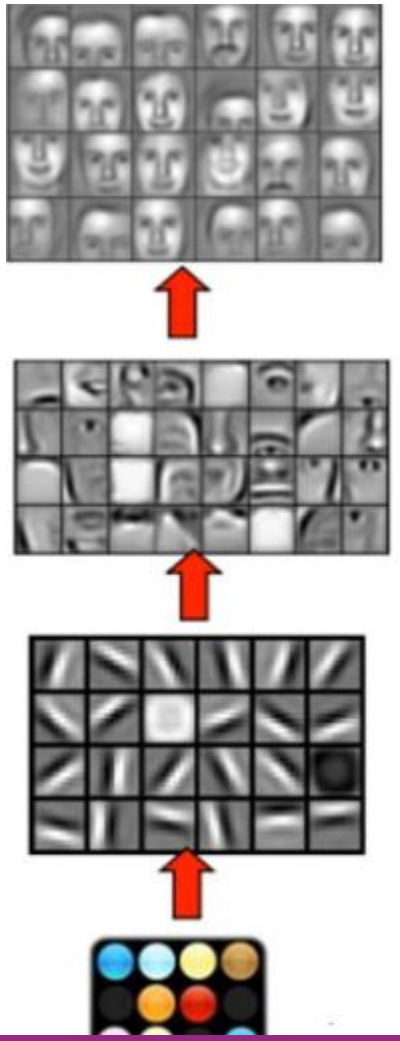
Basis of Convolutional Neural Networks



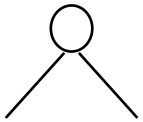
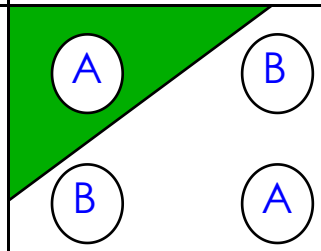
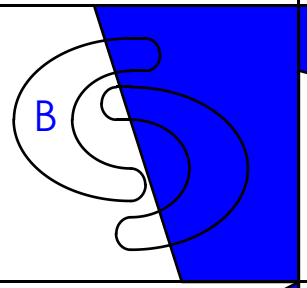
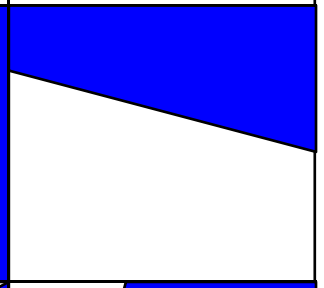
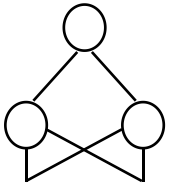
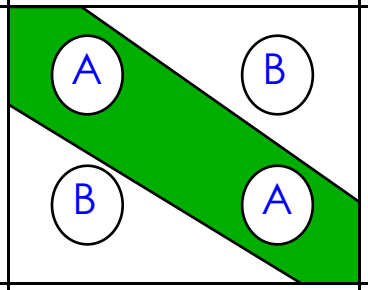
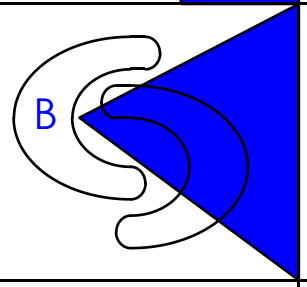
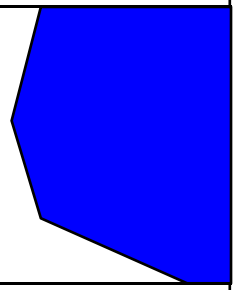
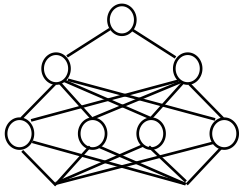
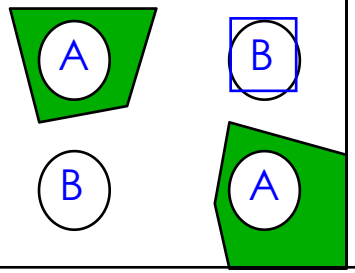
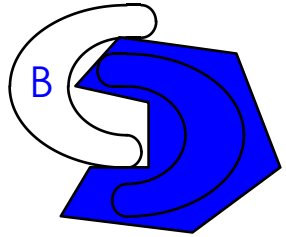
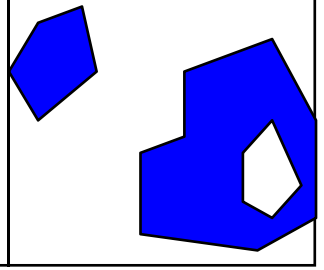
Mingmin Chi

Email: mmchi@fudan.edu.cn

Homepage: http://homepage.fudan.edu.cn/mingmin

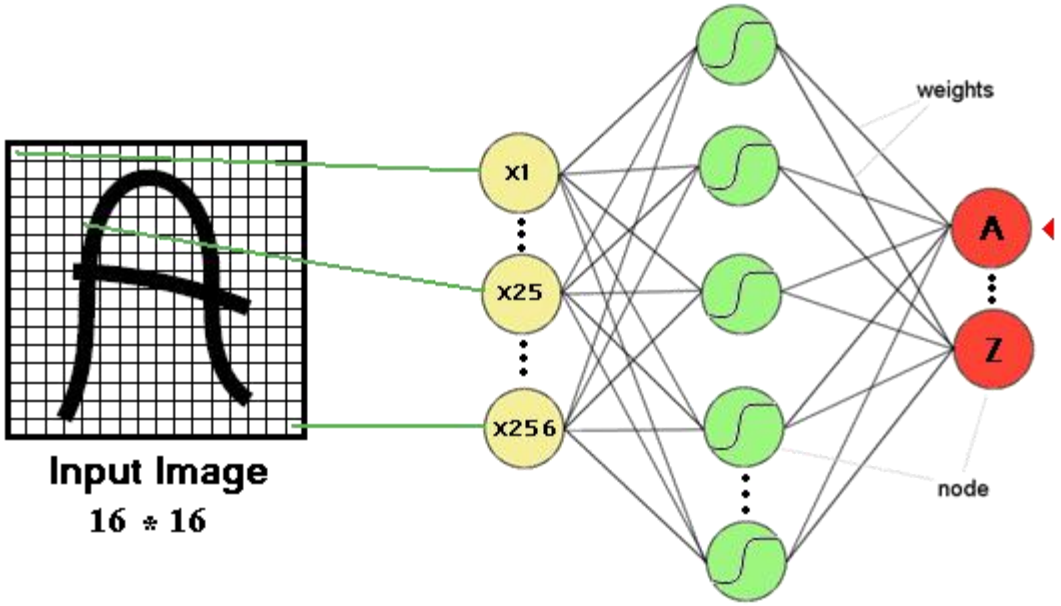
Drawbacks of MLPNN

Behavior of MLPNN

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyper plane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

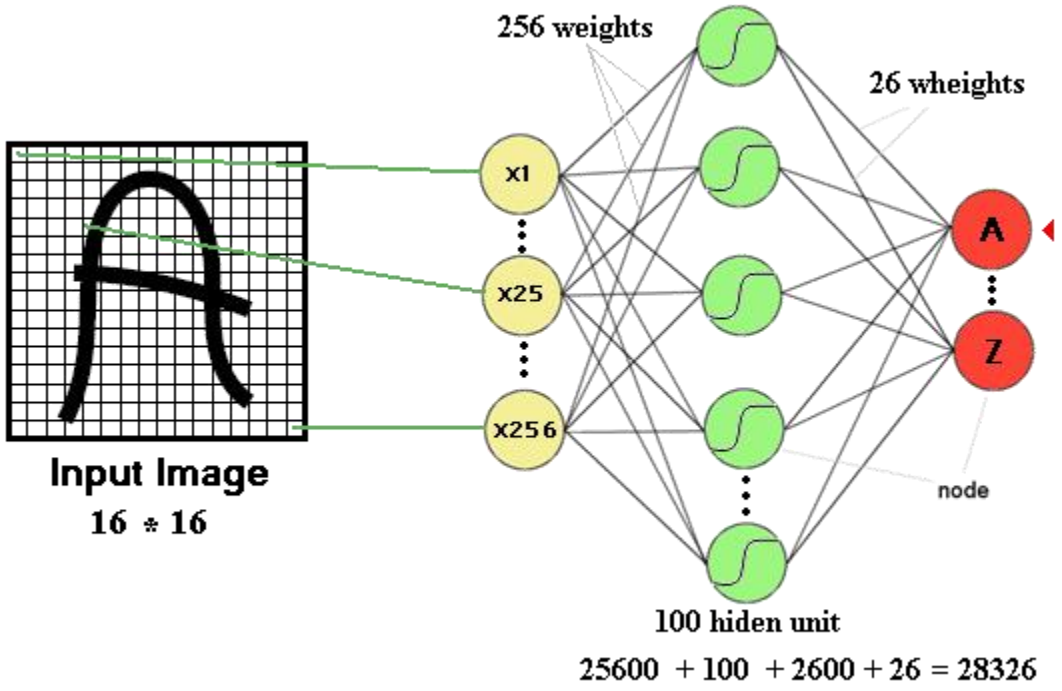
Multi-layer perceptron and image processing

- One or more hidden layers
- Sigmoid activations functions



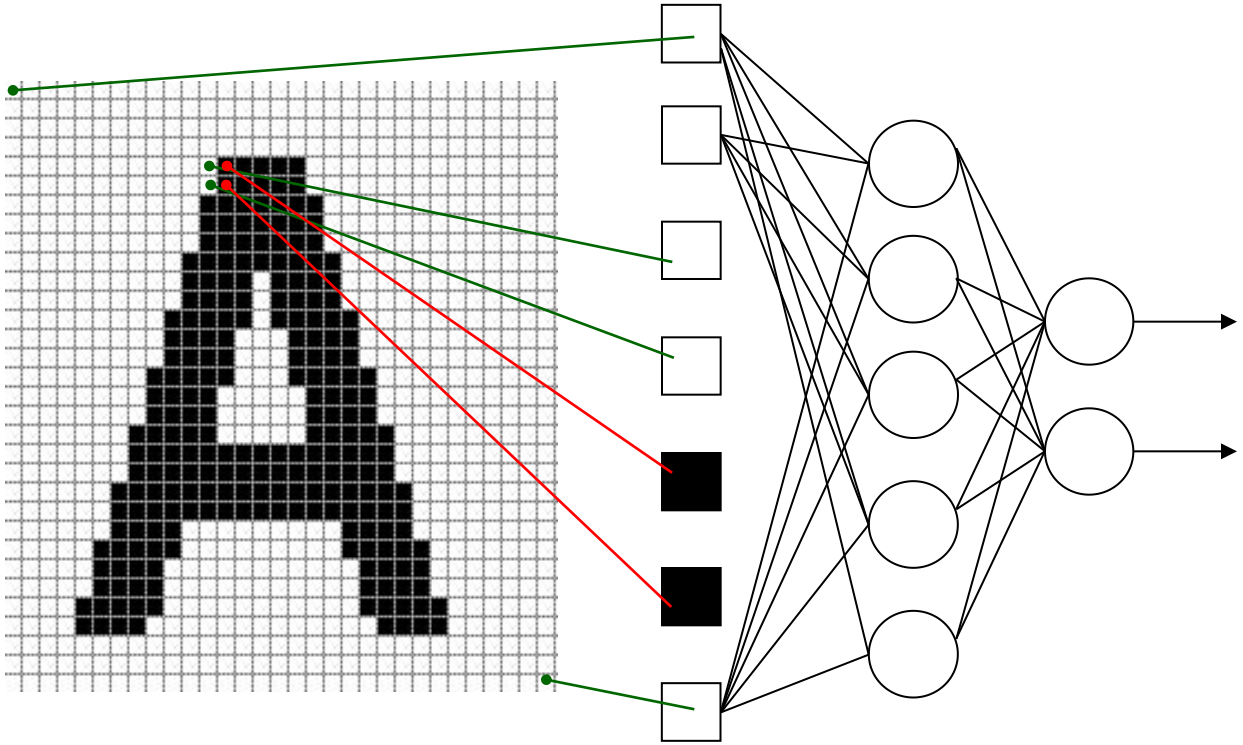
Drawbacks of MLPNN

- The number of trainable parameters becomes extremely large



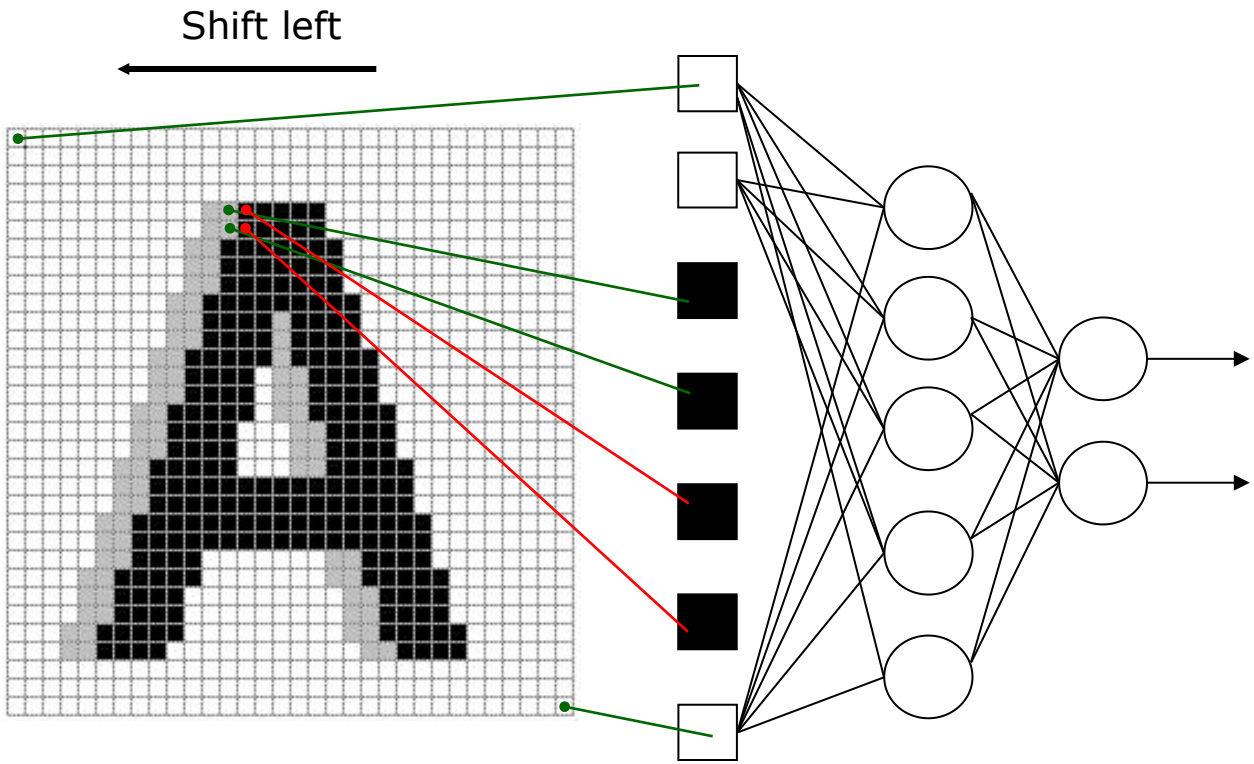
Drawbacks of MLPNN

- Little or no invariance to shifting, scaling, and other forms of distortion

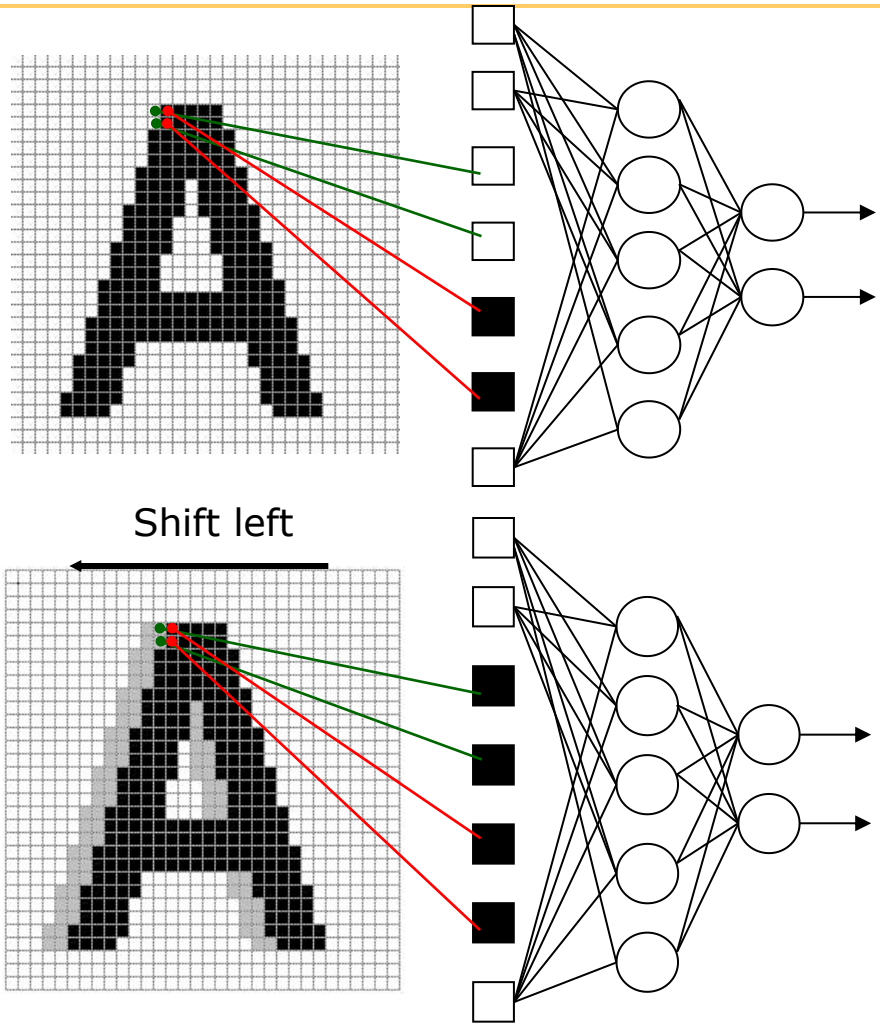


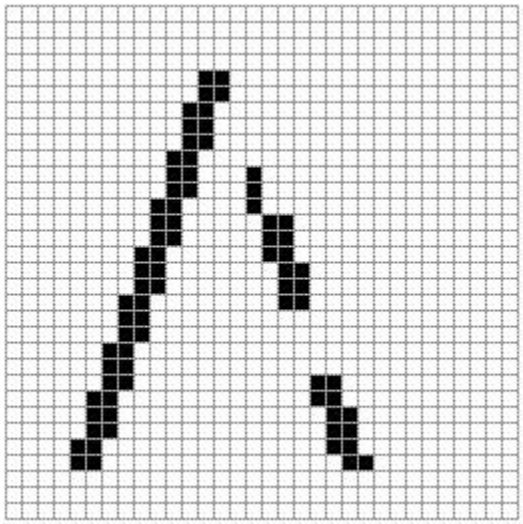
Drawbacks of MLPNN

- Little or no invariance to shifting, scaling, and other forms of distortion



Drawbacks of MLPNN





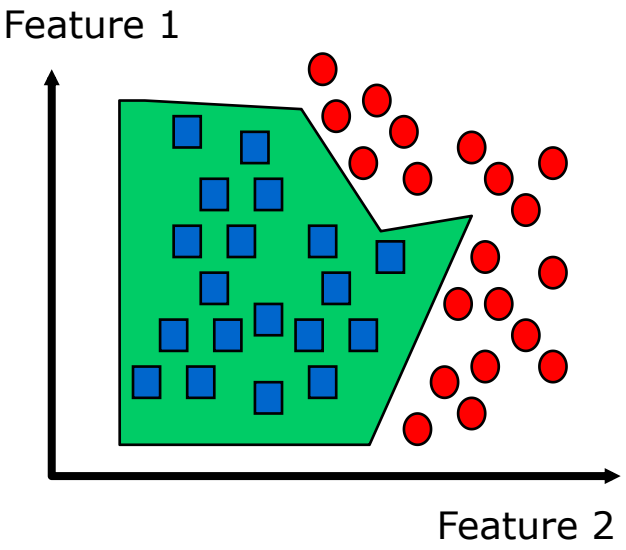
154 input change
from 2 shift left
77 : black to white
77 : white to black

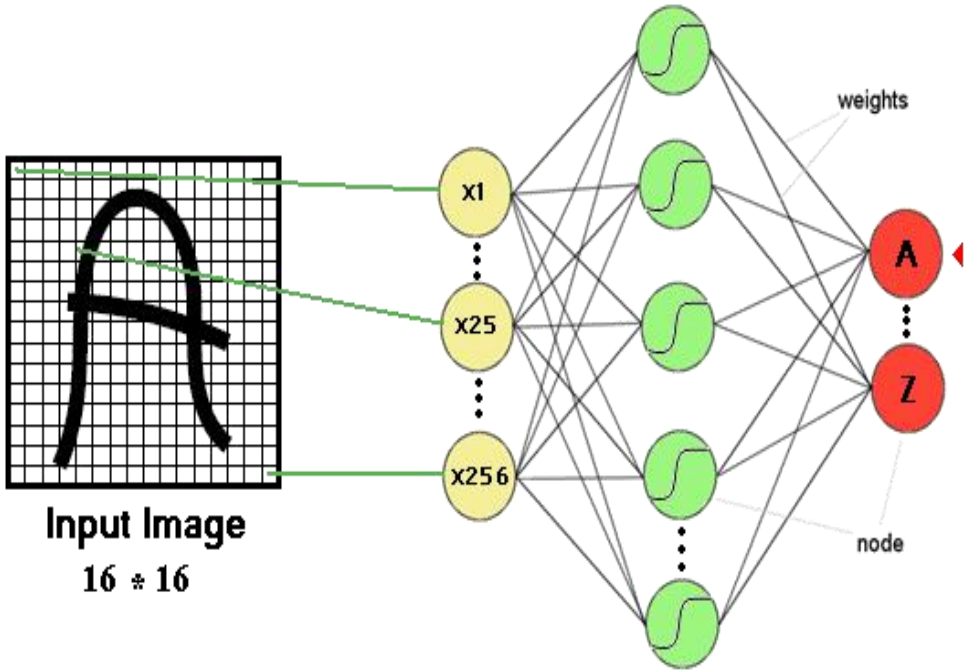
Drawbacks of MLPNN

- Scaling, and other forms of distortion

Drawbacks of MLPNN

- The topology of the input data is completely ignored
- Work with raw data

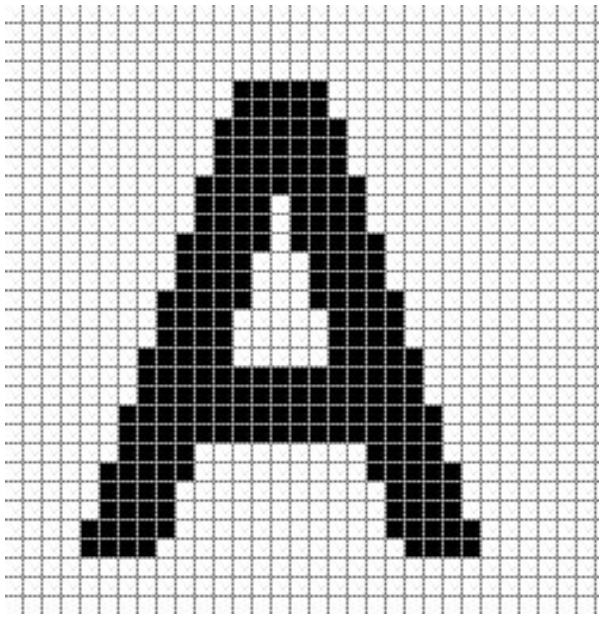




Drawbacks of MLPNN

Black and white patterns: $2^{32*32} = 2^{1024}$

Gray scale patterns : $256^{32*32} = 256^{1024}$

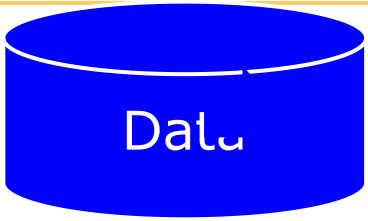


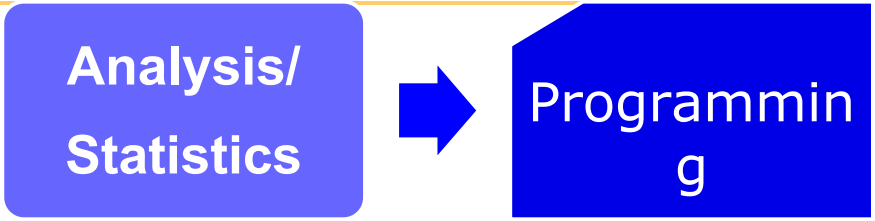
32 * 32 input image

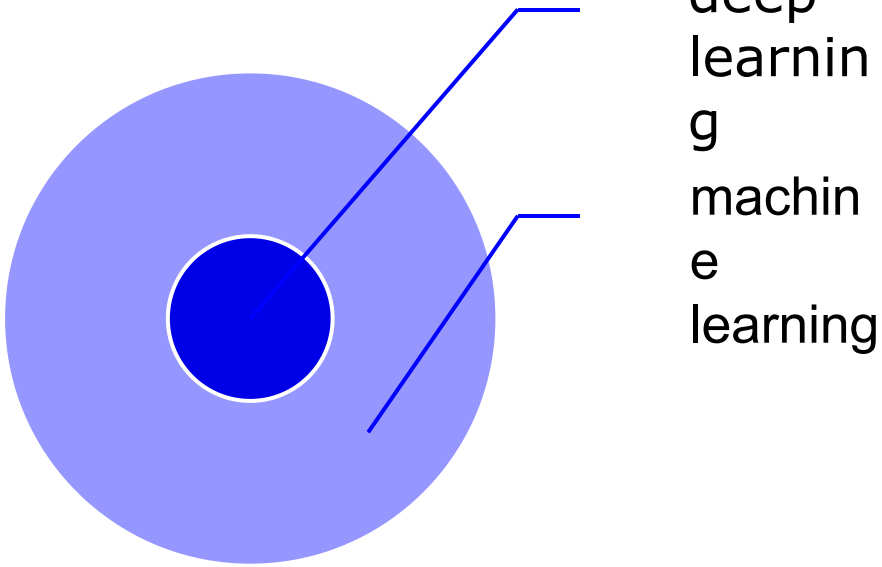
Deep Representations

Machine Learning & Deep Learning

Machine learning







DL is a branch of ML

- Based on massive historical data-big data
- “An algorithm” assumption

How computers to sense input signals?

image/video

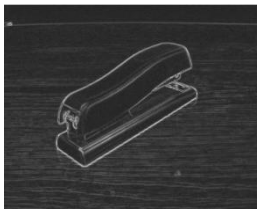
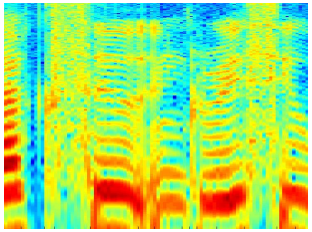


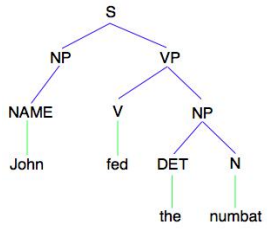
image Visual feature Object detected

speech



audio Audio feature Speaker identification

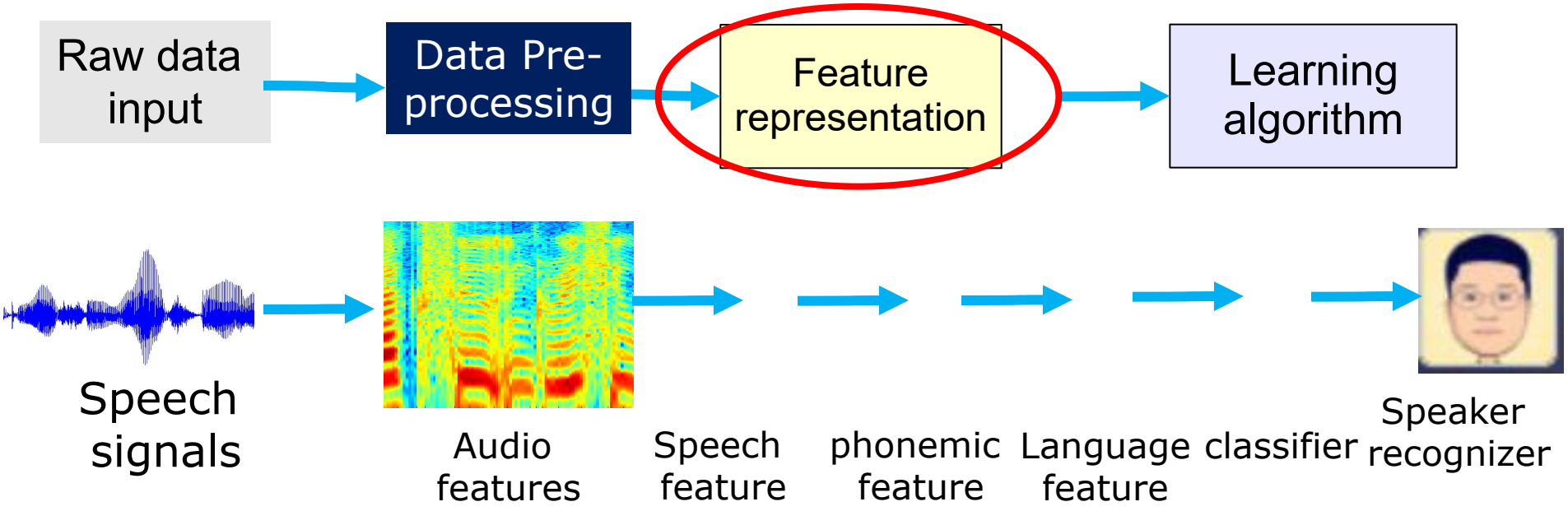
Text



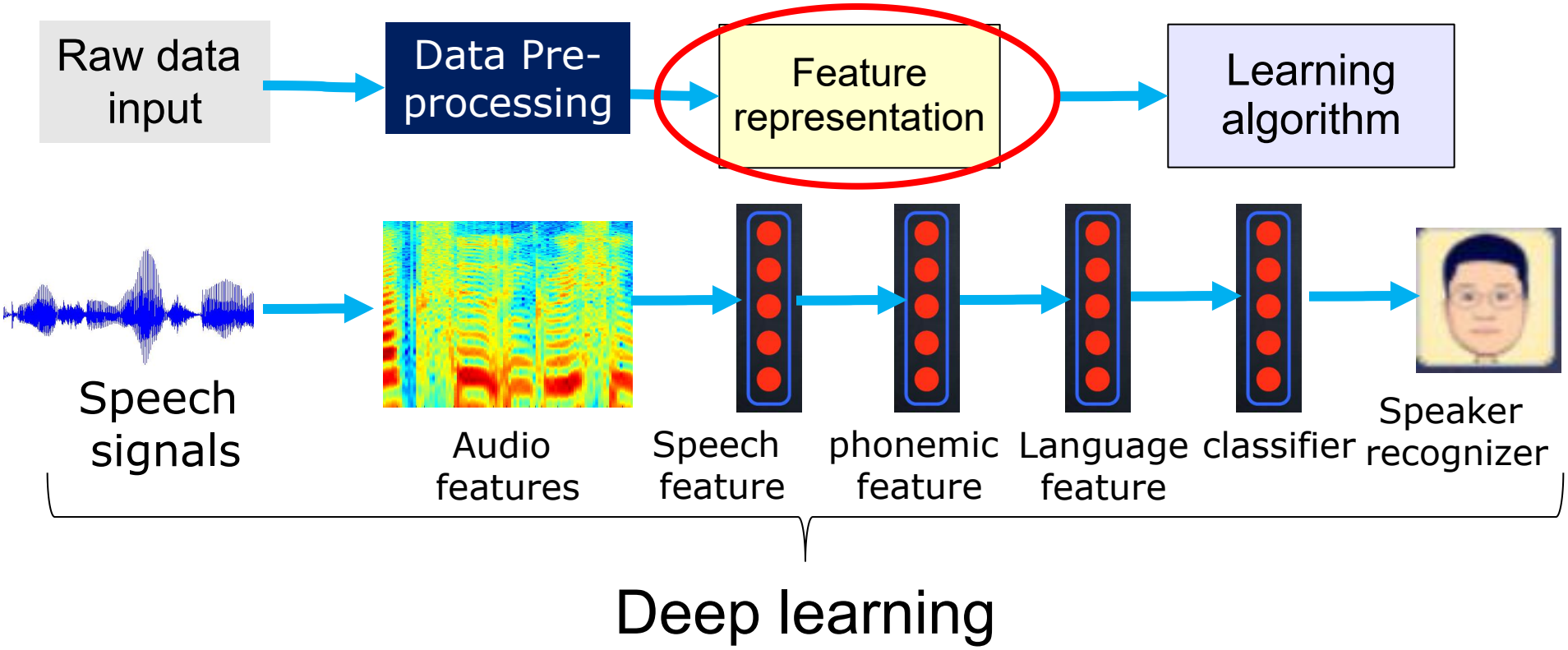
Text categorization
Machine translation
Information retrieval

text Textual features

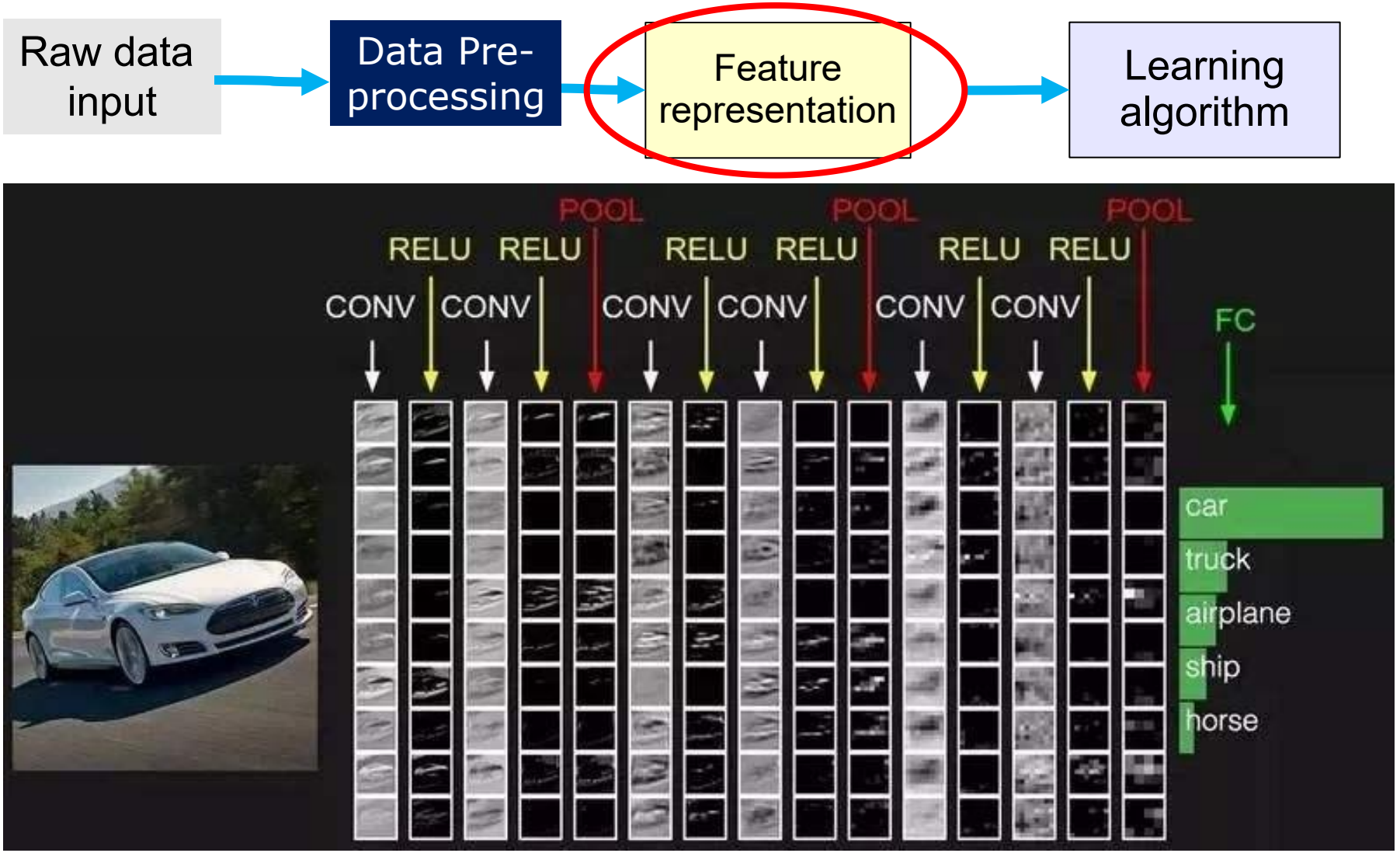
Data processing system



Data processing system

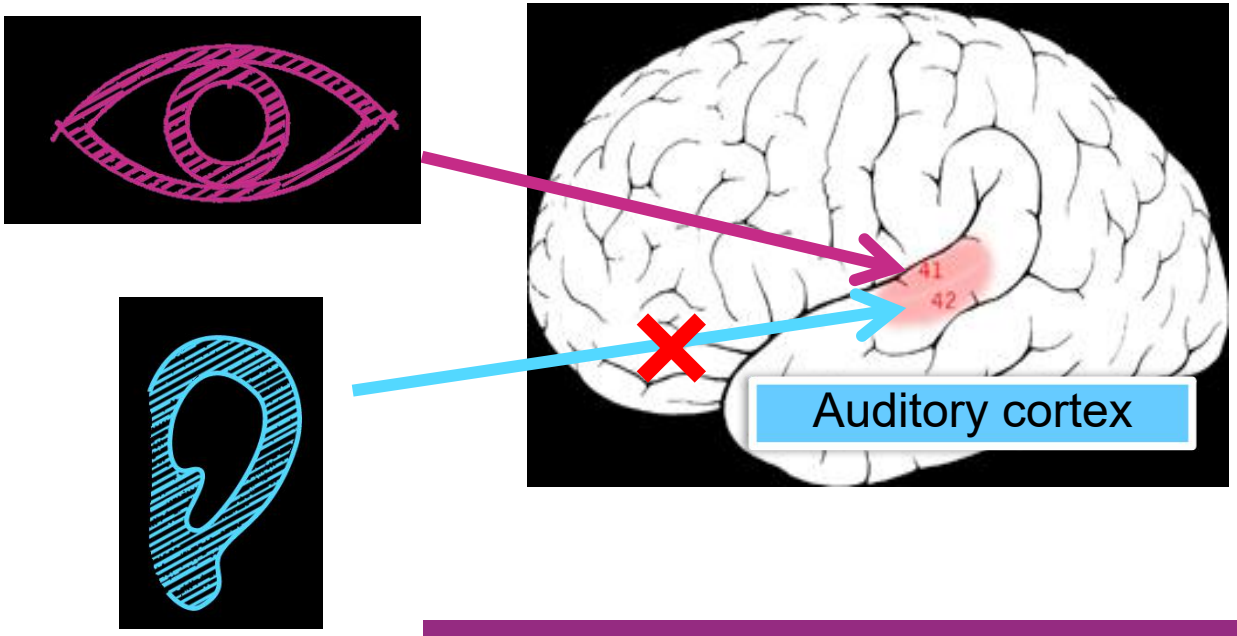


Data processing system



“Unified algorithm” assumption

Jeff Hawkins

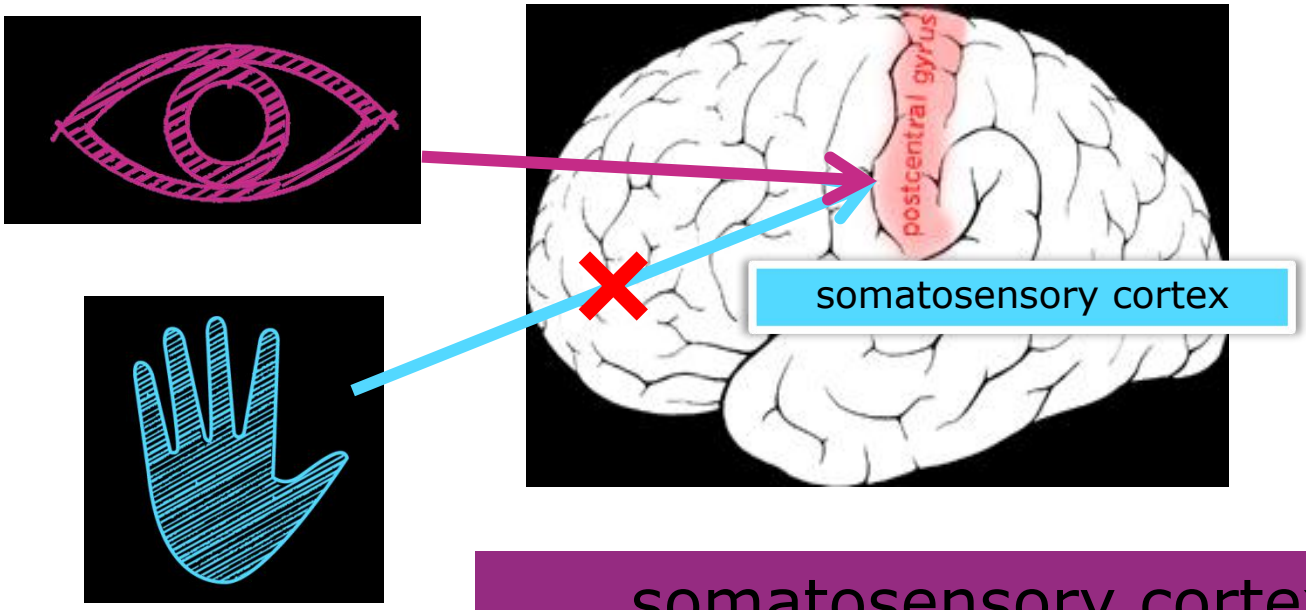


Auditory cortex generates auditory sense as well as visual sense

Roe et al., 1992]

“Unified algorithm” assumption

Jeff Hawkins



somatosensory cortex
Generates tactile sensation as well as auditory sense

Metin & Frost, 1989]

Receptive fields

A bit of history:

Hubel & Wiesel,

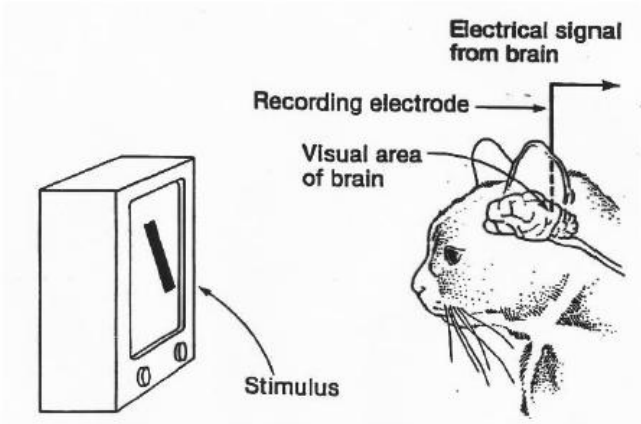
1959

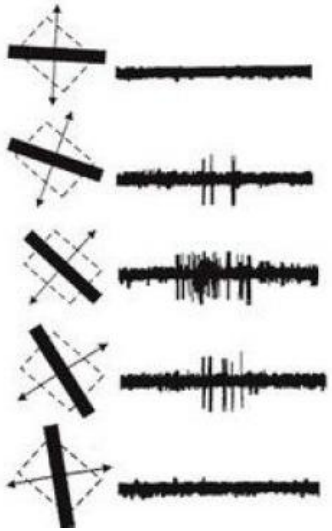
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

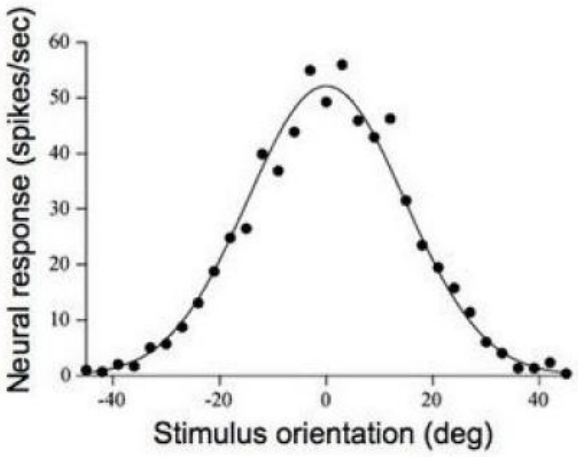
1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

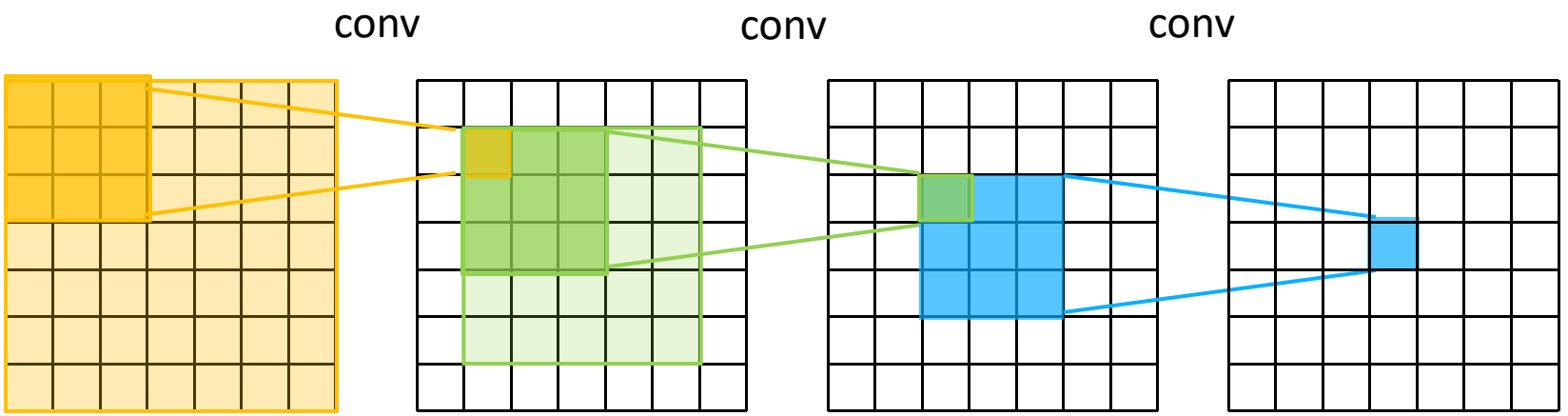
1968…



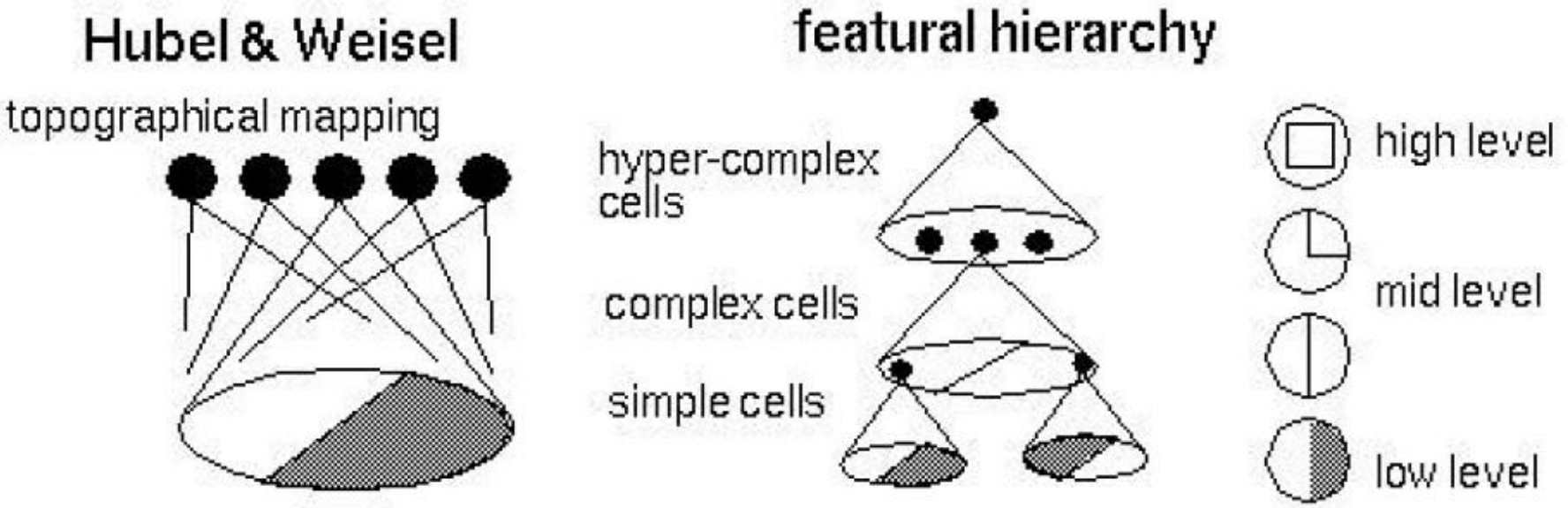




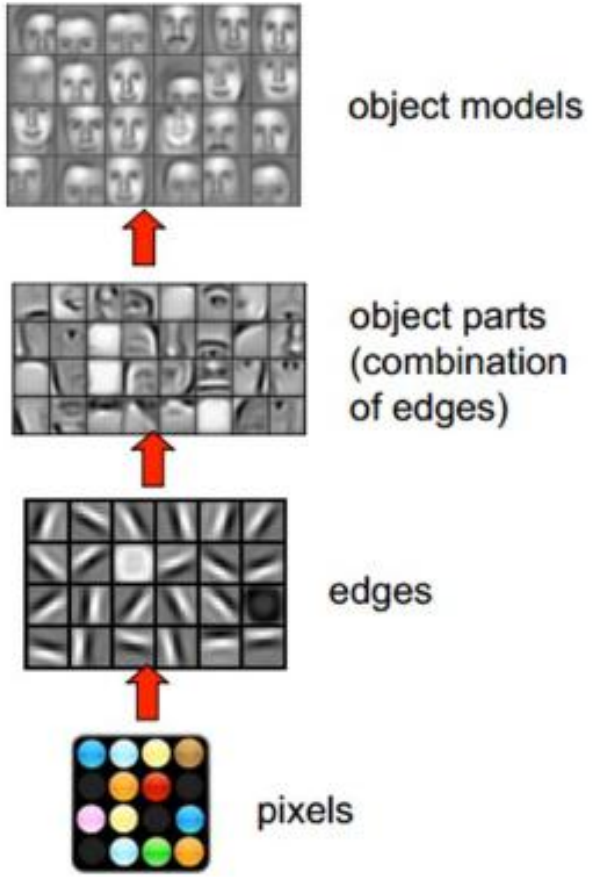
Receptive fields

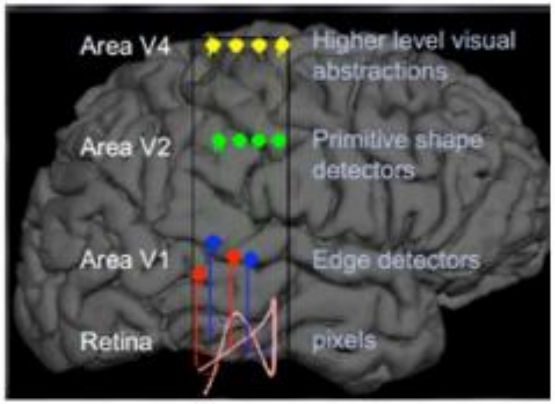


Hierarchical organization

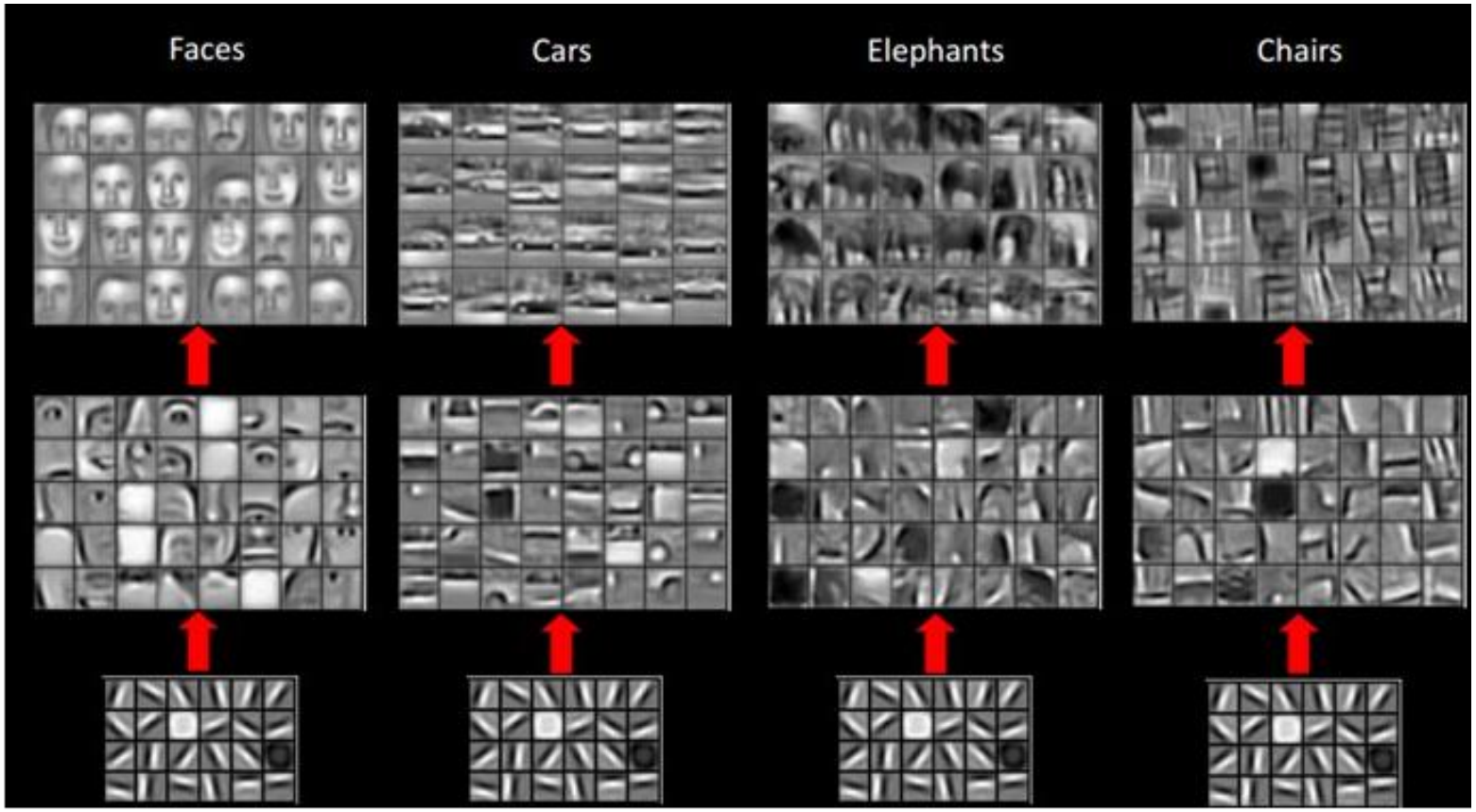


Hierarchical organization





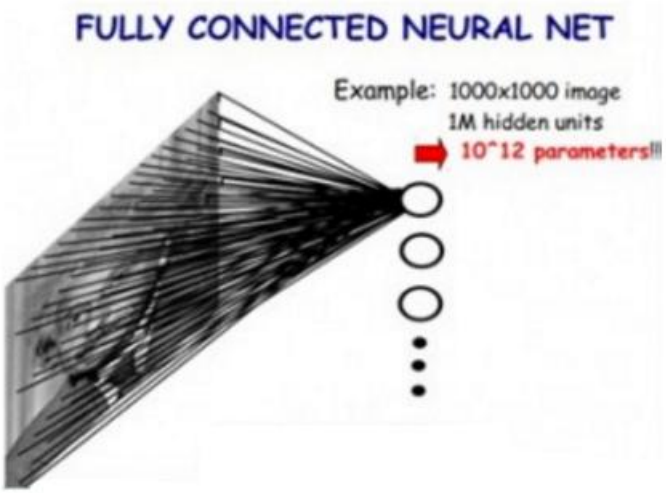
Hierarchical organization

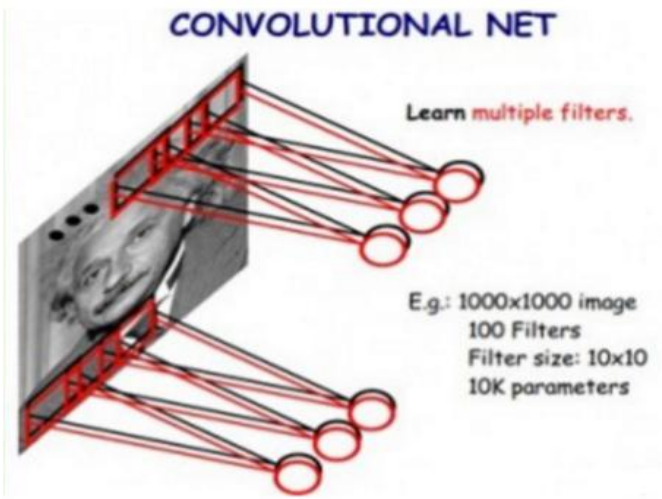


Introduction to CNNs

Convolution

Neural network –
fully connected or convolutional?

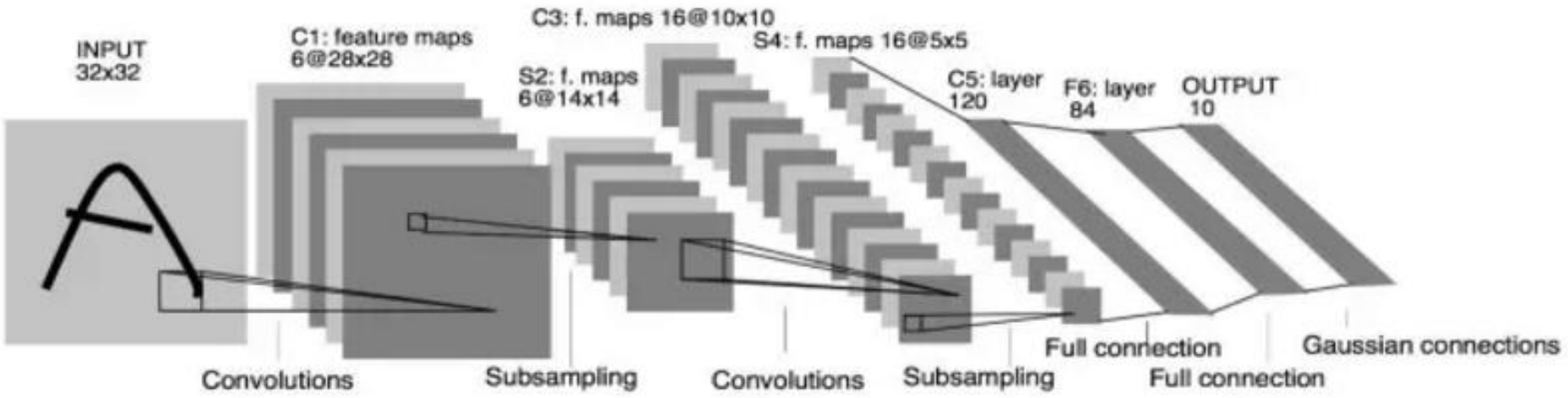




LeNet5

The earliest deep CNN model：

- Convolution, pool, nonlinear operators



Y. Lecun, L. Bottou, Y. Bengio and P. Haffner. *Gradient-Based Learning Applied to Document Recognition.* Proceedings of the IEEE, vol. 86, no. 11, 1998

CNN architecture

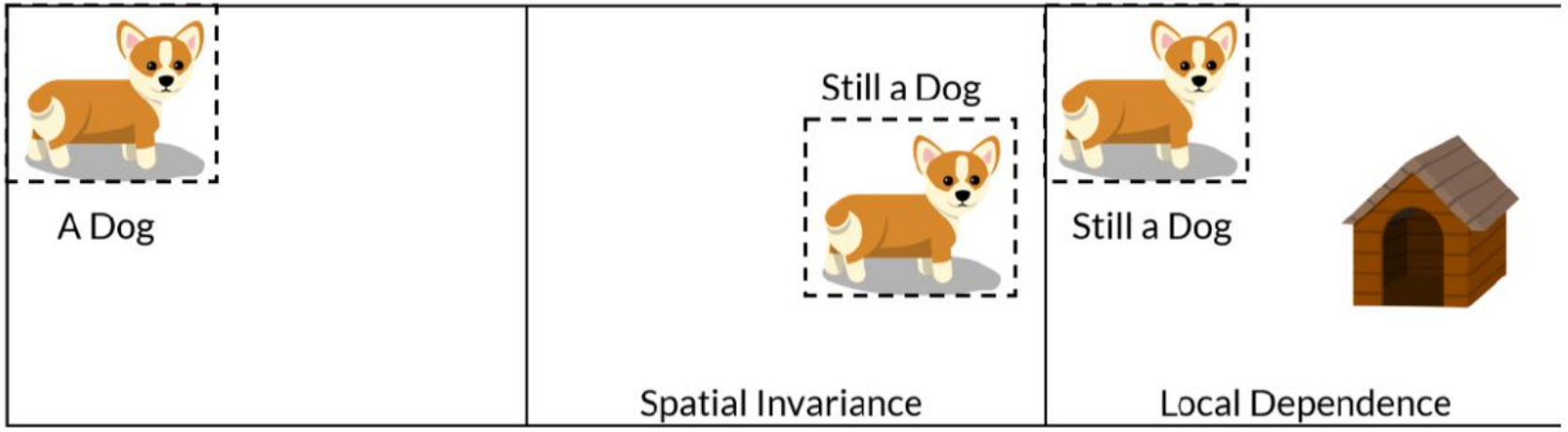
CNN：convolution + pooling + full connection

Convolution：

✓Local connection——local receptive fields——local features

✓Weighting sharing——reducing computation

✓Multiple kernels



Local translation invariance

CNN architecture

CNN：convolution + pooling + full connection

Convolution：

✓Local connection——local receptive fields——local features

✓Weighting sharing——reducing computation

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

W_{11}	W_{12}
W_{21}	W_{22}

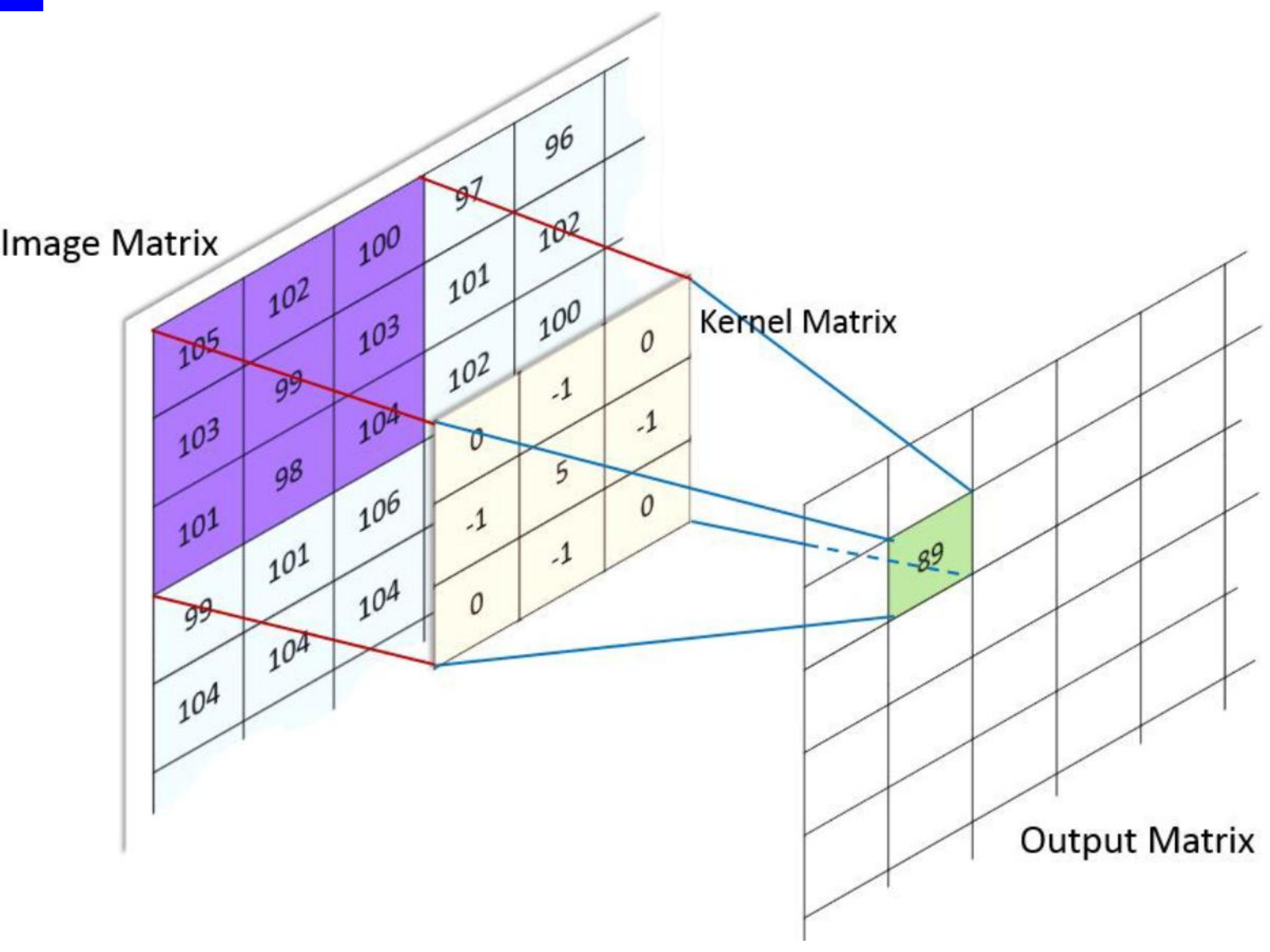


Image Matrix
105
102
100
97
96
103
99
103
101
102
101
98
104
99
100
0
-1
-1
0
5
0
-1
-1
0
Kernel Matrix
89
99
101
106
104
104
104
Output Matrix

CNN architecture

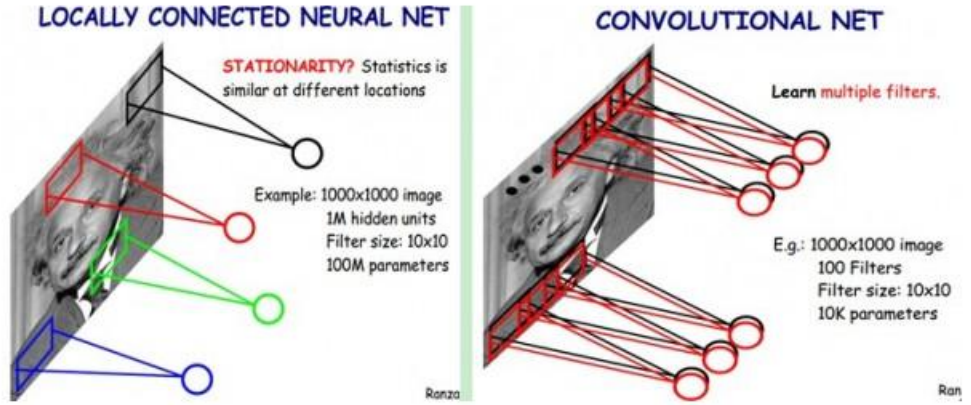
CNN：convolution + pooling + full connection

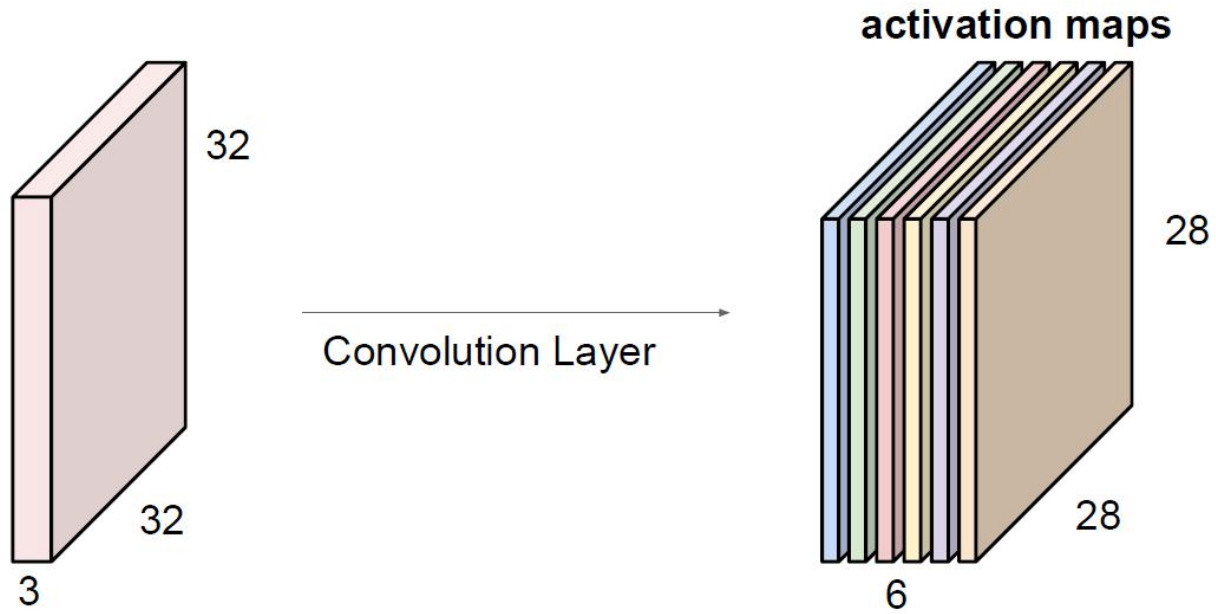
Convolution：

✓Local connection——local recep

✓Weighting sharing——reducing

✓Multiple kernels





CNN architecture

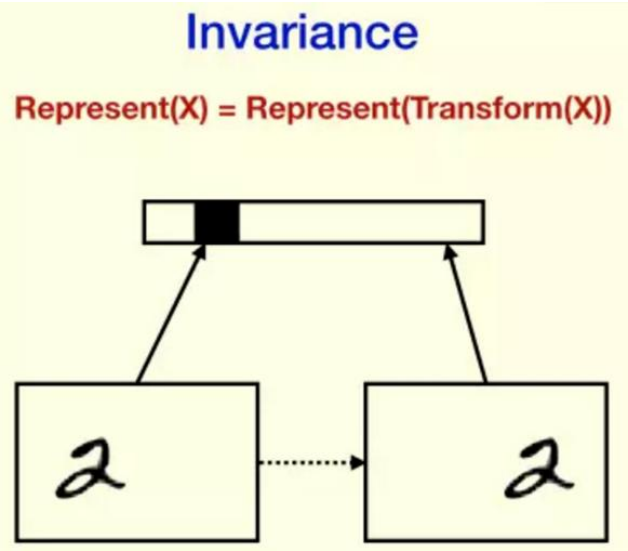
CNN：convolution + pooling + full connection

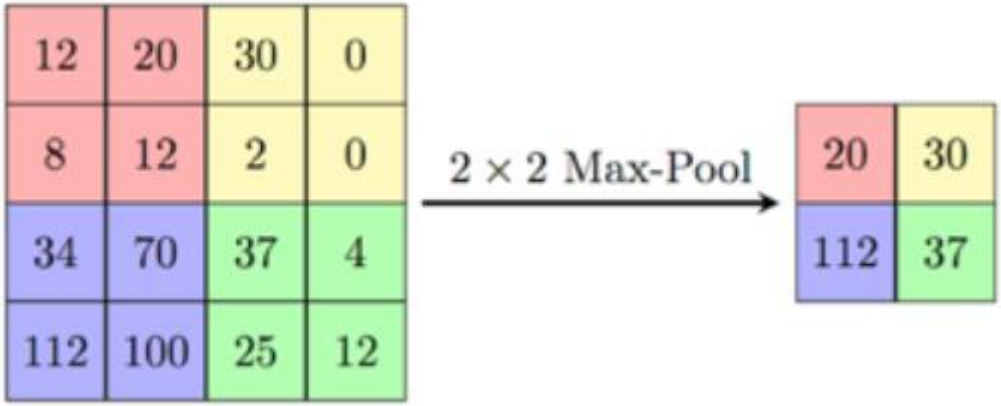
Convolution：

✓Local connection——local receptive fields——local features
✓Weighting sharing——reducing computation
✓Multiple kernels

Pooling（subsampling）：

✓ Reducing data dimensions to avoid overfitting
✓ Enlarge local receptive fields
✓ Improving translation invariance





CNN architecture

CNN：convolution + pooling + full connection

Convolution：

✓Local connection——local receptive fields——local features
✓Weighting sharing——reducing computation
✓Multiple kernels

Pooling（subsampling）：

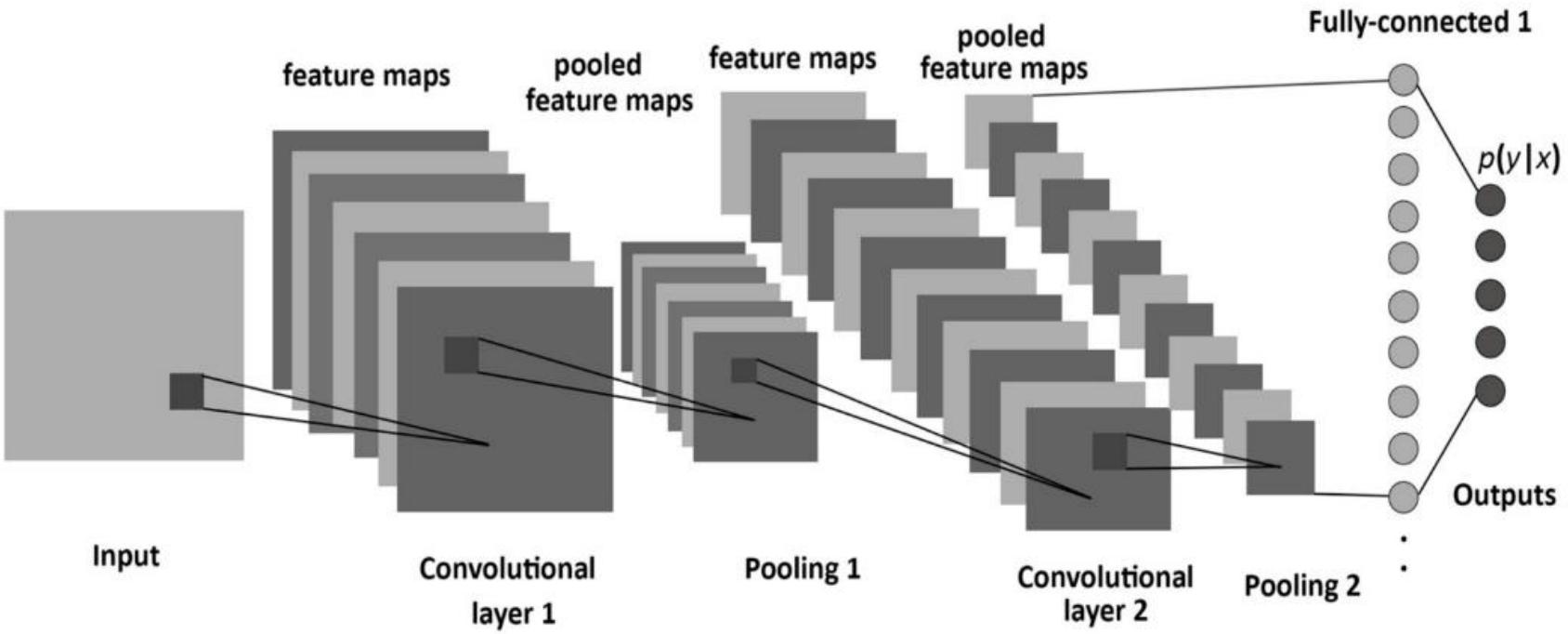
✓ Reducing data dimensions to avoid overfitting
✓ Enlarge local receptive fields
✓ Improving translation invariance

Fully connected layers：

✓ Bridge to connecting feature extraction to classification

Fully connection

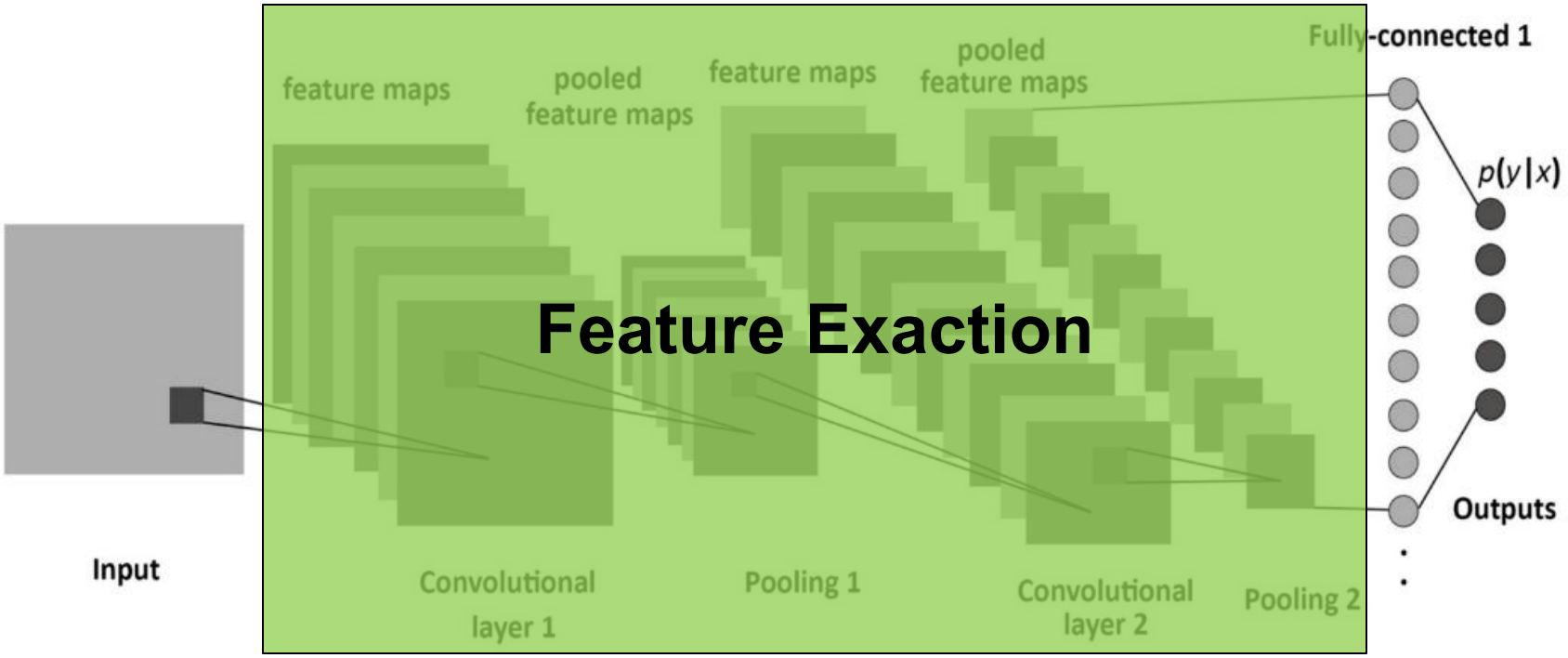
CNN：convolution + pooling + full connection



Full connection —— Bridge to connecting feature extraction to classification

Fully connection

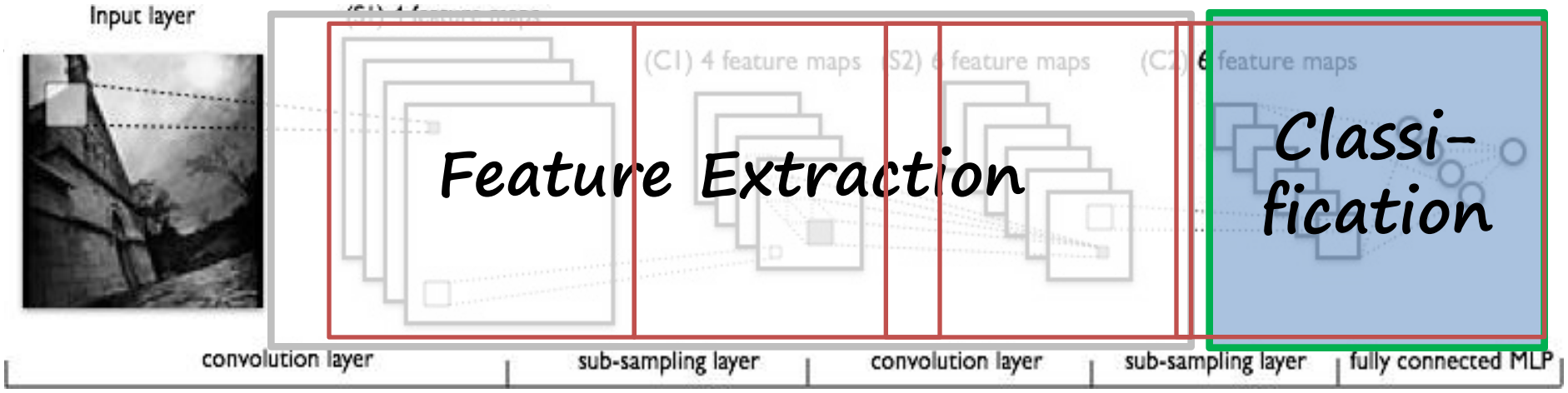
CNN：convolution + pooling + full connection



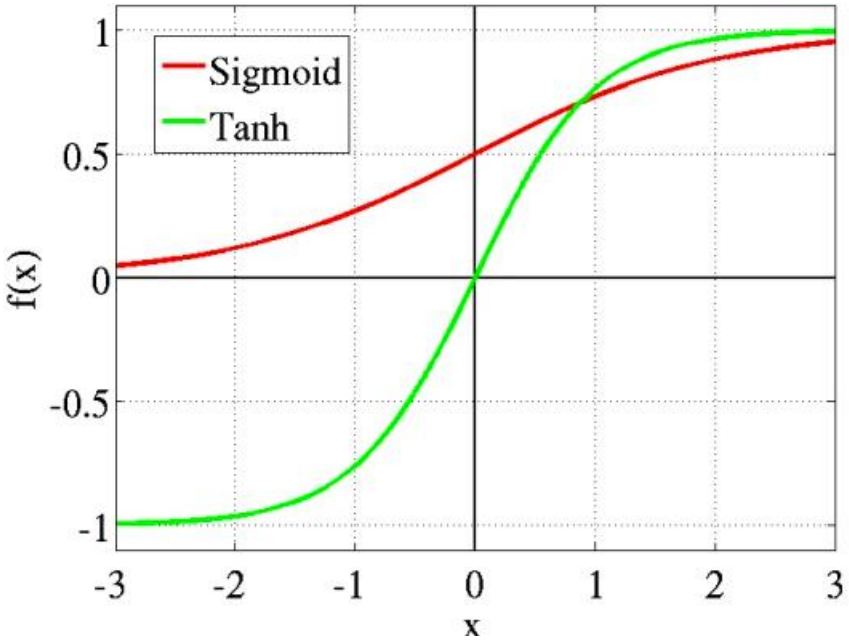
Full connection —— Bridge to connecting feature extraction to classification

CNN - multi-layer NN architecture

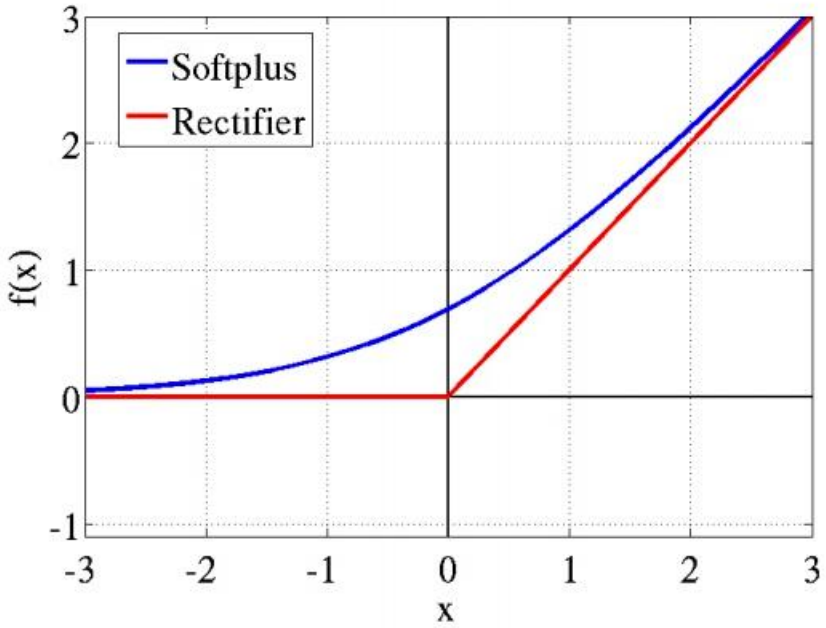
- Convolutional + Non-Linear Layer
- Sub-sampling Layer
- Convolutional +Non-L inear Layer
- Fully connected layers



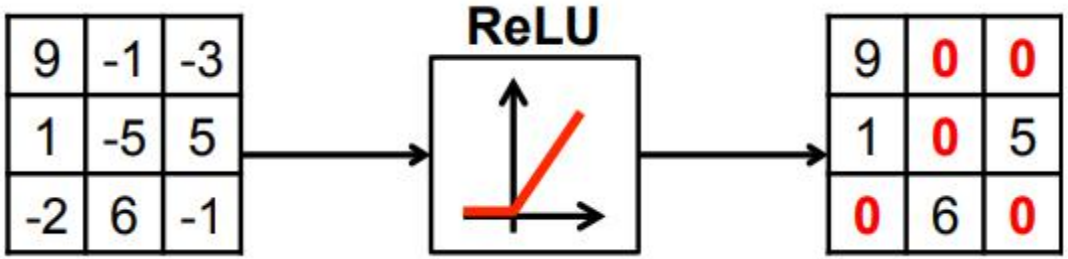
Activation Functions

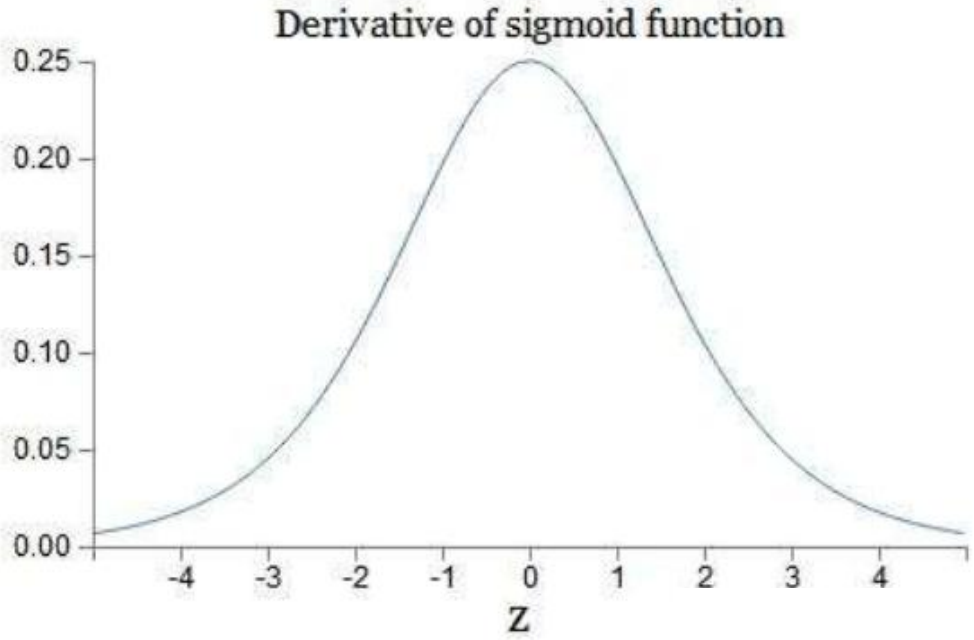


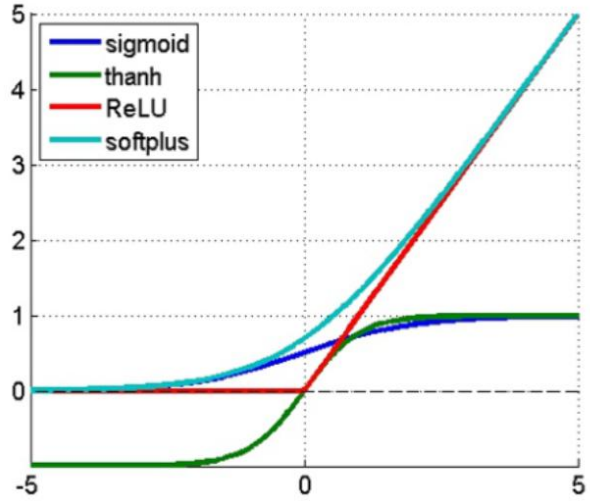
Fully connected network



CNNs



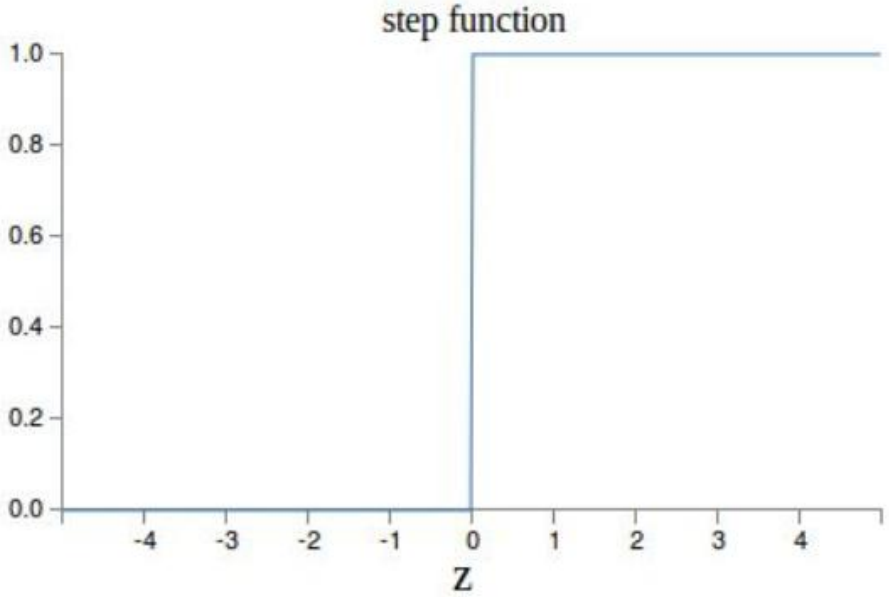


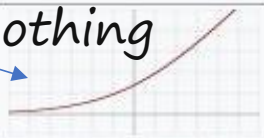


Derivative of Relu function

Relu function:

1. avoiding vanishing gradient
2. sparse activation
3. Speeding up computation

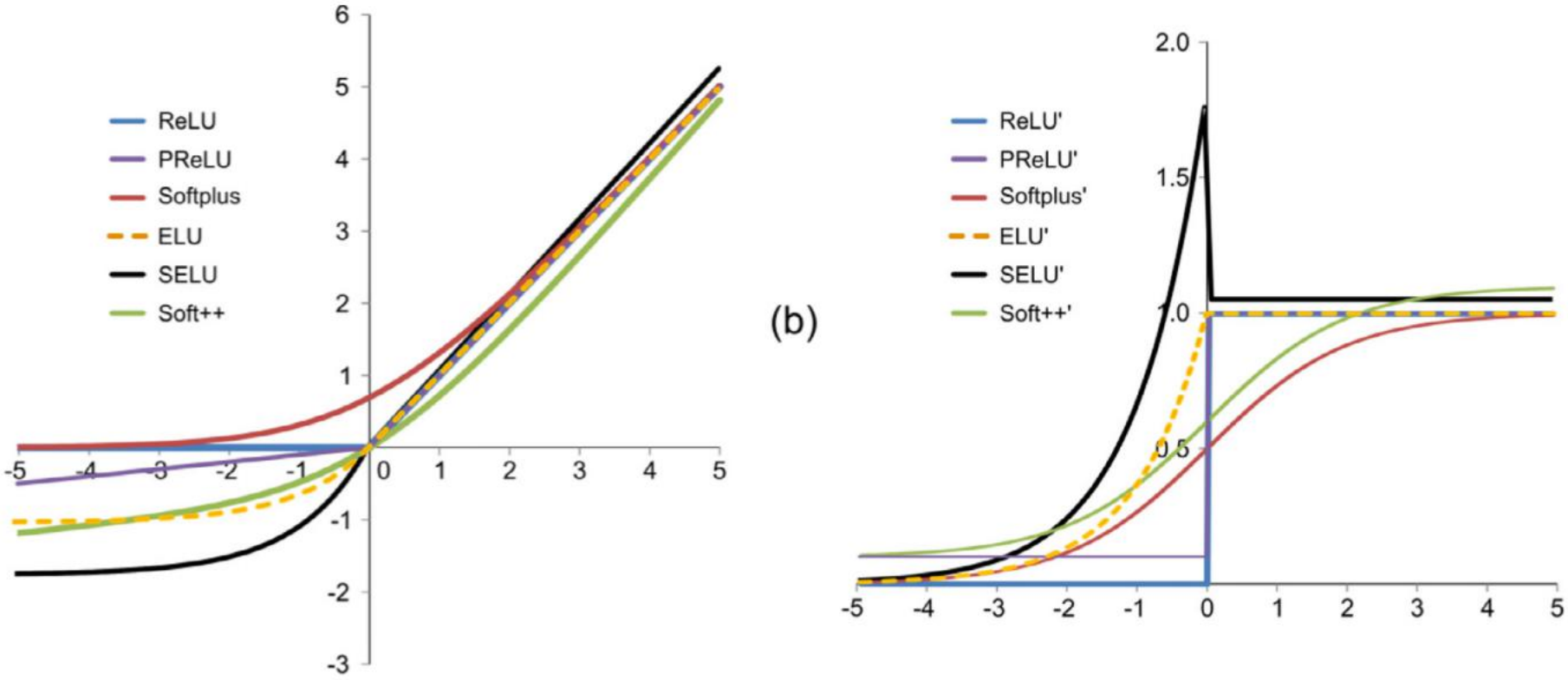


Rectified linear unit (ReLU)[12]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	Inverse square root linear unit (ISRLU)[11]		$f(x) = \begin{cases} \frac{x}{\sqrt{1+\alpha x^2}} & \text{fo} \\ x & \text{fo} \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[13]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	Adaptive piecewise linear (APL)[19]		$f(x) = \max(0, x) + \sum$
Parameteric rectified linear unit (PReLU)[14]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	SoftPlus[20]		$f(x) = \ln(1 + e^x)$
Randomized leaky rectified linear unit (RReLU)[15]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ [3]	Bent identity		$f(x) = \frac{\sqrt{x^2+1}-1}{2}$
Exponential linear unit (ELU)[16]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$			
Scaled exponential linear unit (SELU)[17]		$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$			
S-shaped rectified linear activation unit (SReLU)[18]		$f_{t_l, a_l, t_r, a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ t_l, a_l, t_r, a_r are parameters.			
Inverse square root linear unit (ISRLU)[11]		$f(x) = \begin{cases} \frac{x}{\sqrt{1+\alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$			
Adaptive piecewise linear		$f(x) = \max(0, x) + \sum_{i}^{S} a_i^s \max(0, -x + b_i^s)$			

Smoothing

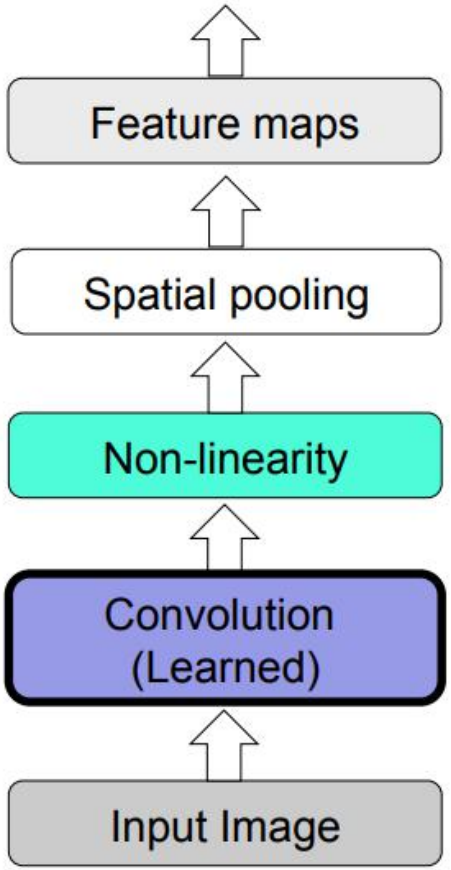
The derivative of softplus is logistic function.

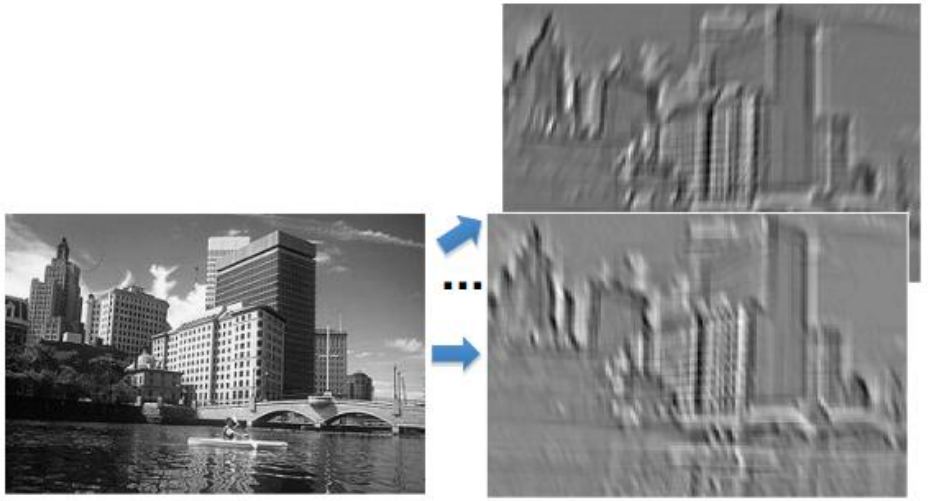
Activation functions and their derivatives



@article{CIUPARU2020376,
title = "Soft++, a multi-parametric non-saturating non-linearity that improves convergence in deep neural architectures",
journal = "Neurocomputing",
volume = "384",
pages = "376 - 388",
year = "2020 “}

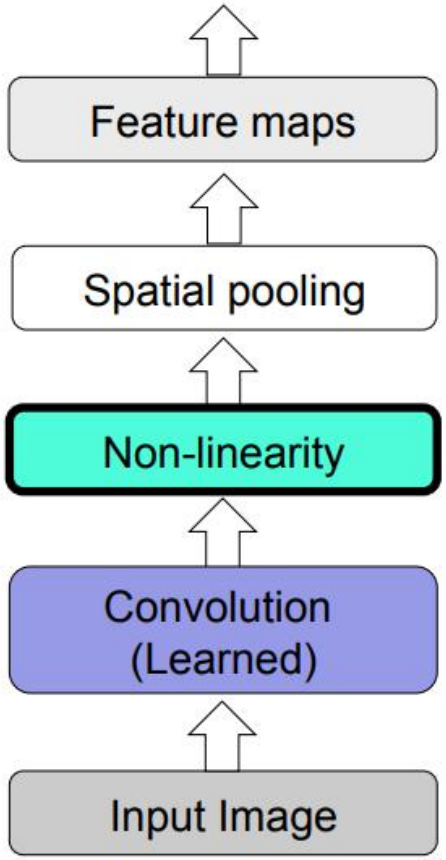
Key Operations in a CNN

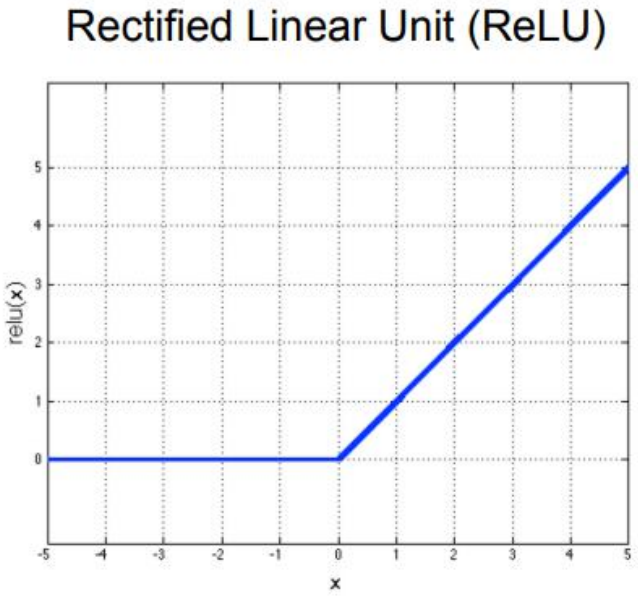




Source: R. Fergus, Y. LeCun

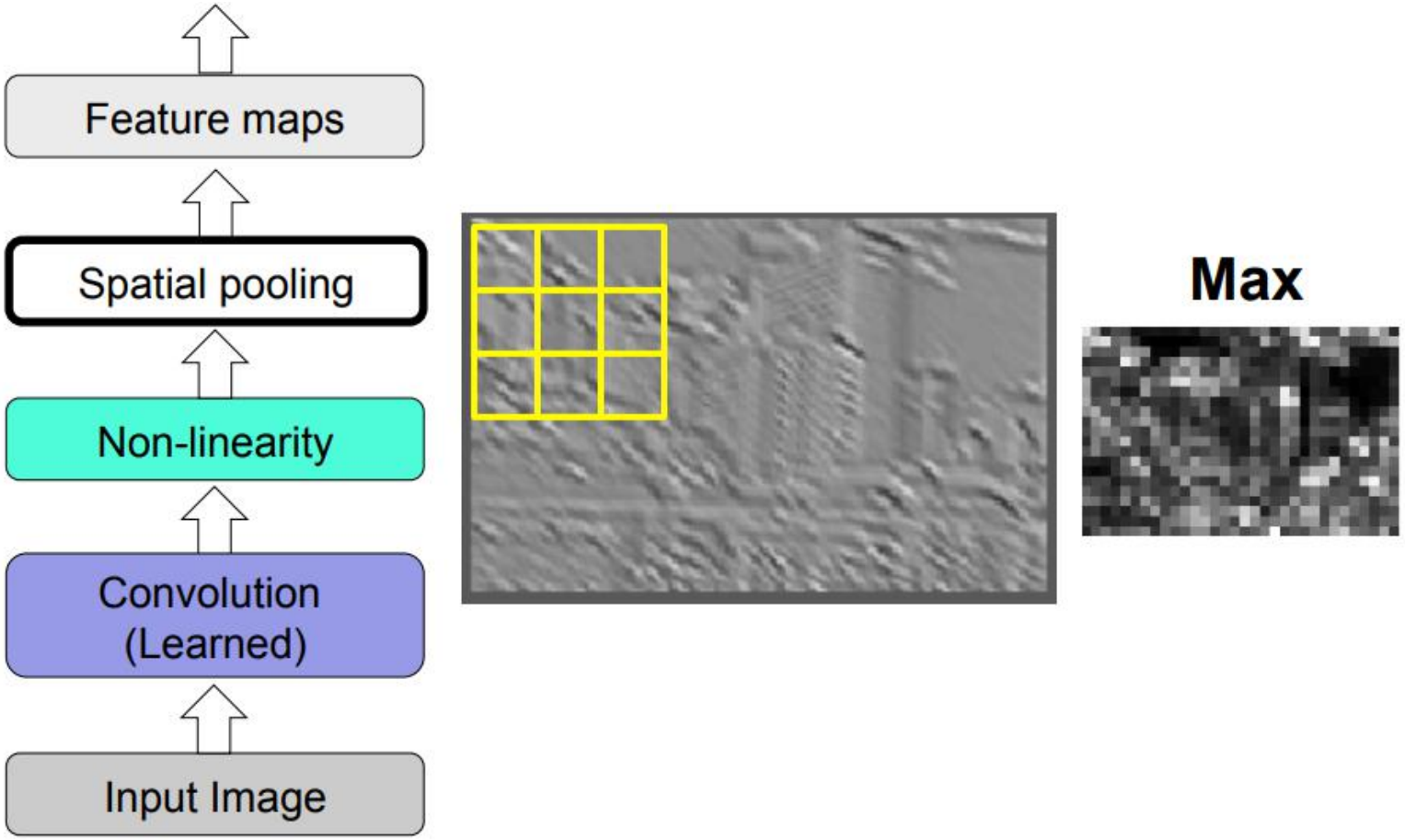
Key Operations in a CNN





Source: R. Fergus, Y. LeCun

Key Operations in a CNN



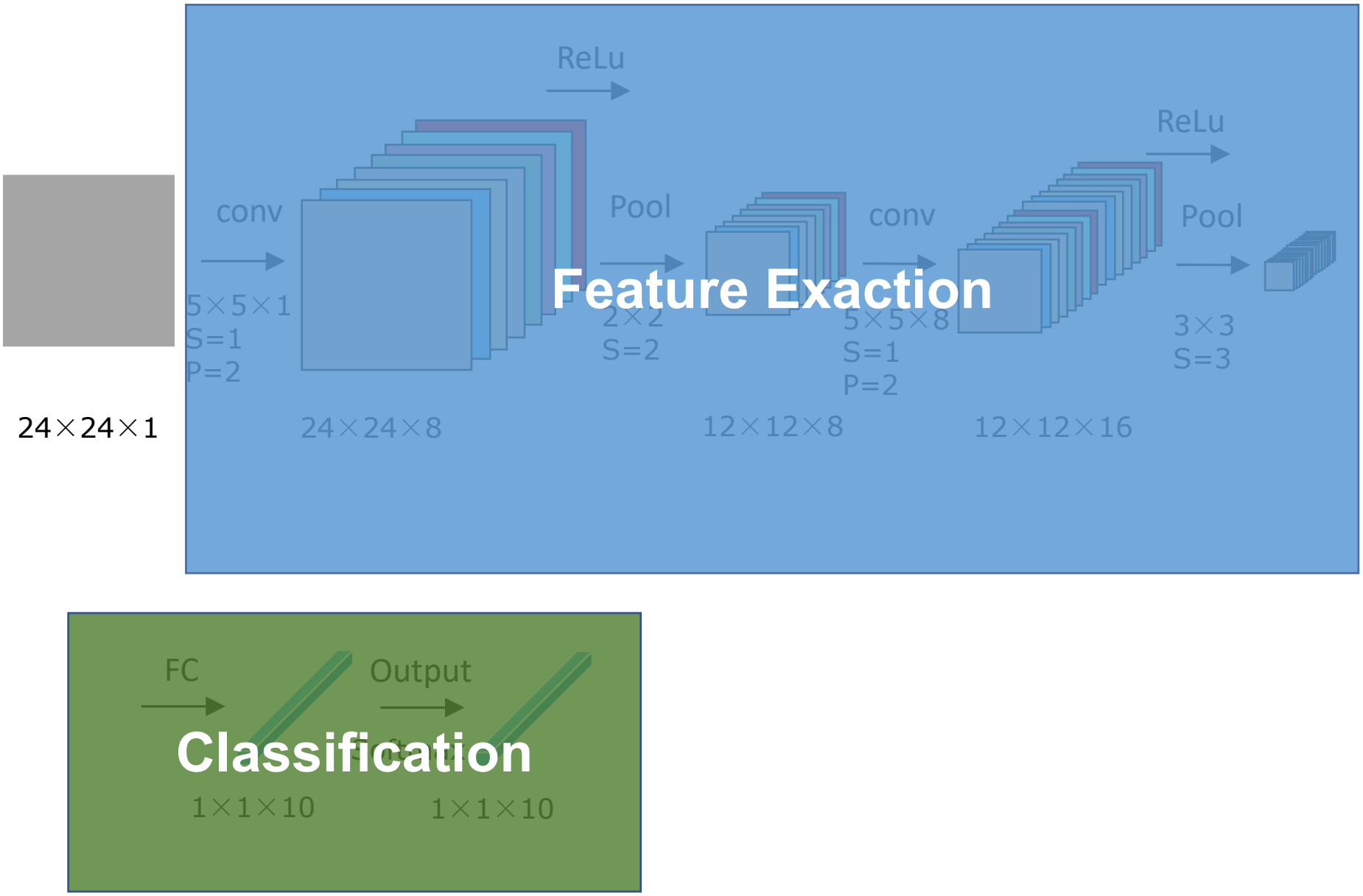
Source: R. Fergus, Y. LeCun

Demo

- http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html
- ConvNetJS by Andrej Karpathy (Ph.D. student at Stanford)

Software libraries

- Caffe (C++, python, matlab)
- Torch7 (C++, lua)
- Theano (python)


ReLu
ReLu
conv
Pool
conv
Pool
Feature Exaction
5×5×1
S=1
P=2
2×2
S=2
5×5×8
S=1
P=2
3×3
S=3
24×24×1
24×24×8
12×12×8
12×12×16
FC
Output
Classification
1×1×10
1×1×10

Network Visualization

input (24x24x1)
max activation: 1, min: 0
max gradient: 0, min: -0.00001

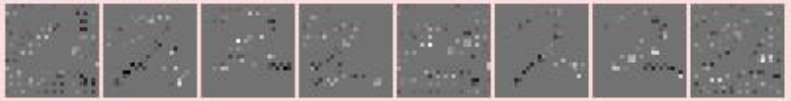
Activations:

Activation Gradients:

conv (24x24x8)
filter size 5x5x1, stride 1
max activation: 2.81644, min: -4.78981
max gradient: 0, min: -0.00001
parameters: 8x5x5x1+8 = 208

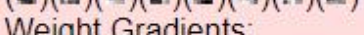
Activations:

Activation Gradients:



Weights:

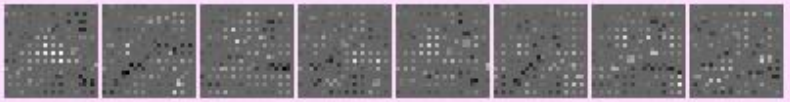
Weight Gradients:



relu (24x24x8)
max activation: 2.81644, min: 0
max gradient: 0, min: -0.00001

Activations:

Activation Gradients:



pool (12x12x8)
pooling size 2x2, stride 2
max activation: 2.81644, min: 0
max gradient: 0, min: -0.00001

Activations:

Activation Gradients:

IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

- ILSVRC 2017
- ILSVRC 2016
- ILSVRC 2015
- ILSVRC 2014
- ILSVRC 2013
- ILSVRC 2012
- ILSVRC 2011
- ILSVRC 2010

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC): 1.2 million training images, 1000 classes

Tasks:

- Classification
- Localization
- Detection
- Etc.

http://www.image-net.org/challenges/LSVRC/

Classification: ImageNet Challenge top-5 error

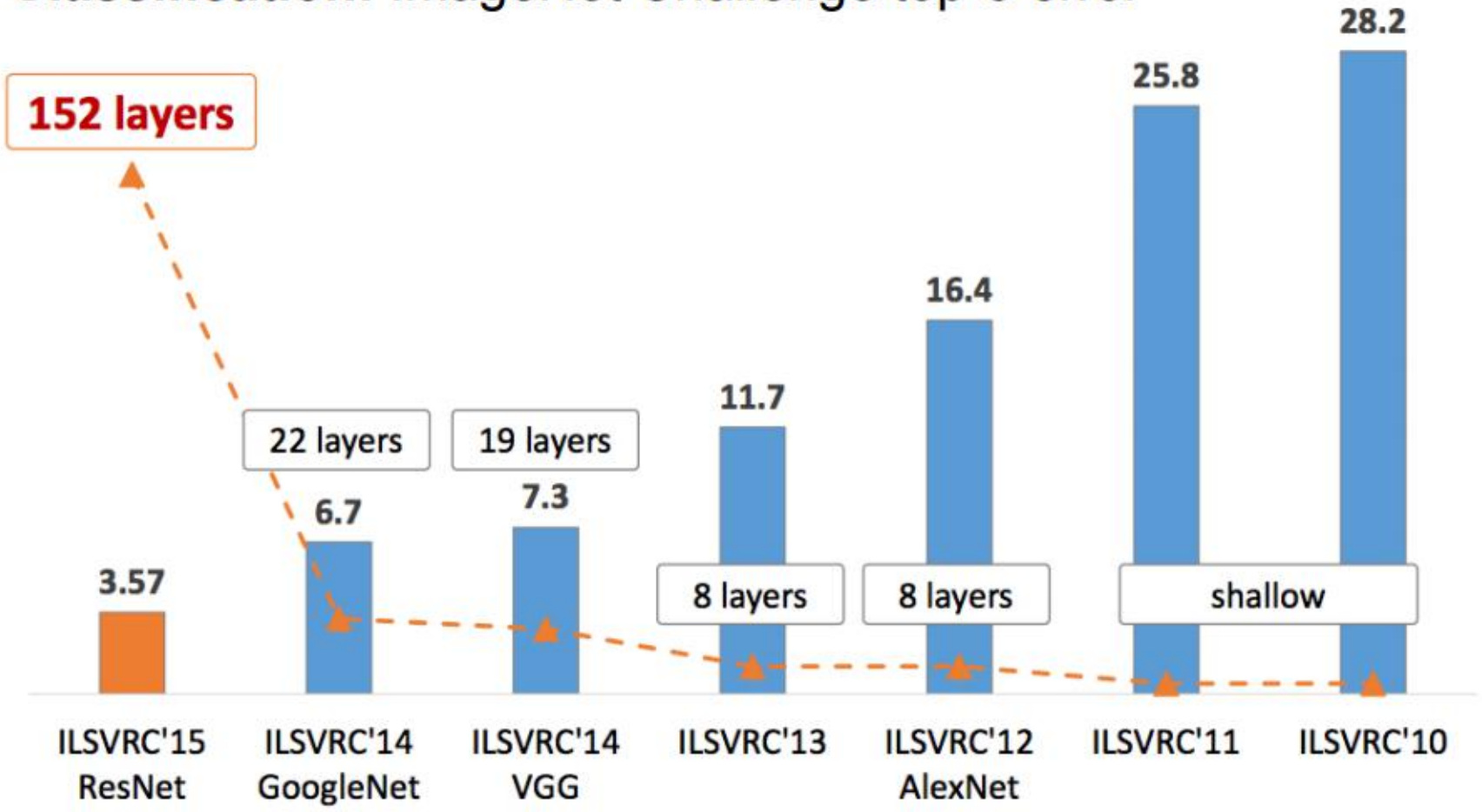


Figure source: Kaiming He

Object Detection: PASCAL VOC mean Average Precision (mAP)

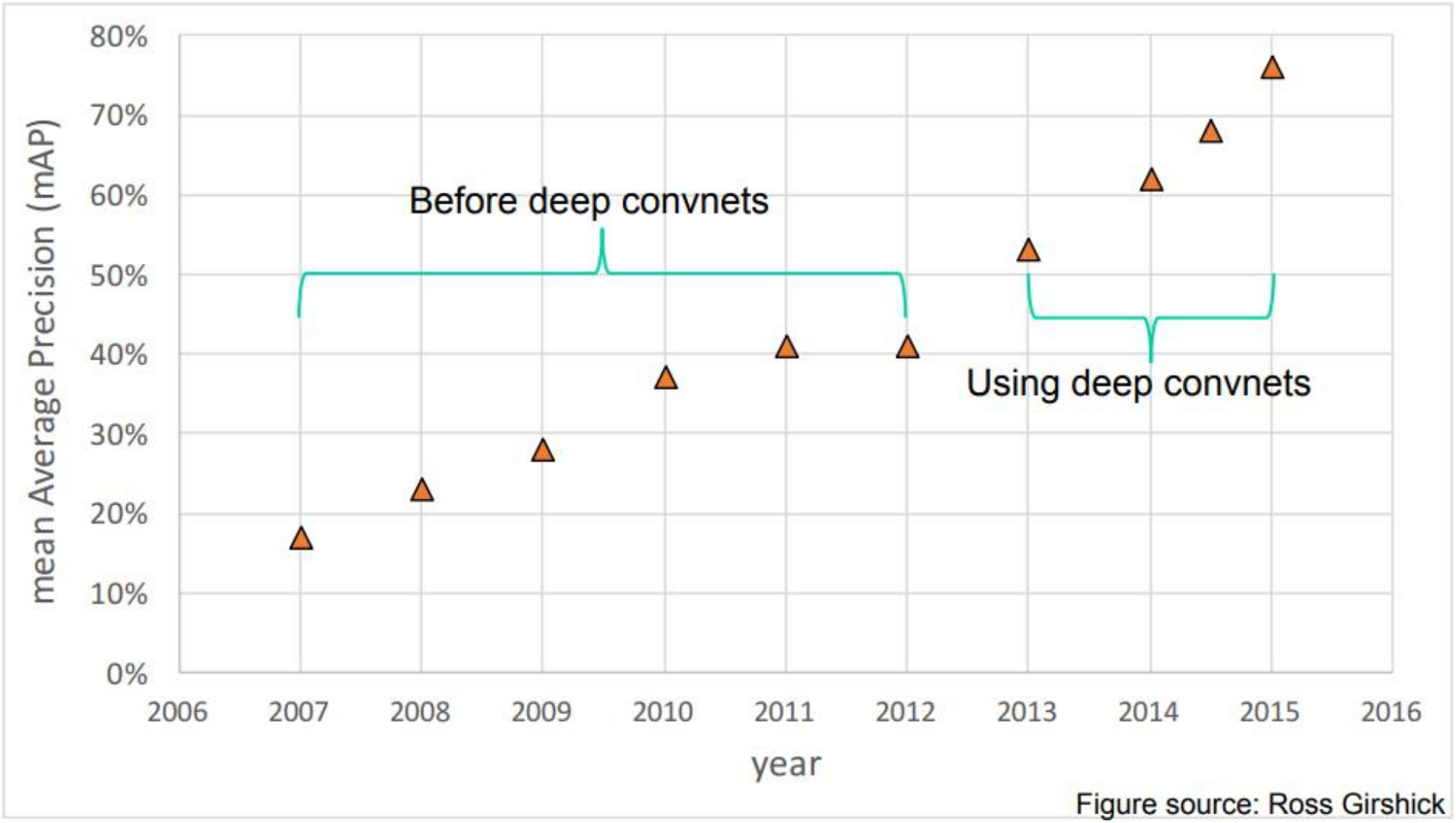


Figure source: Ross Girshick

ILSVRC 2012: top rankers

http://www.image-net.org/challenges/LSVRC/2012/results.html

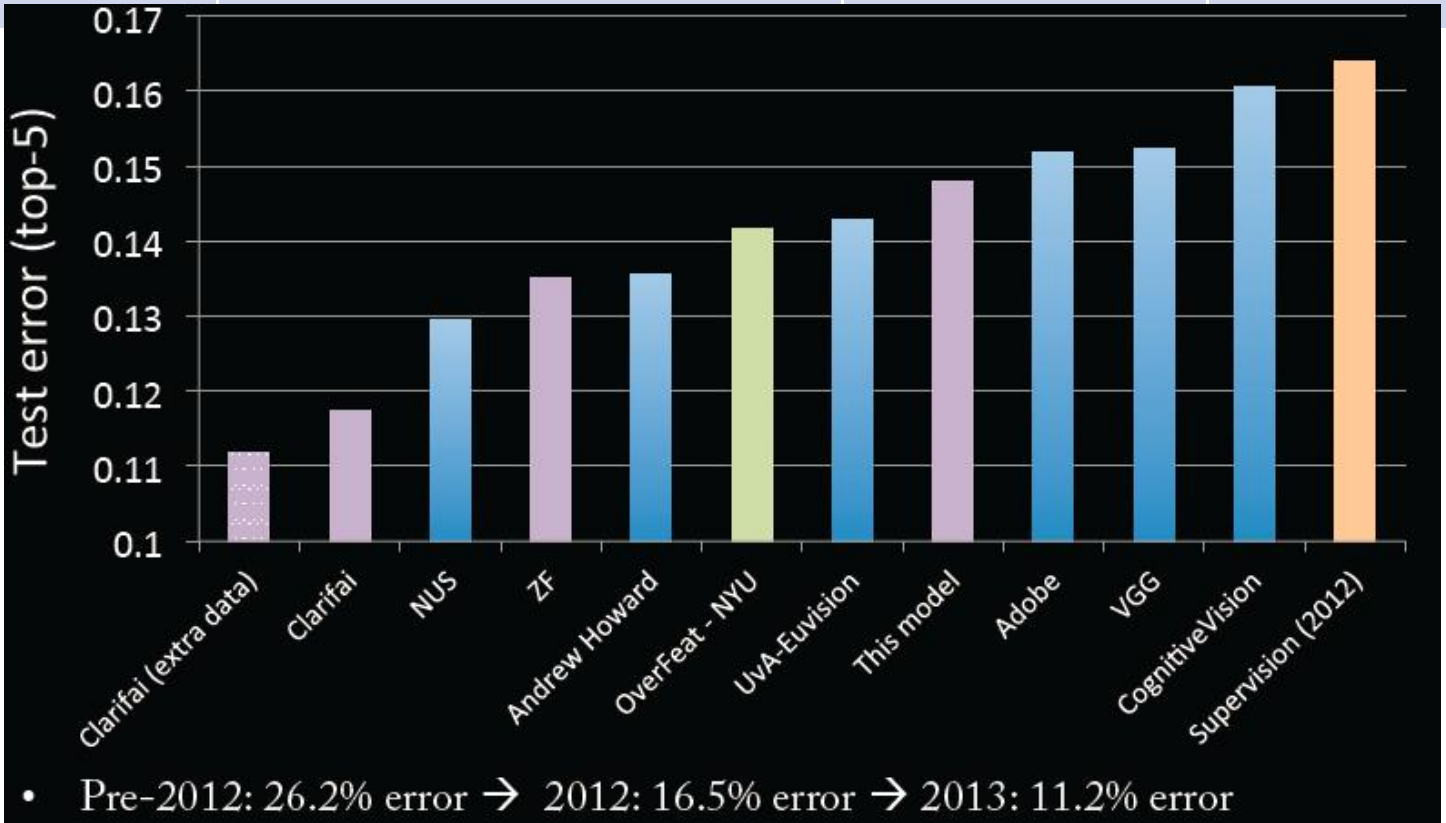
N	Error-5	Algorithm	Team	Authors
1	0.153	Deep Conv. Neural Network	Univ. of Toronto	Krizhevsky et al
2	0.262	Features + Fisher Vectors + Linear classifier	ISI	Gunji et al
3	0.270	Features + FV + SVM	OXFORD_VGG	Simonyan et al
4	0.271	SIFT + FV + PQ + SVM	XRCE/INRIA	Perronin et al
5	0.300	Color desc. + SVM	Univ. of Amsterdam	van de Sande et al

A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional
Neural Networks, NIPS 2012 (AlexNet)

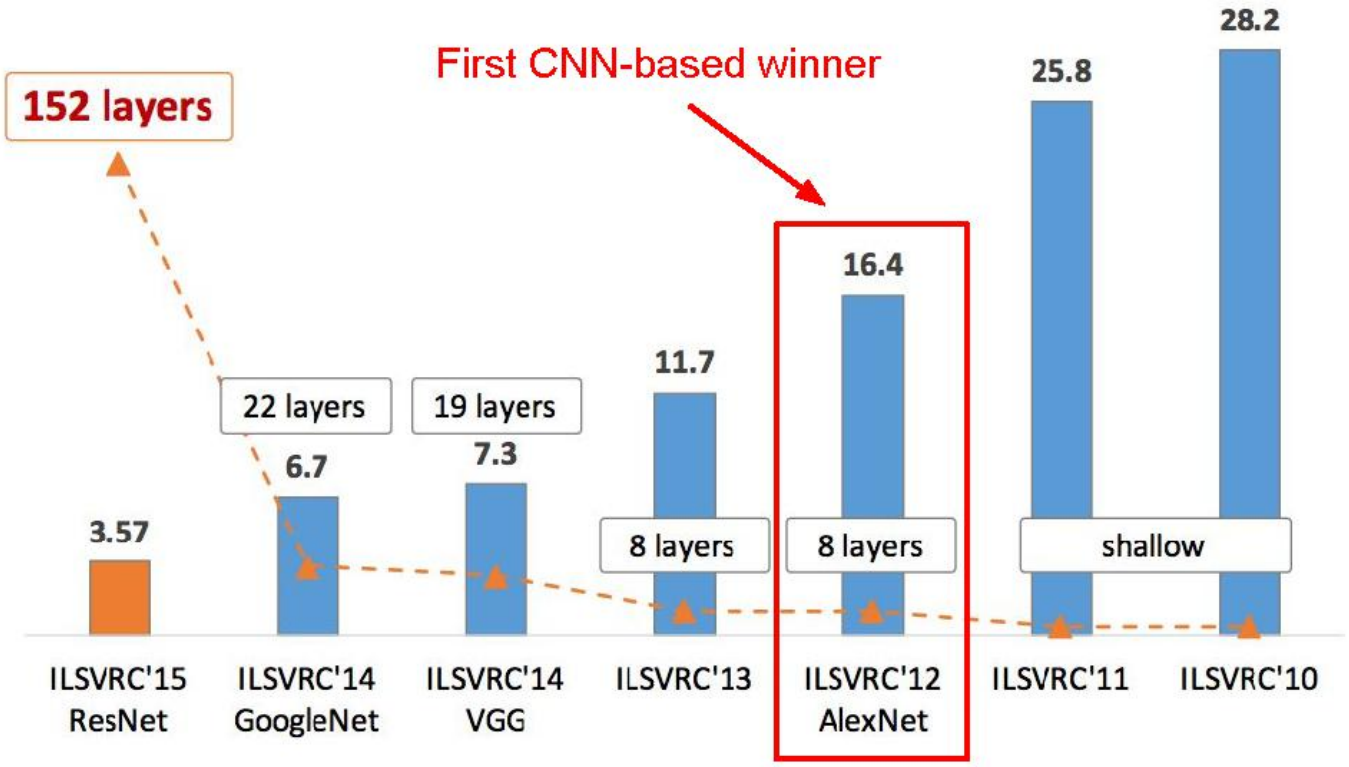
Imagenet 2013: top rankers

http://www.image-net.org/challenges/LSVRC/2013/results.php

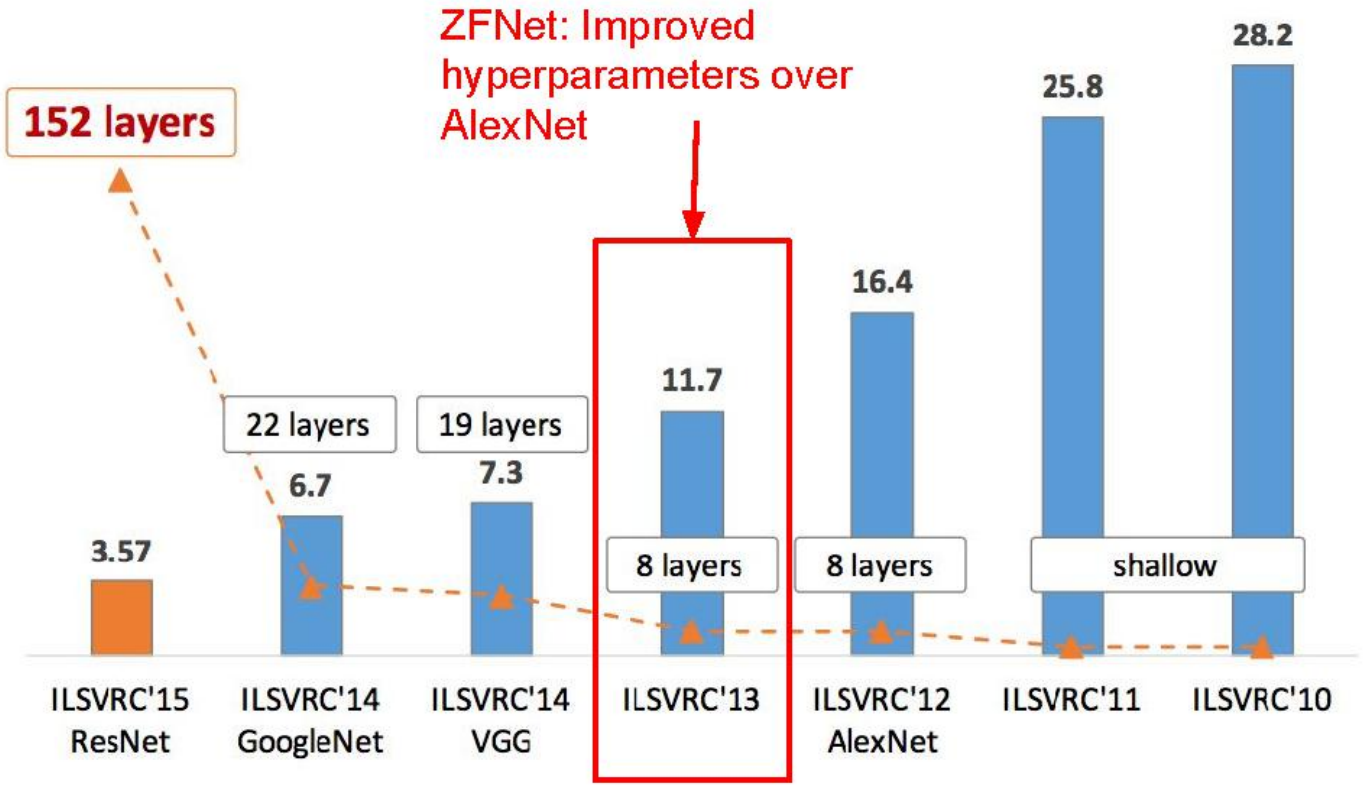
N	Error-5	Algorithm	Team	Authors
1	0.117	Deep Convolutional Neural Network	Clarifi	Zeiler
2	0.129	Deep Convolutional Neural Networks	Nat.Univ Singapore	Min LIN
3	0.135	Deep Convolutional Neural Networks	NYU	Zeiler Fergus
4	0.135	Deep Convolutional Neural Networks		Andrew Howard
5	0.137	Deep Convolutional Neural Networks	Overfeat NYU	Pierre Sermanet et al



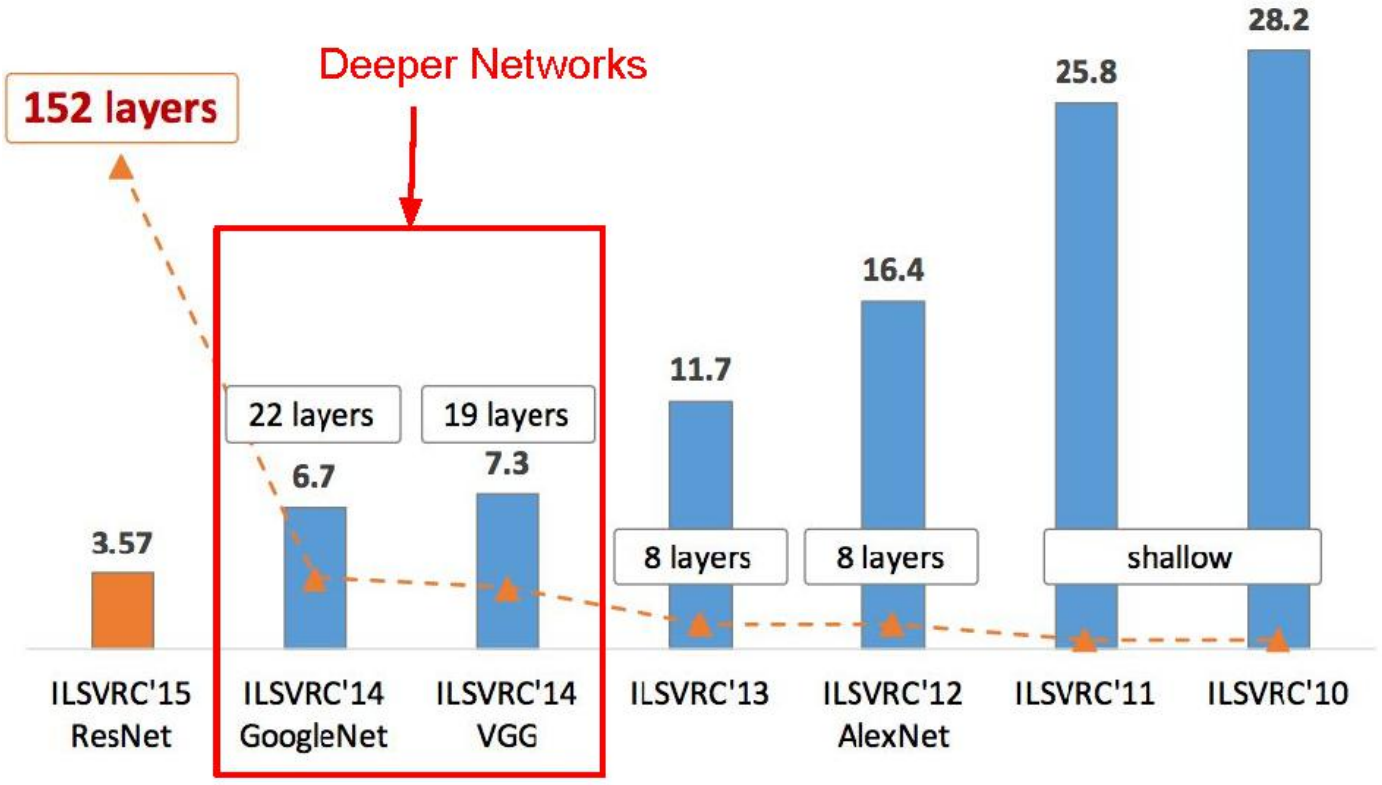
ILSVRC Winners

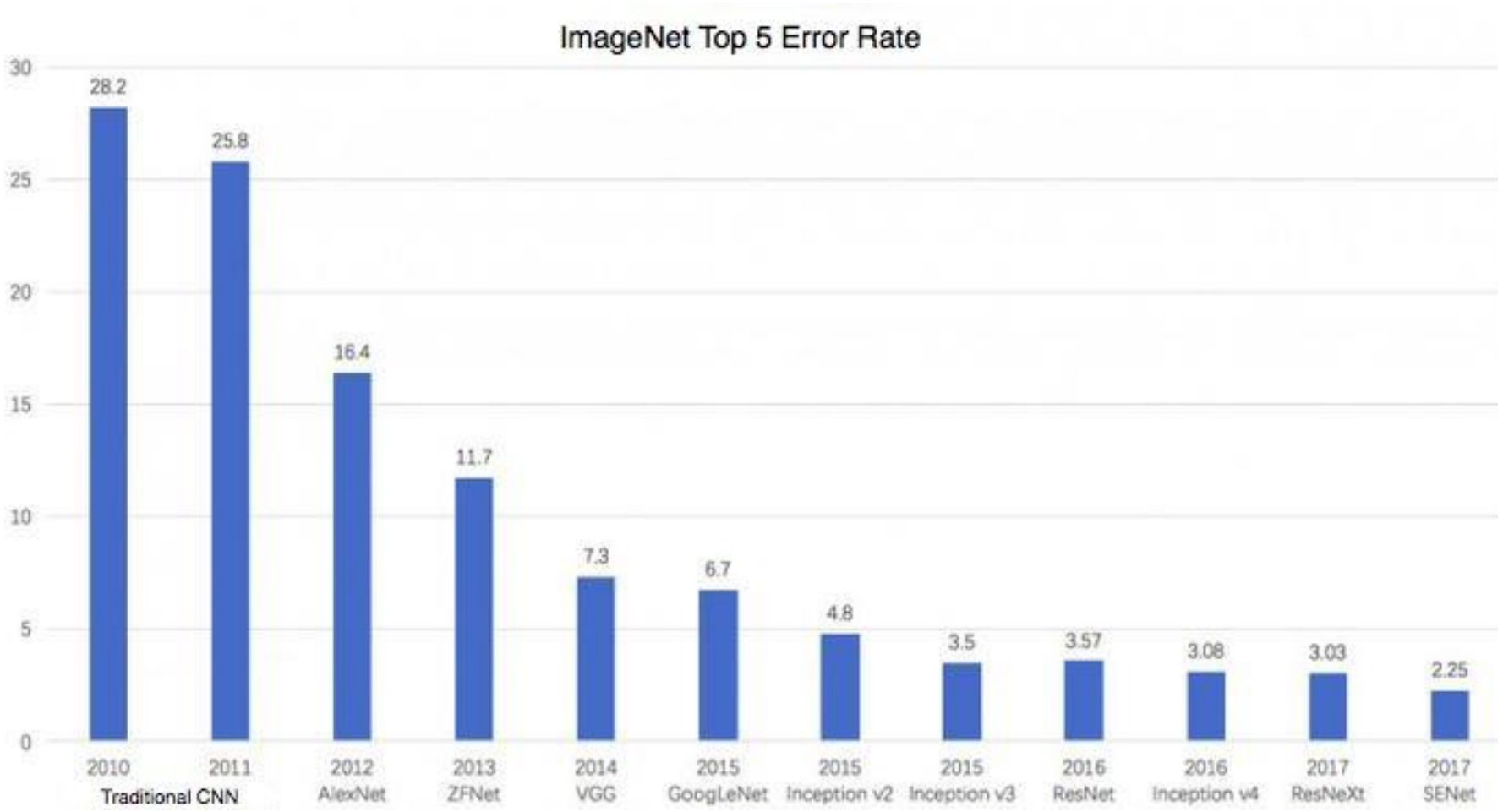


ILSVRC Winners



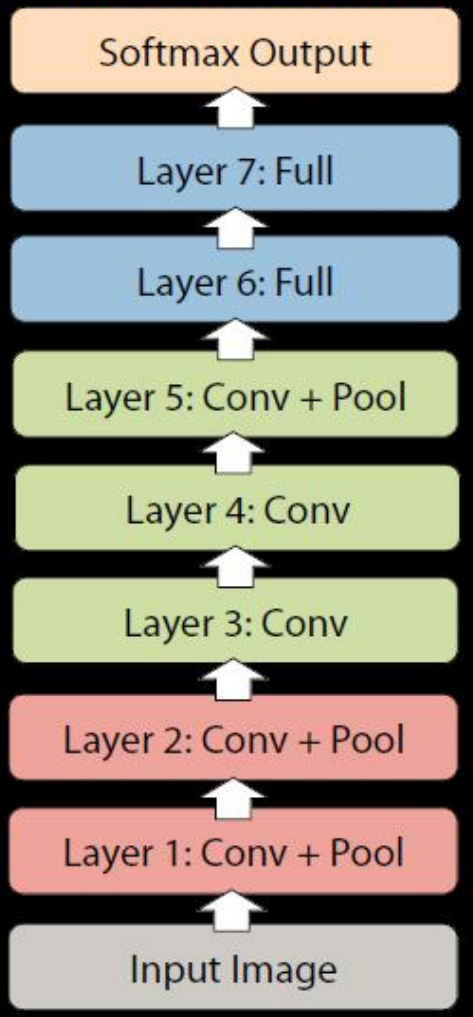
ILSVRC Winners




ImageNet Top 5 Error Rate
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
4.8
3.5
3.57
3.08
3.03
2.25
2010
Traditional CNN
2011
2012
AlexNet
2013
ZFNet
2014
VGG
2015
GoogLeNet
2015
Inception v2
2015
Inception v3
2016
ResNet
2016
Inception v4
2017
ResNeXt
2017
SENet

Why CNNs should be Deep?

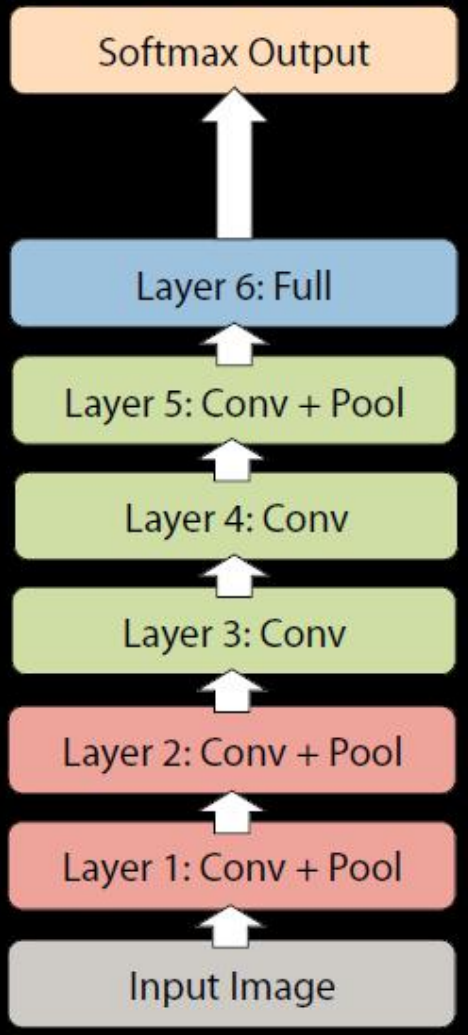
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:
  18.1% top-5 error



Rob Fergus, NIPS 2013

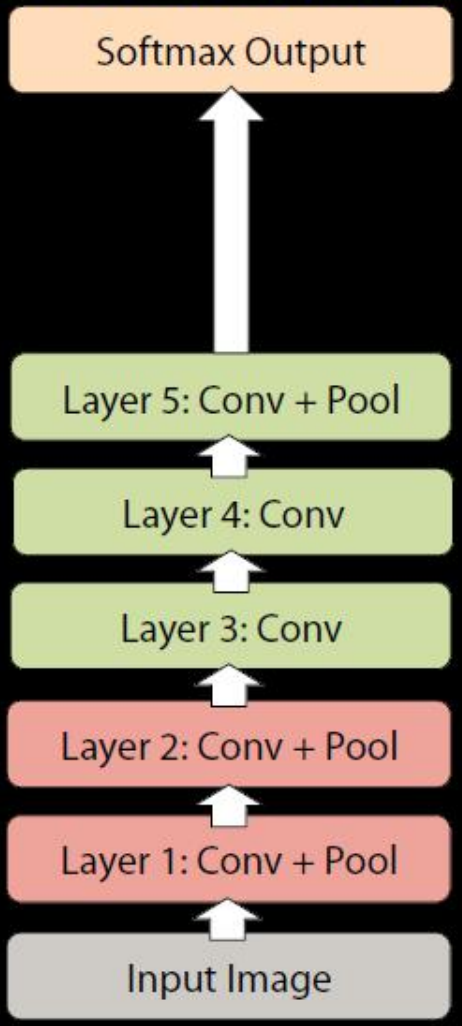
Why ConvNet should be Deep?

- Remove top fully connected layer
  - Layer 7
- Drop 16 million parameters
- Only 1.1% drop in performance!



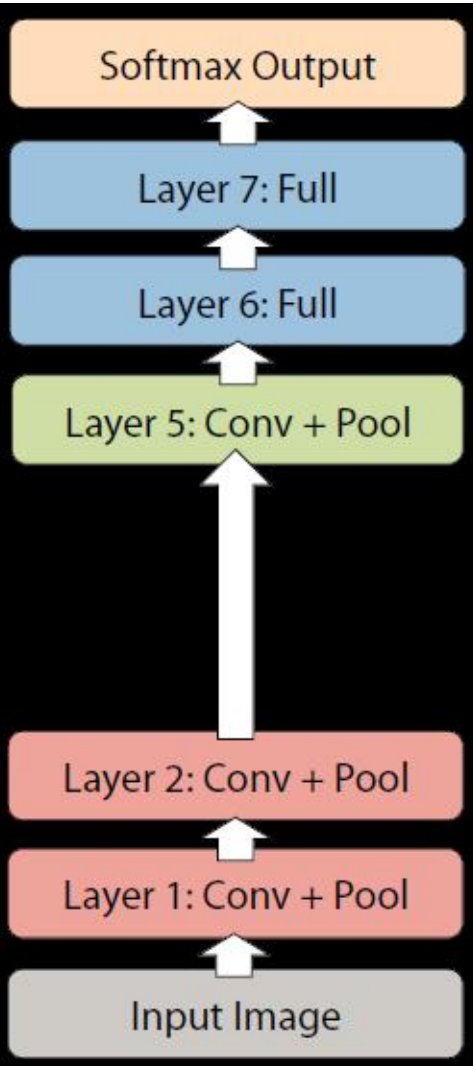
Why CNNs should be Deep?

- Remove both fully connected layers
  - Layer 6 & 7
- Drop ~50 million parameters
- 5.7% drop in performance



Why CNNs should be Deep?

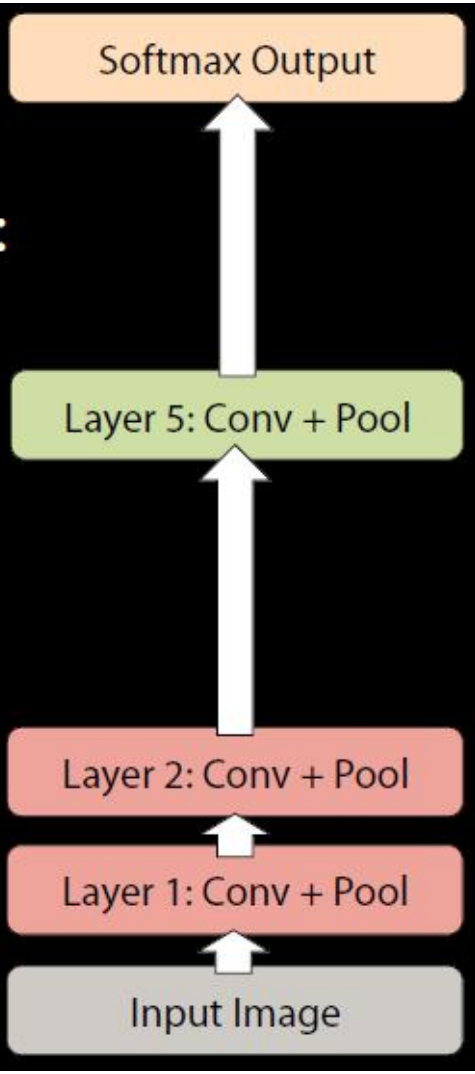
- Now try removing upper feature extractor layers:
  - Layers 3 & 4
- Drop ~1 million parameters
- 3.0% drop in performance



Why CNNs should be Deep?

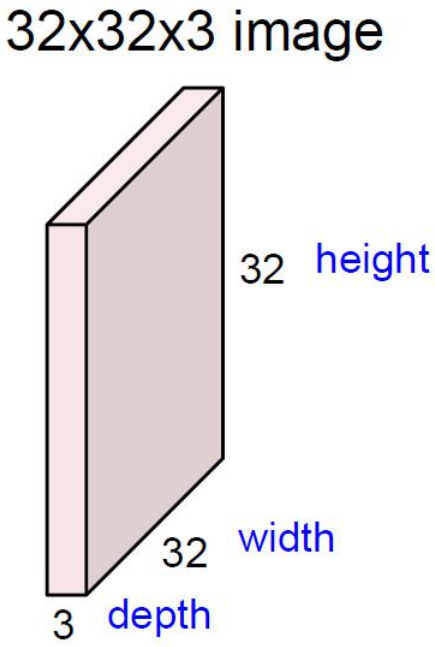
- Now try removing upper feature extractor layers & fully connected:
  - Layers 3, 4, 6 ,7
- Now only 4 layers
- 33.5% drop in performance

→Depth of network is key

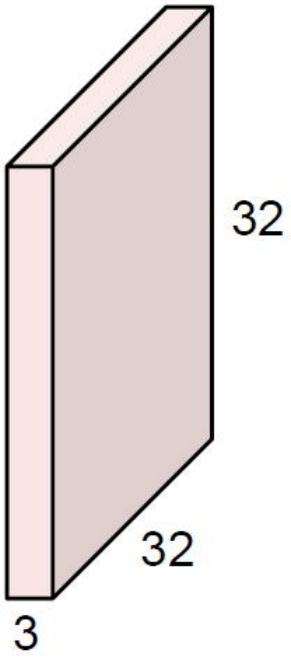


Convolution Operations

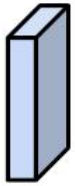
Convolution Layer



32x32x3 image



5x5x3 filter

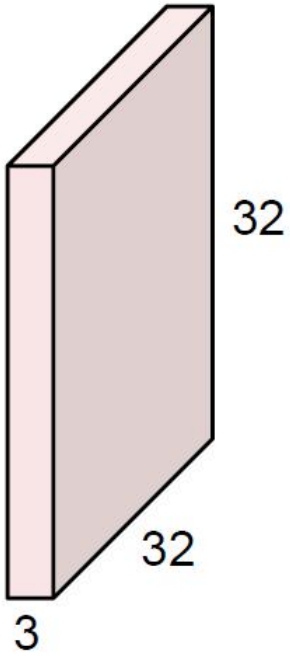


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

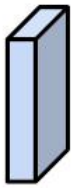
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

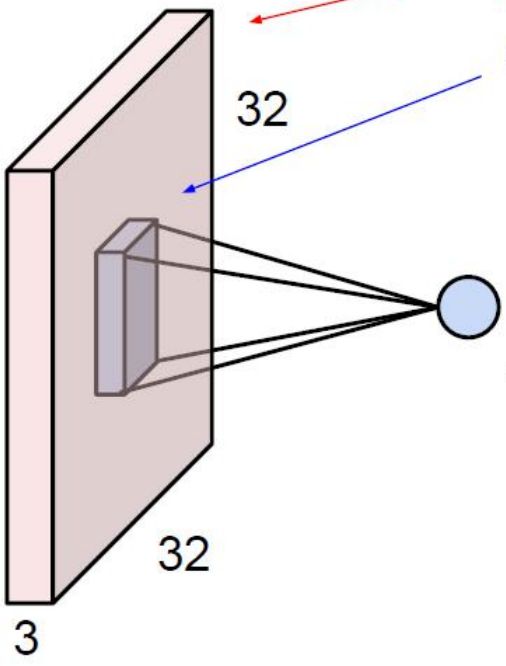


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

We call the layer convolutional because it is related to convolution of two signals:

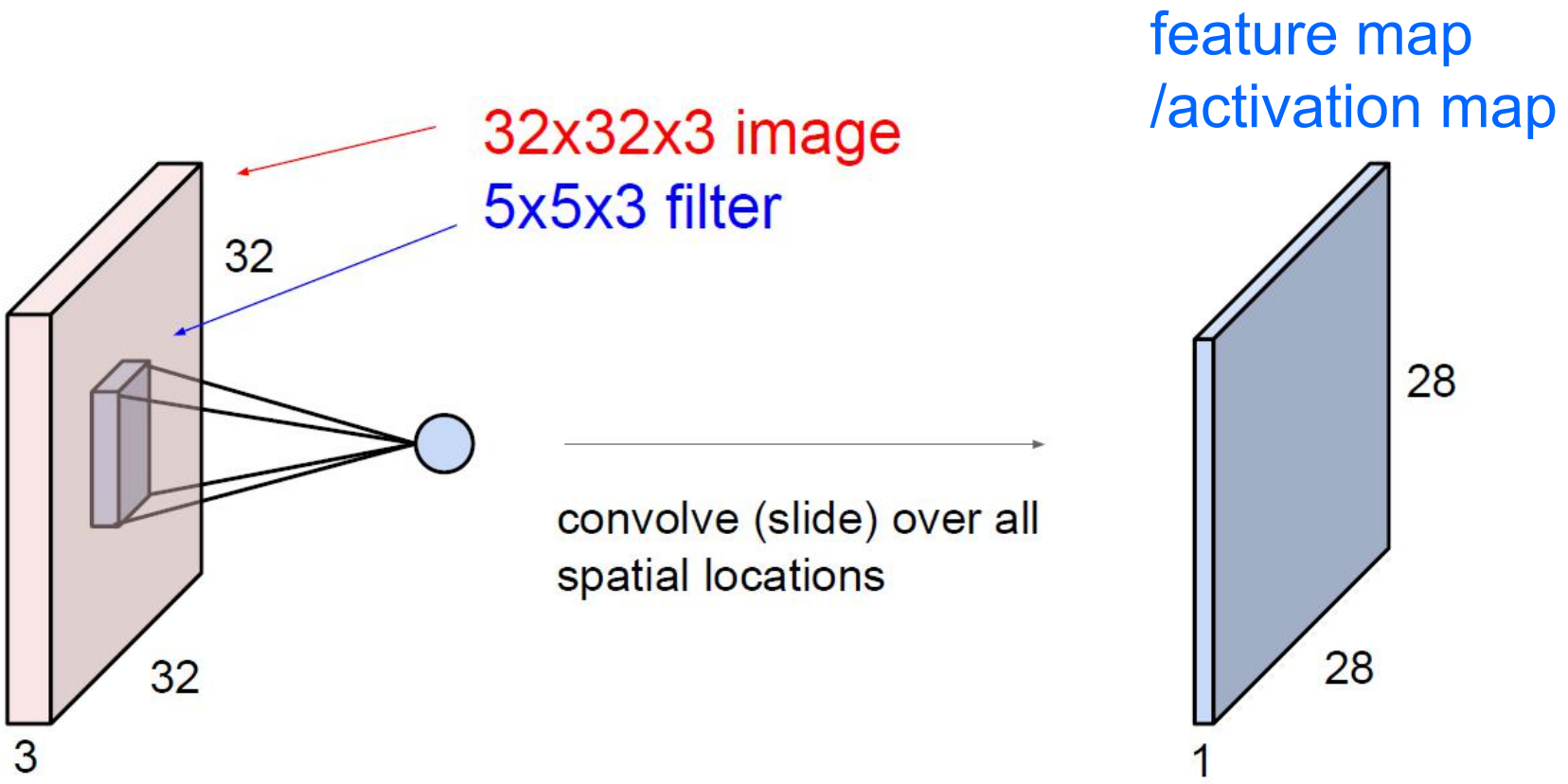
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

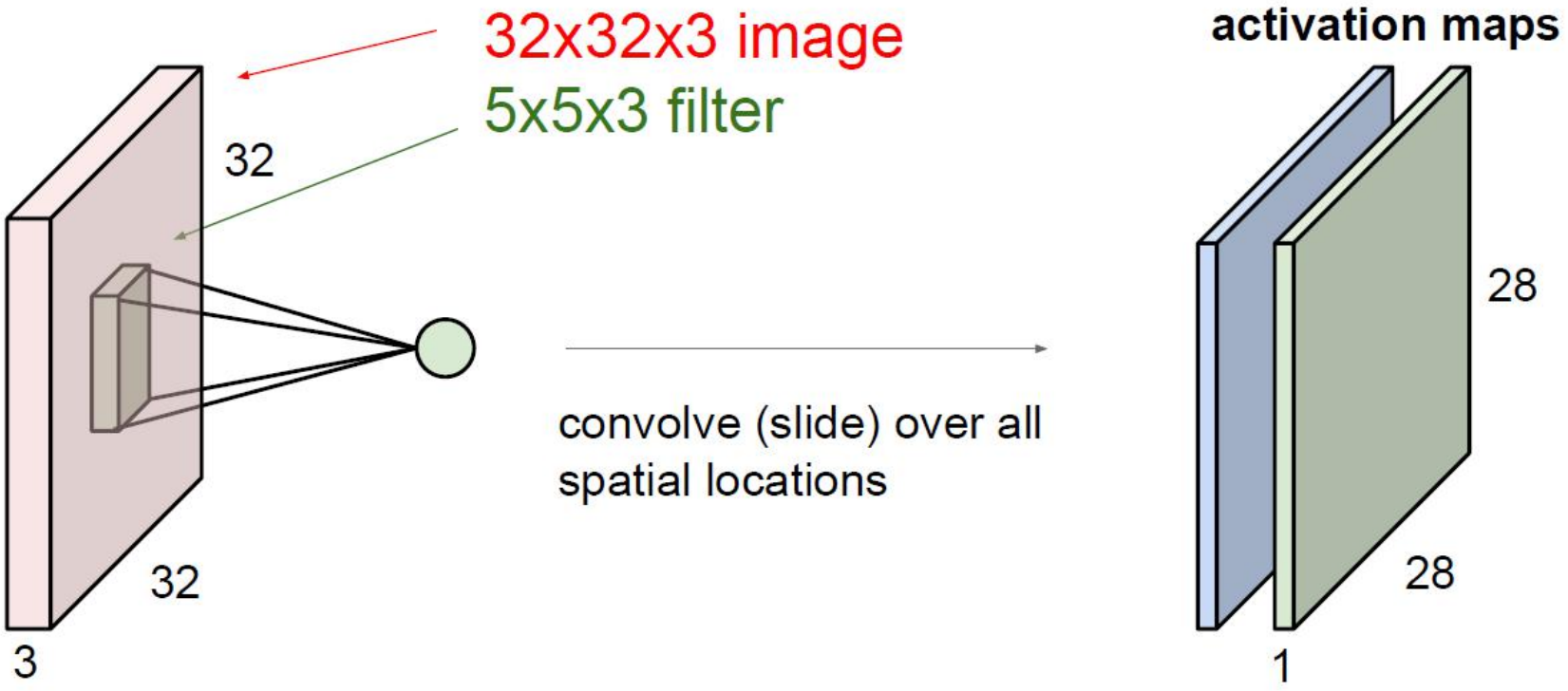


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

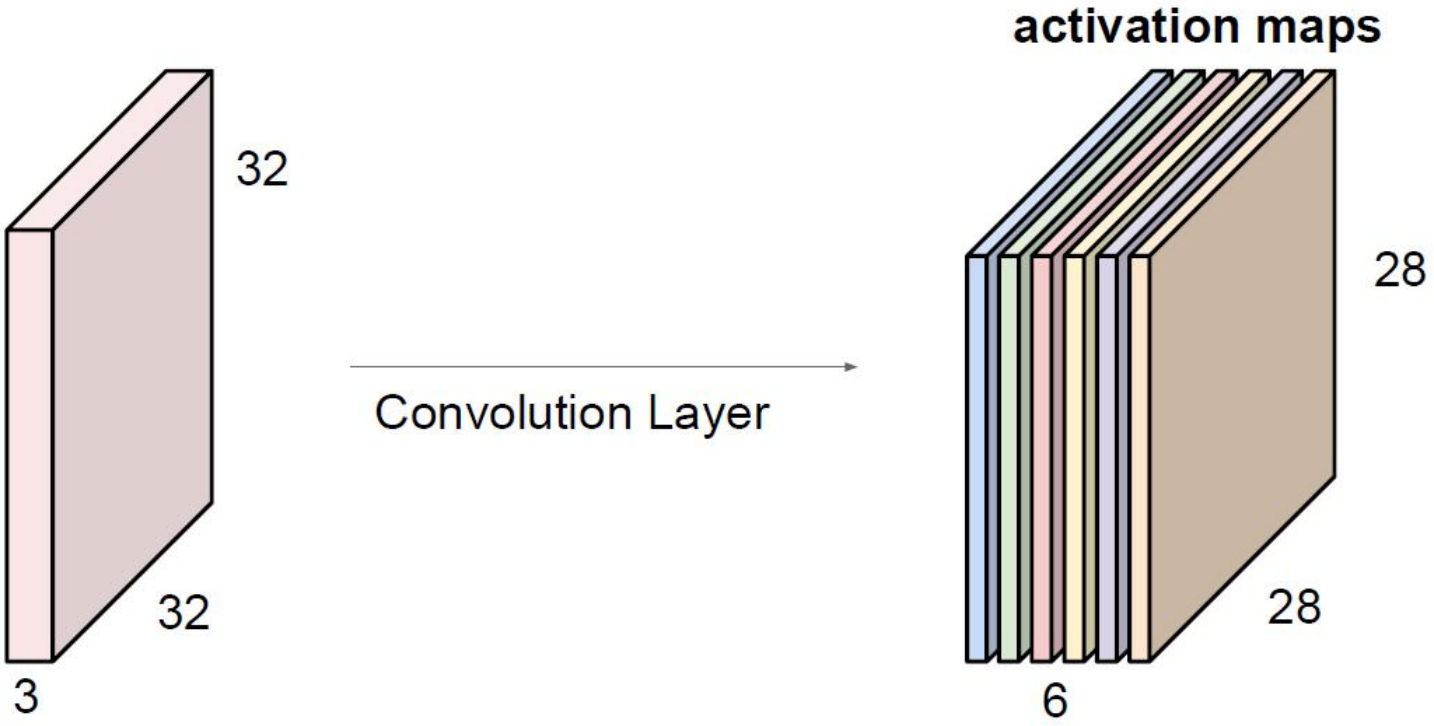
$$w^T x + b$$


32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
feature map
/activation map
28
28
1

Consider a second, green kernel

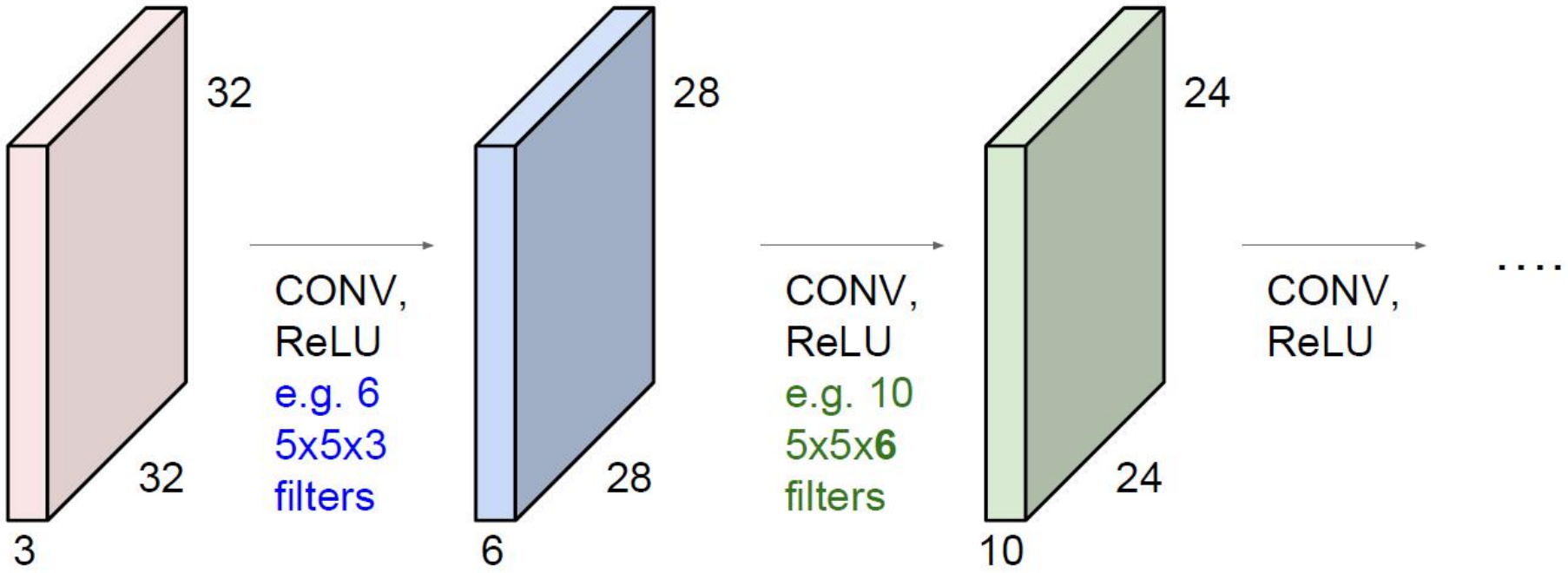


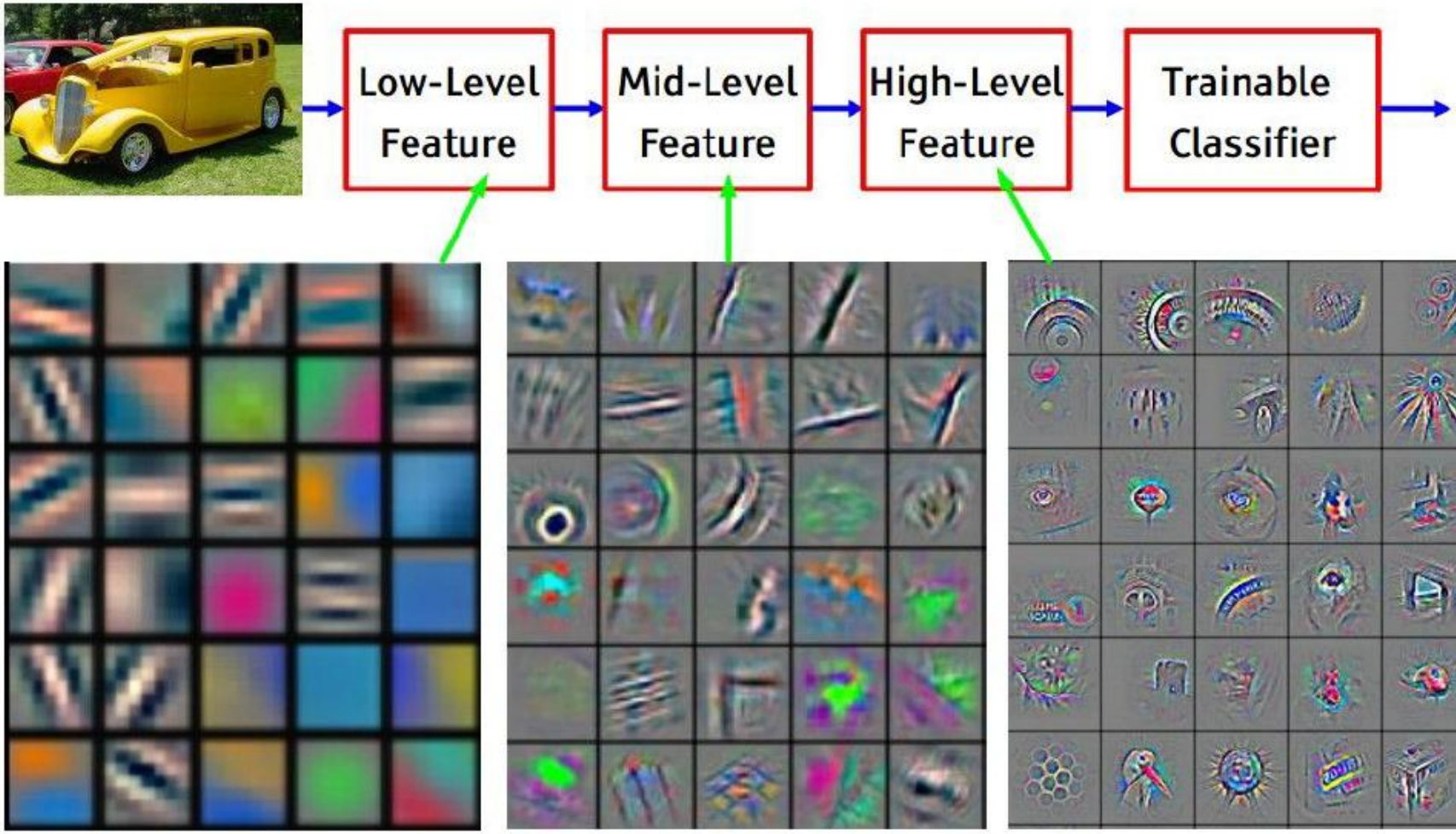
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



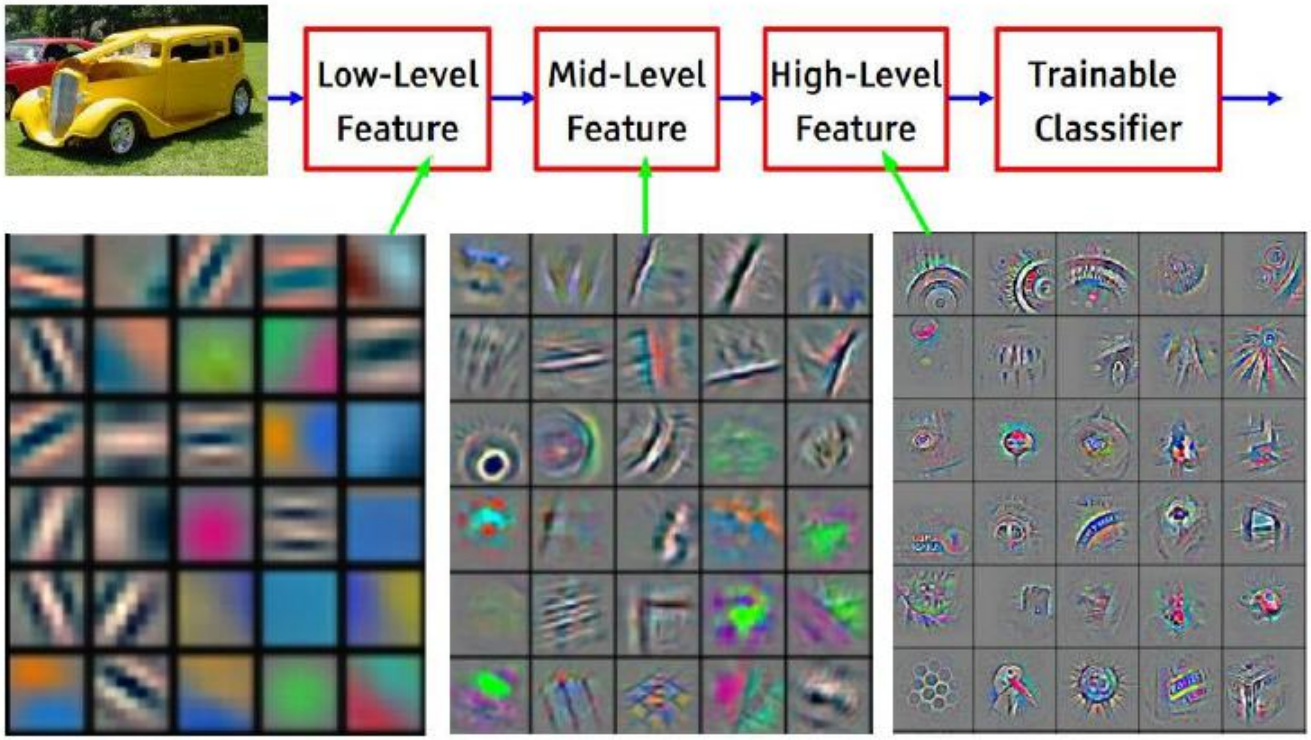
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

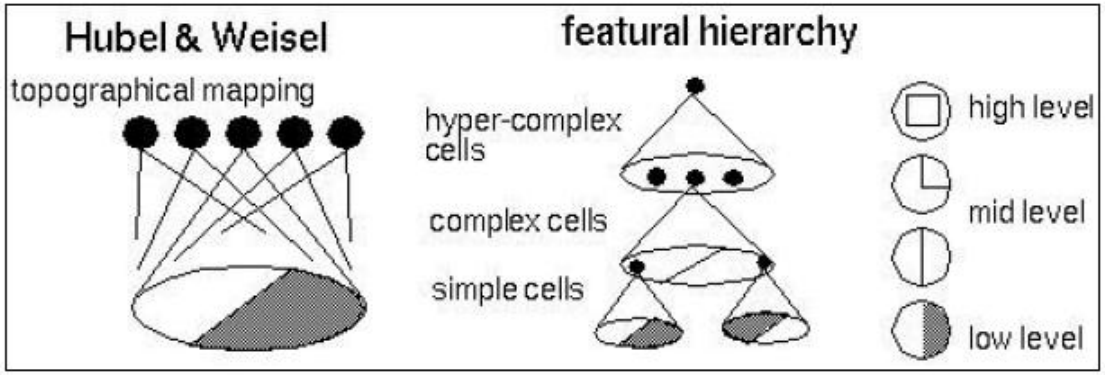




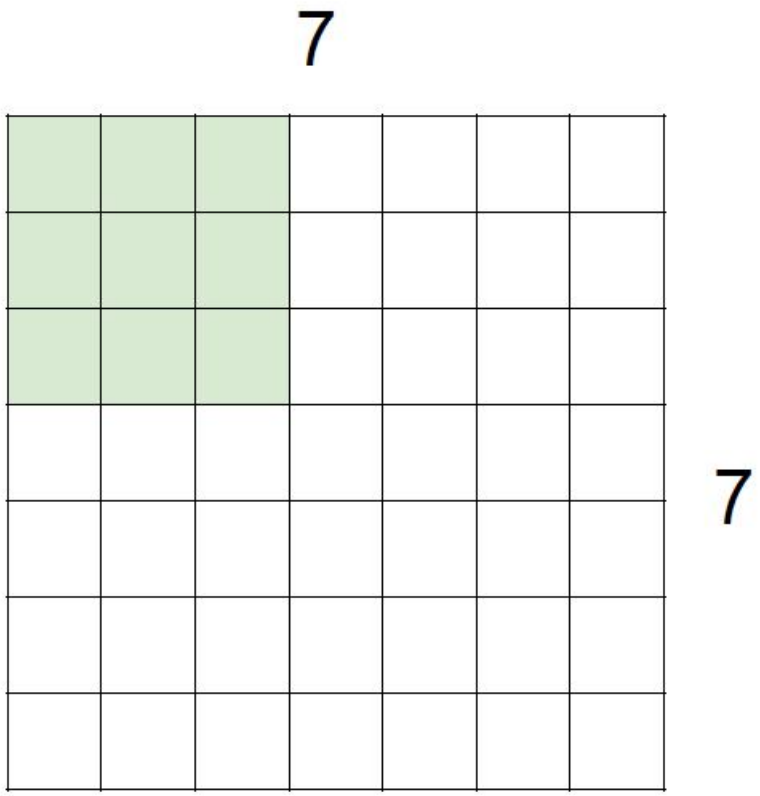
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



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

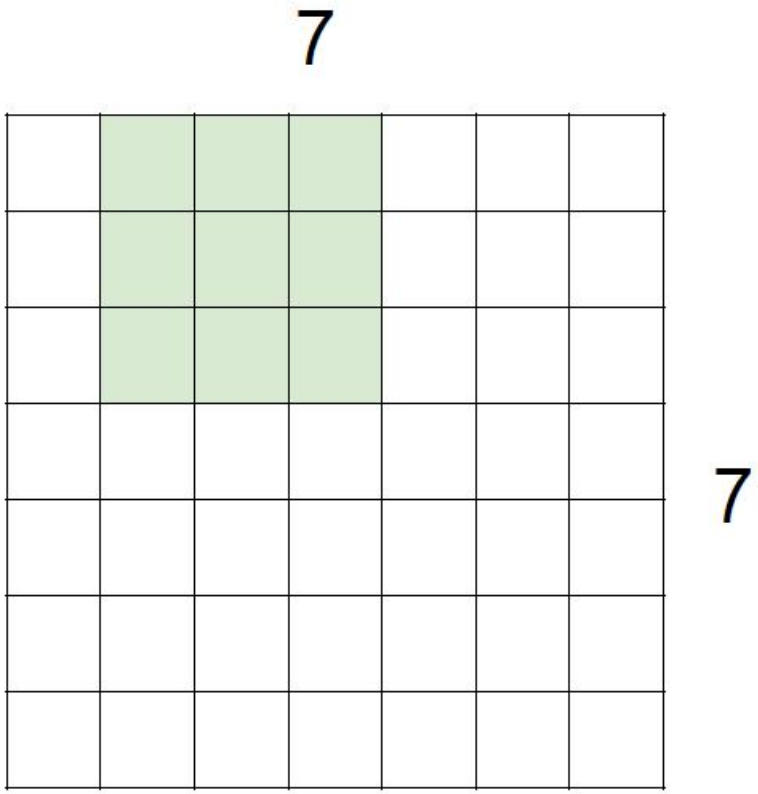


Stride=1



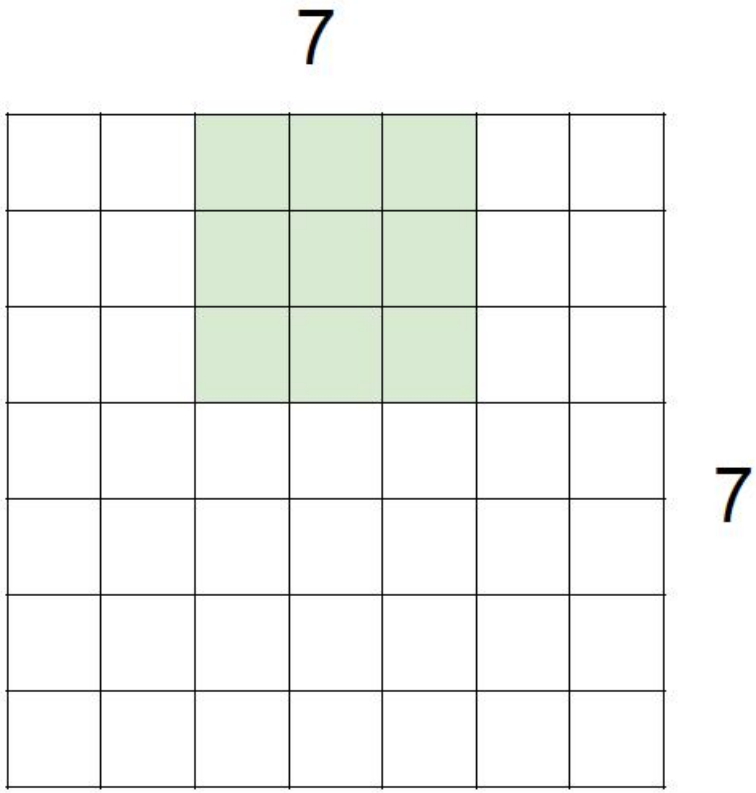
7x7 input (spatially)
assume 3x3 filter

Stride=1



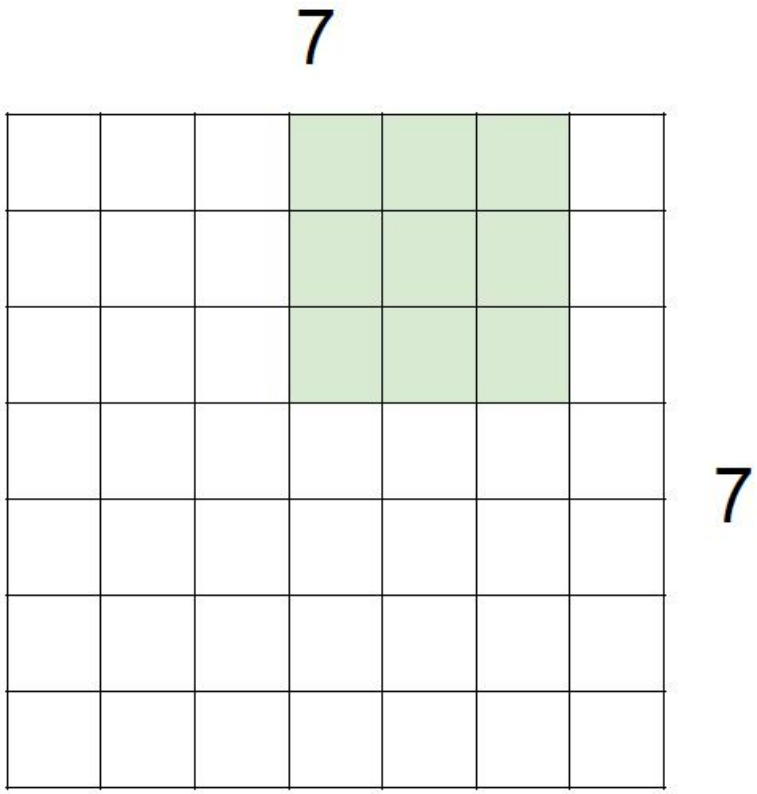
7x7 input (spatially)
assume 3x3 filter

Stride=1



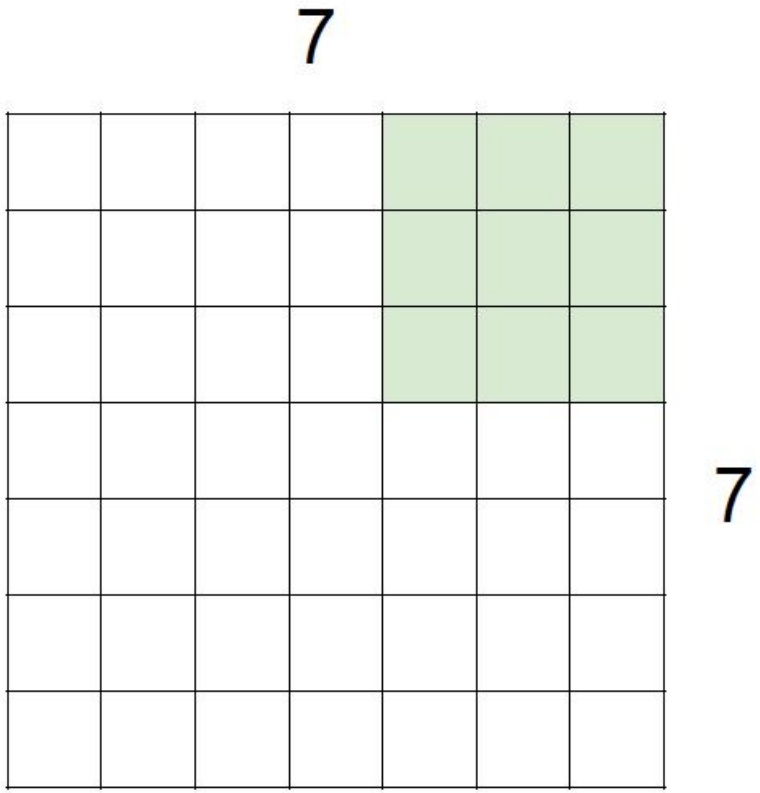
7x7 input (spatially)
assume 3x3 filter

Stride=1



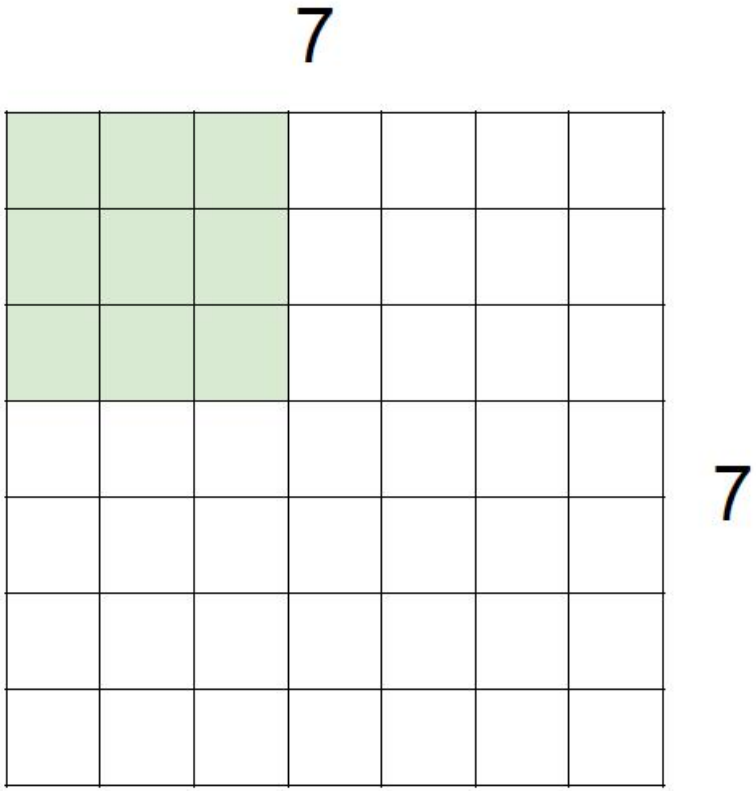
7x7 input (spatially)
assume 3x3 filter

Stride=1

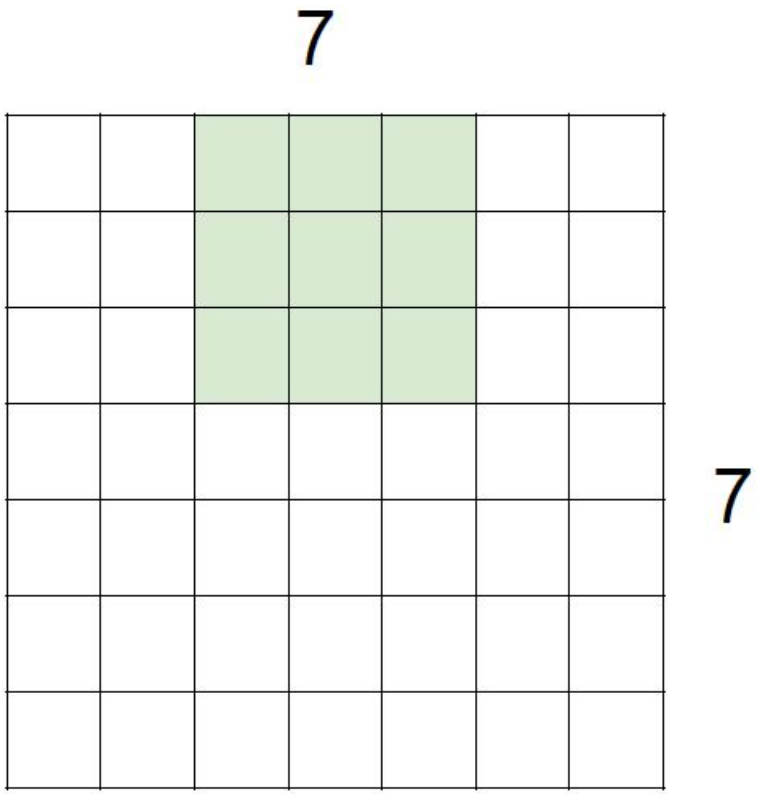


7x7 input (spatially)
assume 3x3 filter

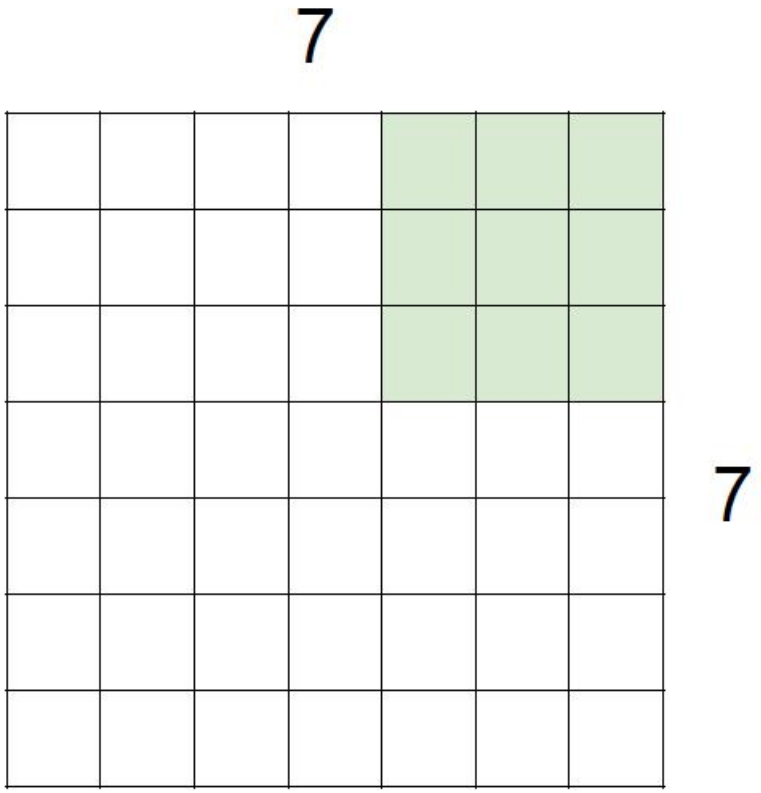
=> 5x5 output



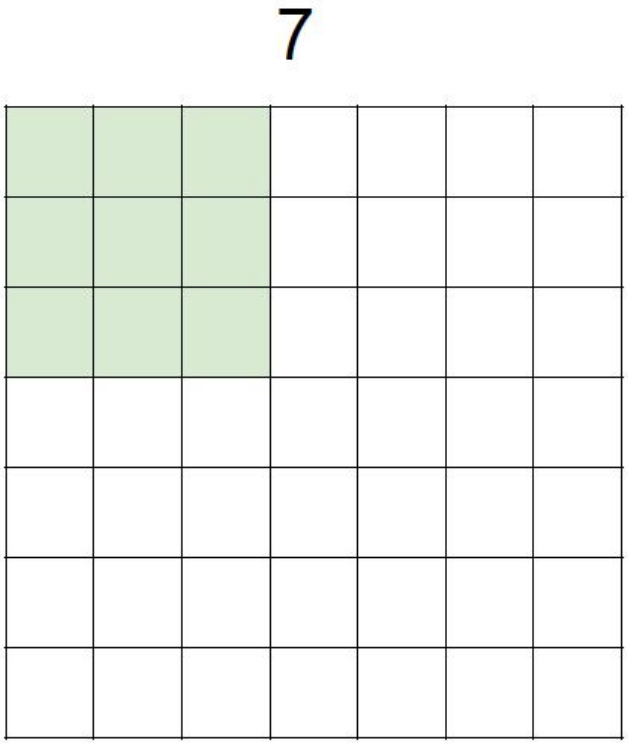
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

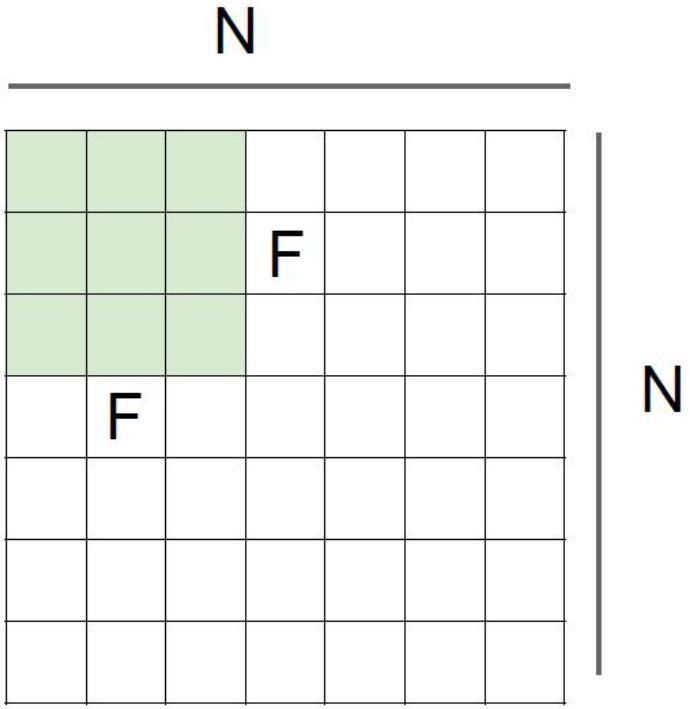


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

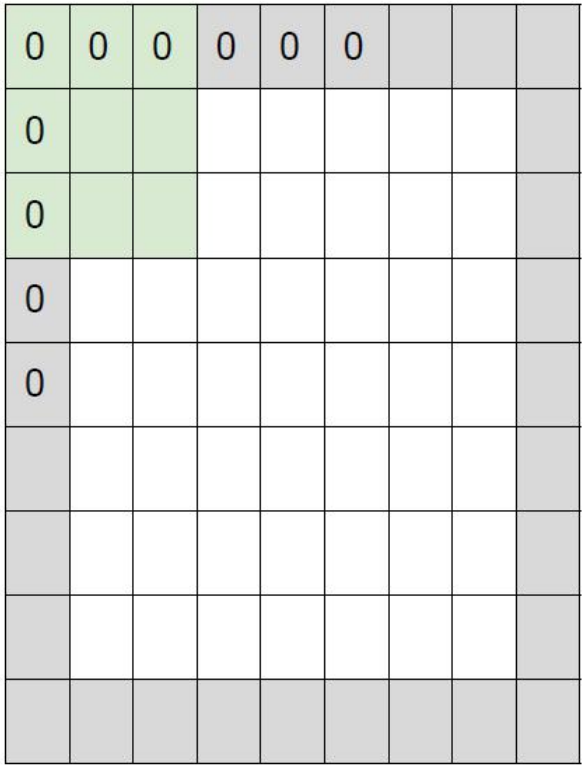
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Output size:
(N - F) / stride + 1

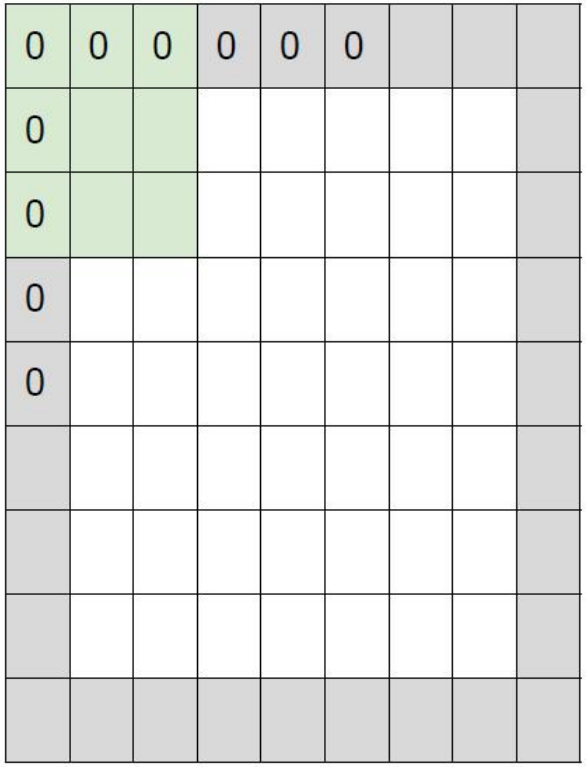
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

In practice: Common to zero pad the border



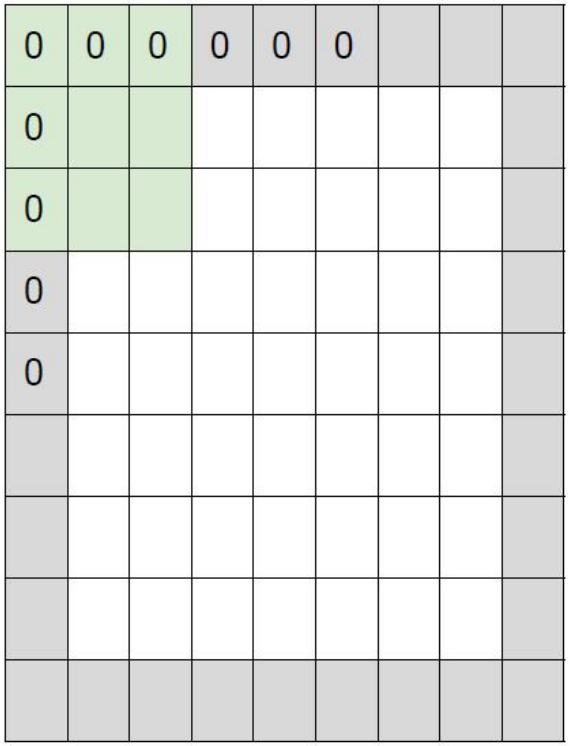
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

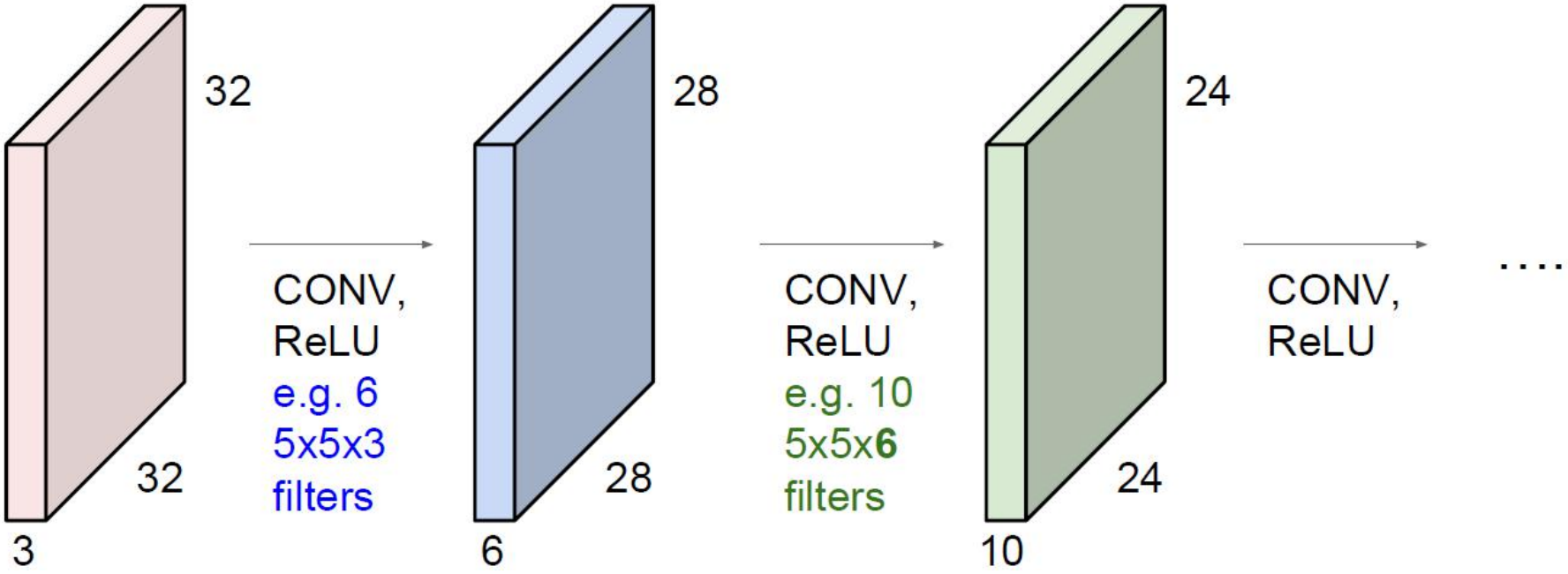


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

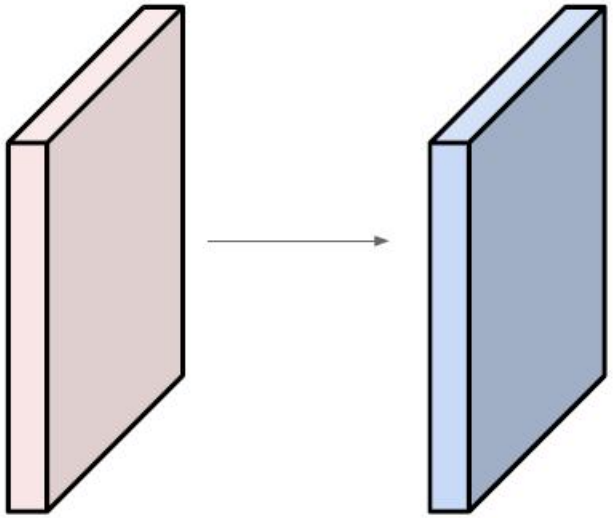
Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



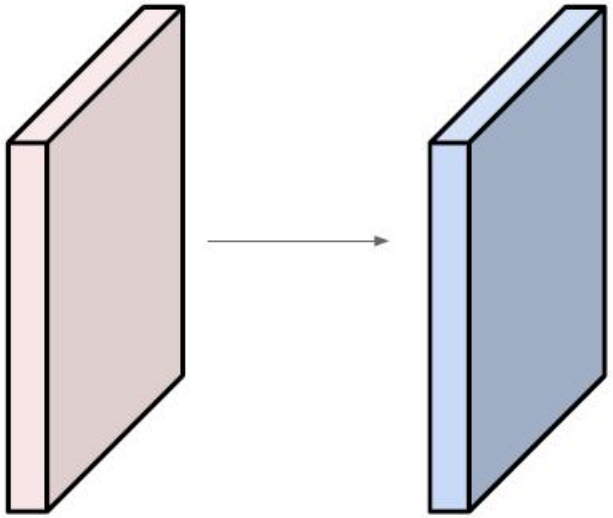
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



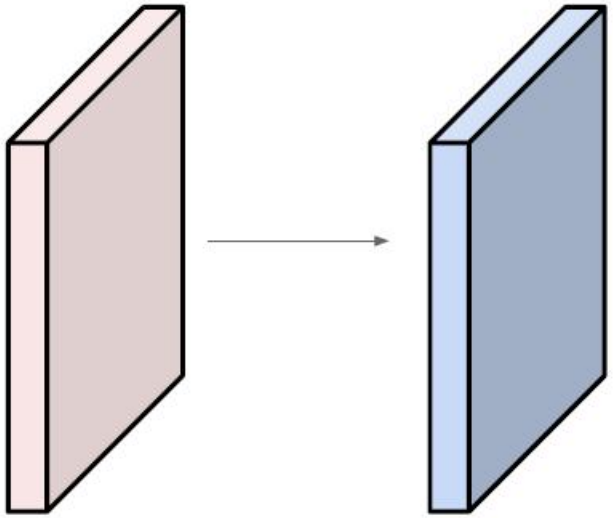
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

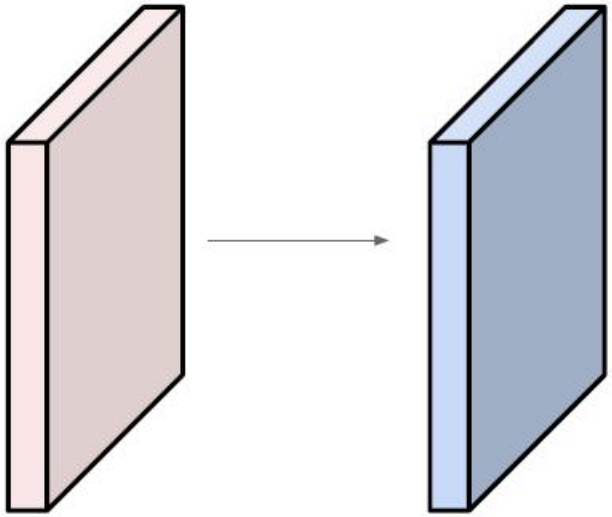
Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?





Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Summary. To summarize, the Conv Layer:

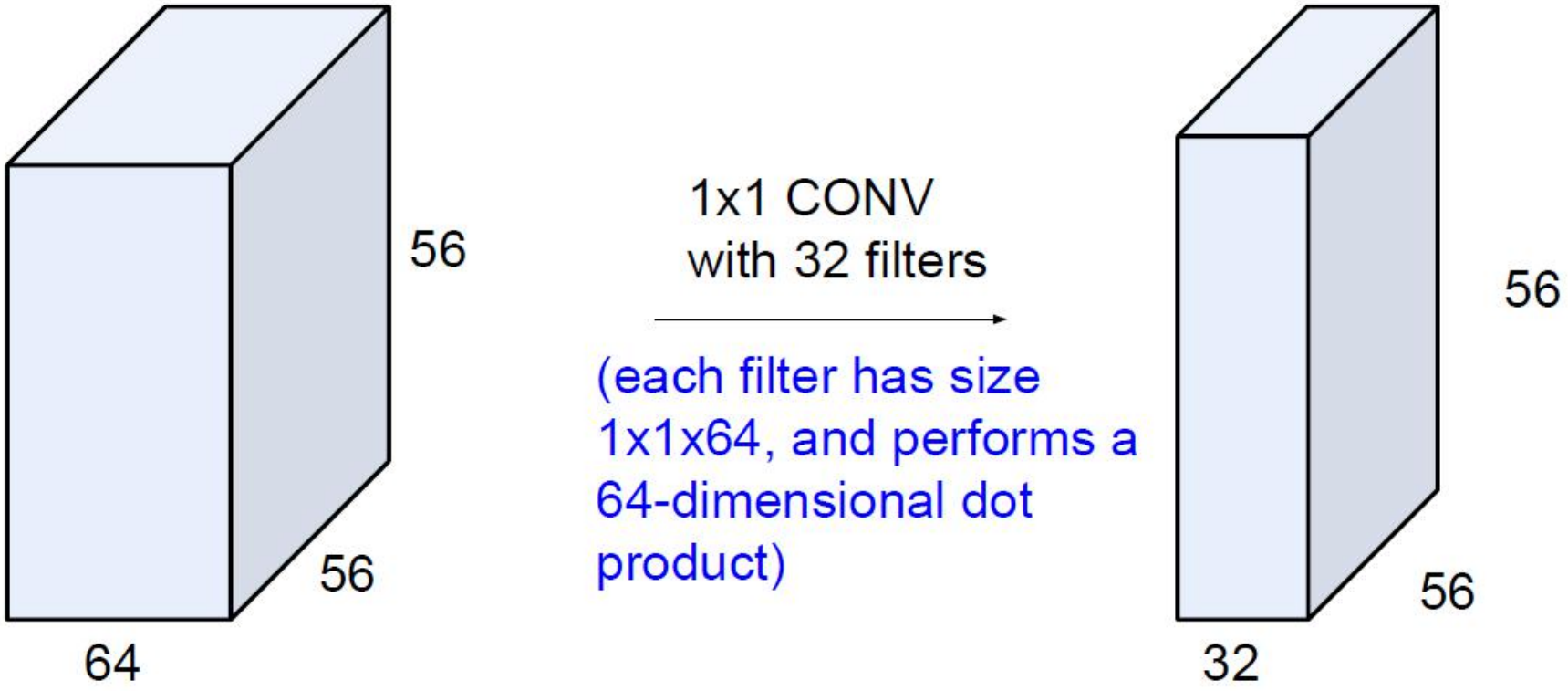
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

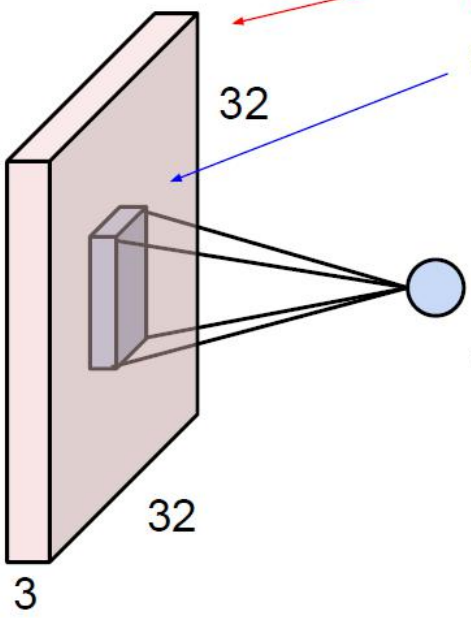
K = (powers of 2, e.g. 32, 64, 128, 512)

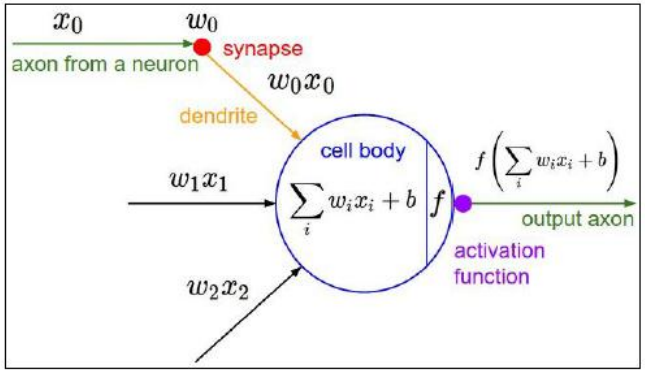
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

1x1 convolution layers make perfect sense



The brain/neuron view of CONV Layer

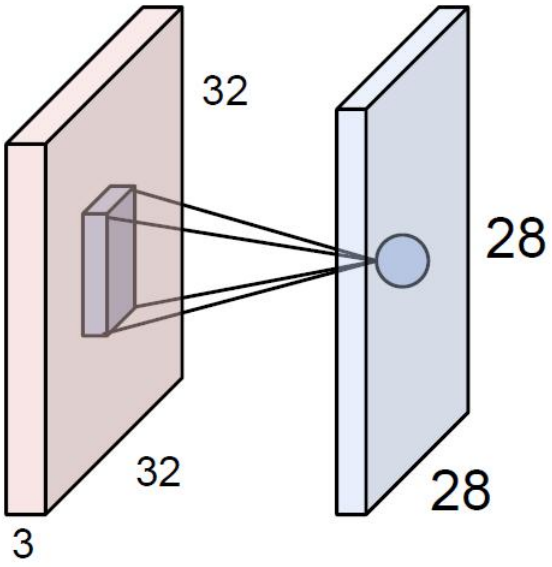


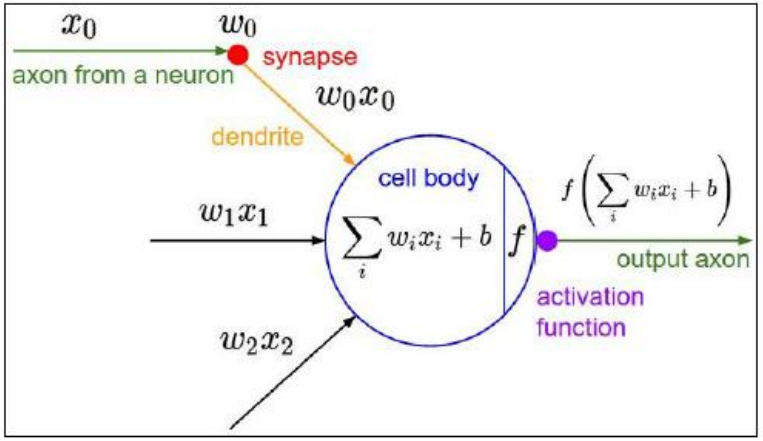


It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

The brain/neuron view of CONV Layer





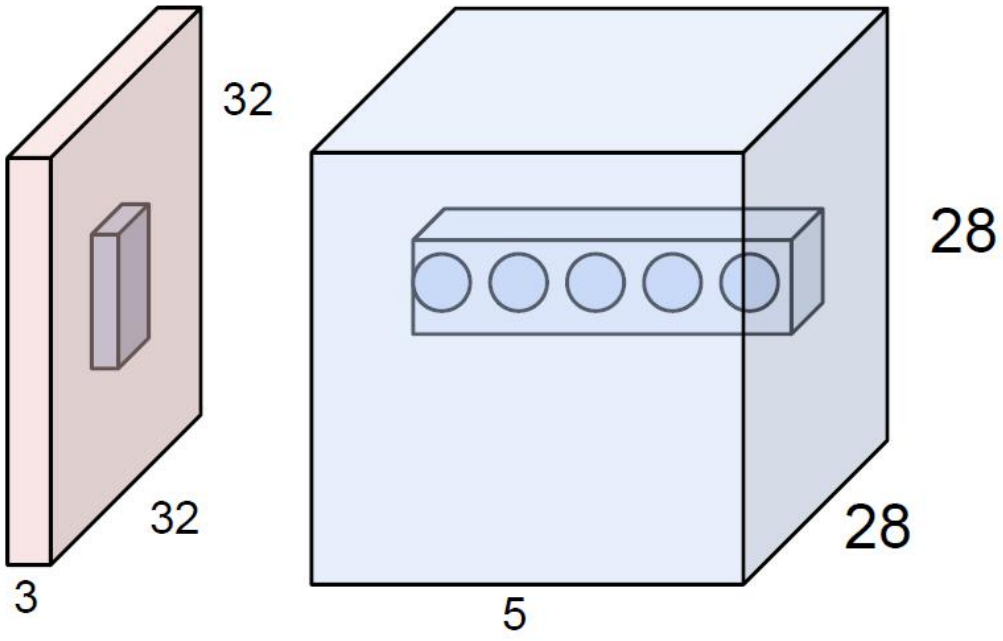
An activation map is a 28x28 sheet of neuron outputs:

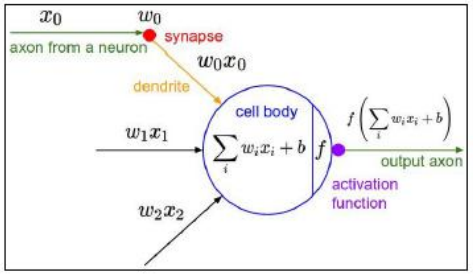
✓ Each is connected to a small region in the input

✓ All of them share parameters

"5x5 filter" -> "5x5 receptive field for each neuron

The brain/neuron view of CONV Layer



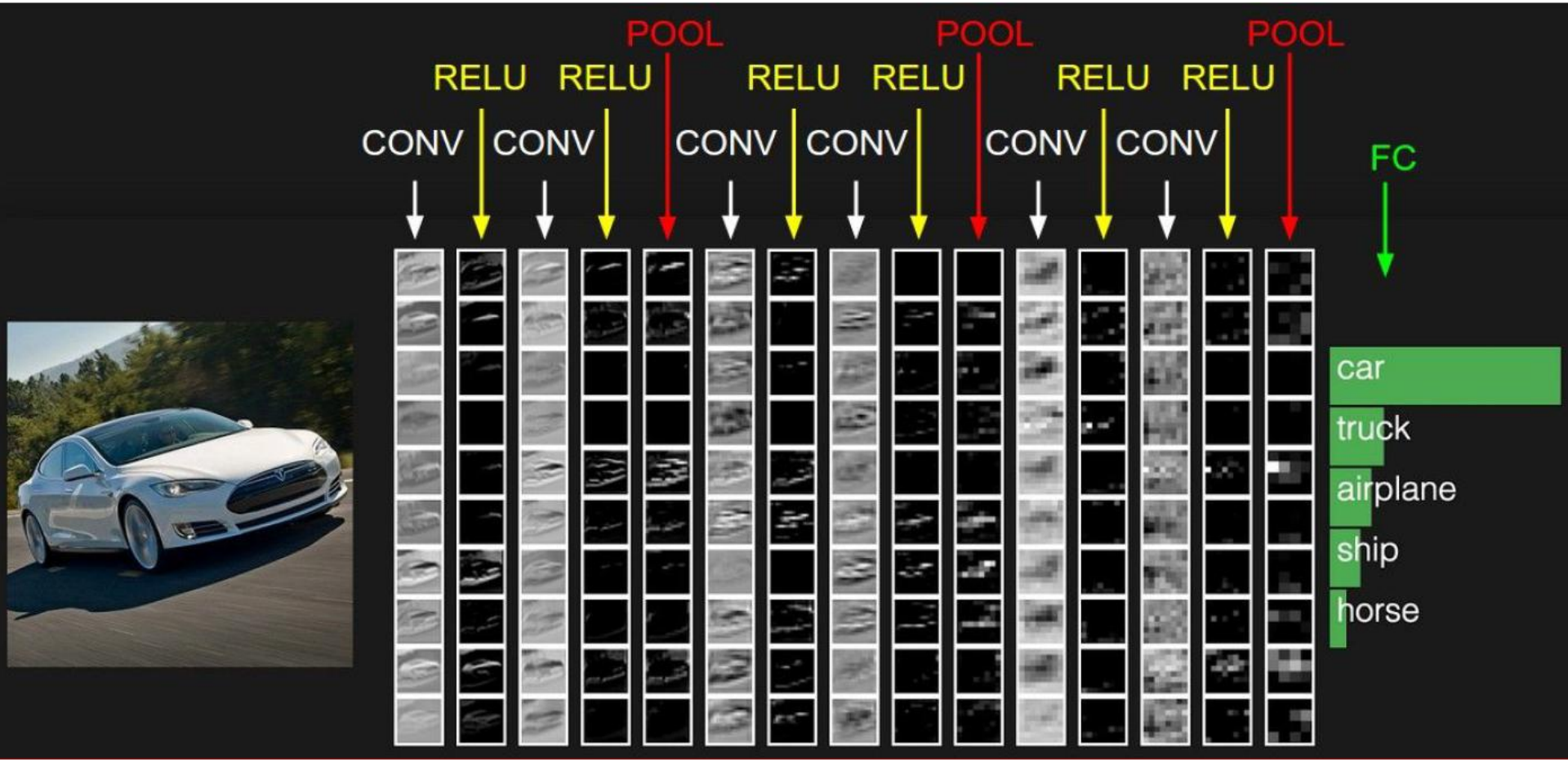


E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume

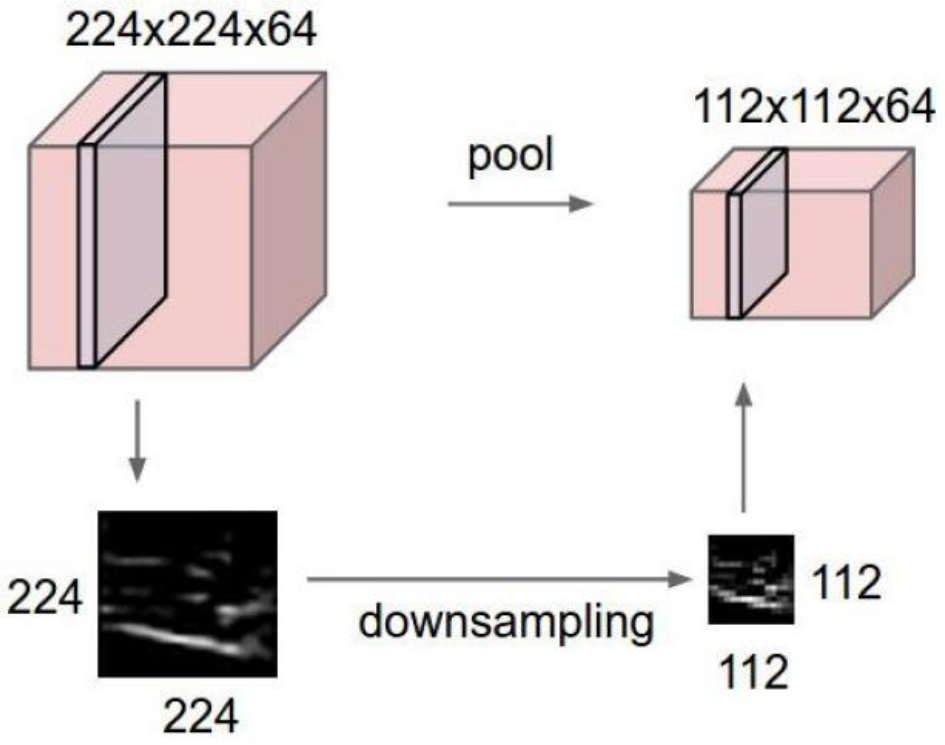
Pooling/FC layers

Two more layers to go: POOLing/FC

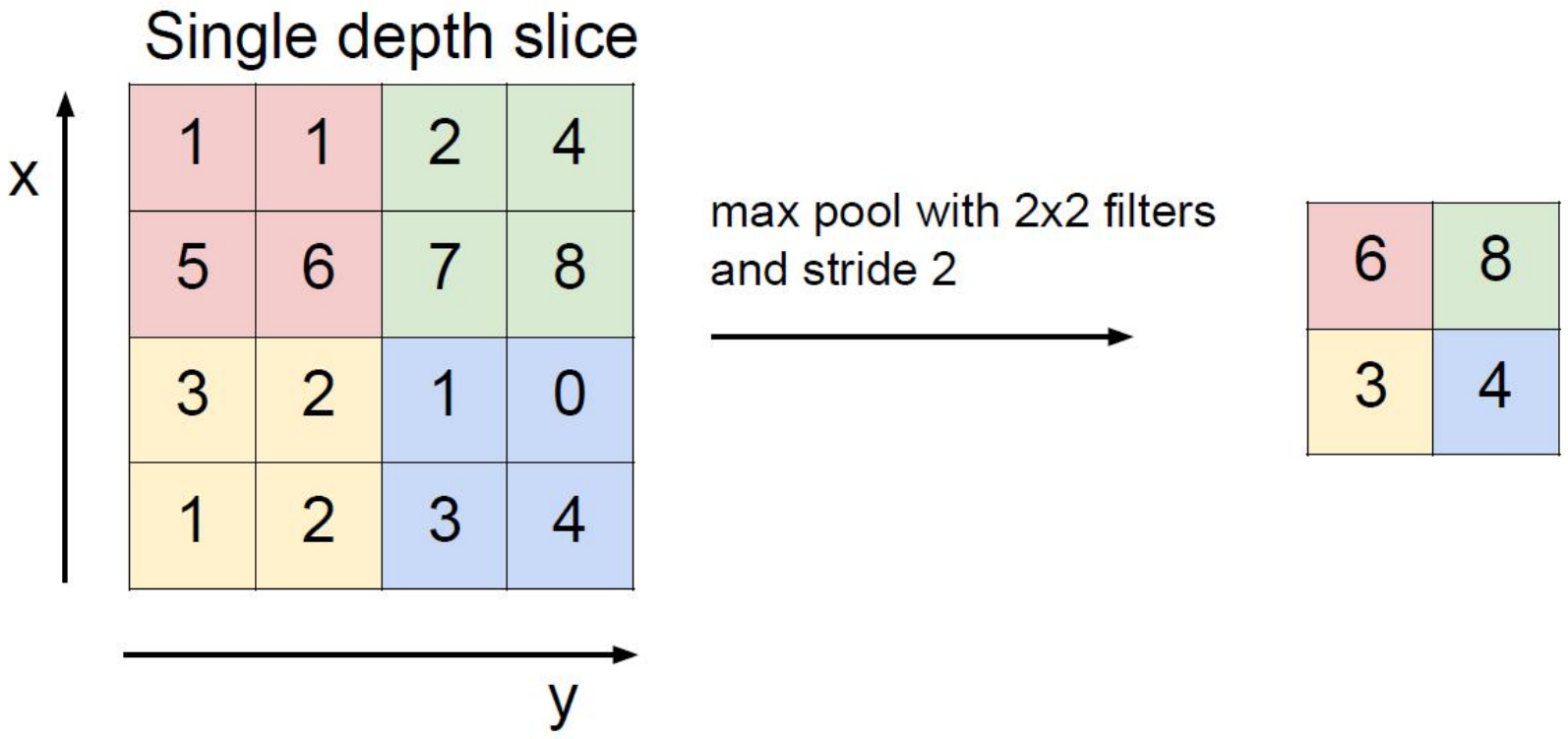


Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks