smaller” outputs (feature maps).



Conv/filter Kernel: Edge detection



$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$

Padding and Strides

Padding

- Multiple layers of convolution may reduce the information available at boundary.
- Padding prevents this problem.
- Adding zeros around the edges such that multiple convolution operation does not lead to information loss.
- Pixels added around edges.
- These pixels are zero in value.

	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	1	0
	0	0	1	0	0	1	0	0
	0	0	0	1	1	0	0	0
	0	1	0	0	0	1	0	0
	0	0	1	0	0	1	0	0
	0	0	0	1	0	1	0	0
p_h	0	0	0	0	0	0	0	0

p_w

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

Padding: In practice

- $p_h = k_h - 1$,
- $p_w = k_w - 1$,
- Kernel dimensions : k_w, k_h are chosen odd numbers (Ex: 1,3,5,7..)
- Padding dimensions are even. $p = k - 1$,
- then, each side padded with $p/2$ *zeros*
- *or,* padding dimensions = *(k-1)/2*

$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

	0	0	0	0	0	0	0	0
	0	1	0	0	0	0	1	0
	0	0	1	0	0	1	0	0
	0	0	0	1	1	0	0	0
	0	1	0	0	0	1	0	0
	0	0	1	0	0	1	0	0
	0	0	0	1	0	1	0	0
p_h	0	0	0	0	0	0	0	0

p_w

Strides

Stride=1

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

Number of rows and columns per slide → stride.

- Useful in reducing information (resolution) drastically.

Convolution : Strides

Stride=1

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

Convolution : Strides

Stride=1

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

Convolution : Strides

Stride=1

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

Convolution : Strides

Stride=1

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

Input : 7x7 (spatially)

Filter kernel: 3X3

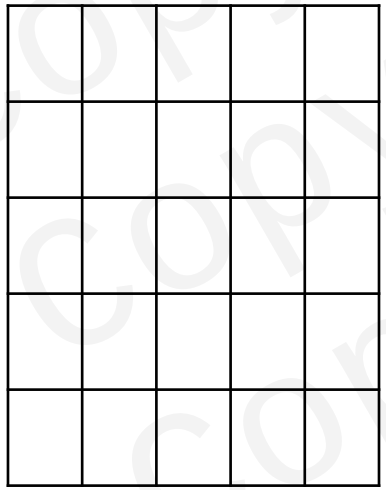
Stride=1

Output =5 x5

Convolution : Strides

Stride=1

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



Feature Map

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=1

Output =5 x5

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3 Feature map matrix

Convolution : Strides

Stride=2

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=2

Output =3 x3 Feature map matrix

In general, with stride =s

output size: $\left(\frac{n_h - \kappa_h + p_h}{s} + 1\right) \times \left(\frac{n_w - k_w + p_w}{s} + 1\right)$

Convolution : Strides

Stride=3

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Cannot apply 3x3 filter kernel on 7X7 input → Does not fit.

Convolution : Strides

Stride=3

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Cannot apply 3x3 filter kernel on 7X7 input → Does not fit.

Convolution : Strides

Stride=3

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

In general, with stride =s

output size: $\left(\frac{n_h - \kappa_h + p_h}{s} + 1\right) \times \left(\frac{n_w - k_w + p_w}{s} + 1\right)$

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Cannot apply 3x3 filter kernel on 7X7 input → Does not fit.

Apply padding

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

Input : 7x7 (spatially)

Filter kernel: 3X3

Stride=3 ?? (stride increased)

Output= 3 X 3

In general, with stride =*s*

output size: $\left(\frac{n_h - \kappa_h + p_h}{s} + 1\right) \times \left(\frac{n_w - k_w + p_w}{s} + 1\right)$

Apply Padding: 1 pixel border on each side (p_h=2 , p_w=2)

Kernel =3X3, Stride =1 ,

output =7 X 7 !!

In practice:

Stride =1,

kernel dim: F X F where F is an odd number (Ex: 1,3,5,7..)

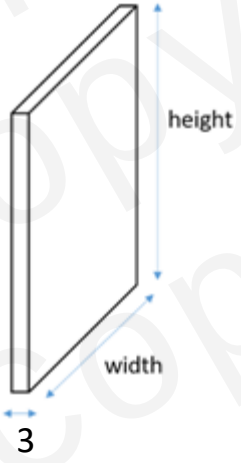
Padding on each side = (F-1)/2

Multi input and output channels

Multi input channels

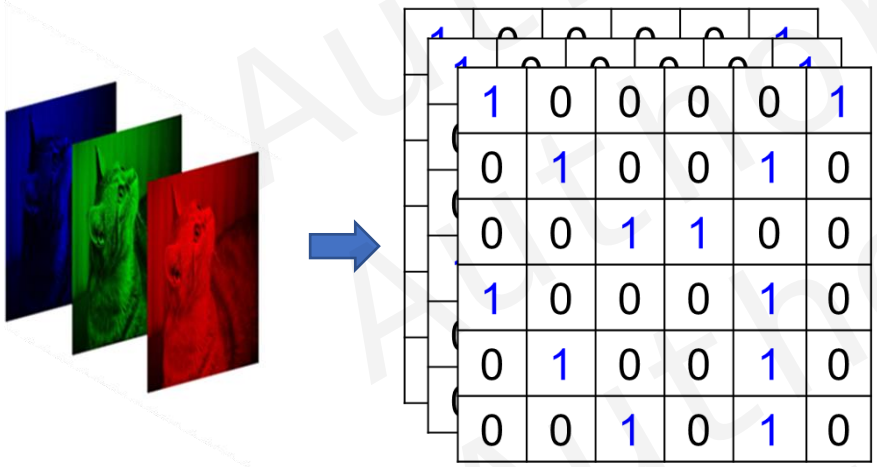
so far, greyscale images→ One channel.

Most images are colorful → 3 channels RGB



➔Input as multi-dimensional array : 3 X *h* X w

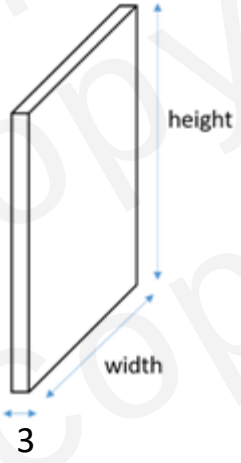
➔construct a convolution kernel with the same number of input channels as the input data (3 here)



Multi input channels

so far, greyscale images→ One channel.

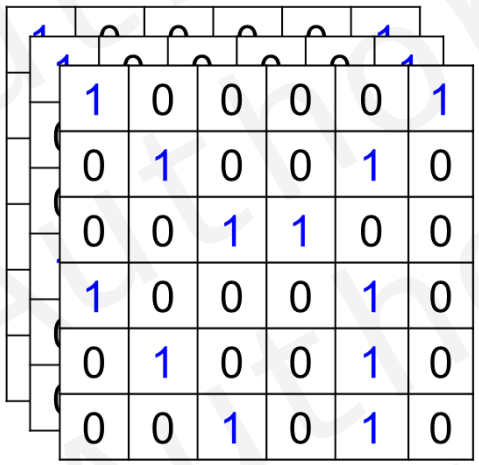
Most images are colorful → 3 channels RGB

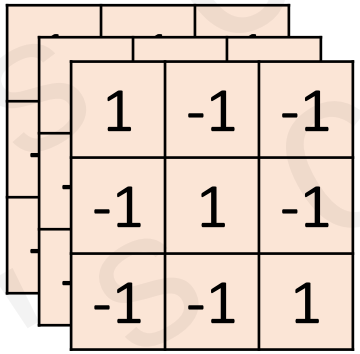


➔Input as multi-dimensional array : 3 X *h* X w

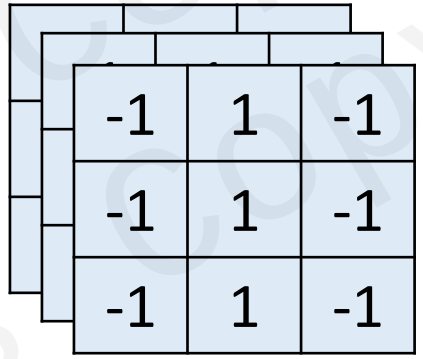
➔construct a convolution kernel with the same number of input channels as the input data (3 here)

➔Assign a 2-D kernel to each channel → concatenation gives 3D conv kernel.





Filter 1



Filter 2

Multi input channels

Convolution with 3 input channels:

- slide the 2D filter kernel on 2D input , for each channel.

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

*

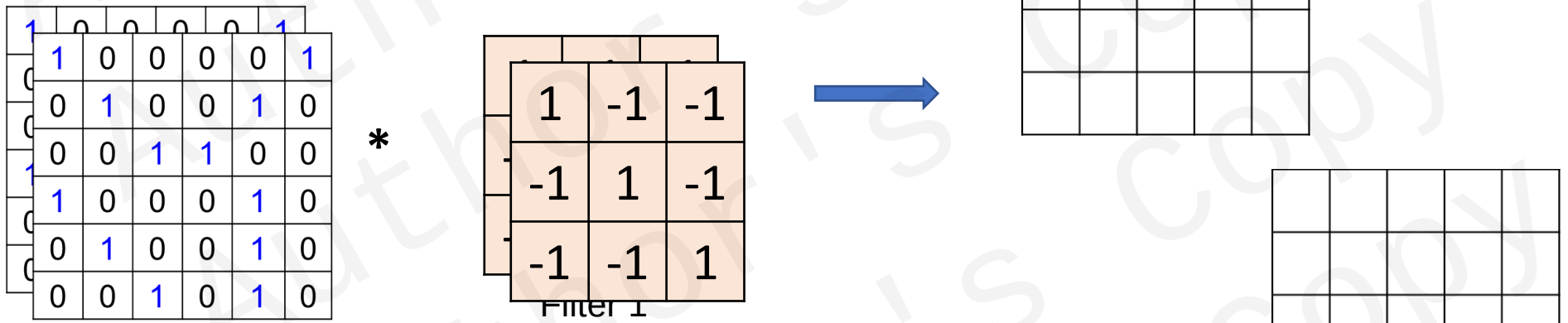
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Multi input channels

Convolution with 3 input channels:

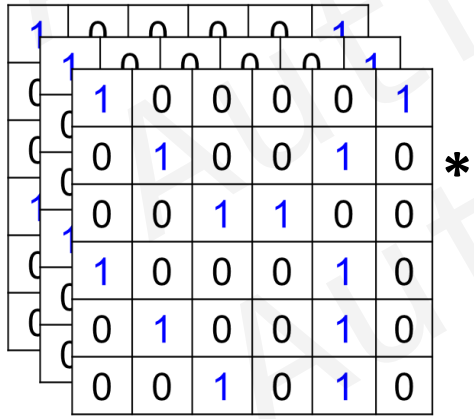
- slide the 2D filter kernel on 2D input , for each channel.

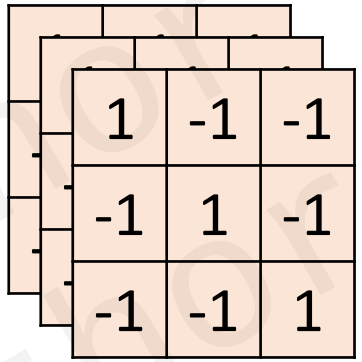


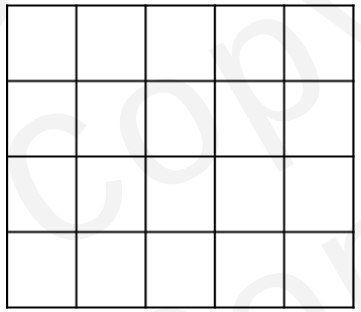
Multi input channels

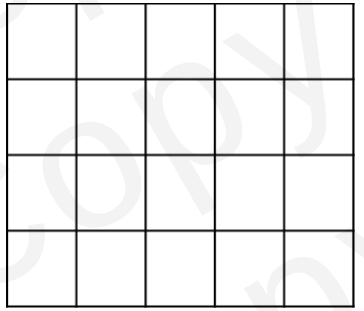
Convolution with 3 input channels:

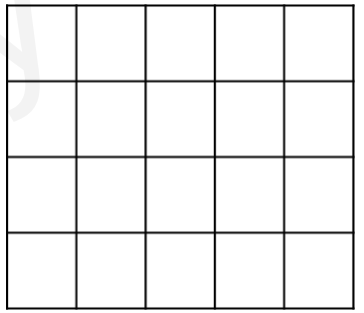
- slide the 2D filter kernel on 2D input , for each channel.







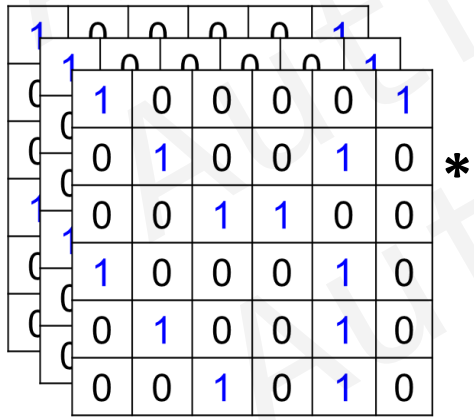


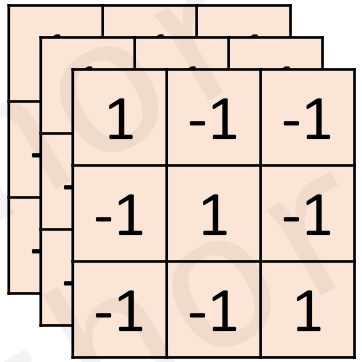


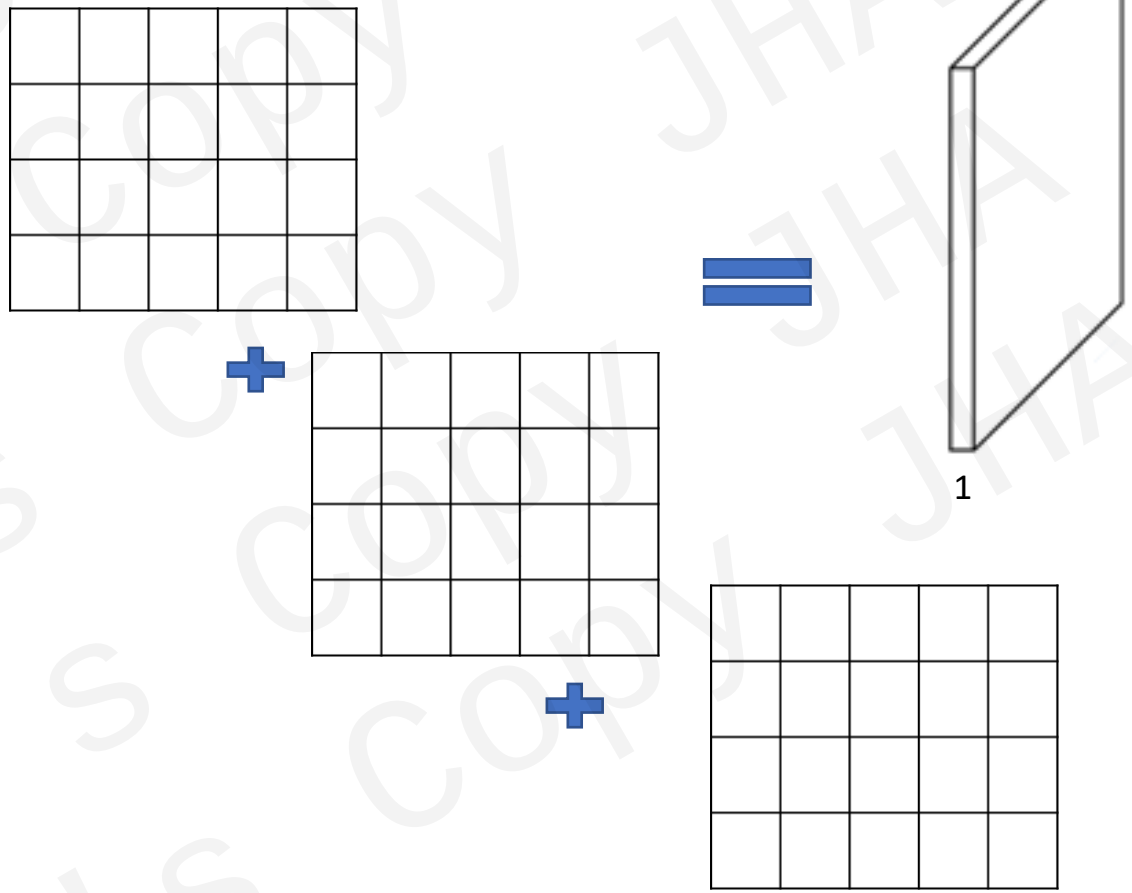
Multi input channels

Convolution with 3 input channels:

- slide the 2D filter kernel on 2D input , for each channel.
- add the three 2D feature maps to get the output➔ feature map (a **2D** array).
- generalizable to *n* input channels.



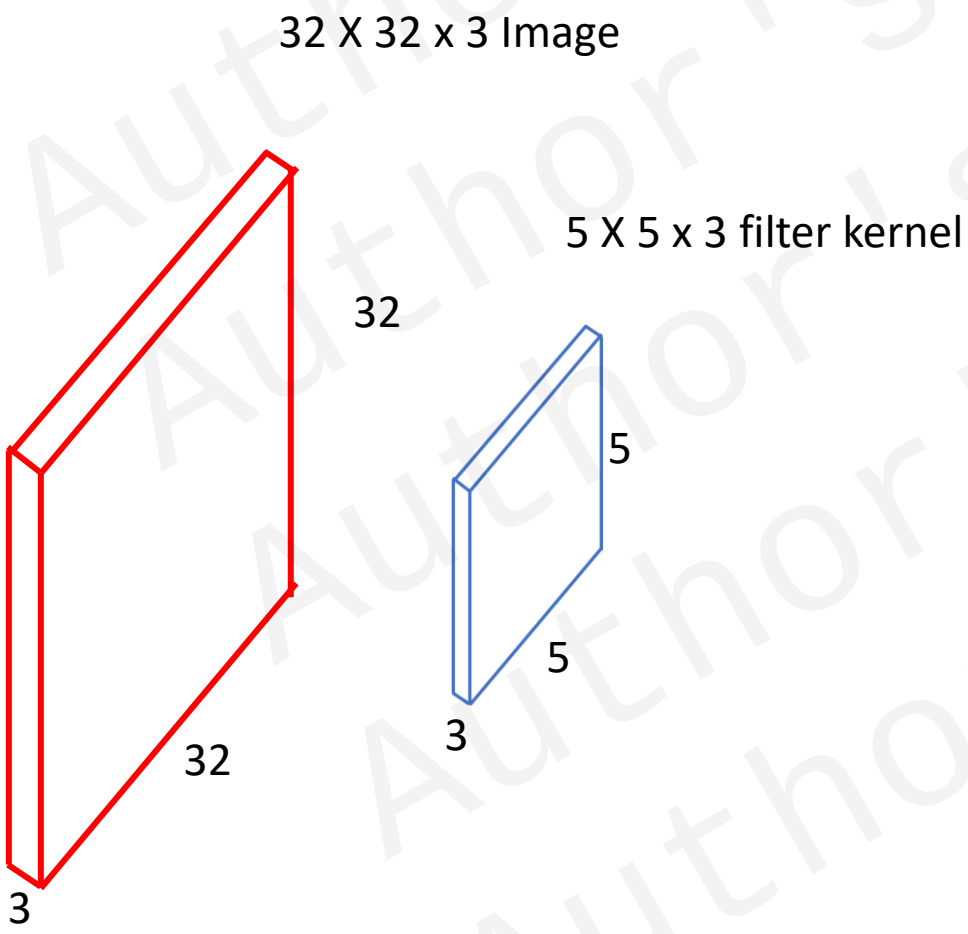




Multi input channels: Summary

Convolution of image (3 channels)

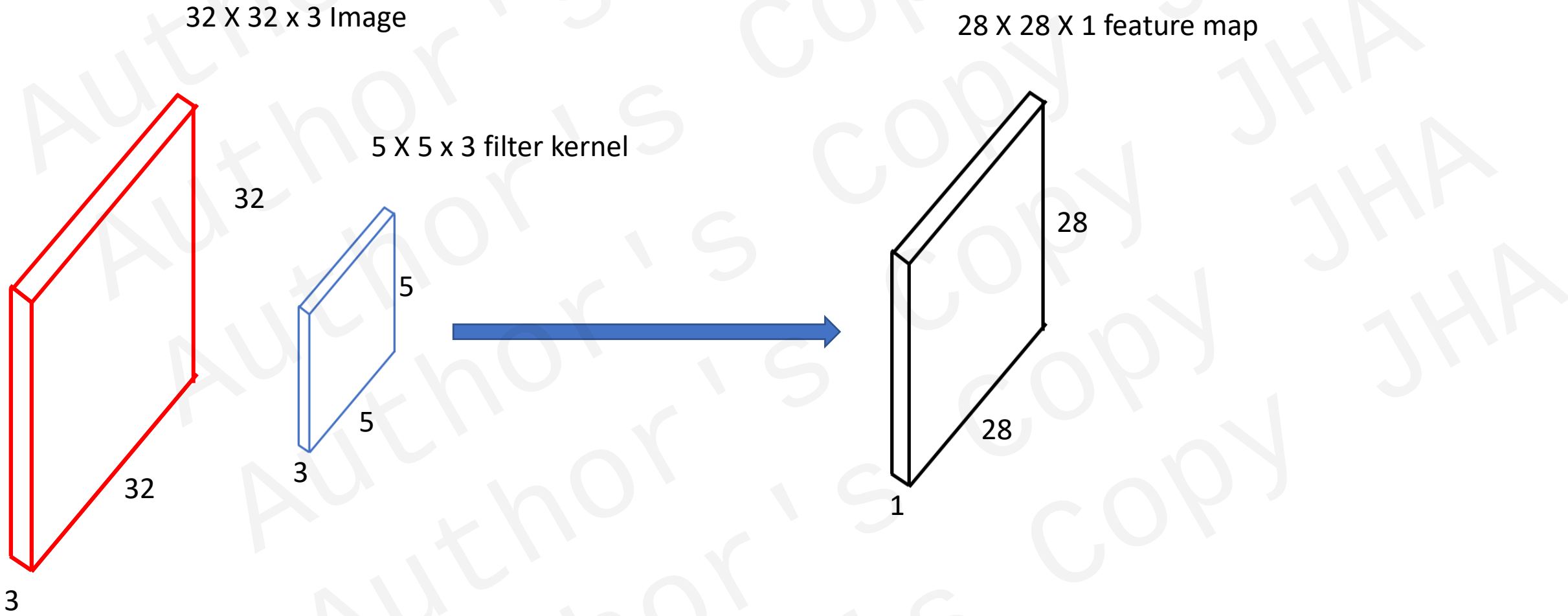
with 3 channel filter ➔ 1-D feature map.



Multi input channels: Summary

Convolution of image (3 channels)

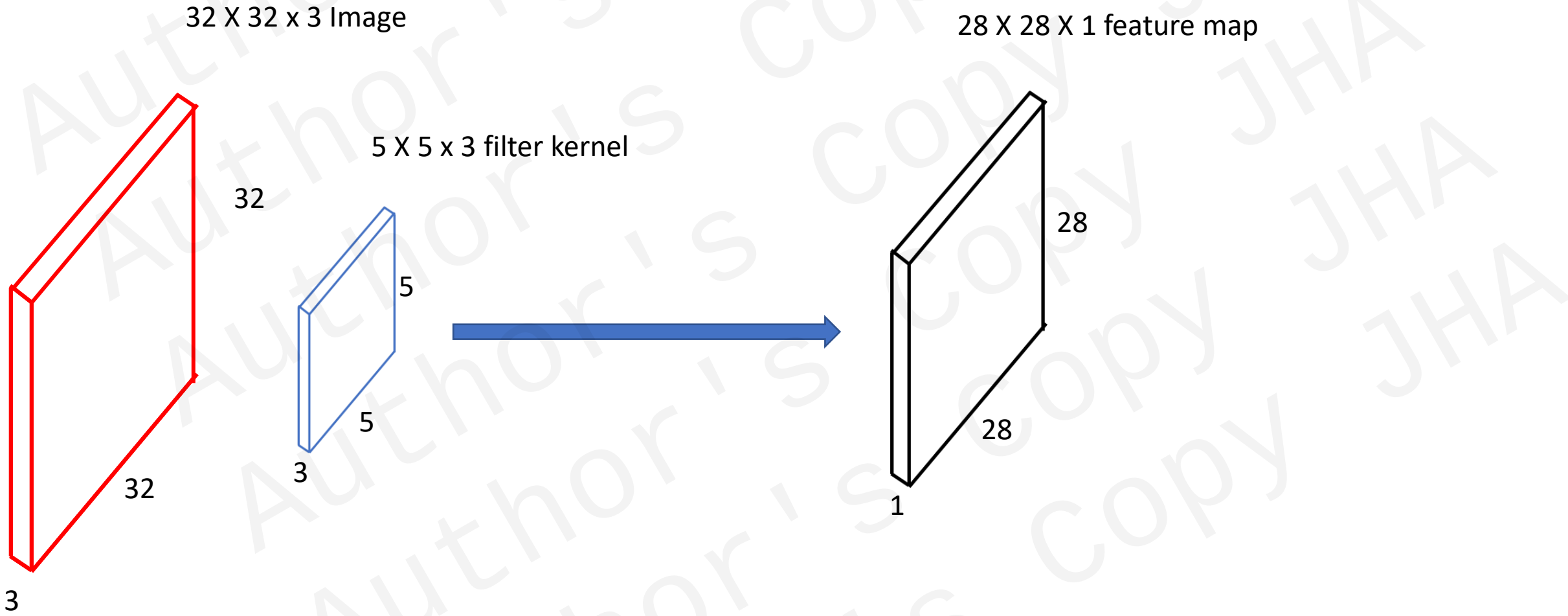
with 3 channel filter ➔ 1-D feature map.



Multi input channels: Summary

Convolution of image (3 channels)

with 3 channel filter ➔ 1-D feature map.



Multi outputs

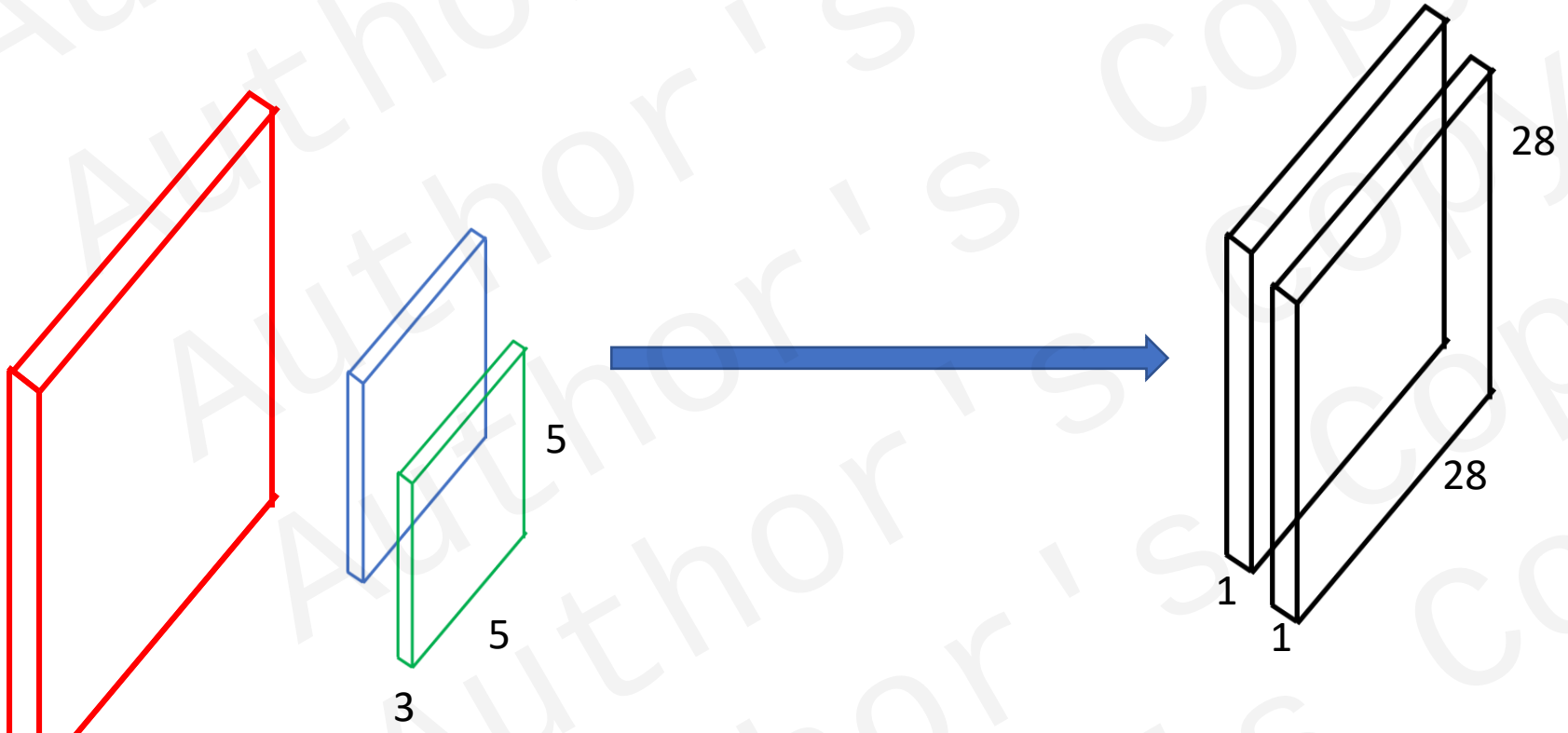
When more than one feature is to be extracted → multiple filters are used.

Output for each filter is desired. Output has multiple channels.

Convolution is performed with each filter kernel , for each output channel.

Output is concatenated along number of filter (output channel) dimension.

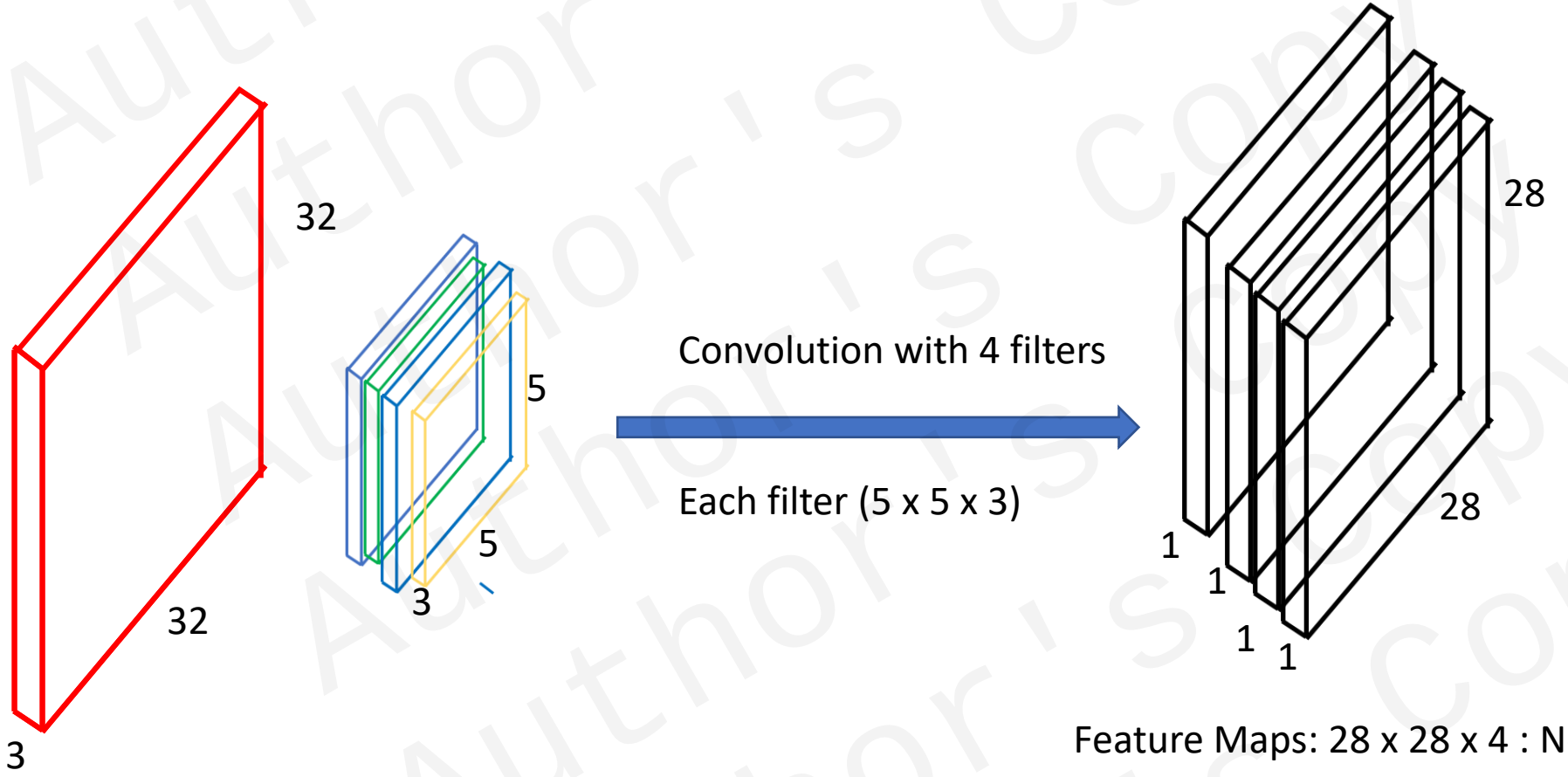
2 different filters ➔ convolution with each filter kernel and concatenated along output channel dimension.



Multi outputs

When more than one feature is to be extracted → multiple filters are used.

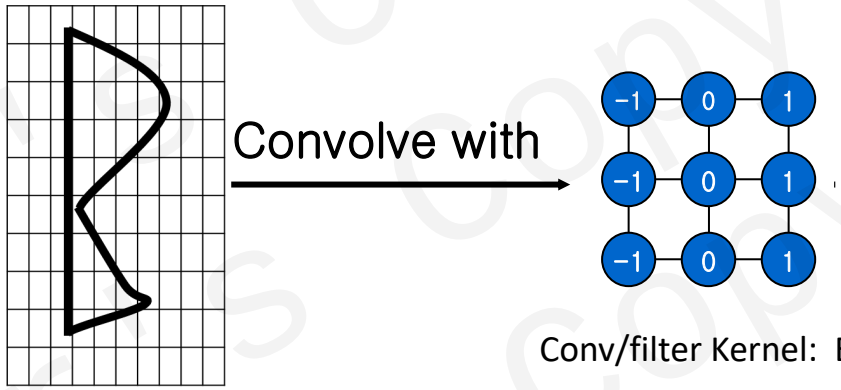
Output for each filter is desired. Output has multiple channels.



Feature Maps: 28 x 28 x 4 : New image!

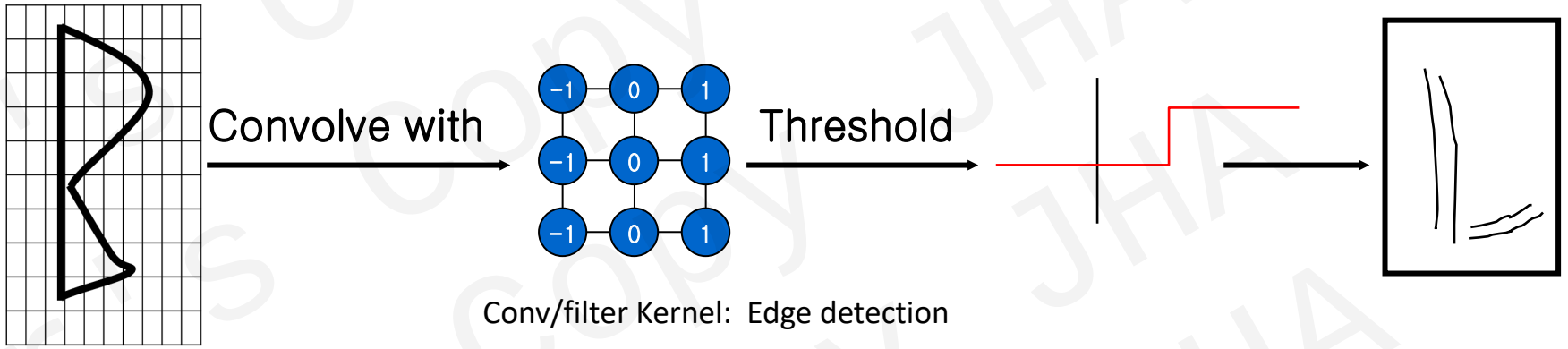
Summary

- So far: Just Convolutions!



Conv/filter Kernel: Edge detection

Summary

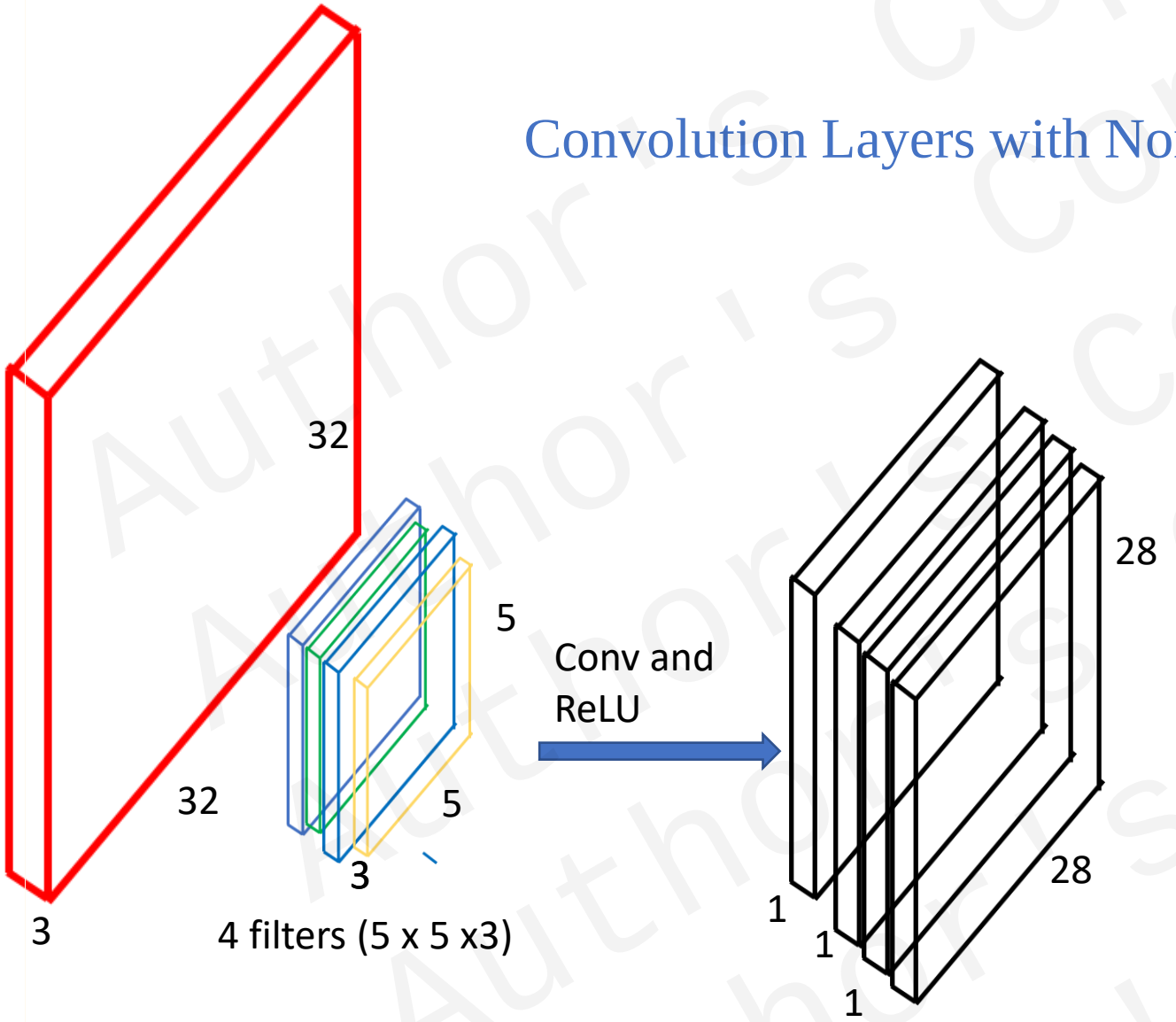


Conv/filter Kernel: Edge detection

- Apply Non-linearity (as seen earlier) : features pass thru activation functions → activation maps (terminology is loose , feature maps/activation maps both are used often to mean the same)

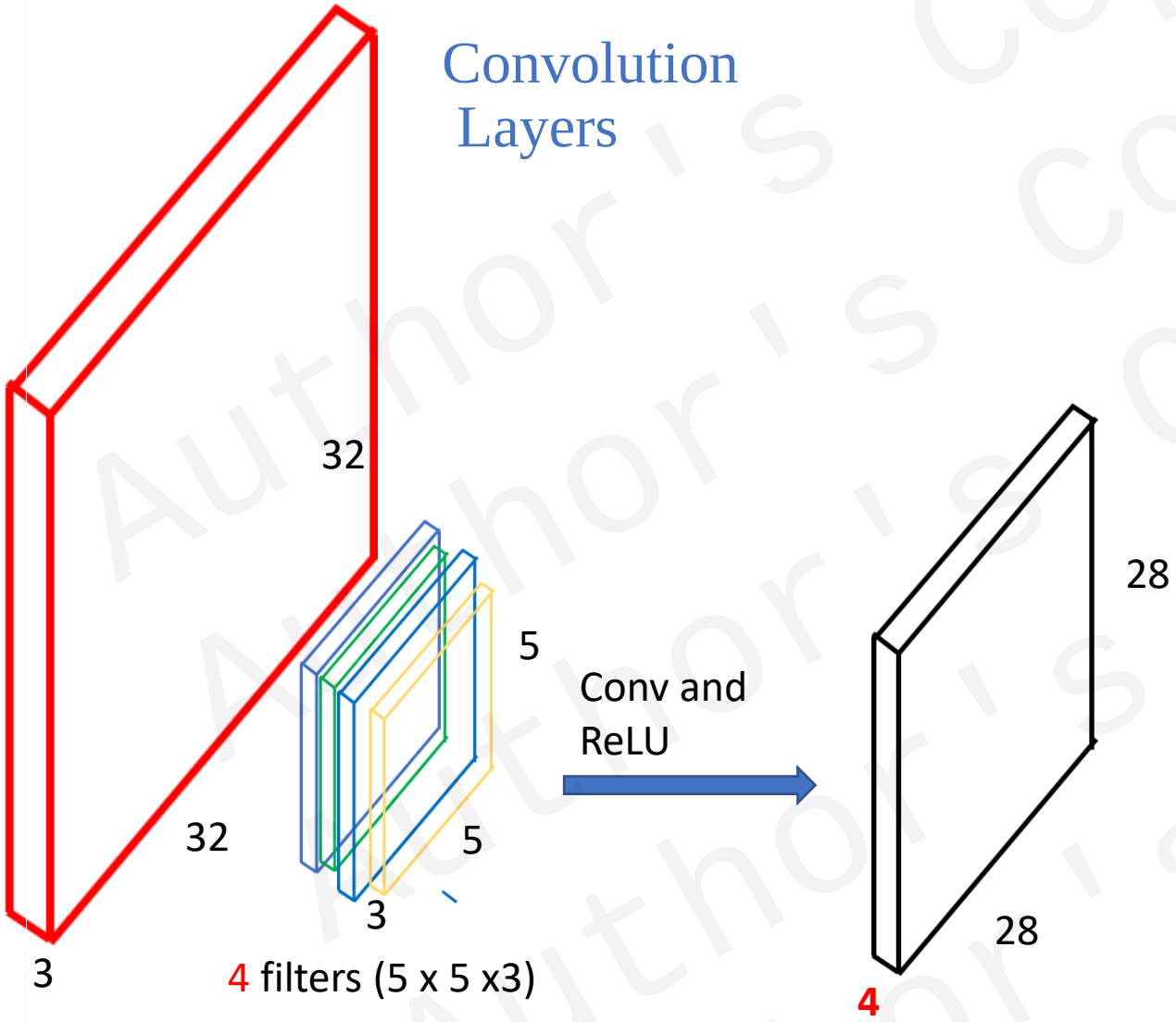
- In practice: ReLu is mostly preferred (fast convergence, no zero-gradient problem..

Convolution Layers with Non linearity Activation



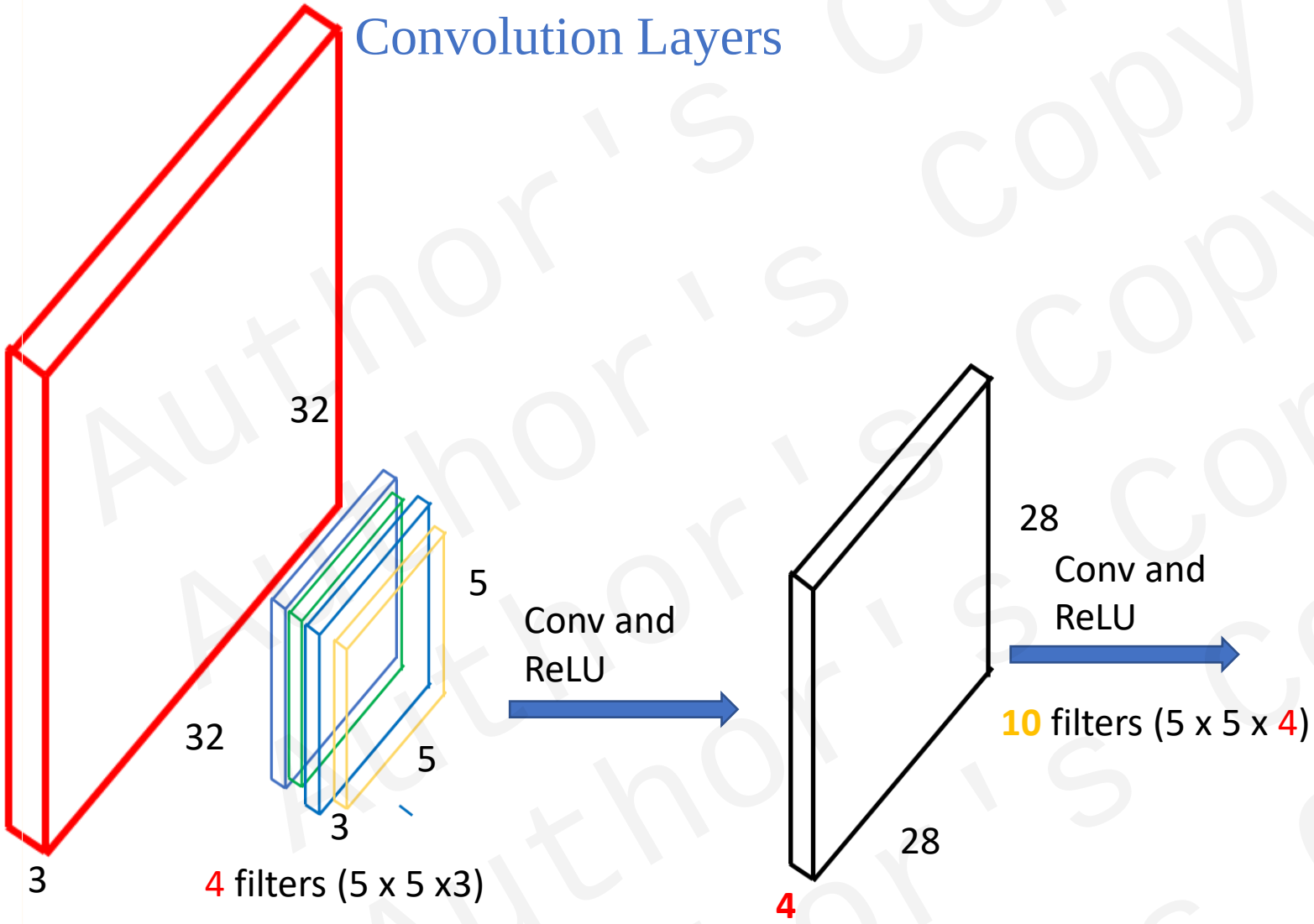
Feature Maps: 28 x 28 x 4 : New image!

Convolution Layers



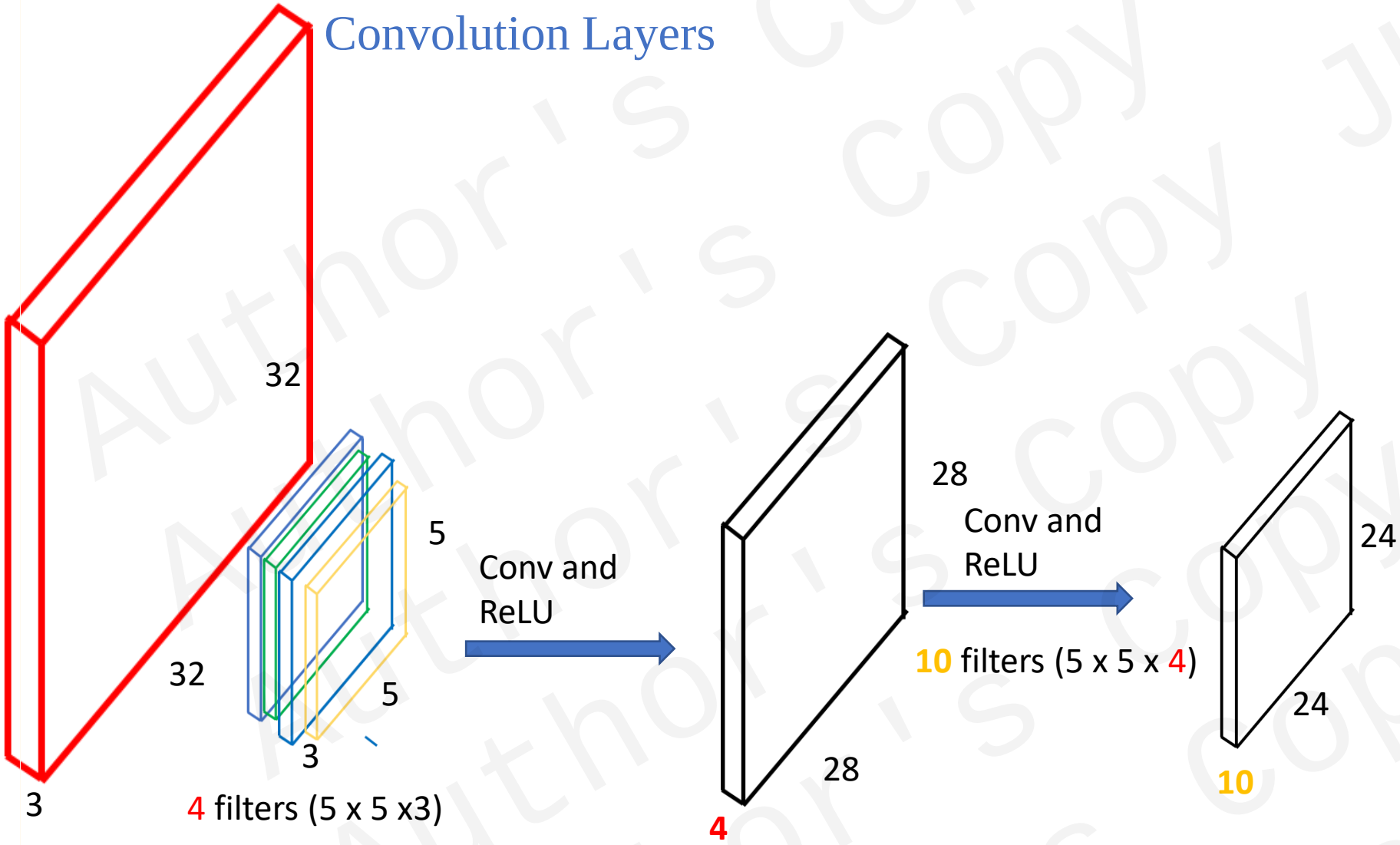
Feature Maps / Activation maps : 28 x 28 x 4

Convolution Layers



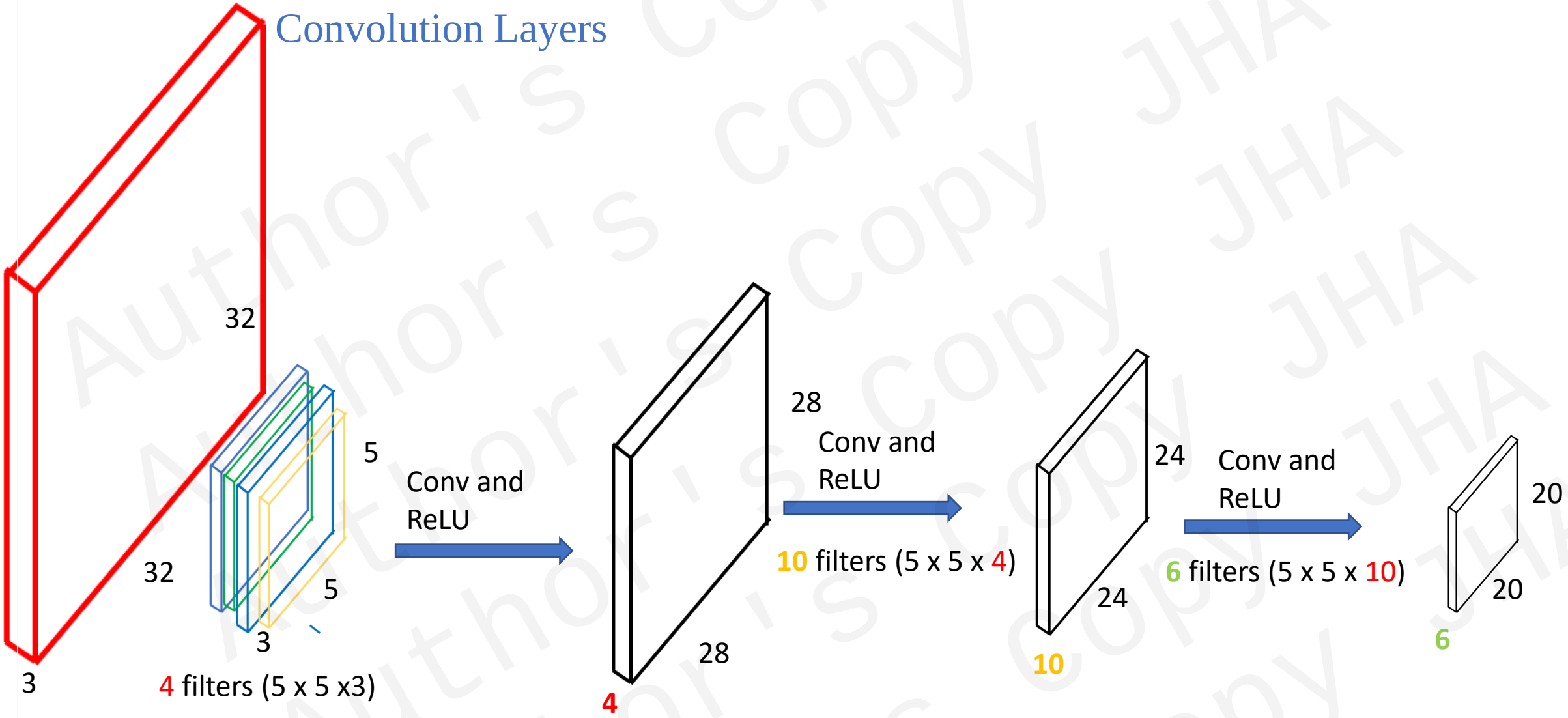
Feature Maps / Activation maps
: 28 x 28 x 4

Convolution Layers



Feature Maps / Activation maps
: 28 x 28 x 4

Convolution Layers



Feature Maps / Activation maps
: 28 x 28 x 4

Convolutional neural network (CovNets) CNNs

JHA Mayank , Email: mayank-shekhar.jha [at] univ-lorraine.fr

Convolution Layers

Remarks: Observe the reduction in size.

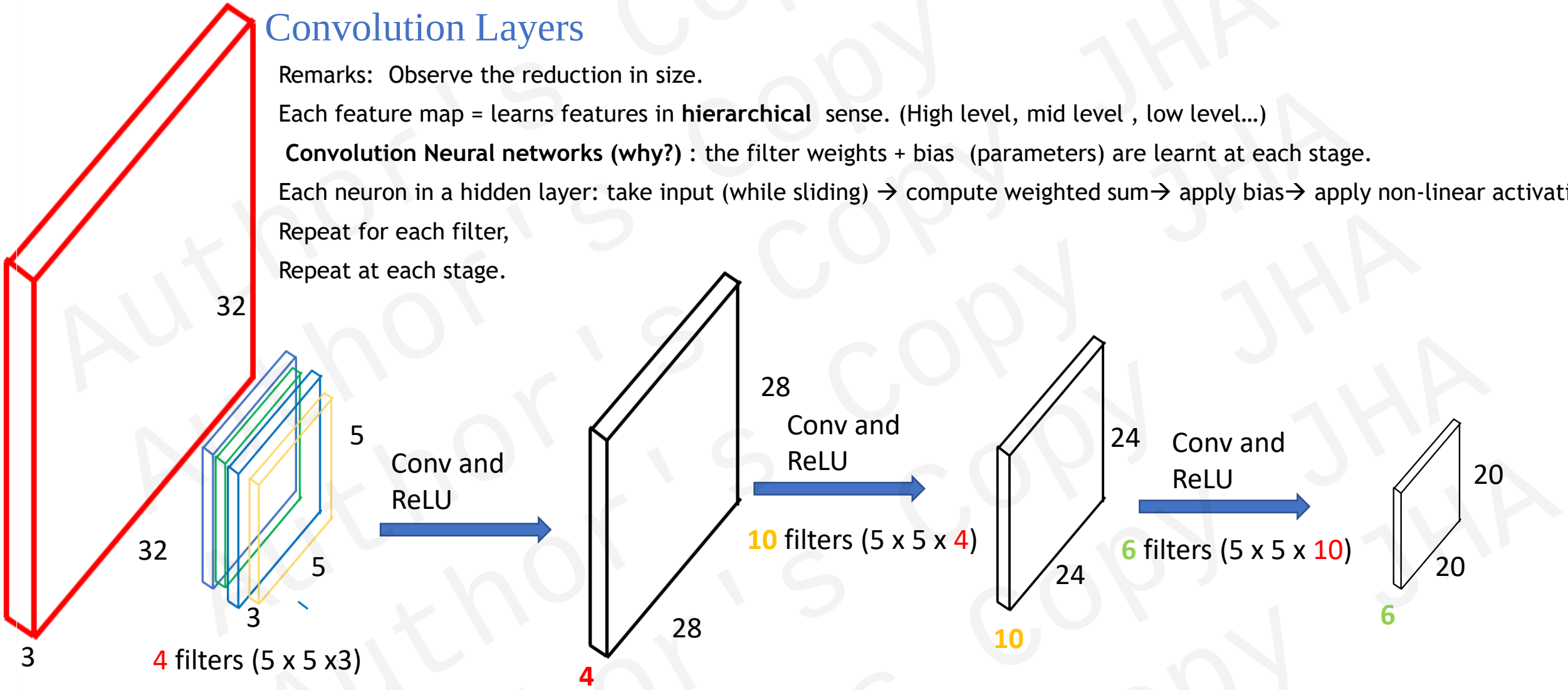
Each feature map = learns features in **hierarchical** sense. (High level, mid level , low level…)

Convolution Neural networks (why?) : the filter weights + bias (parameters) are learnt at each stage.

Each neuron in a hidden layer: take input (while sliding) → compute weighted sum→ apply bias→ apply non-linear activation

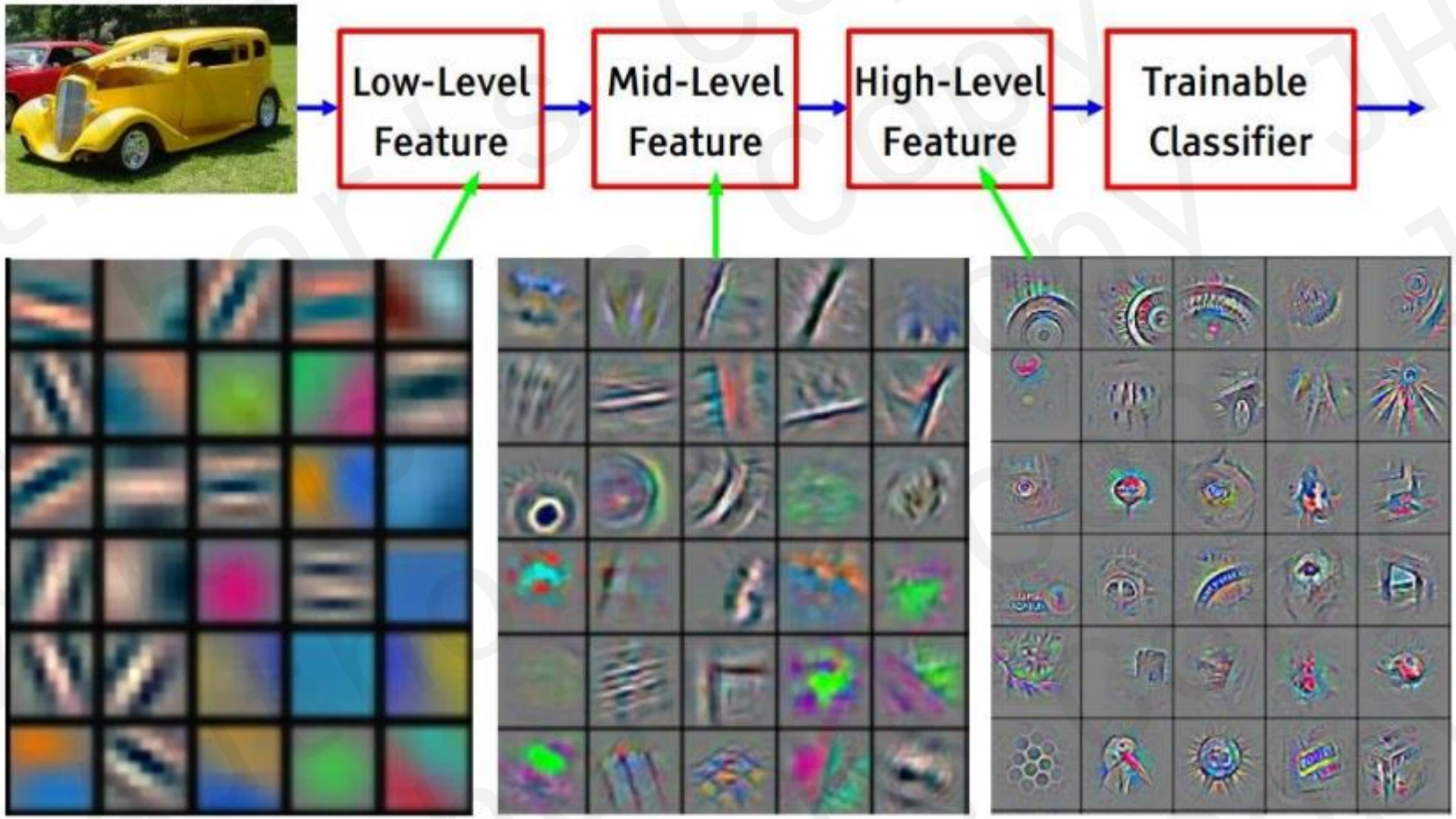
Repeat for each filter,

Repeat at each stage.



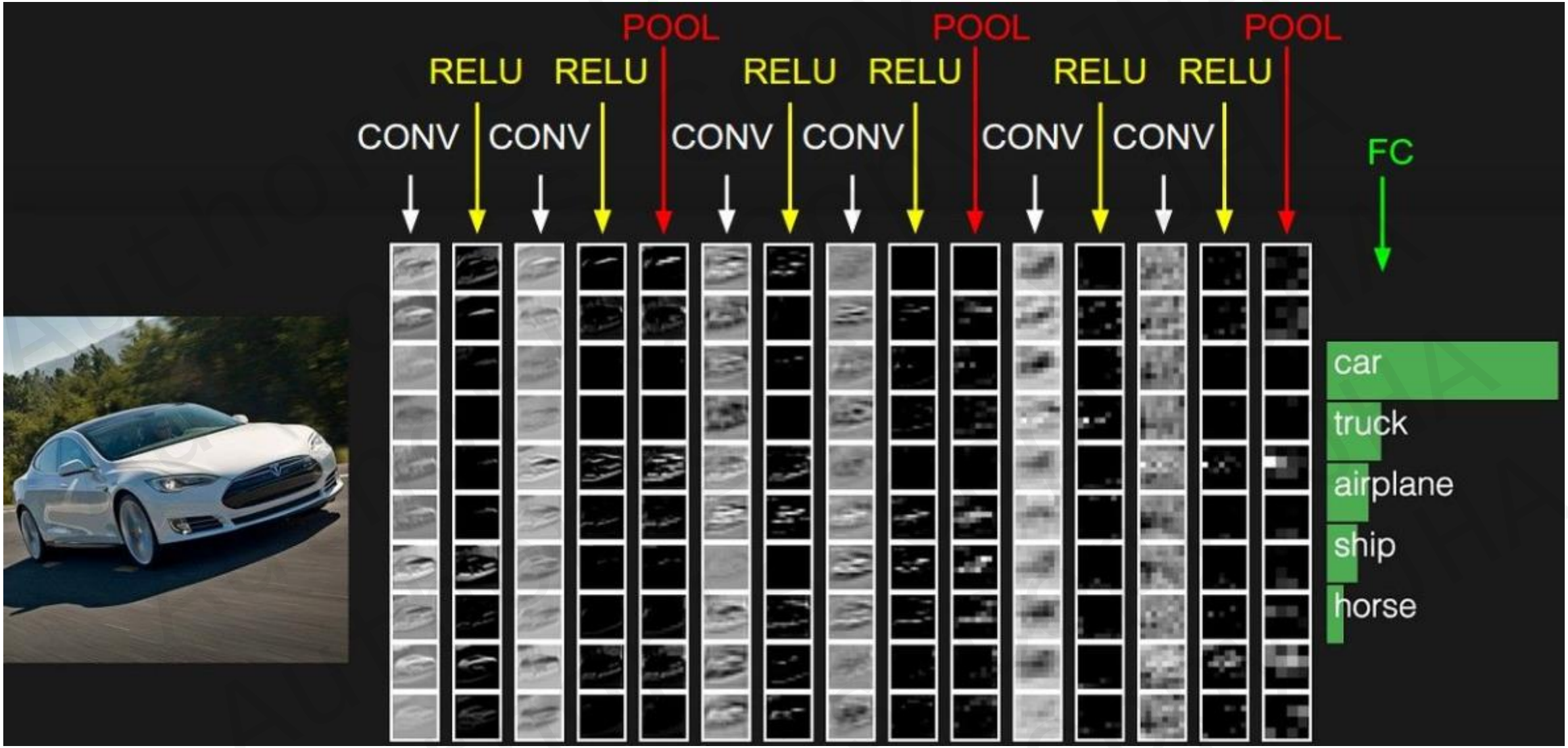
$$\sum_{i=1}^{5}\sum_{j=1}^{5} w_{i,j}x_{i+p,j+q} + b$$

Feature Maps / Activation maps : 28 x 28 x 4



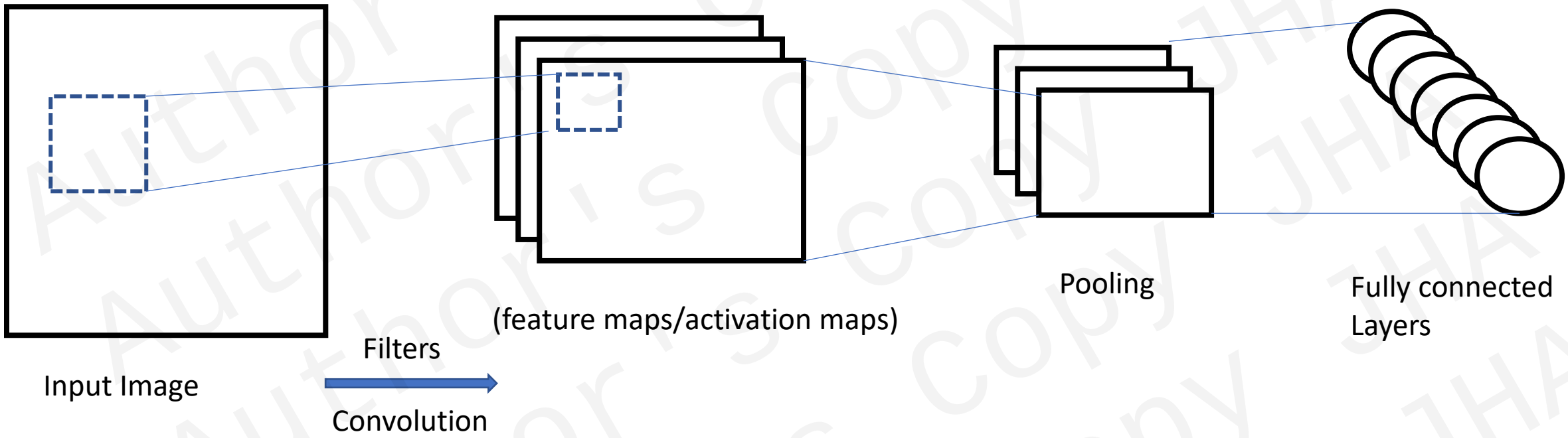
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Source: Dr. Fie Fie Li slides



Source: Prof. Fie Fie Li slides

CNNs for classification



- We discussed convolution operation and feature maps.
- Pooling:

Pooling

Pooling: Motivations

Down sampling:

- We want to reduce the resolution of images.
- The output should not depend on the dimensionality of the original image.

Invariance to translation:

In reality, objects hardly ever occur exactly at the same place.

- Detection should be invariant to translation to some extent.

Example: For instance, image with sharp feature and shifted by one pixel → detection result should not be vastly different from original image.

Pooling layers:

- reduce the sensitivity of Conv layer to location
- reduce the resolution through the processing pipeline.

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1		1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	6

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	2.25

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	6
1	

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	2.25
-0.25	

Pooling layer : Max pooling or Average Pooling

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Max Pooling

Choose the maximum value in pooling window

3	6
1	3

0	3	1	6
2	1	0	2
-1	1	1	1
-1	0	-1	3

Average Pooling

Choose the average value in pooling window

1.5	2.25
-0.25	1

Strides and padding also available for pooling.

In practice, pooling window size: 2 x 2 , stride = 2.

Note: Zero padding is NOT common for pooling layers.

Pooling

Pooling layers / Subsampling pixels does not change the object.

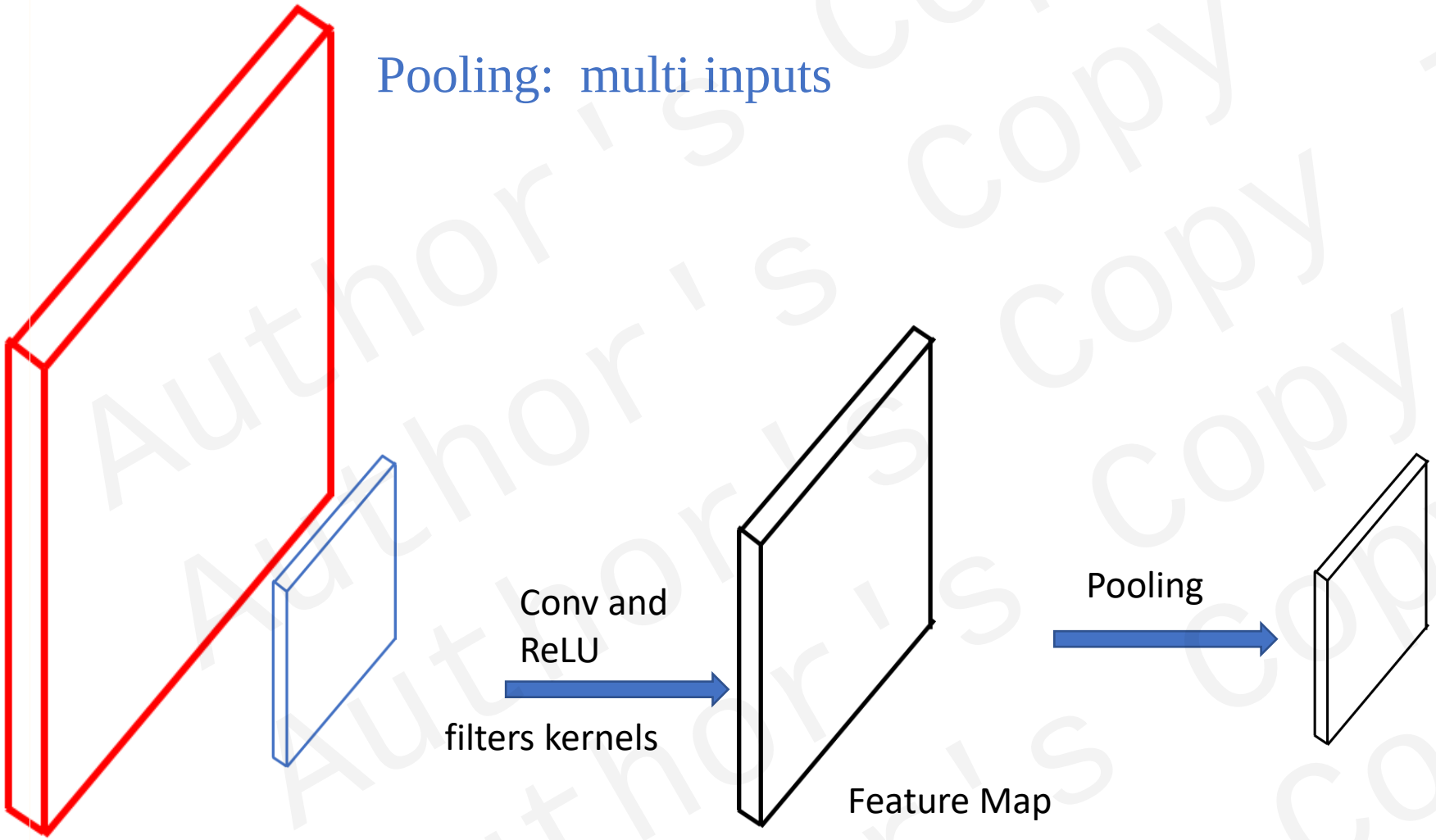
Changes the resolution , fewer parameters to characterize the image.

The subsampling layers reduce the spatial resolution of each feature map

By reducing the spatial resolution of the feature map, a certain degree of shift and distortion invariance is achieved.

Reduces the effect of noises and shift or distortion

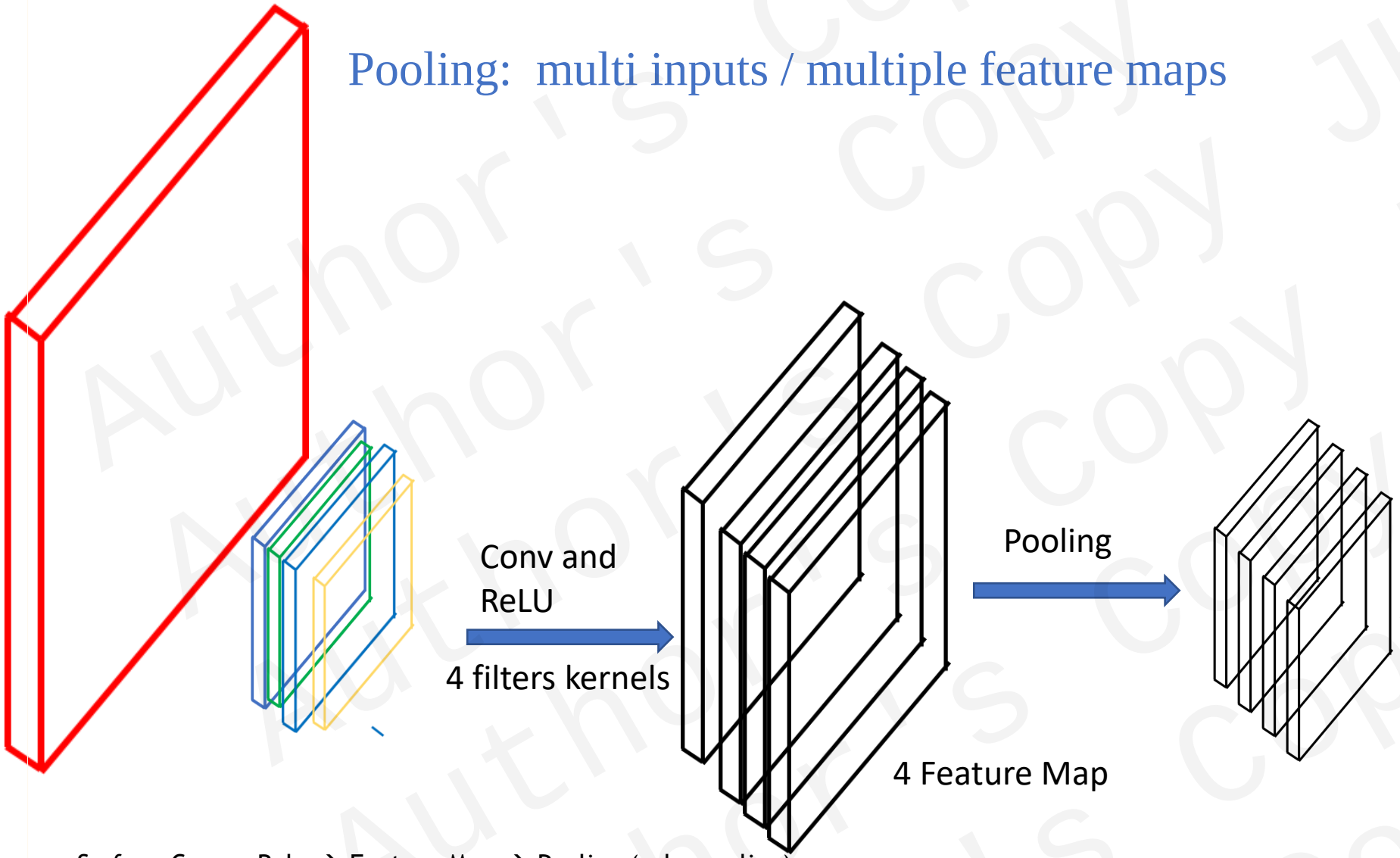
Pooling: multi inputs



So far: Conv + Relu → Feature Map → Pooling (subsampling)

When, multiple filters used:

Pooling: multi inputs / multiple feature maps

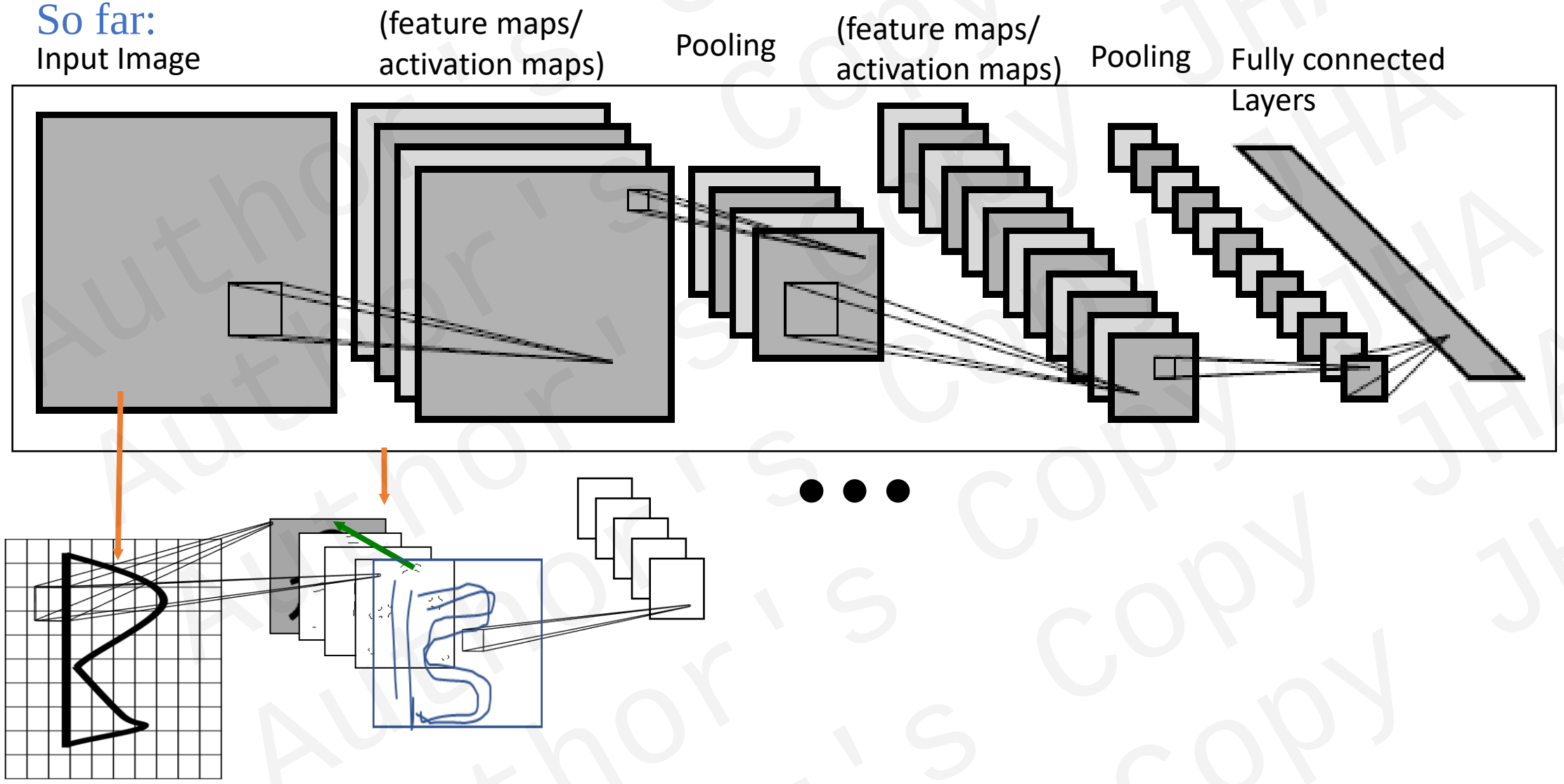


So far: Conv + Relu → Feature Map → Pooling (subsampling)

When, multiple filters used: Pooling done on each input feature map.

New set of images but smaller images.

So far:

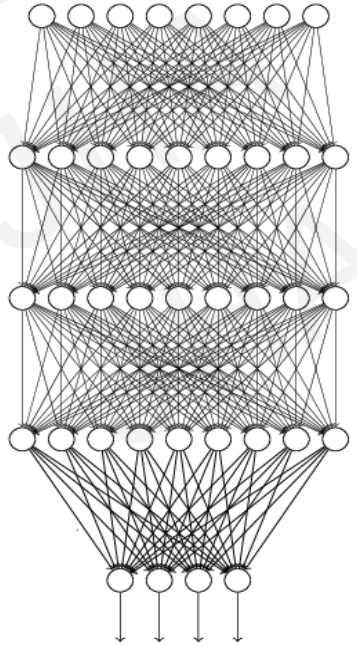


Convolution → Max Pooling → Convolution → Max Pooling → Flattening

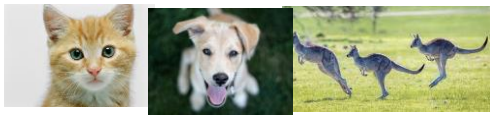
A new image

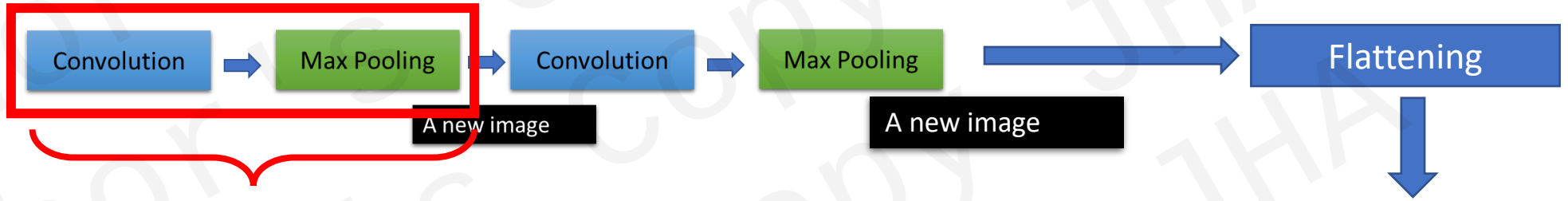
A new image

Fully Connected Feedforward network



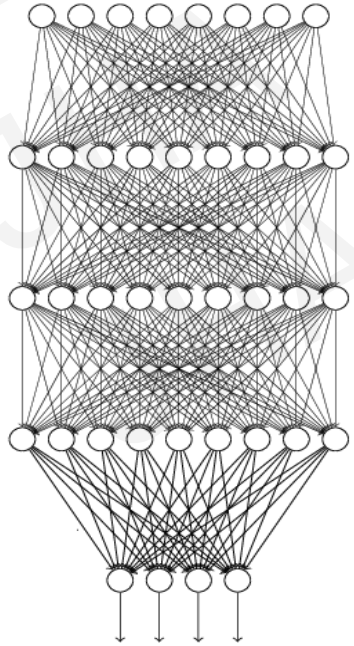
cat , dog , kangaroo……





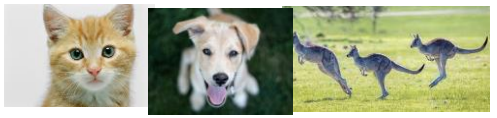
This can repeat many times
Pooling leads to subsampling

Fully Connected
Feedforward network

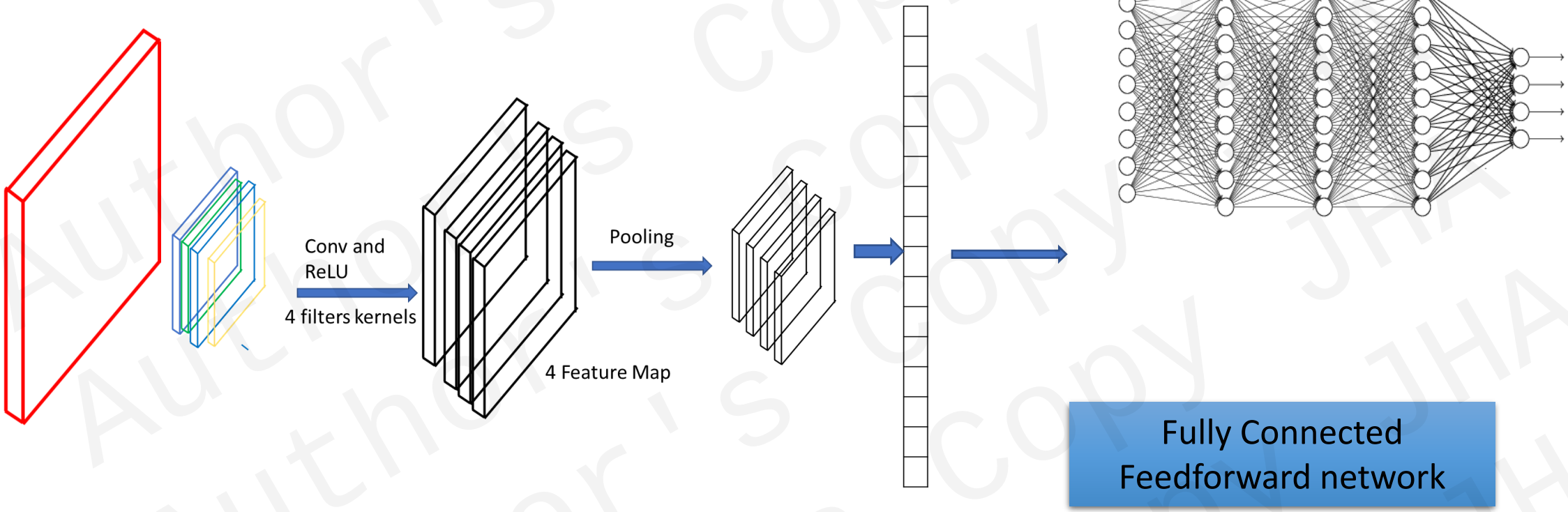


- At each stage, a new image (reduced resolution) is obtained, ready for convolution.
- At the end, the output structure is “flattened” to create a single long feature vector to be used by the dense layer for the final classification.

cat , dog , kangaroo……



Flattening



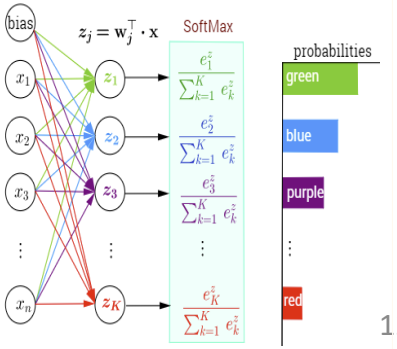
For classification, expect the net outputs distribution of probability of each class (multi class , softmax)

This has little to do with “spatial” 2D information.

This is abstract representation.

Output information → flatten→ create single long feature vector (like last lecture)→ feed to Dense ANNs.

Fully connected Dense Feed foreword networks can propagate this information.



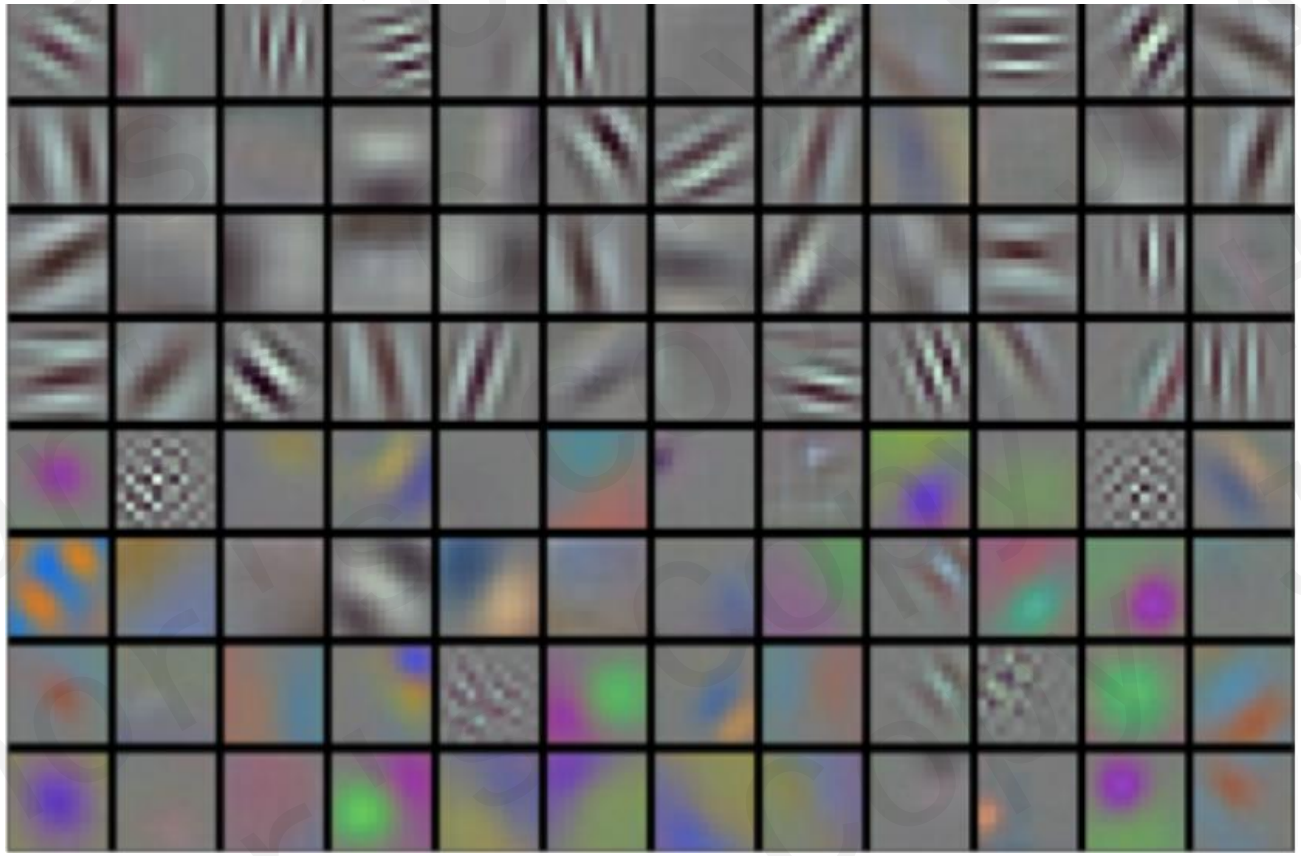
Demo: training on CIFAR-10 dataset

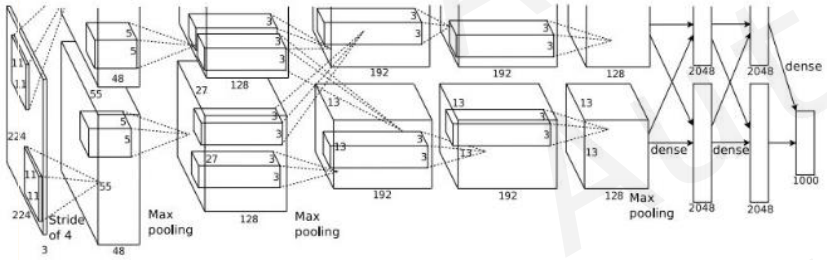
https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.htmlt

AlexNet: obtained above par state of art results on ImageNet challenge,

learnt good low level features,

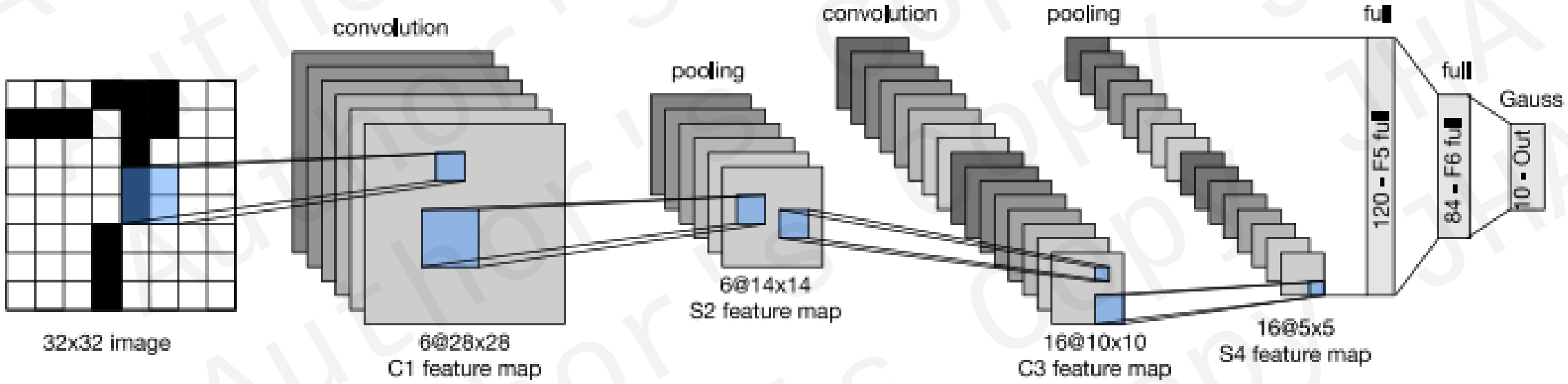
higher level features built upon these.





LeNet-5 (LeCun et al. 1998)

- state-of-the-art performance on hand digit recognition tasks.



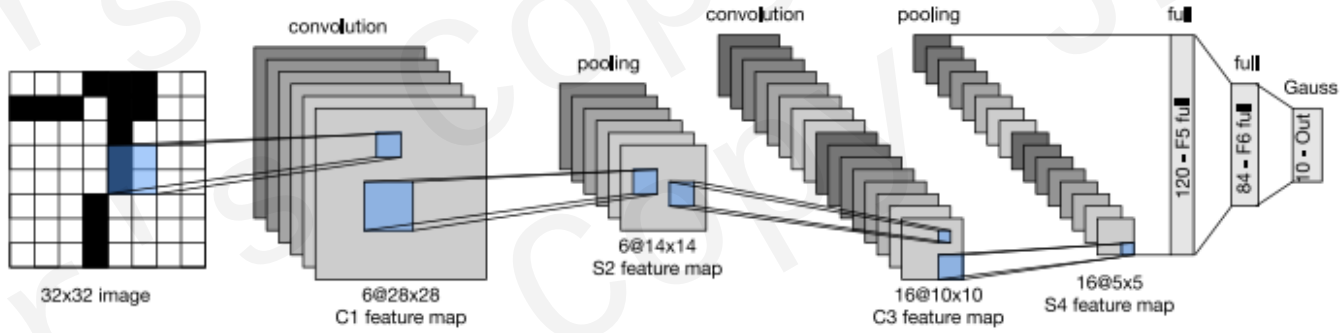
LeNet-5 (LeCun et al. 1998)

Advantages :

- convolution with learnable parameters (sharable parameters) ➔ effective way to extract similar features at multiple locations with few parameters .
- correlation with neighboring pixels (data) considered.
- optical character, fingerprint recognition…

Limitations:

- High computational burden: each pixel as separate input .
- Traditional activations functions: slow learning.



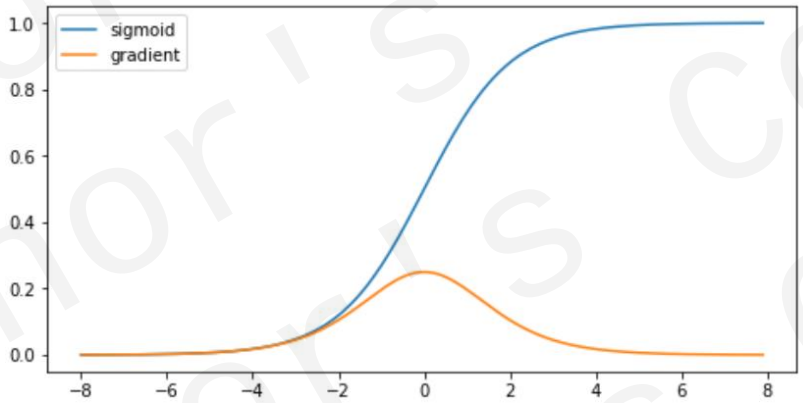
Stagnation of CNN : Early 2000

ML paradigm in 1990-1998:

- Typically such datasets were **hand generated** using very expensive sensors.
- Lacked richness, diversity ➔ insignificant improvement of performance (lack of complex training data, representations etc.
- Till 2012, feature representation had to be thought, or based on intuition.

CNNs:

- Backpropagation → not effective to reach global minima.
- Activation functions: Sigmoid function (variants)
  - vanishing gradient problem (exponential decay)
  - exploding gradient problem.



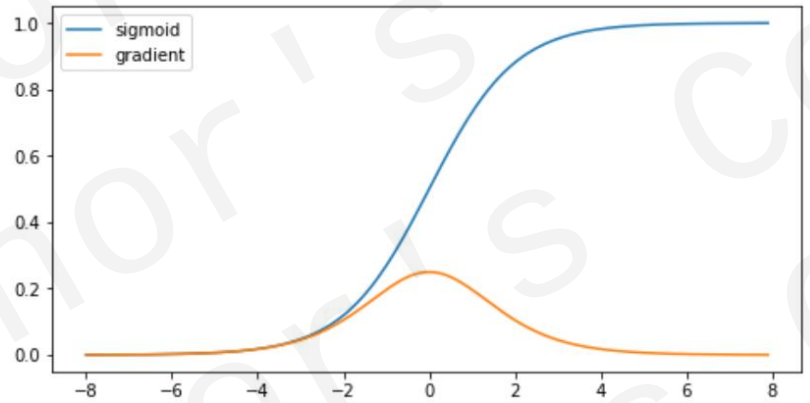
Stagnation of CNN : Early 2000

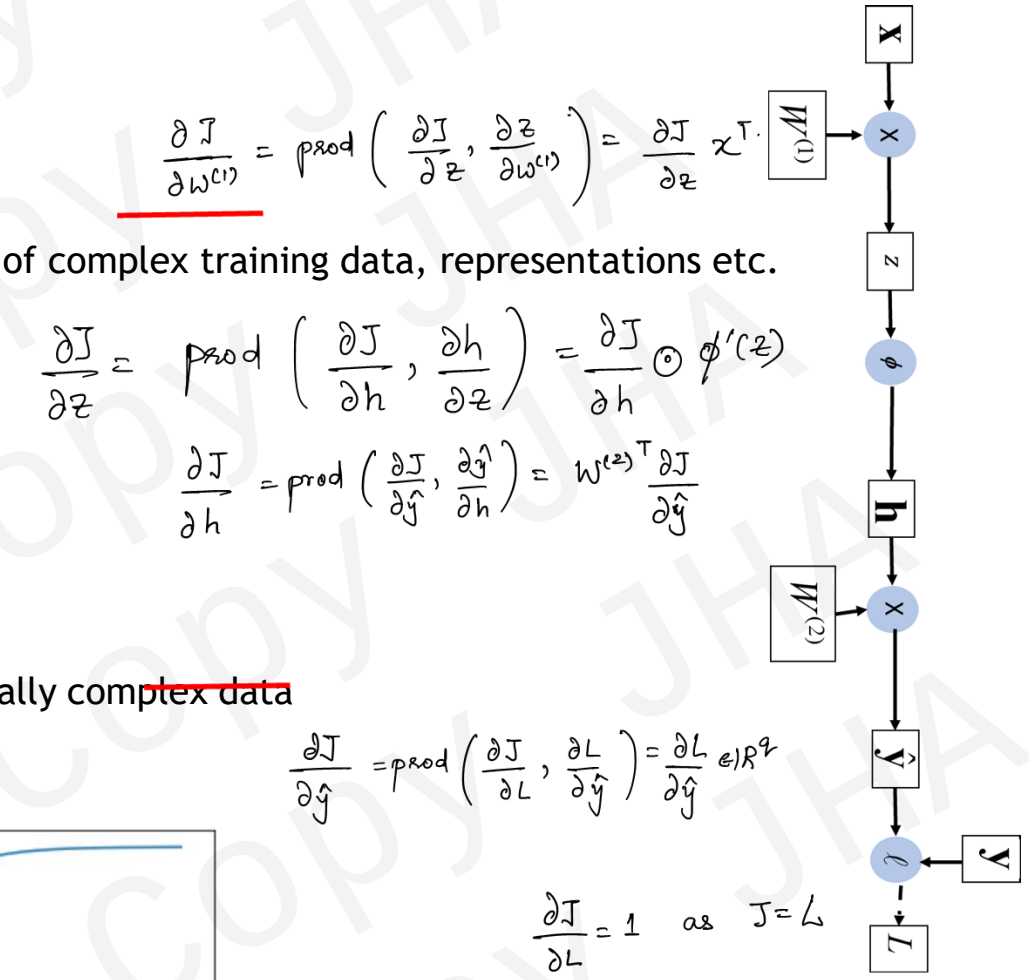
ML paradigm in 1990-1998:

- Typically such datasets were **hand generated** using very expensive sensors.
- Lacked richness, diversity ➔ insignificant improvement of performance (lack of complex training data, representations etc.
- Till 2012, feature representation had to be thought, or based on intuition.

CNNs:

- Backpropagation → not effective to reach global minima.
- Activation functions: sigmoid function (variants)
  - vanishing gradient problem (exponential decay)
  - exploding gradient problem (no efficient initialization methods)
- Little attention : object detection, classification/prediction of spatio-temporally complex data
- Limited computational resources (no GPUs)

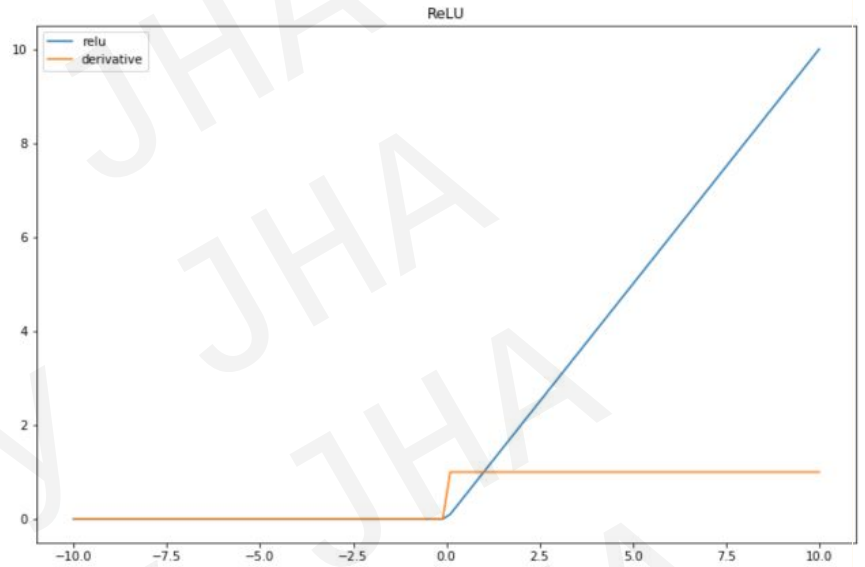


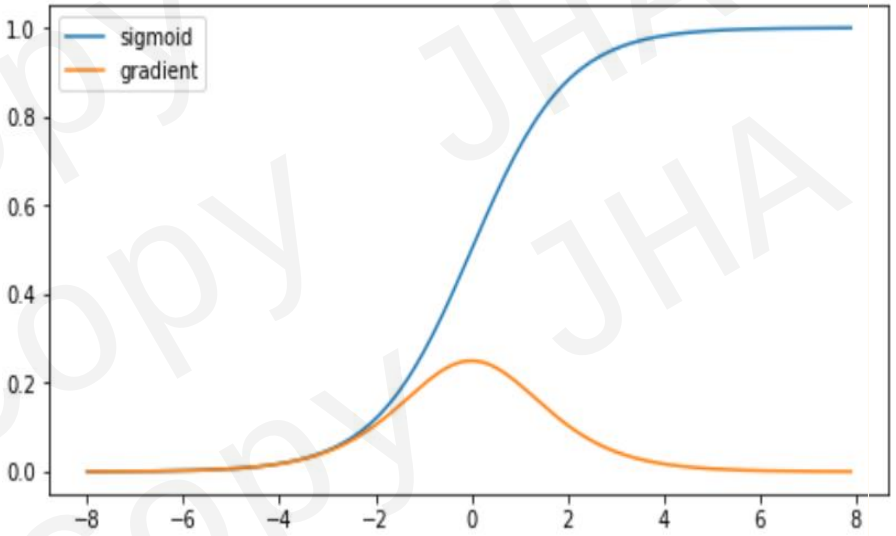


Revival of CNNs: 2006-2011

- Efficient initialization techniques:
  - greedy layer-wise pre-training (Hinton et al. 2006)
  - unsupervised/supervised training-based pre-training
  - Xavier initialization (Glorot and Bengio, 2010)
- Use of Non-saturating Activation Functions : ReLu (Glorot and Bengio, 2010)
- Max-pooling > Sub-sampling (Ranzato et al, 2007) → learnt better invariant features.

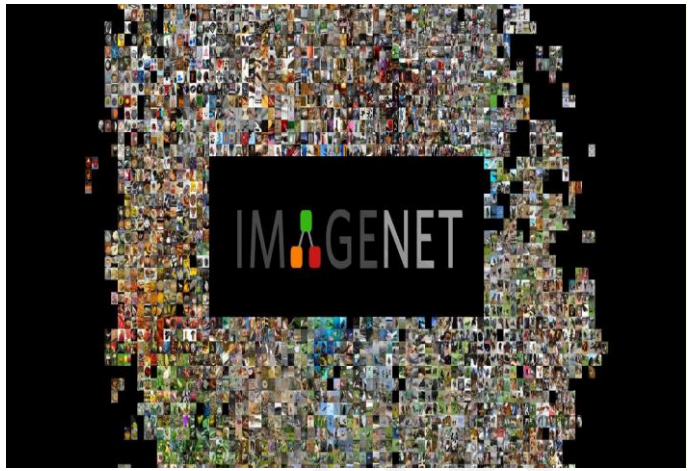
- Late 2006: GPUs for training CNNs.
- 2007: NVIDIA → CUDA programming → harness parallel processing power of GPUs





Revival of CNNs: 2006-2011

- Efficient initialization techniques:
  - greedy layer-wise pre-training (Hinton et al. 2006)
  - unsupervised/supervised training-based pre-training
  - Xavier initialization (Glorot and Bengio, 2010)
- Use of Non-saturating Activation Functions : ReLu (Glorot and Bengio, 2010)
- Max-pooling > Sub-sampling (Ranzato et al, 2007) → learnt better invariant features.

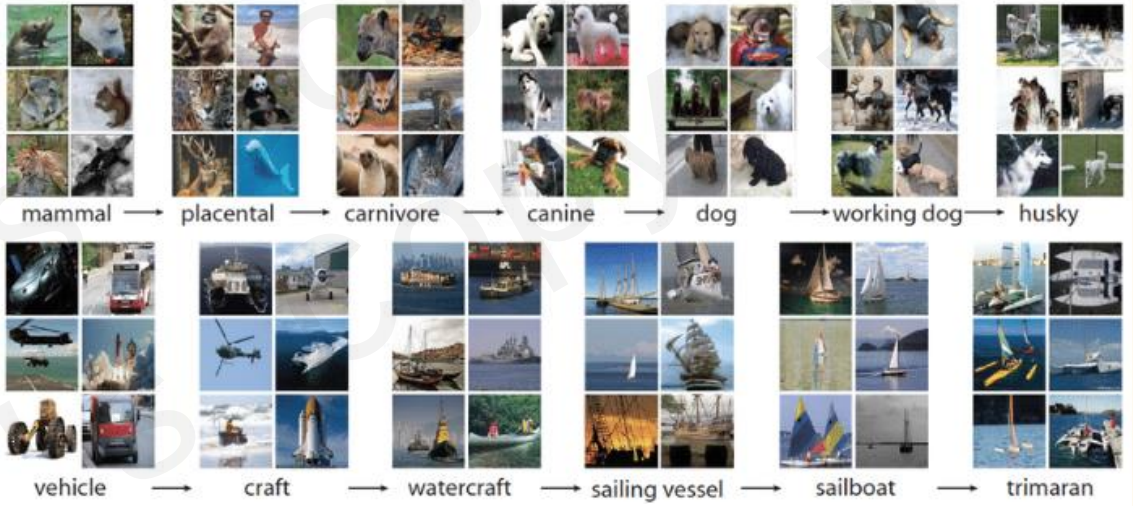


- Late 2006: GPUs for training CNNs.
- 2007: NVIDIA → CUDA programming → harness parallel processing power of GPUs

- 2010: Dr. Fei-Fei Li group (Stanford) → ImageNet platform

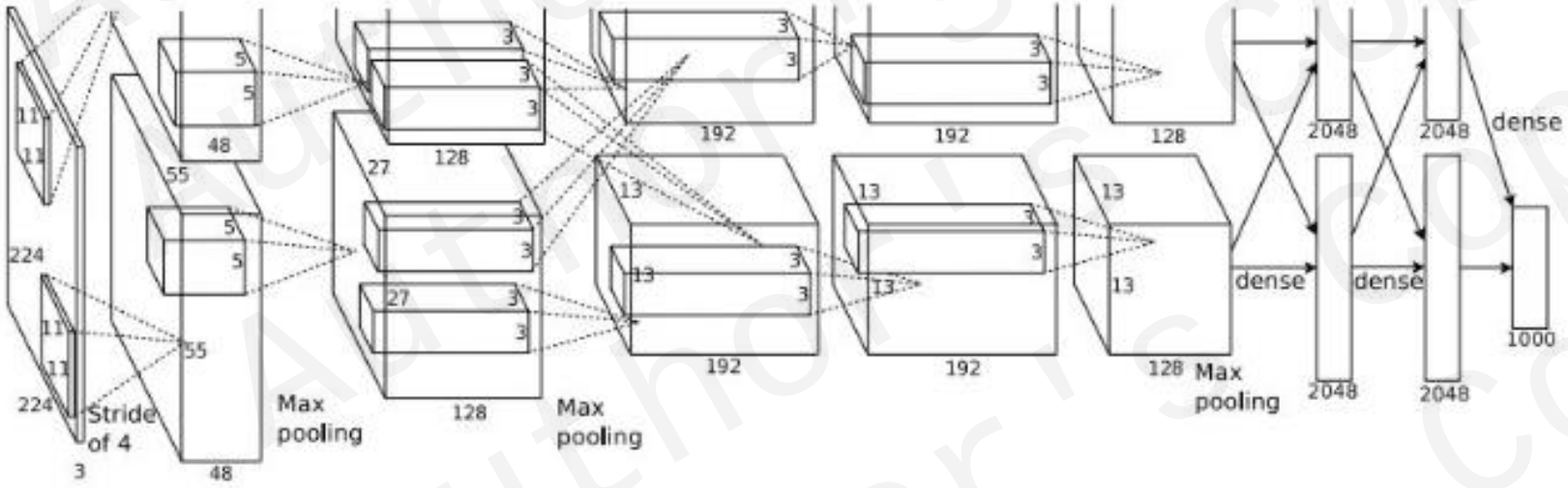
today ImageNet → 15 millions, large number categories and classes (target labels).

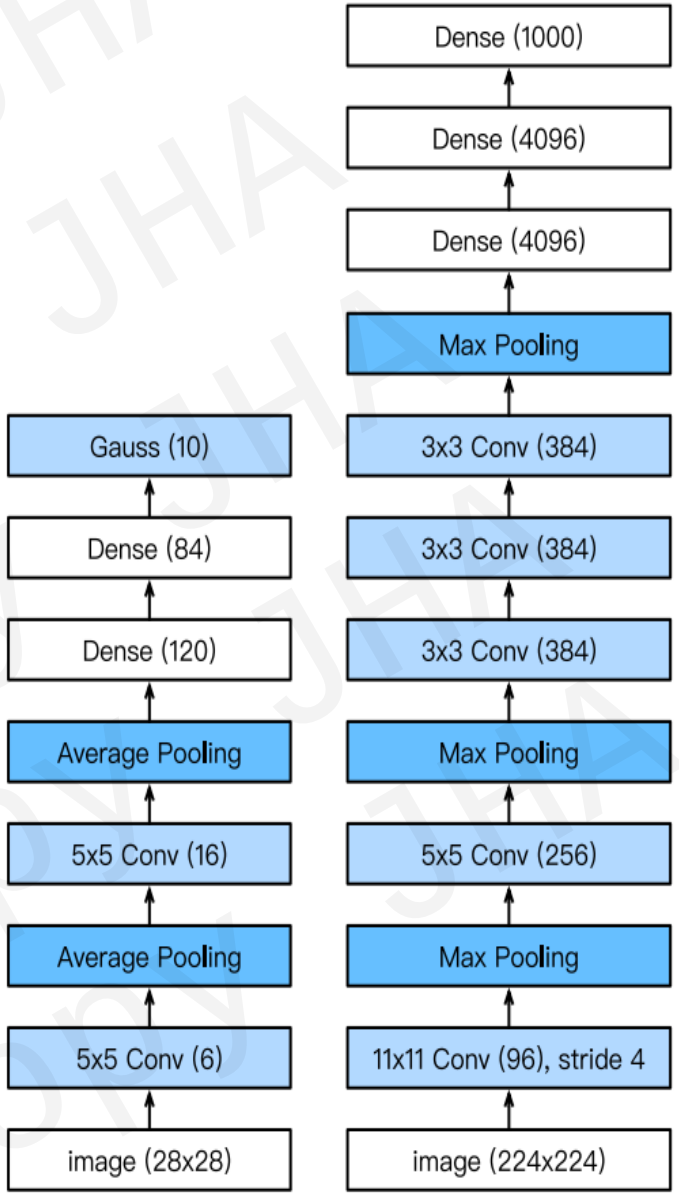
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (2010-2017)



AlexNet (Krizhevsky et al. 2012)

- Considered first “modern deep architecture”
- Deeper than LeNet-5: from 5 to 8 layers

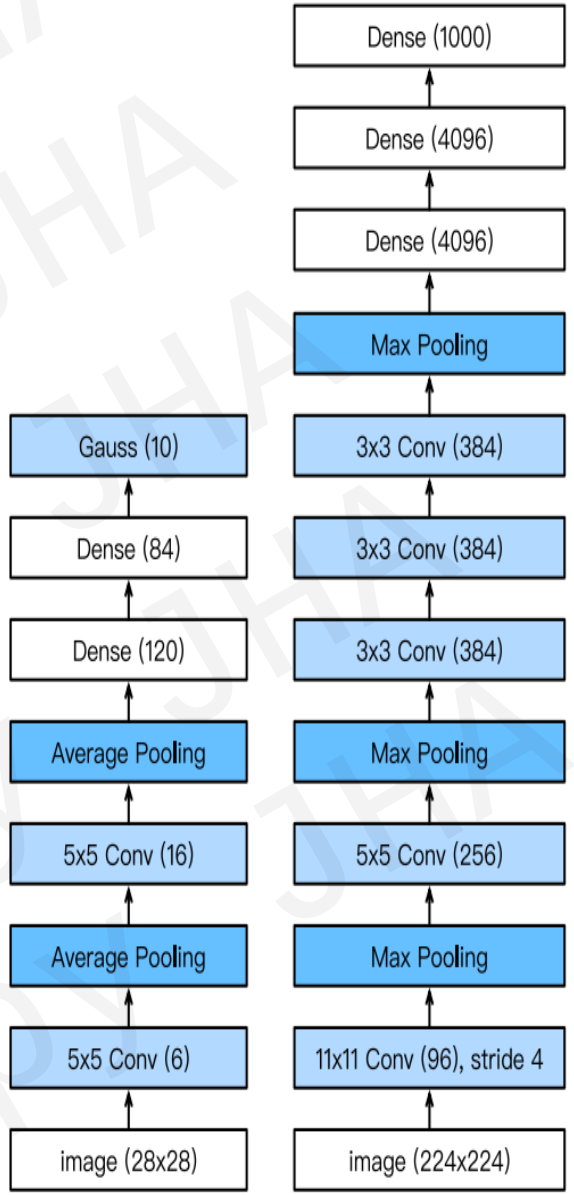




LeNet AlexNet

AlexNet (Krizhevsky et al. 2012)

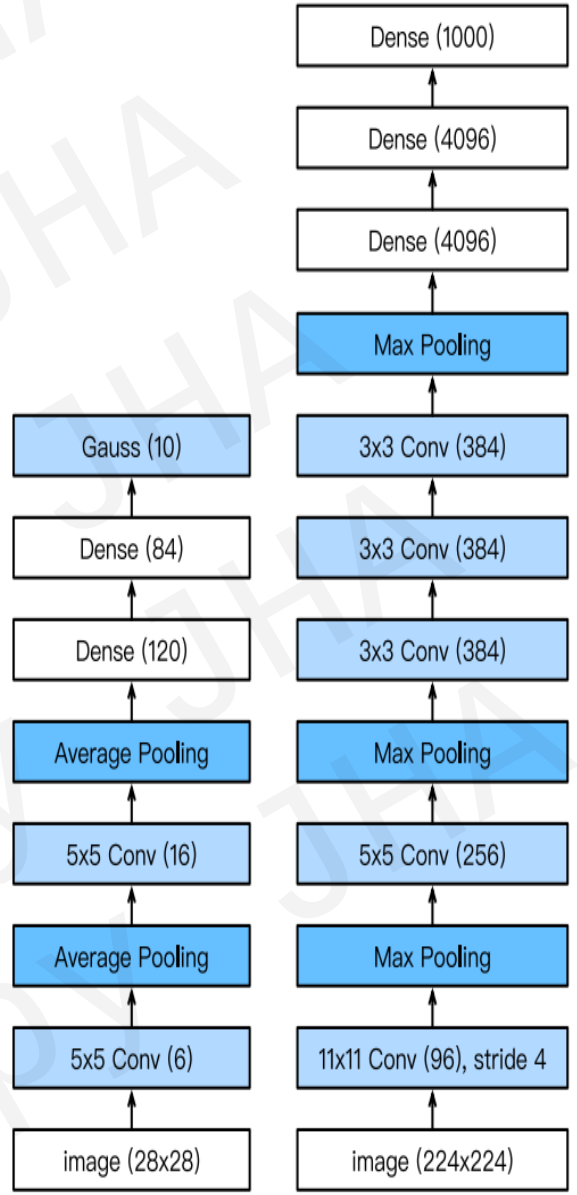
- Considered first “modern deep architecture”, deeper than LeNet-5: from 5 to 8 layers,
- 60 Million parameters
- Depth increases overfitting: learning algo: skips some transformational units.
- ReLU : improve convergence ➔ reduce vanishing gradient problem.
- Heavy data augmentation for training: flipping, clipping, color change etc.
- Use of multiple GPUs for training : trained in parallel on two NVIDIA GTX 580
- Use of large filter (11X11, 5X5) as initial layers
- Overlapping pooling layers: (0.5% reduction in overfitting).
- Other adjustments:
  - Dropout for regularization
  - SGD Momentum



LeNet AlexNet

AlexNet (Krizhevsky et al. 2012)

- Considered first “modern deep architecture”, deeper than LeNet: from 5 to 8 layers,
- 60 Million parameters
- Depth increases overfitting: learning algo: skips some transformational units.
- ReLU : improve convergence ➔ reduce vanishing gradient problem.
- Heavy data augmentation for training: flipping, clipping, color change etc.
- Use of multiple GPUs for training : trained in parallel on two NVIDIA GTX 580
- Use of large filter (11X11, 5X5) as initial layers
- Overlapping pooling layers: (0.5% reduction in overfitting).
- Other adjustments:
  - Dropout for regularization: 0.5
  - SGD Momentum
- Winner of ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012
  - Recognize off-center objects.
- Beginning of Modern era of Deep learning: SOTA
  - Deep Learning to new fields: medical imaging, data extraction, end to end learning…
- Missing ➔ A template for Deep NN design.



LeNet AlexNet

Visual Geometry Group or VGG (Simonyan and Zisserman 2015)

19 layers deeper compared to AlexNet

Addition:

- Studied the relation of depth with the representational capacity of the network.
- Replaced: large kernel-sized with small receptive field (multiple 3×3 kernels).
- All hidden layers: ReLu activation.
- Suggested that small size filters can improve the performance of the CNNs

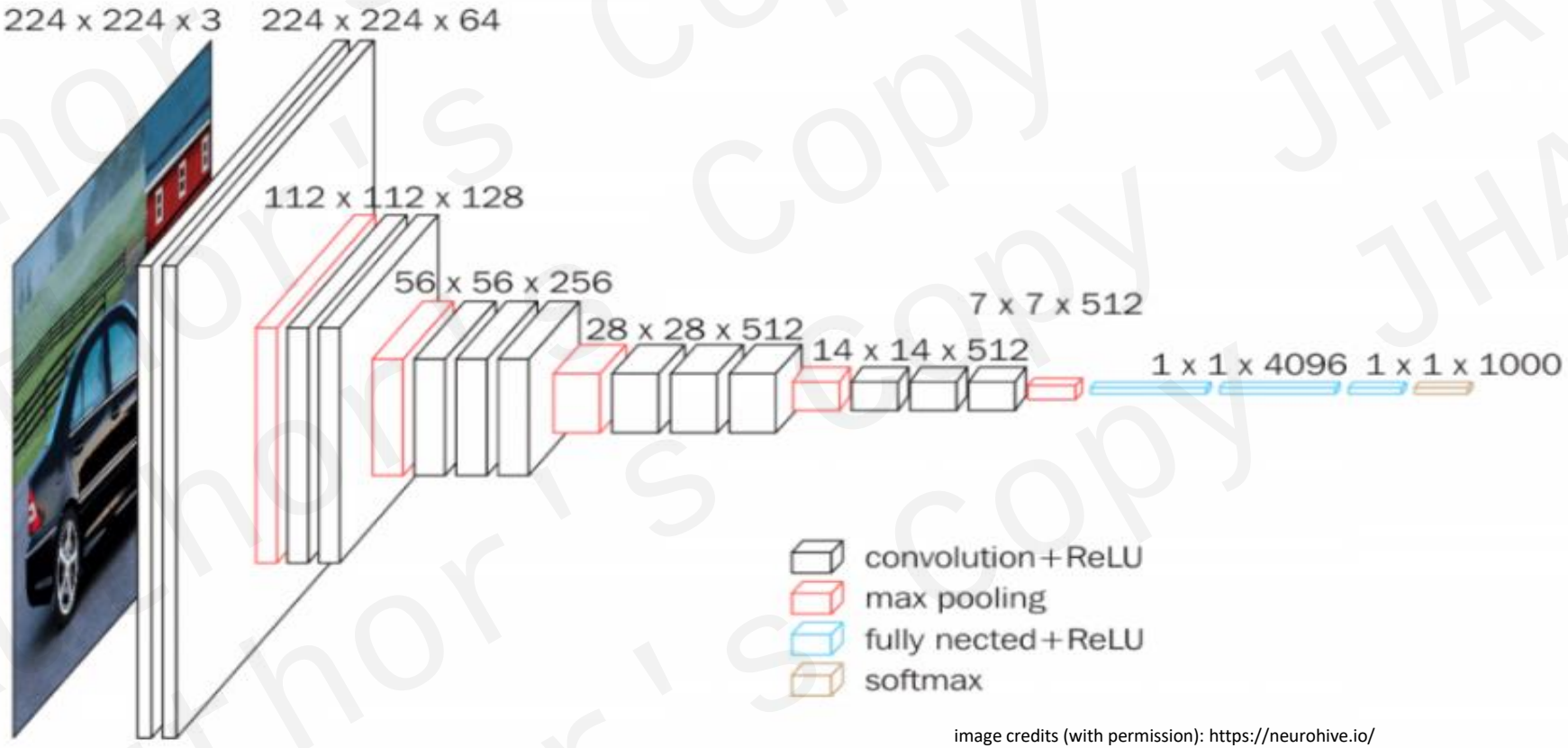


image credits (with permission): https://neurohive.io/

Visual Geometry Group or VGG (Simonyan and Zisserman 2015)

Dataset:

- ImageNet , inputs down-sampled → 256×256

Architecture:

- Image passed through a stack of convolutional (conv.) layers, with filters → with a very small receptive field: 3×3
  - The convolution stride is fixed to 1 pixel
  - the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers.
  - Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling).
  - Max-pooling is performed over a 2×2 pixel window, with stride 2.
  - Complexity regulation: 1X1 convolutions between conv layers (learn linear combination of resultant feature maps)
  - Followed by: Three Fully-Connected (FC) layers : 4096, 4096,1000 (for ILSVRC classification)
  - The final layer is the soft-max layer.
  - The configuration of the fully connected layers is the same in all networks.

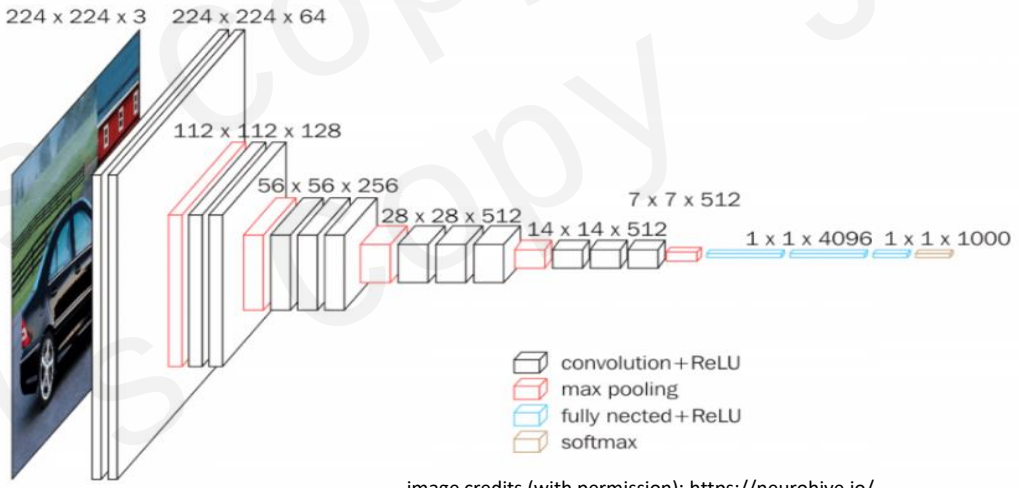


image credits (with permission): https://neurohive.io/

Visual Geometry Group or VGG (Simonyan and Zisserman 2015)

19 layers deeper compared to AlexNet

Addition:

- Studied the relation of depth with the representational capacity of the network.
- Replaced: large kernel-sized with small receptive field (multiple 3×3 kernels).
- All hidden layers: ReLu activation.

Advantages:

- Significantly outperformed previous generation models with respect to classification accuracy.
- Representation depth is beneficial for the classification accuracy.
- Suggested that small size filters can improve the performance of the CNNs.
- Several layers of deep and narrow convolutions (i.e., 3×3) were more effective than fewer layers of wider convolutions.
- 2nd Place 2014-ILSVRC

Set the trend: smaller sized filters.

Limitations:

- Very slow to train (For example: VGG16 was trained for weeks , NVIDIA Titan Black GPU's)
- Large no pf parameters 138 million parameters
- Heavy architecture → 533MB

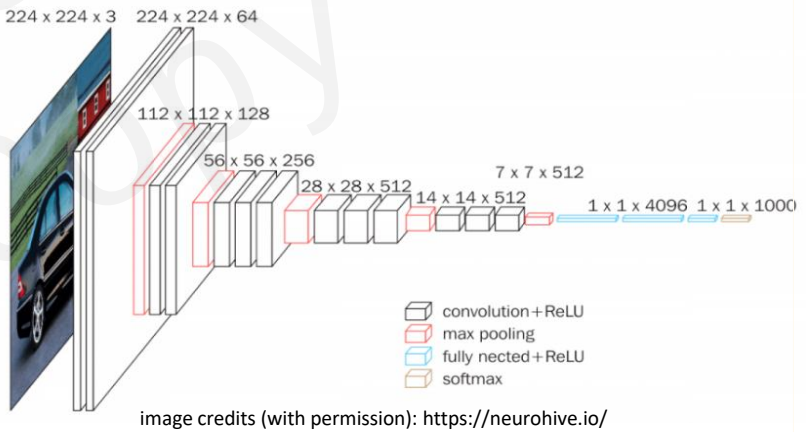
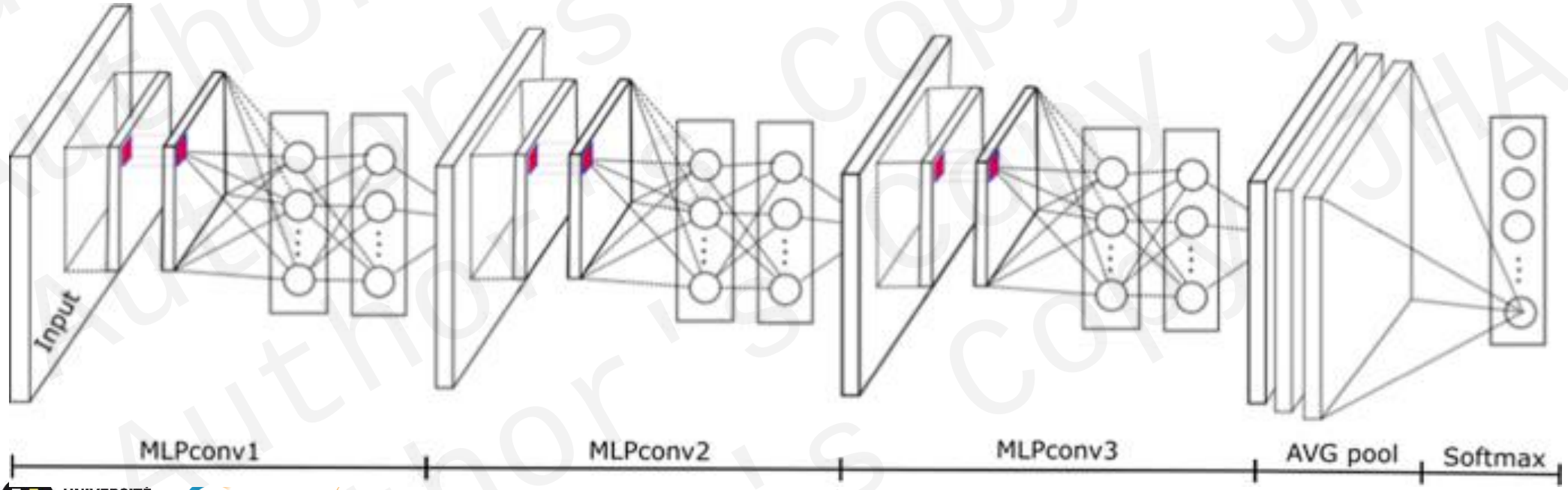


image credits (with permission): https://neurohive.io/

Network in Network (NiN) (Lin et al., 2013)

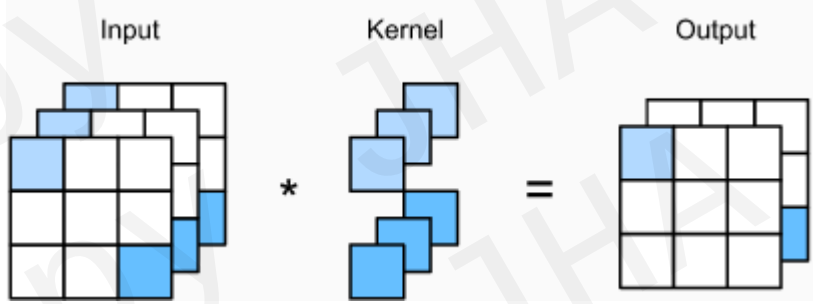
- Intuition:
  - to use an MLP on the channels for each pixel separately.
  - Apply a fully-connected layer at each pixel location (for each height and width).

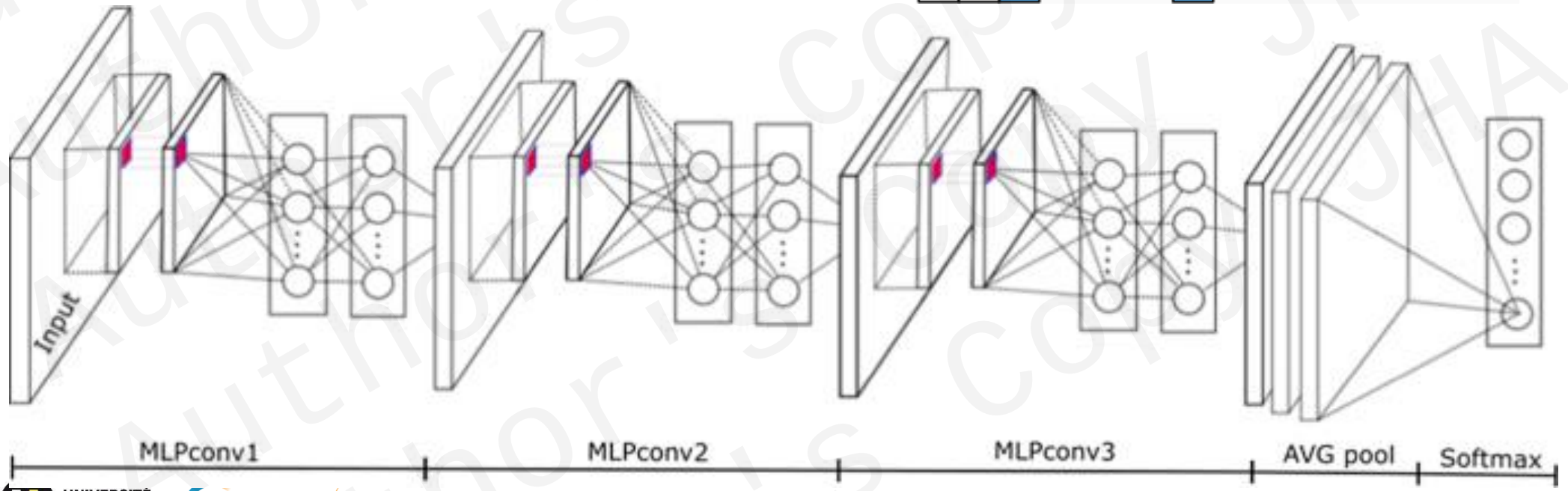


Source: Alaeddine, H., Jihene, M. Deep network in network. *Neural Comput & Applic* (2020).

Network in Network (NiN) (Lin et al., 2013)

- Intuition:
  - to use an MLP on the channels for each pixel separately.
  - Apply a fully-connected layer at each pixel location (for each height and width).
  - If we tie the weights across each spatial location becomes ➔ 1X1 convolution layer.
    or
    fully-connected layer acting independently on each pixel location

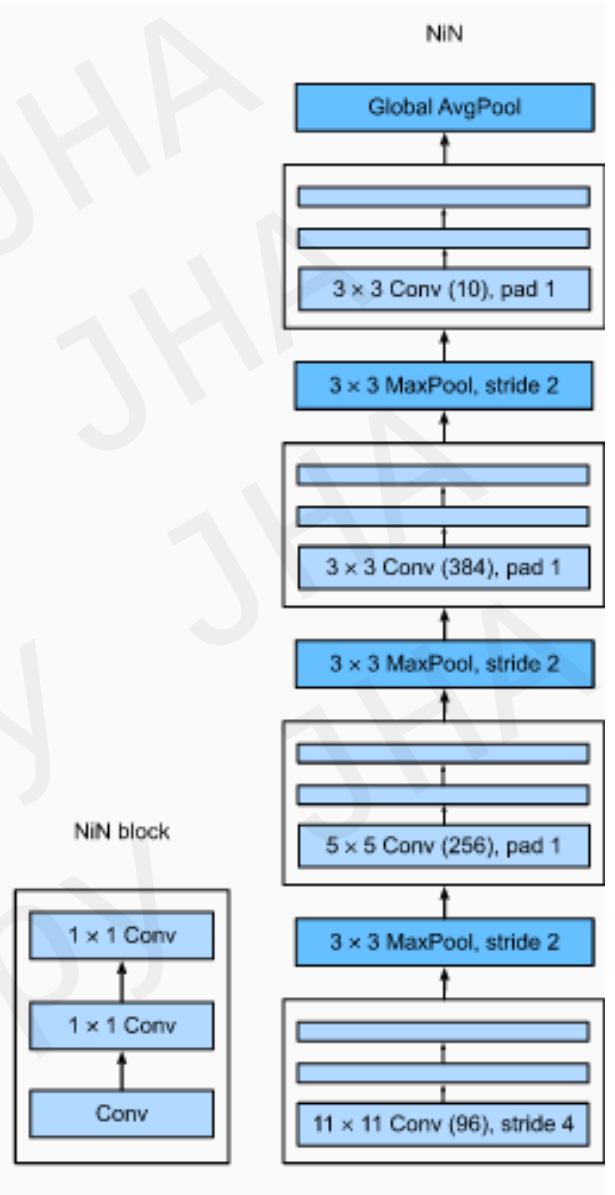




Source: Alaeddine, H., Jihene, M. Deep network in network. *Neural Comput & Applic* (2020).

Network in Network (NiN) (Lin et al. 2013)

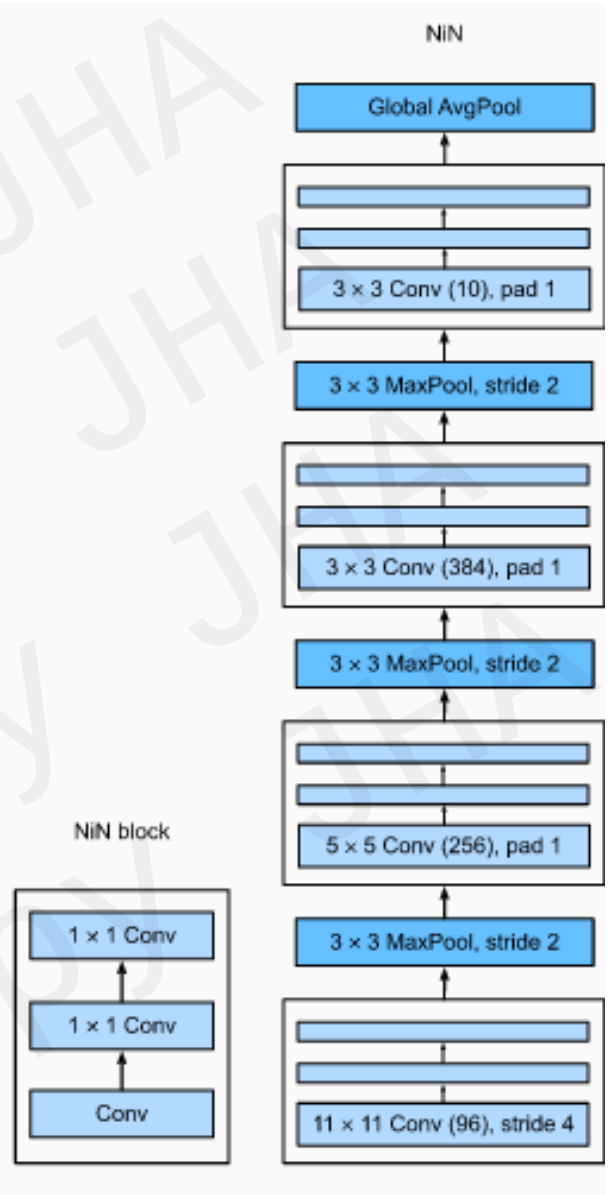
- Architecture:
  - inspired from AlexNet.
  - Convolutional layers: 11×11, 5×5, and 3×3
    - followed by two 1×1 convolutional layers that act as per-pixel fully-connected layers with ReLU activations
    - Each NiN block is followed by a maximum pooling layer (stride 2, window shape of 3×3).
  - The convolution window shape of the first layer is typically set by the user.
  - Output: number of output channels equal to the number of label classes, followed by a *global* average pooling layer.
  - Avoids fully-connected layers totally (against AlexNet, LeNet…)
- Advantages:
  - 1X1 convolutions ➔ allow for more per-pixel nonlinearity within convolutional stack.
  - NiN removes the fully-connected layers and replaces them with global average pooling.
    - Removing fully-connected layers reduces overfitting.
  - NiN has dramatically less parameters.



Source: Dive into Deep learning book

Network in Network (NiN) (Lin et al. 2013)

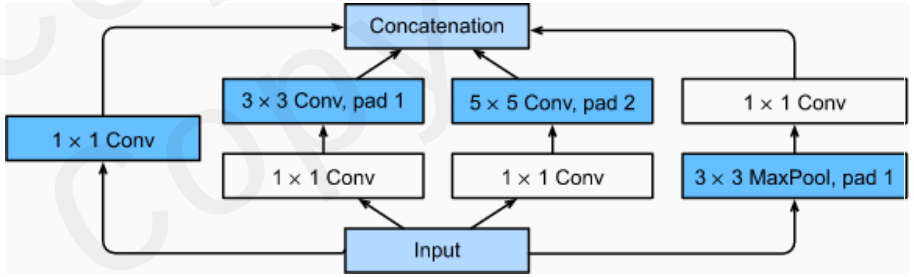
- Architecture:
  - inspired from AlexNet.
  - Convolutional layers: 11×11, 5×5, and 3×3
    - followed by two 1×1 convolutional layers that act as per-pixel fully-connected layers with ReLU activations
    - Each NiN block is followed by a maximum pooling layer (stride 2, window shape of 3×3).
  - The convolution window shape of the first layer is typically set by the user.
  - Output: number of output channels equal to the number of label classes, followed by a *global* average pooling layer.
  - Avoids fully-connected layers totally (against AlexNet, LeNet…)
- Advantages:
  - 1X1 convolutions ➔ allow for more per-pixel nonlinearity within convolutional stack.
  - NiN removes the fully-connected layers and replaces them with global average pooling.
    - Removing fully-connected layers reduces overfitting.
  - NiN has dramatically less parameters.



Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015)

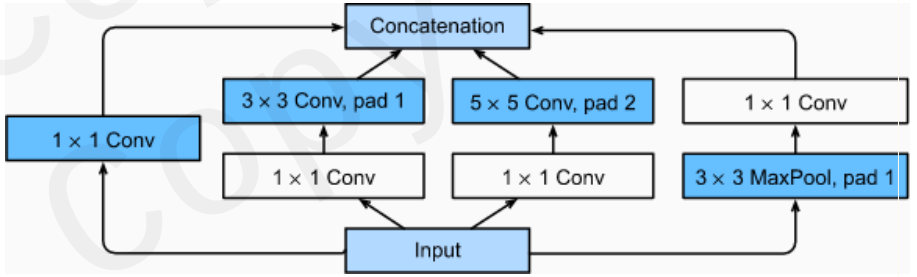
- Winner of 2014 ILSVRC
- One focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
  - incorporates multi-scale convolutional transformations using split, transform and merge idea.
  - encapsulates filters of different sizes (1x1, 3x3, and 5x5)
  - captures spatial information at different scales: fine and coarse grain level.



Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015)

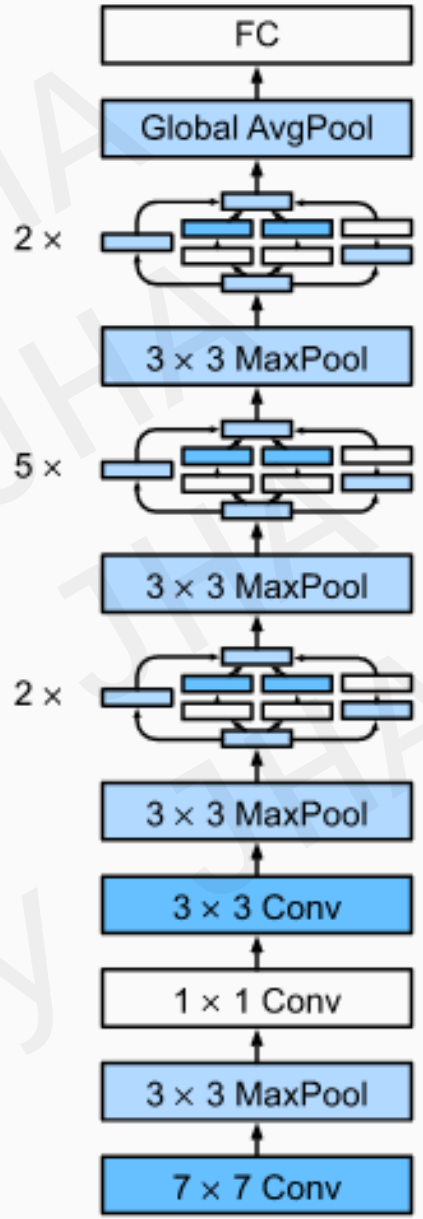
- Winner of 2014 ILSVRC
- One focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
  - incorporates multi-scale convolutional transformations using split, transform and merge idea.
  - encapsulates filters of different sizes (1x1, 3x3, and 5x5)
  - captures spatial information at different scales: fine and coarse grain level.
  - computation regularization ➔ adding a bottleneck layer of 1x1 convolutional filter, before employing large size kernels.



Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015) or Inception V1

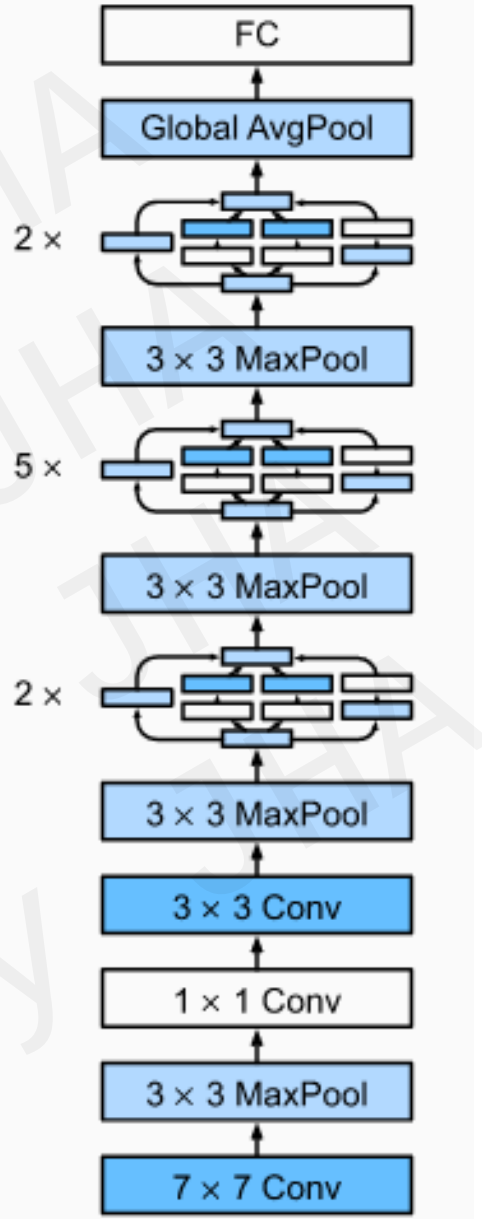
- Winner of 2014 ILSVRC
- One of the focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
  - Incorporates multi-scale convolutional transformations using **split, transform and merge idea.**
  - Encapsulates filters of different sizes (1x1, 3x3, and 5x5)
  - Captures spatial information at different scales: fine and coarse grain level.
  - Computation regularization ➔ adding a bottleneck layer of 1x1 convolutional filter, before employing large size kernels.
- Advantages:
  - Density reduced → use of global average pooling at the last layer and NOT instead of using a fully connected layer
  - Significant decrease in parameters: from 138 Million to 4 Million parameters.
  - Other novelties:
    - Batch Normalization
    - RmsProp as optimizer,…



Source: Dive into Deep learning book

GoogLeNet (Szegedy et al., 2015) or Inception V1

- Winner of 2014 ILSVRC
- One of the focus: Which sized convolution kernels are best (1X1, 3X3, 11X11 …) ?
- Introduced *Inception* block:
  - Incorporates multi-scale convolutional transformations using **split, transform and merge idea.**
  - Encapsulates filters of different sizes (1x1, 3x3, and 5x5)
  - Captures spatial information at different scales: fine and coarse grain level.
  - Computation regularization ➔ adding a bottleneck layer of 1x1 convolutional filter, before employing large size kernels.
- Advantages:
  - Density reduced → use of global average pooling at the last layer and NOT instead of using a fully connected layer
  - Significant decrease in parameters: from 138 Million to 4 Million parameters.
  - Other novelties:
    - Batch Normalization
    - RmsProp as optimizer,…
- Limitations:
  - heterogeneous topology that needs to be customized from module to module
  - representation bottleneck that drastically reduces the feature space
  - in the next layer and thus sometimes may lead to loss of useful information.
- Variants: Inception V2, Inception V3



Source: Dive into Deep learning book

ResNet (He et al., 2015)

Problem: Deeper networks do not necessarily lead to better accuracy.

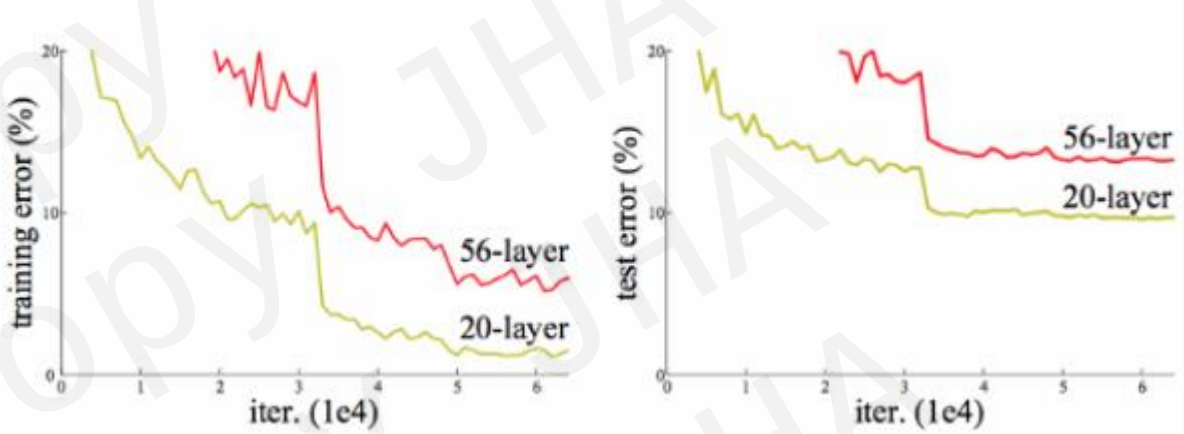
ResNet (He et al. 2015)

Problem: Deeper networks do not necessarily lead to better accuracy. WHY?

Vanishing gradients? (infinitesimally small gradients?)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

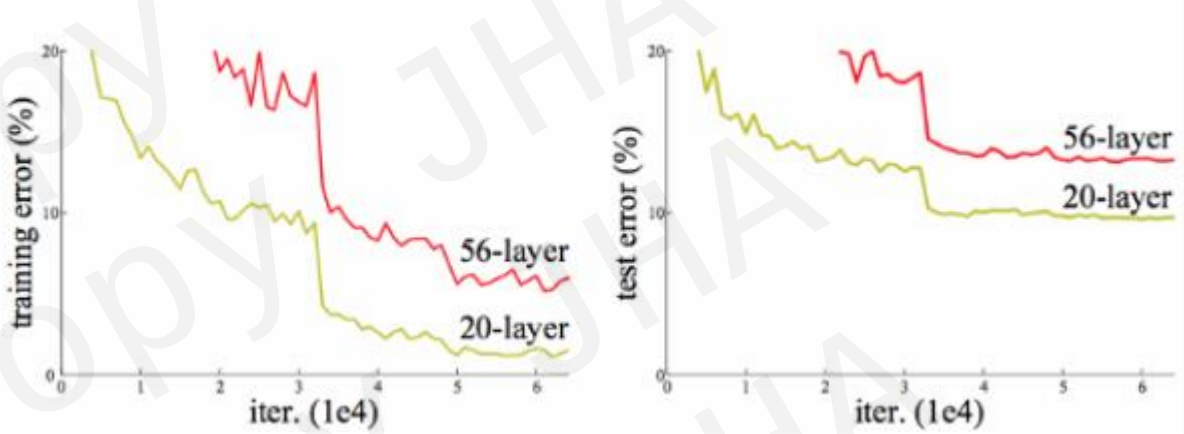


Source: Dive into Deep learning book

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Overfitting ?



Source: Dive into Deep learning book

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

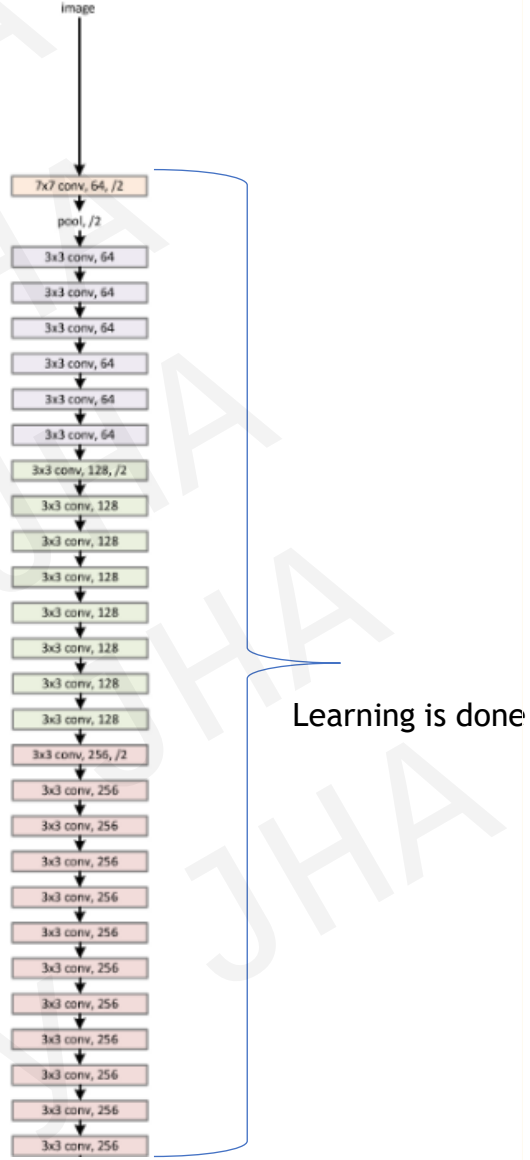


Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

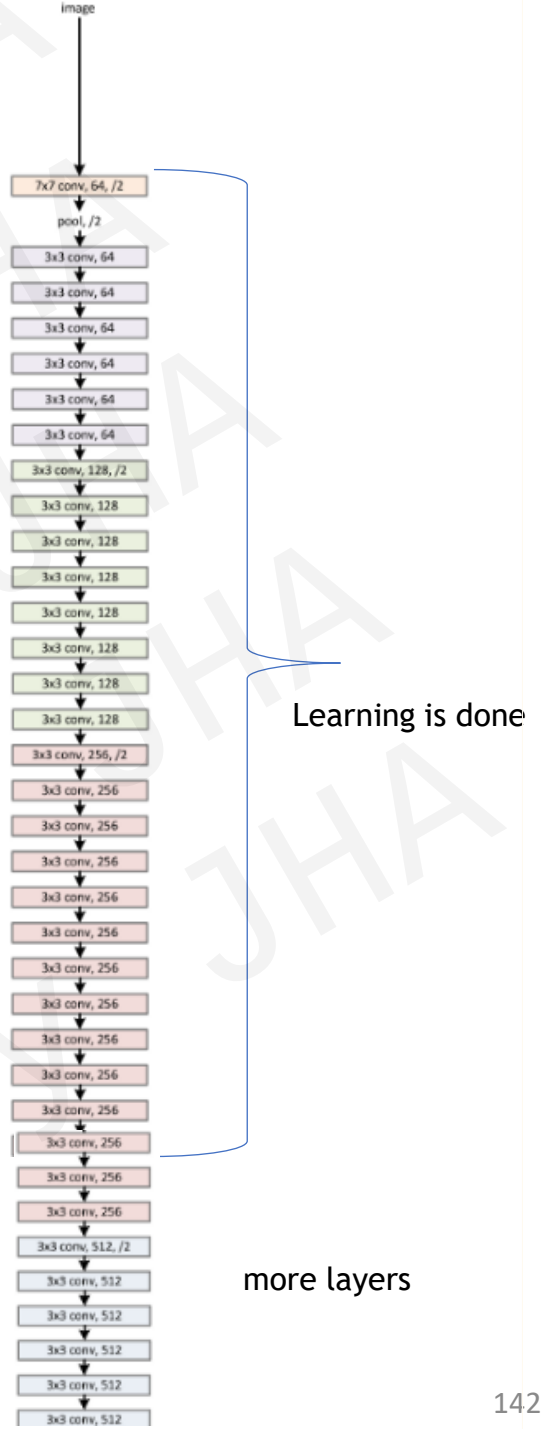
Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.



Should behave as Identity Function (let the input from previous layer flow ahead)

$$f(x) = x$$

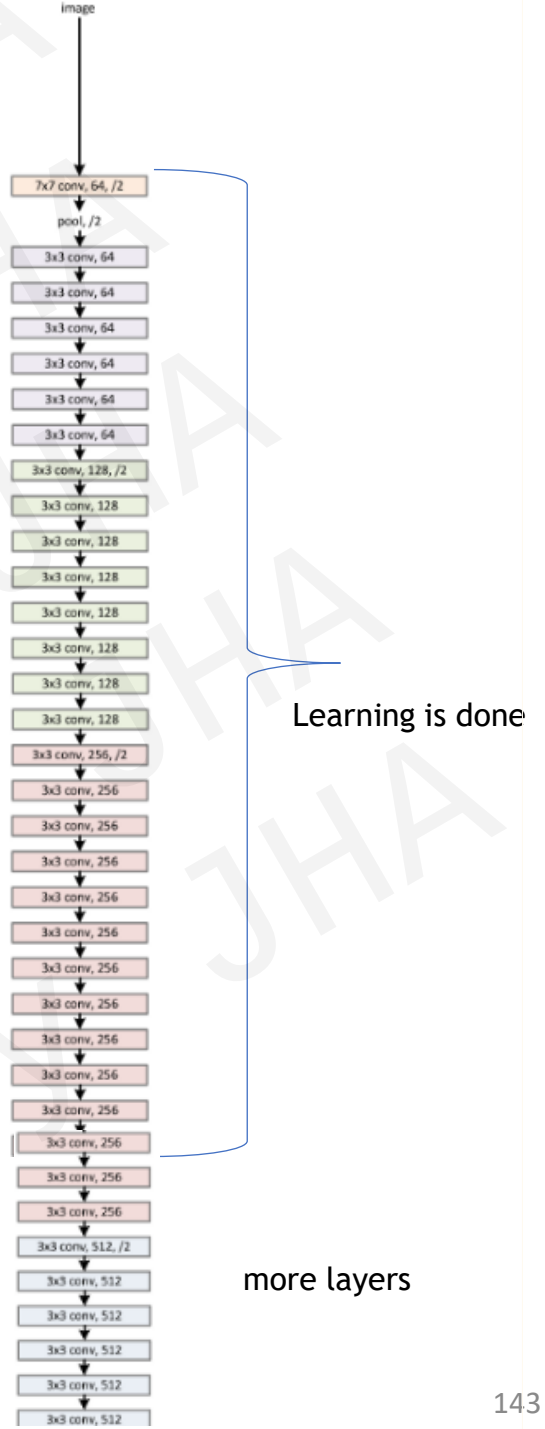
Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.



Should behave as Identity Function (let the input from previous layer flow ahead)

$$f(x) = x$$

Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

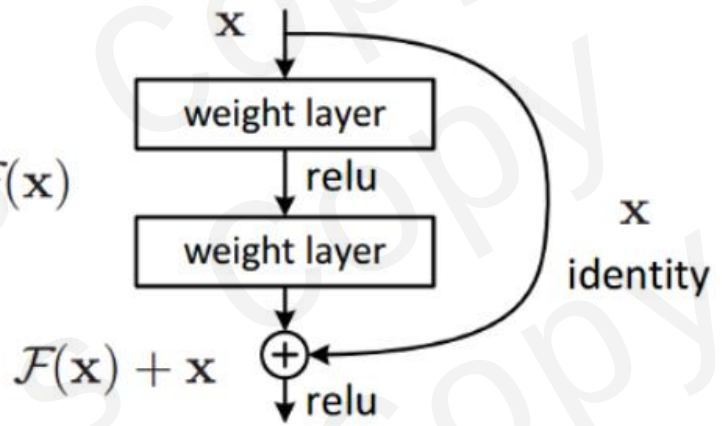
Observation: Training accuracy dropped when the count of layers was increased.

Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

Intuition :

- Learn Residual mapping



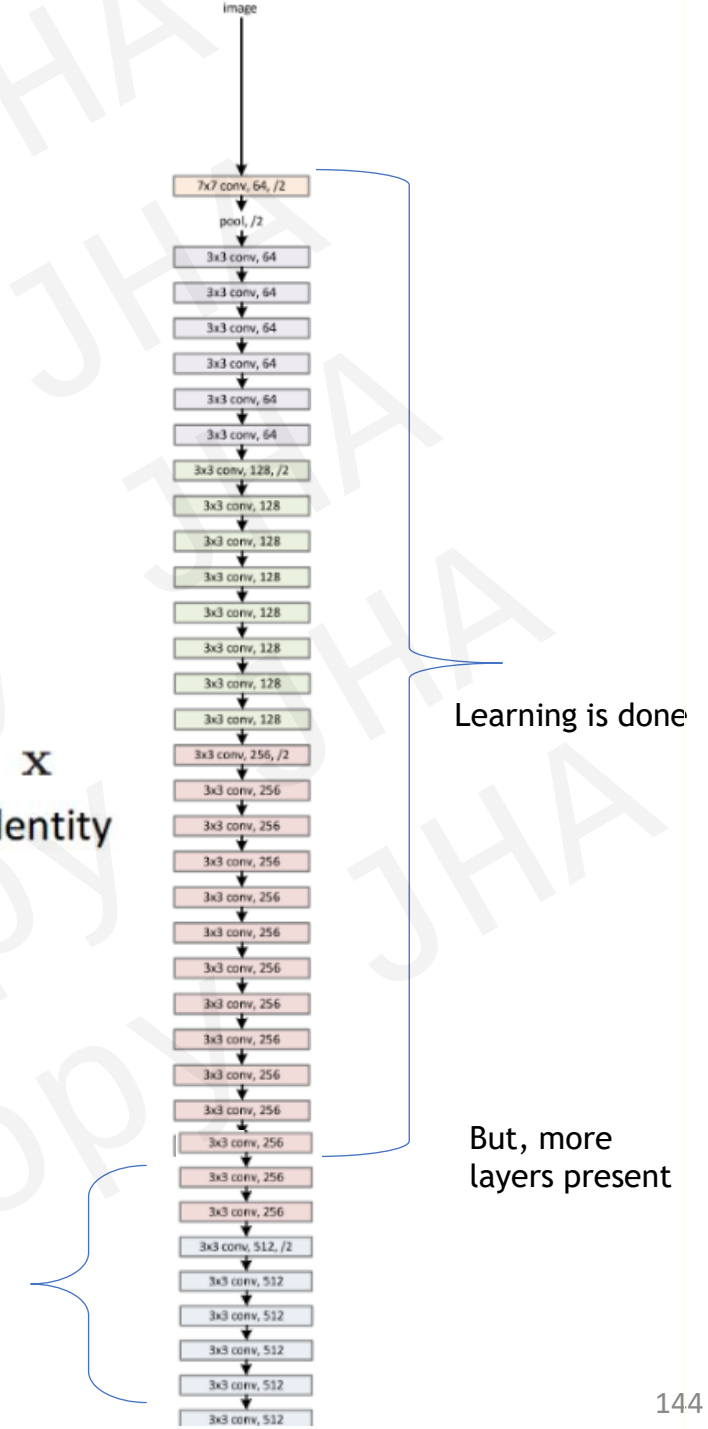


Image Credits: (He et al. 2015)

ResNet (He et al. 2015)

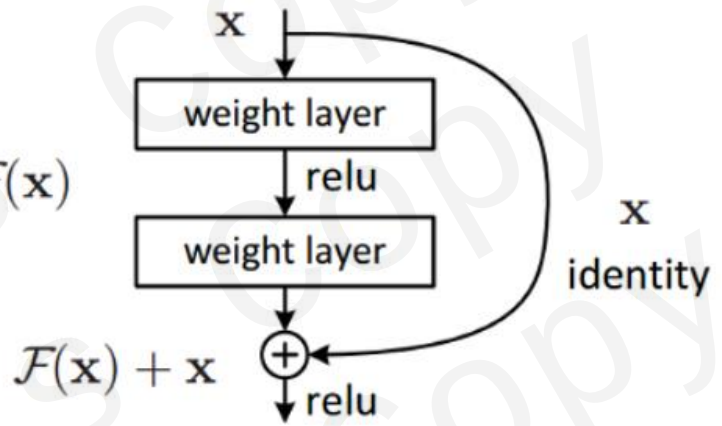
Observation: Training accuracy dropped when the count of layers was increased.

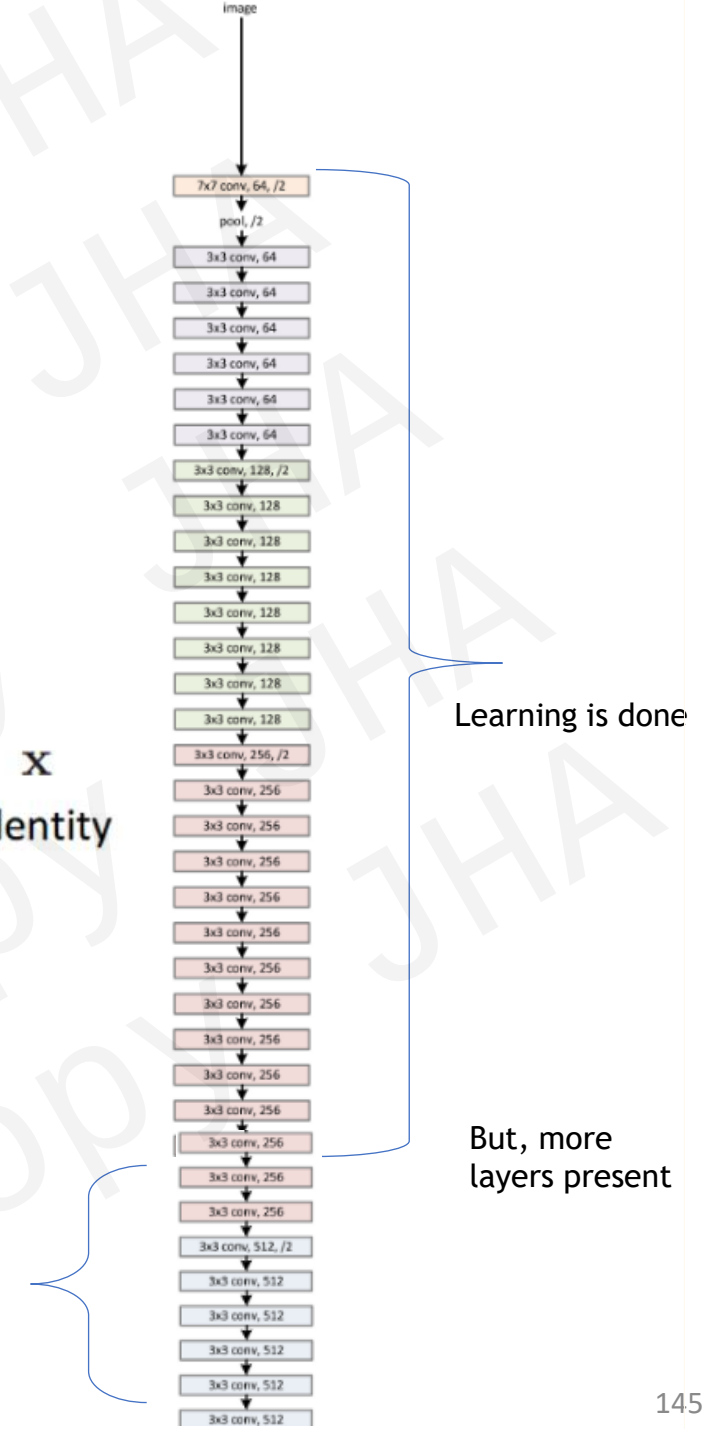
Degradation Problem:

With the network depth increasing, the accuracy saturates and then begins to degrade rapidly if more layers are introduced.

Intuition :

- Learn Residual mapping
- Use skip connections
- If any layer hurts performance ➔ skip it!
- Easier to learn $F(x) = 0$ so that it behaves as identity function.





Learning is done

But, more layers present

ResNet (He et al. 2015)

Architecture:

- Identity Block: skip connections
- Conv block: restructure incoming data
- 153 layers Deep
- Less computational complexity (but deeper : 20 X AlexNet, 8 X VGG)

Advantage:

- Residual mapping can learn the identity function more easily
- Stacking more layers ➔ equivalent to stacking identity mappings
- Inputs can forward propagate faster through the residual connections across layers.

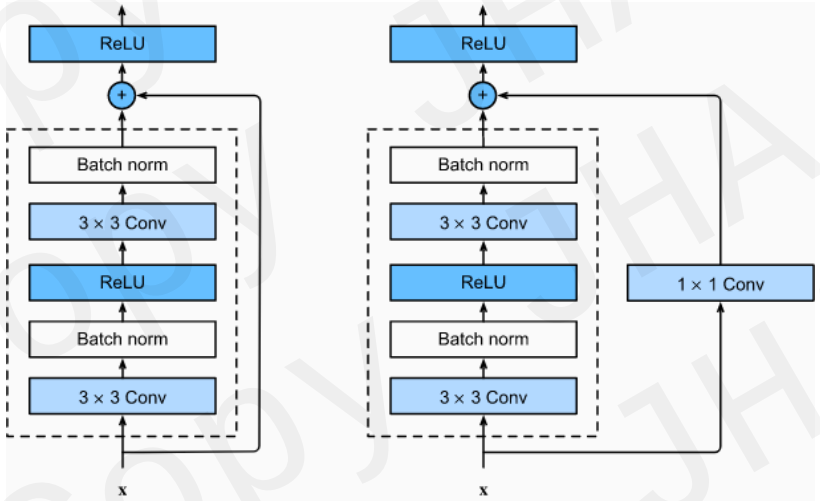


Image Credits: Dive into Deep learning

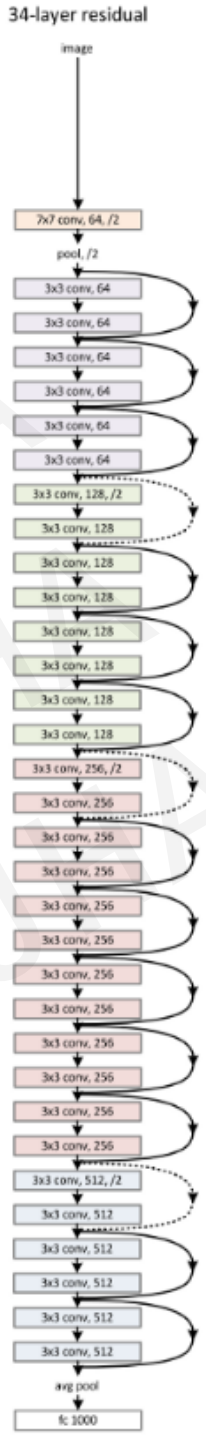
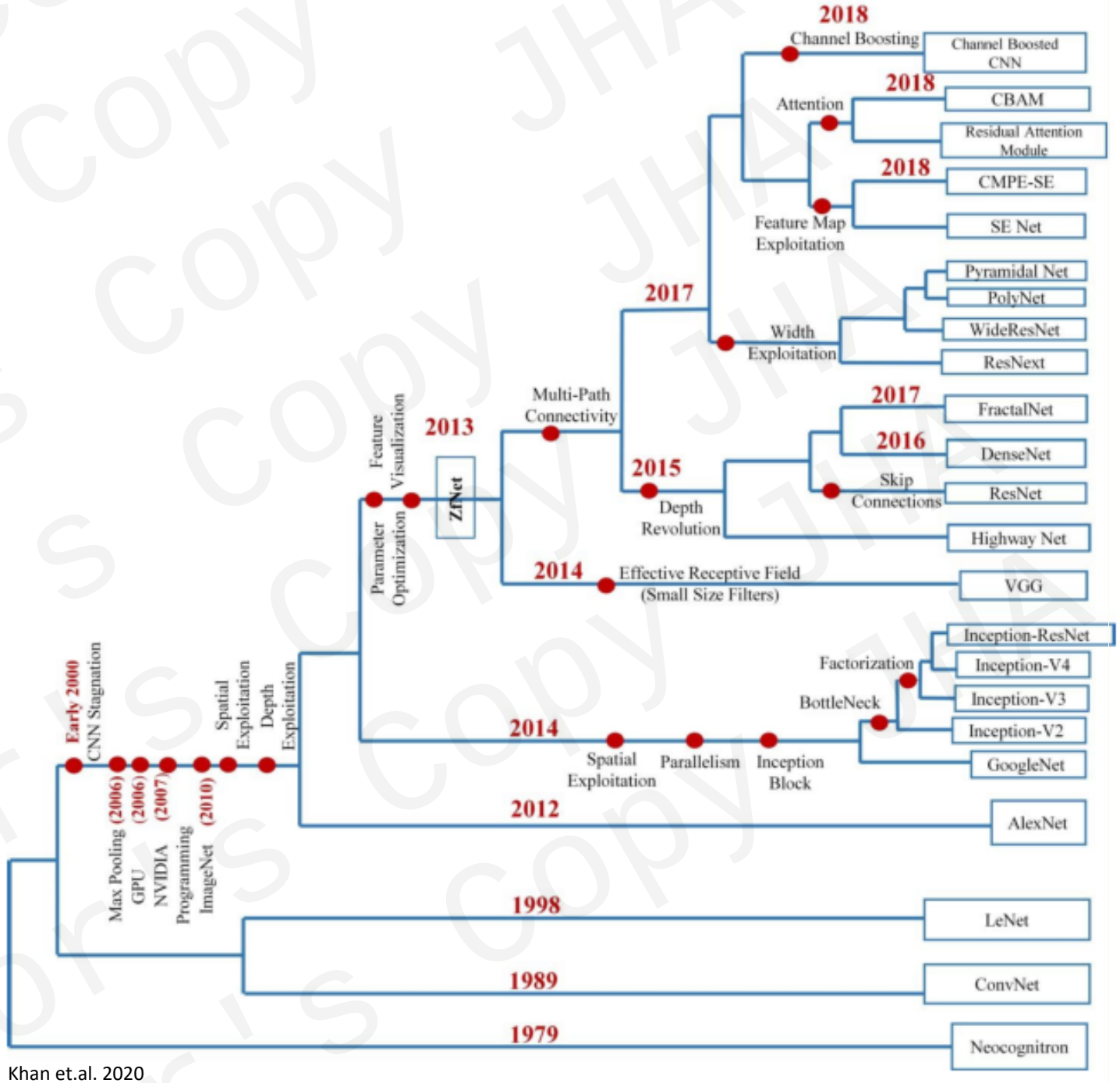


Image Credits: (He et al. 2015)

Where are we?



Khan et.al. 2020