

Shanghai Jiao Tong University

Computer Vision

Instructor: Xu Zhao
Class No.: C032703 F032528

Spring 2020



Xu Zhao @ Shanghai Jiao Tong university

Lecture 11: Deep Neural Networks - Part 1

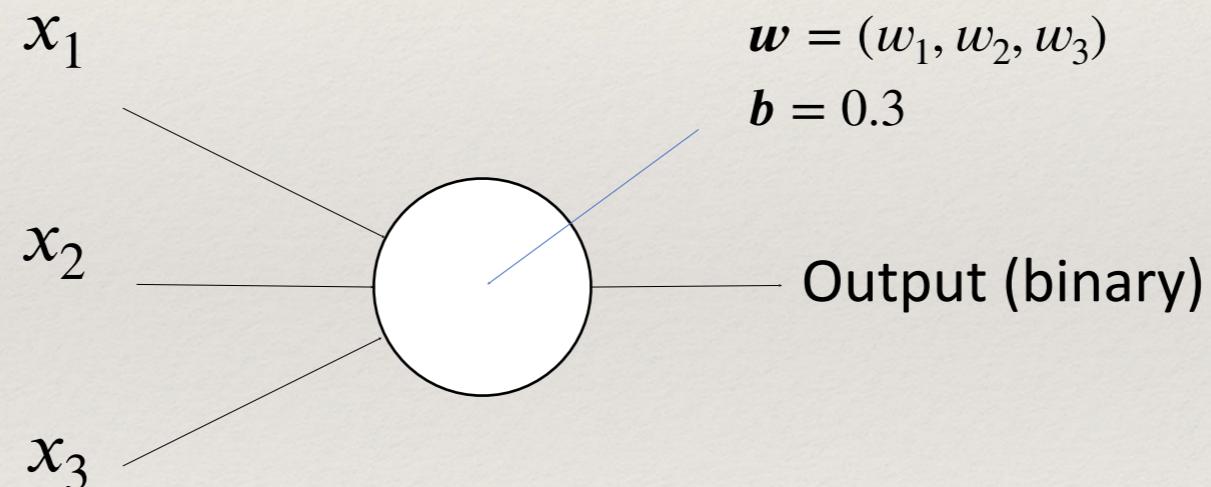
Contents

- ❖ **Neural Networks**
- ❖ **ConvNets**
- ❖ **Network Architectures and Applications**

Neural Networks

Basic building block is a *perceptron* (Rosenblatt c.1960)

Linear classifier – vector of weights w and a ‘bias’ b



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j$$

Binary classifying an image

Each pixel of the image would be an input.

So, for a 28×28 image, we vectorize:

$$\mathbf{x} = 1 \times 784$$

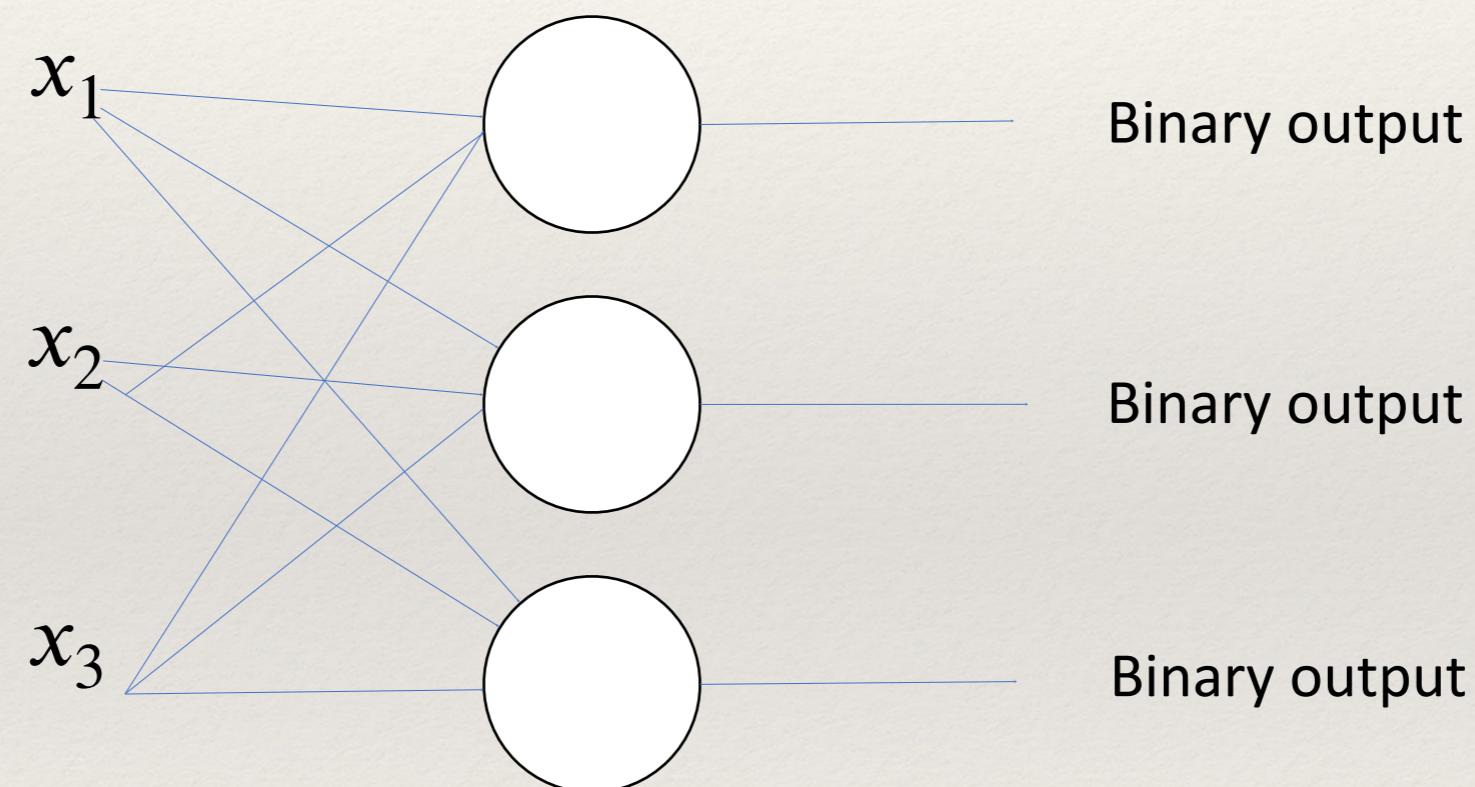
\mathbf{w} is a vector of weights for each pixel, 784×1

b is a scalar bias per perceptron (neuron)

$$\text{Result} = \mathbf{xw} + b \rightarrow (1 \times 784) \times (784 \times 1) + b = (1 \times 1) + b$$

Neural Networks - multiclass

Add more perceptrons



Multi-class classifying an image

Each pixel of the image would be an input.

So, for a 28×28 image, we vectorize.

$$\mathbf{x} = 1 \times 784$$

\mathbf{W} is a matrix of weights for each pixel/each perceptron

$$\mathbf{W} = 10 \times 784 \text{ (10-class classification)}$$

\mathbf{b} is a bias *per perceptron* (vector of biases); (1×10)

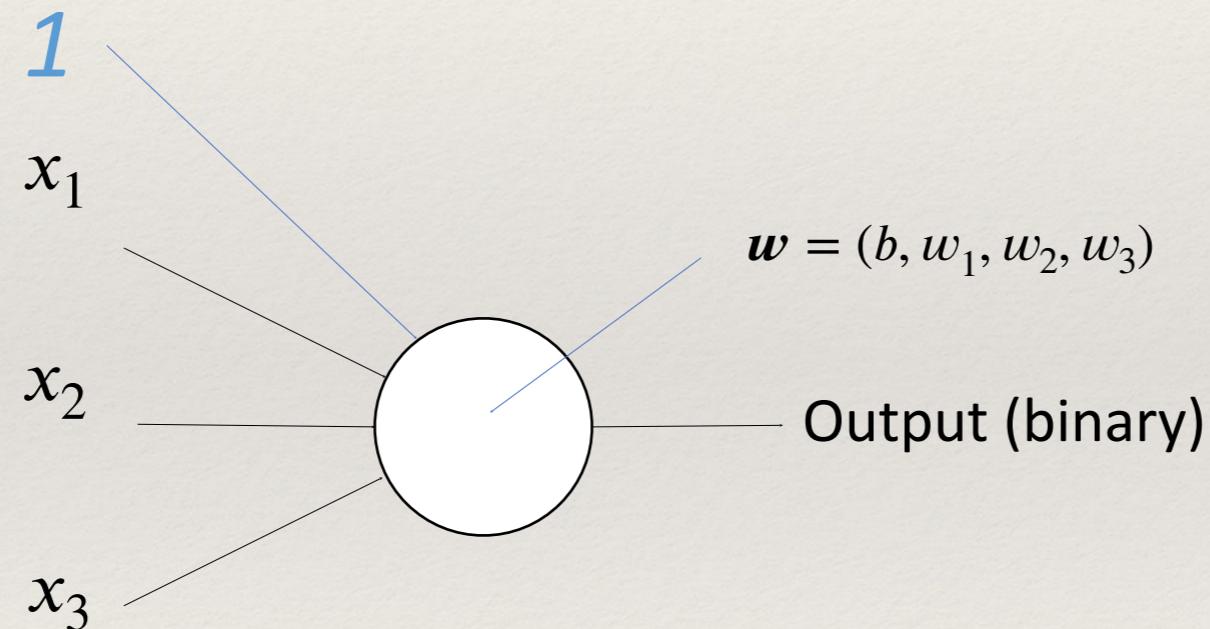
$$\text{Result} = \mathbf{xW} + \mathbf{b} \rightarrow (1 \times 784) \times (784 \times 10) + \mathbf{b}$$

$$\rightarrow (1 \times 10) + (1 \times 10) = \text{output vector}$$

Bias convenience

Let's turn this operation into a multiplication only:

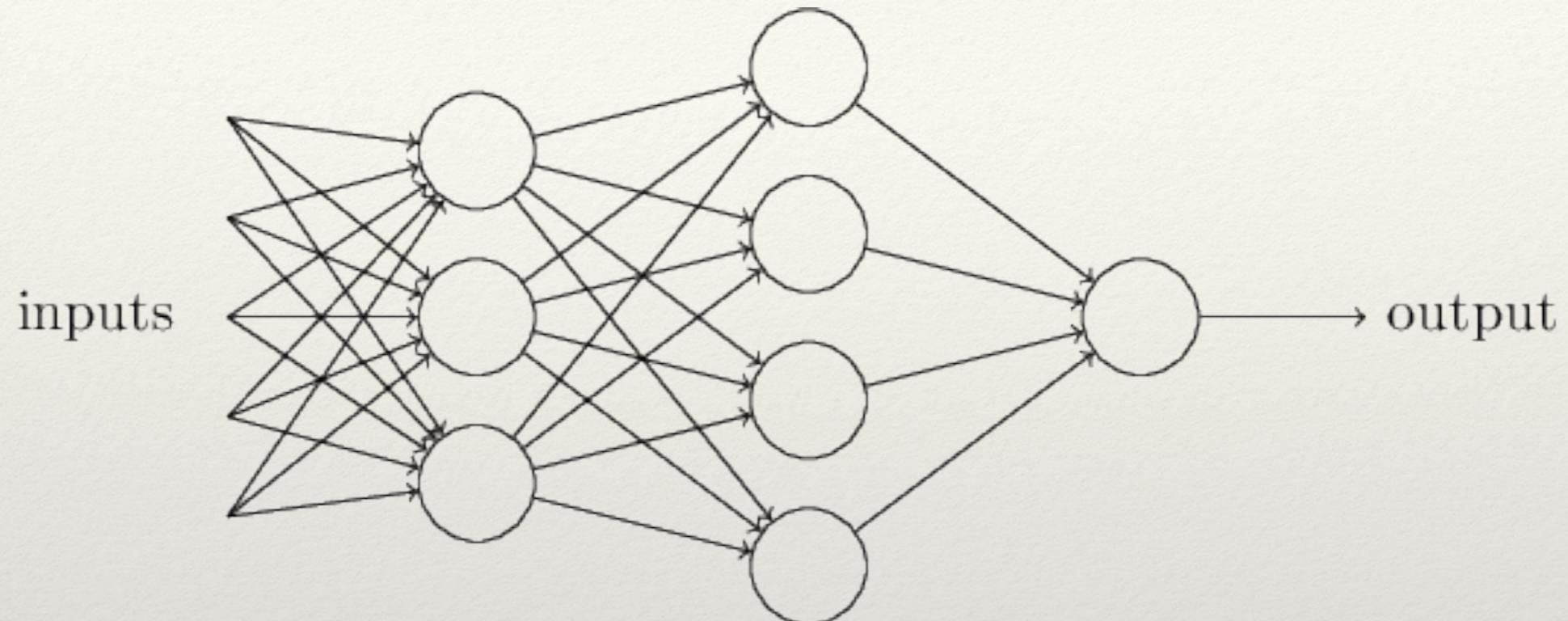
- ❖ Create a 'fake' feature with value 1 to represent the bias
- ❖ Add an extra weight that can vary



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

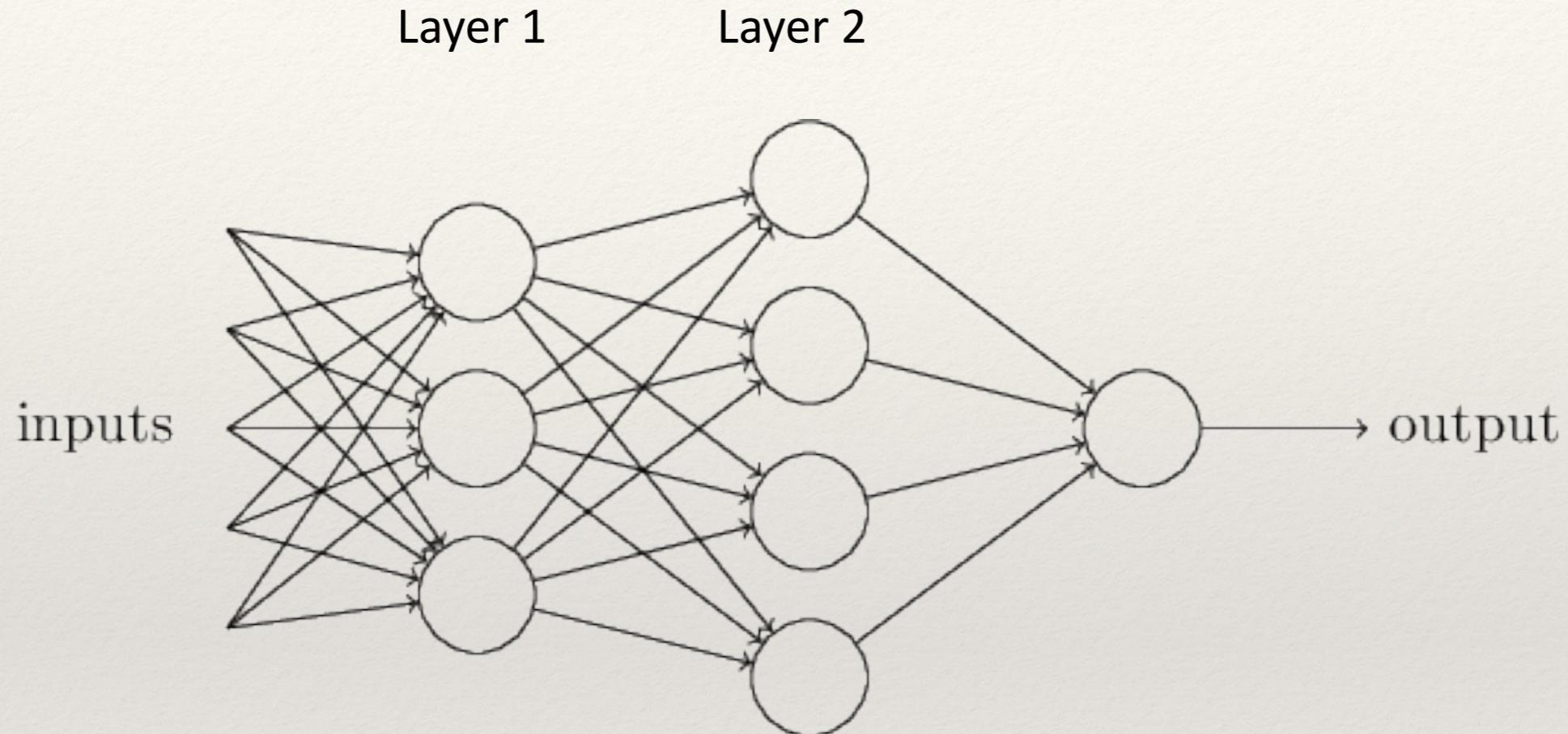
$$w \cdot x \equiv \sum_j w_j x_j$$

Composition



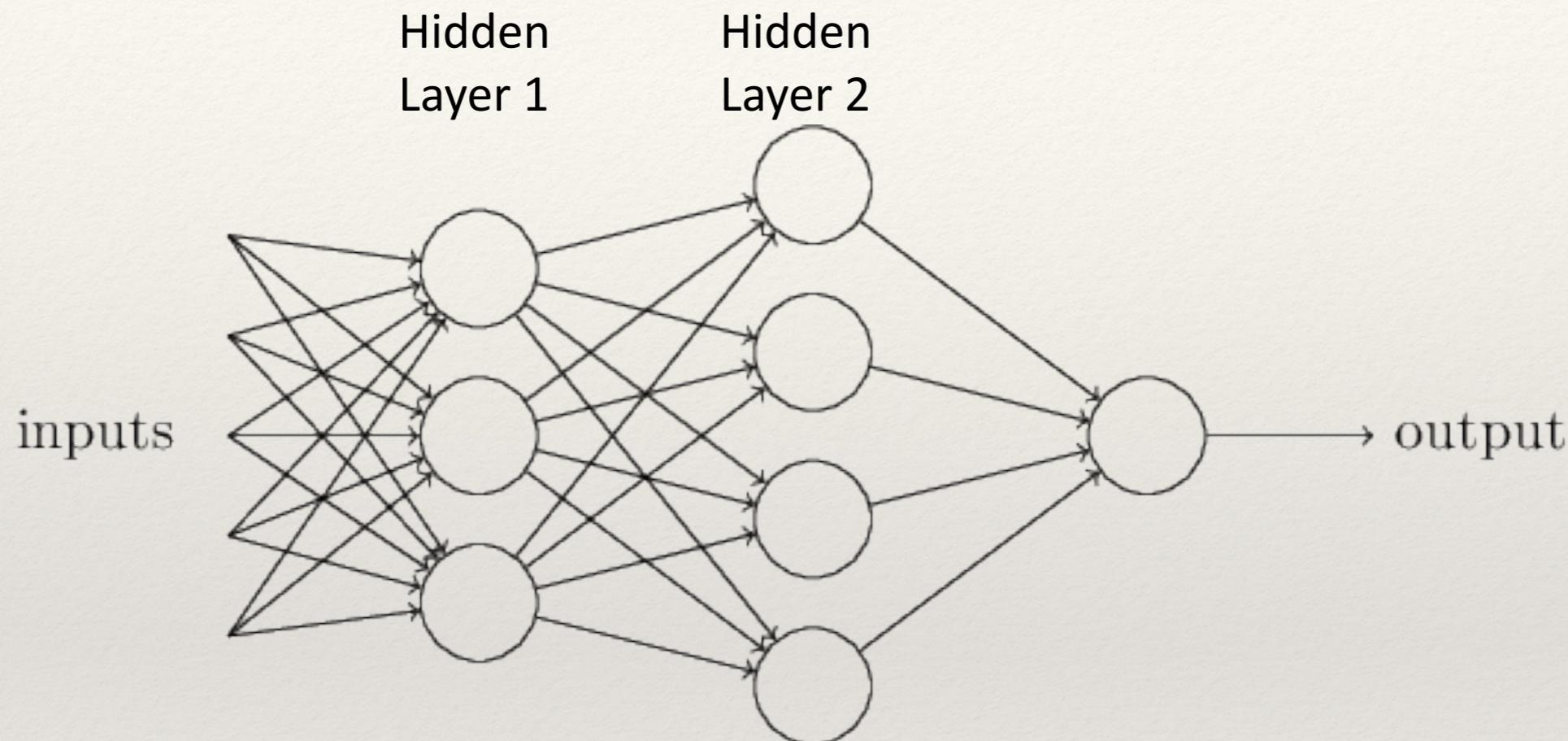
- ❖ Attempt to represent complex functions as compositions of smaller functions.
- ❖ Outputs from one perception are fed into inputs of another perceptron.

Composition



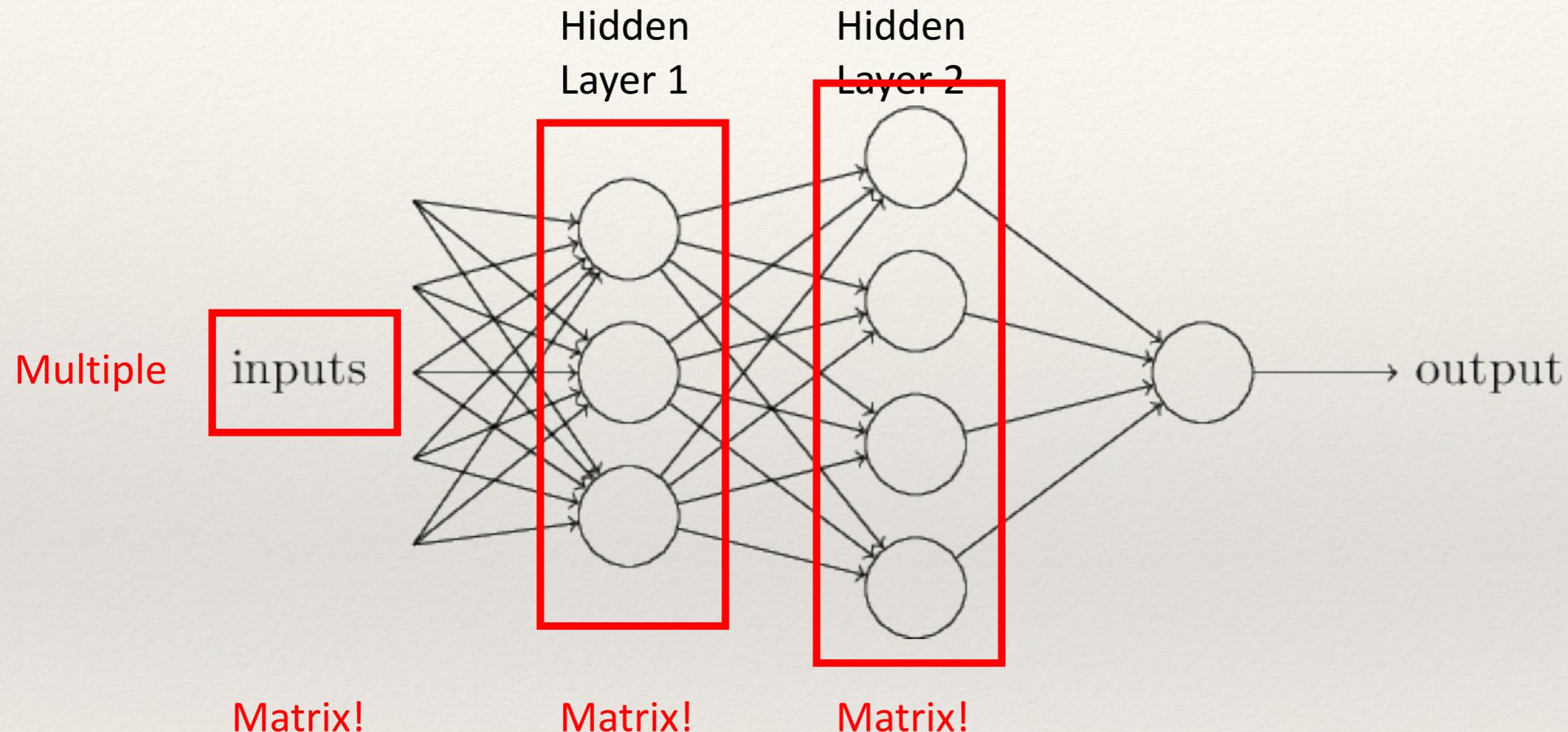
- ❖ Sets of layers and the connections (weights) between them define the *network architecture*.

Composition



- ❖ Layers that are in between the input and the output are called *hidden layers*, because we are going to *learn* their weights via an optimization process.

Composition



It's all just matrix multiplication!

GPUs -> special hardware for fast/large matrix multiplication.

Problem 1 with all linear functions

We have formed chains of linear functions.

We know that linear functions can be reduced

- ❖ $g = f(h(x))$

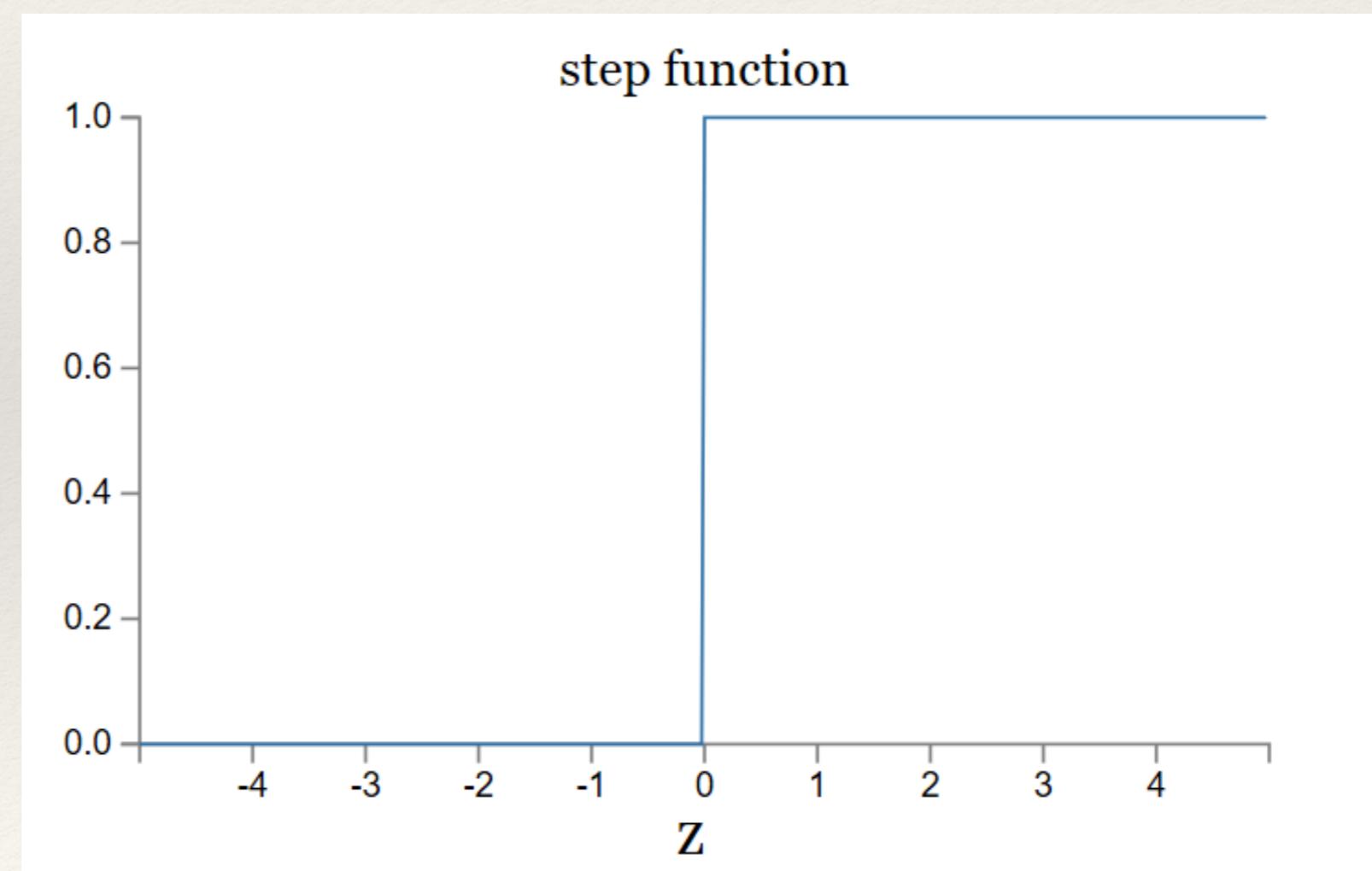
Our composition of functions is really just a single function

Problem 2 with all linear functions

Linear classifiers: small change in input can cause large change in binary output

= problem for composition of functions

*Activation
function*

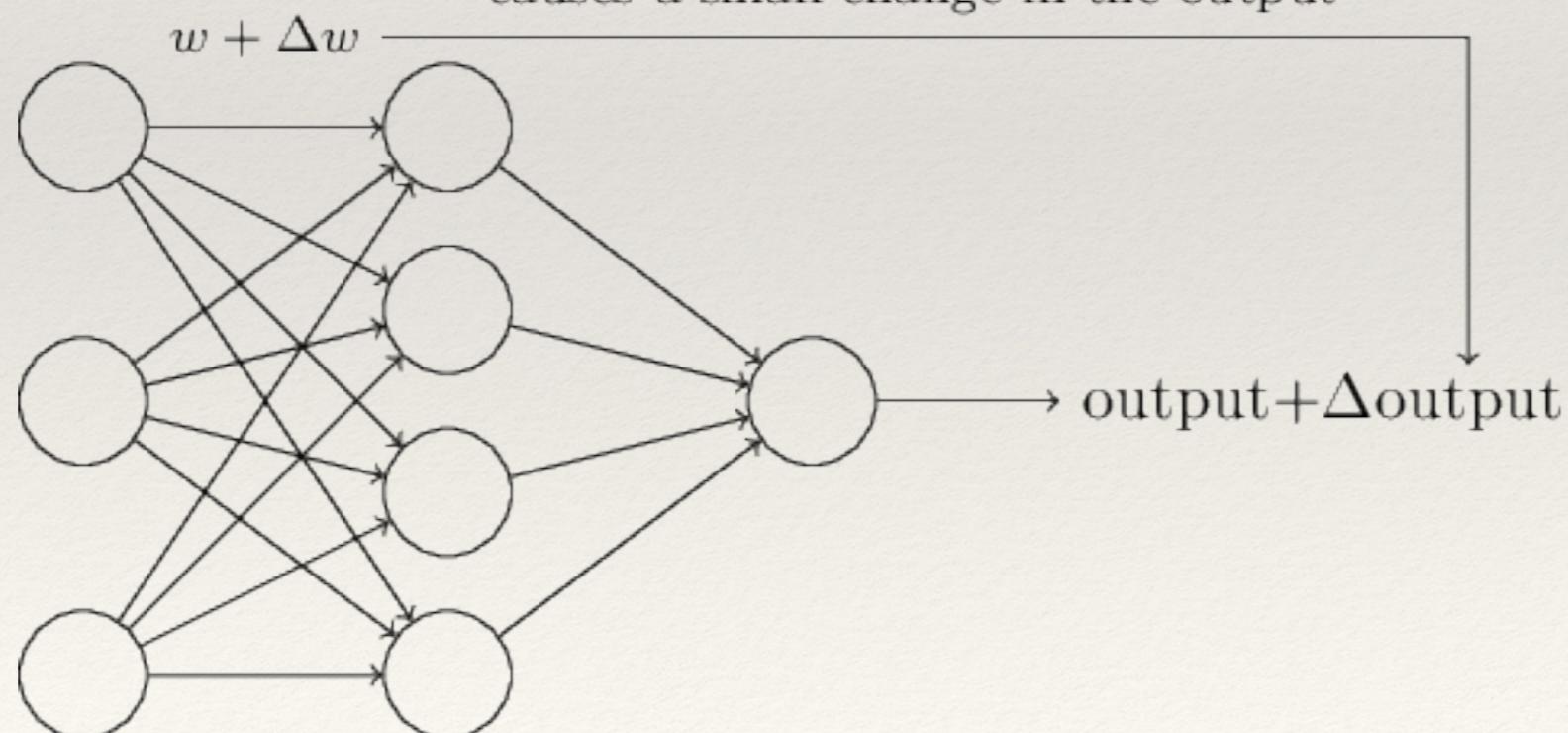


Problem 2 with all linear functions

Linear classifiers: small change in input can cause large change in binary output.

We want:

small change in any weight (or bias)
causes a small change in the output

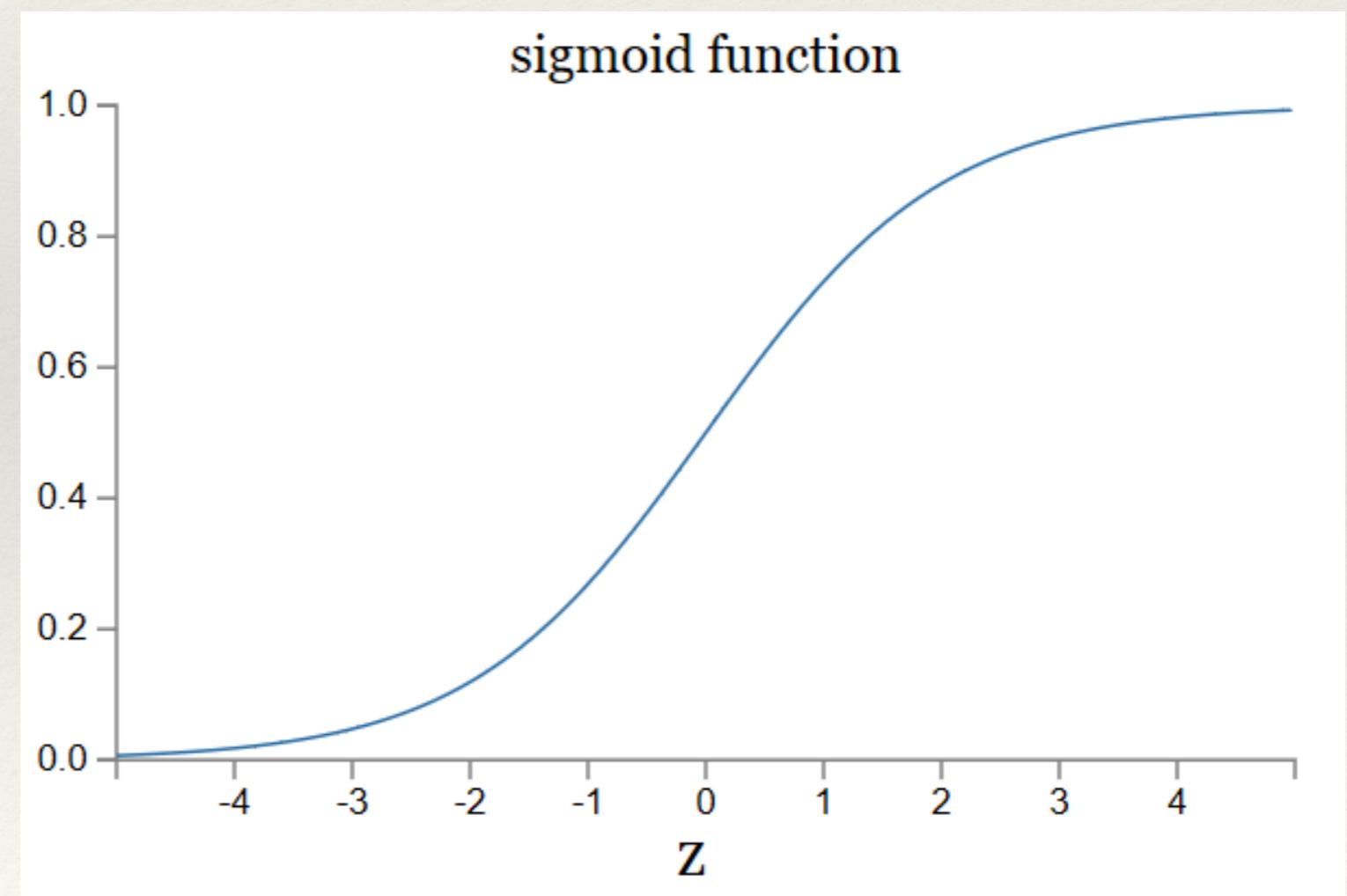


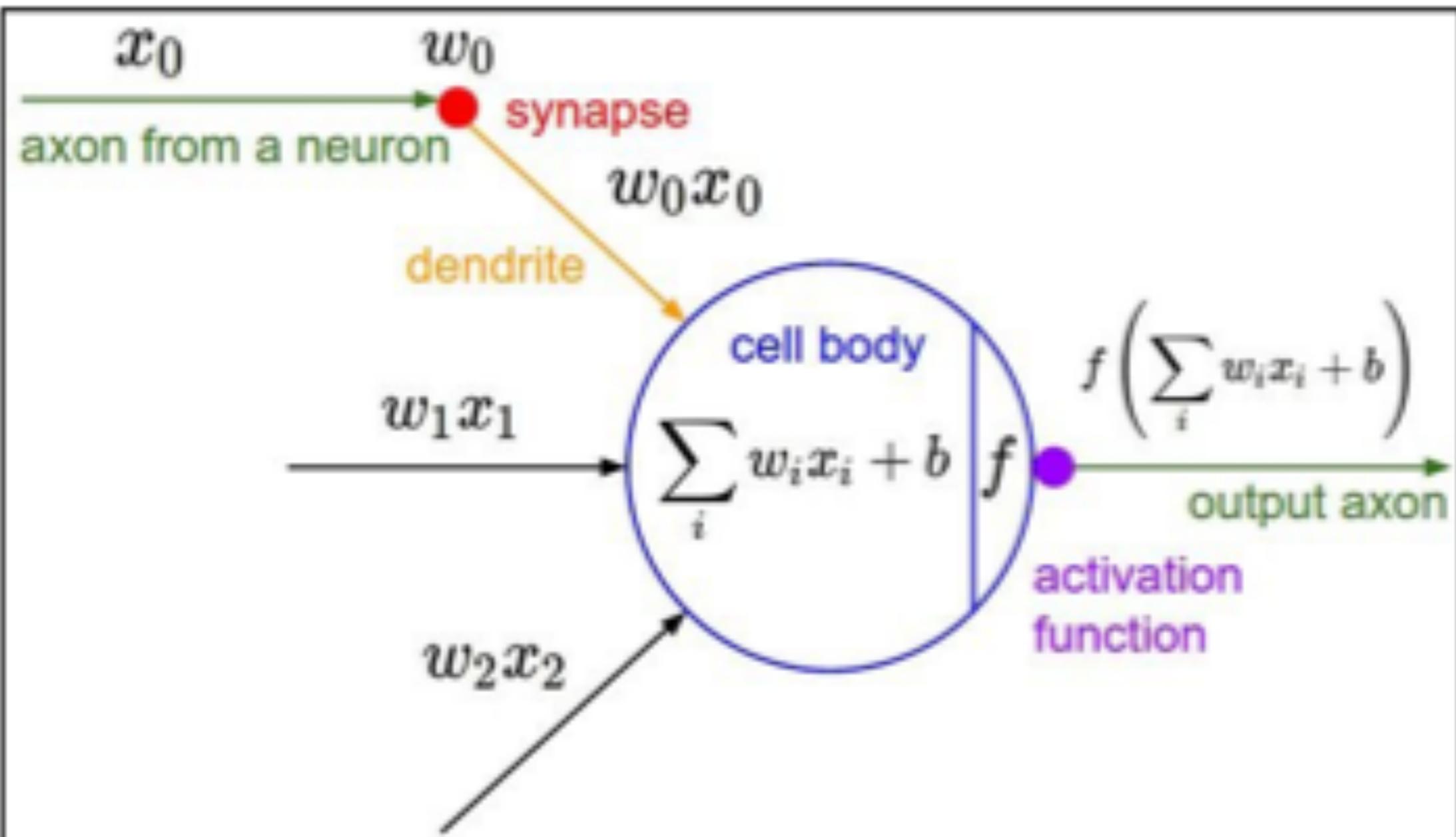
Introduce non-linearities

We're going to introduce non-linear functions to transform the features.

$$\sigma(w \cdot x + b)$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$





Universality

A single-layer of perceptrons can learn any univariate function:

- ❖ So long as it is differentiable
- ❖ To some approximation;
- More perceptrons = a better approximation

Visual proof (Michael Nielson):

<http://neuralnetworksanddeeplearning.com/chap4.html>

Perceptron model

- ❖ Use is grounded in theory
 - ❖ Universal approximation theorem (Goodfellow 6.4.1)
- ❖ Can represent a NAND circuit, from which any binary function can be built by compositions of NANDs
- ❖ With enough parameters, it can approximate any function.

*If a single-layer network can learn any function...
...given enough parameters...*

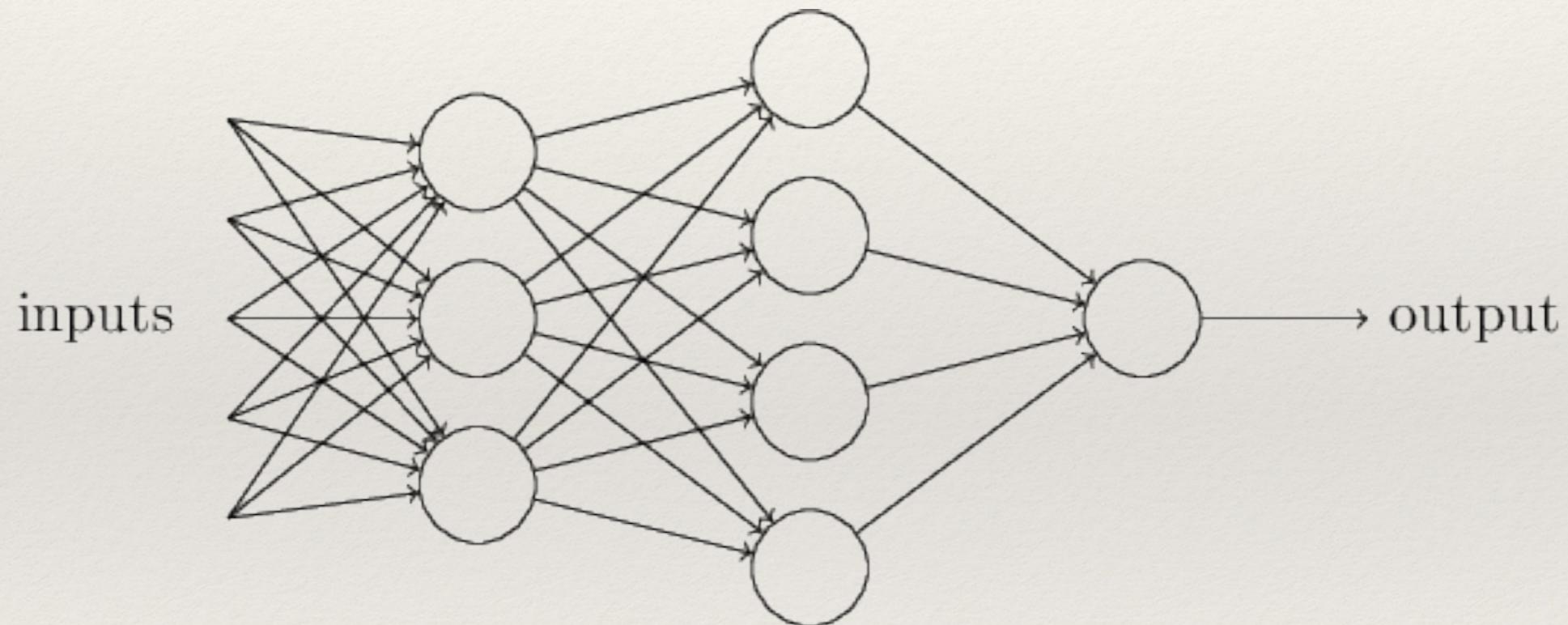
...then why do we go deeper?

Intuitively, composition is efficient because it allows *reuse*.

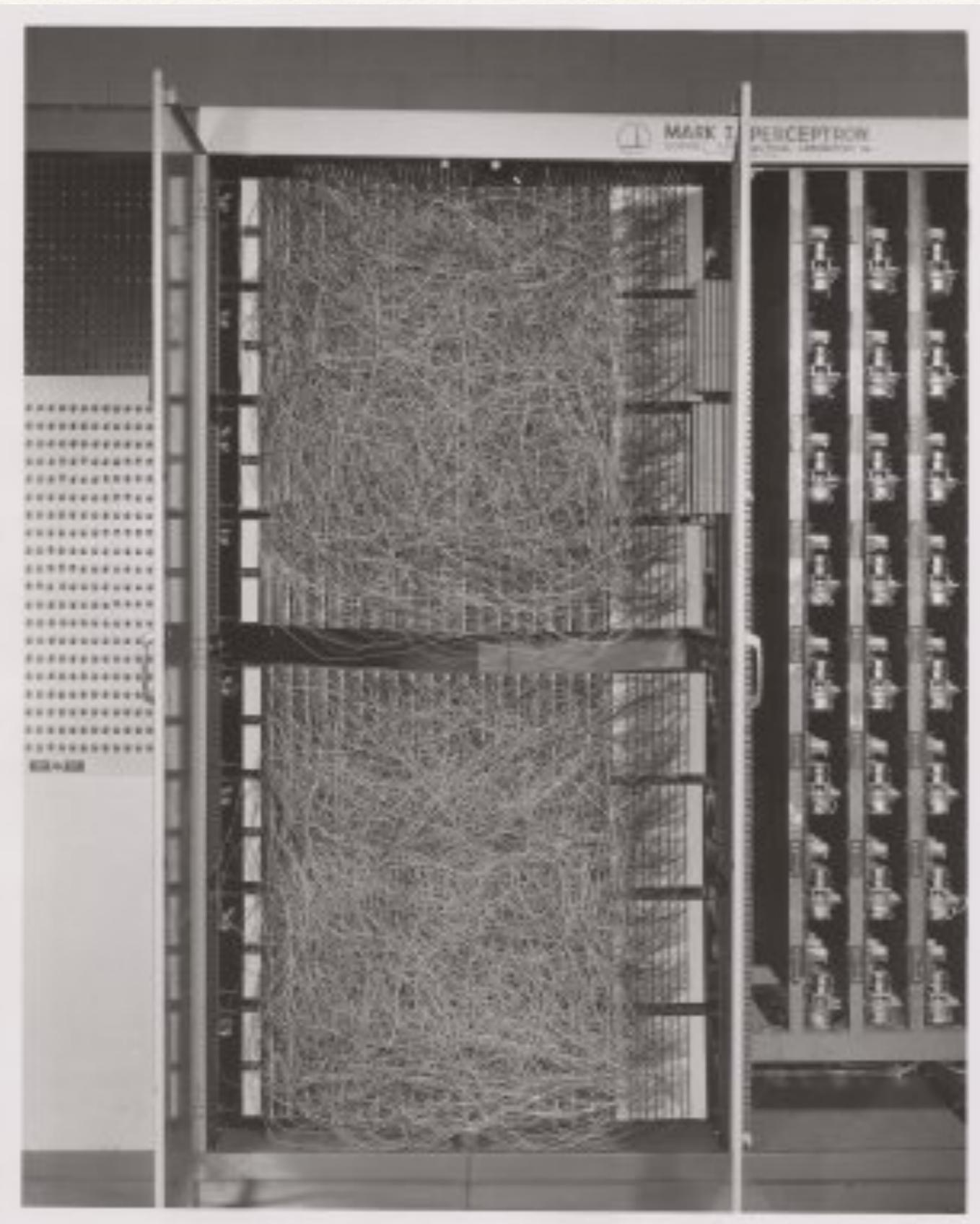
Empirically, deep networks do a better job than shallow networks at learning such hierarchies of knowledge.

Multi-layer perceptron (MLP)

- ❖ ...is a '*fully connected*' neural network with non-linear activation functions.



- ❖ '*Feed-forward*' neural network

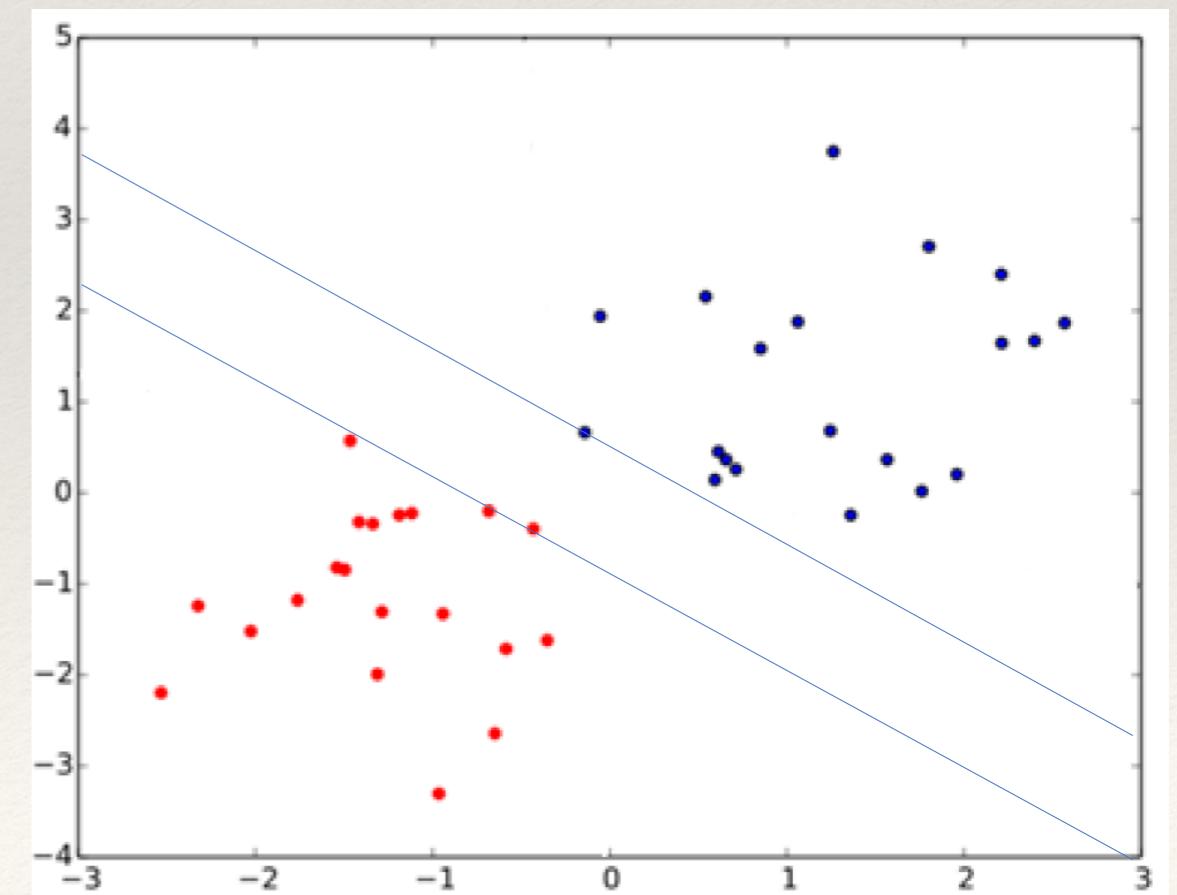


Mark 1 Perceptron
c.1960

20x20 pixel
camera feed

What is the relationship between SVMs and perceptrons?

SVMs attempt to learn the support vectors which maximize the margin between classes.



What is the relationship between SVMs and perceptrons?

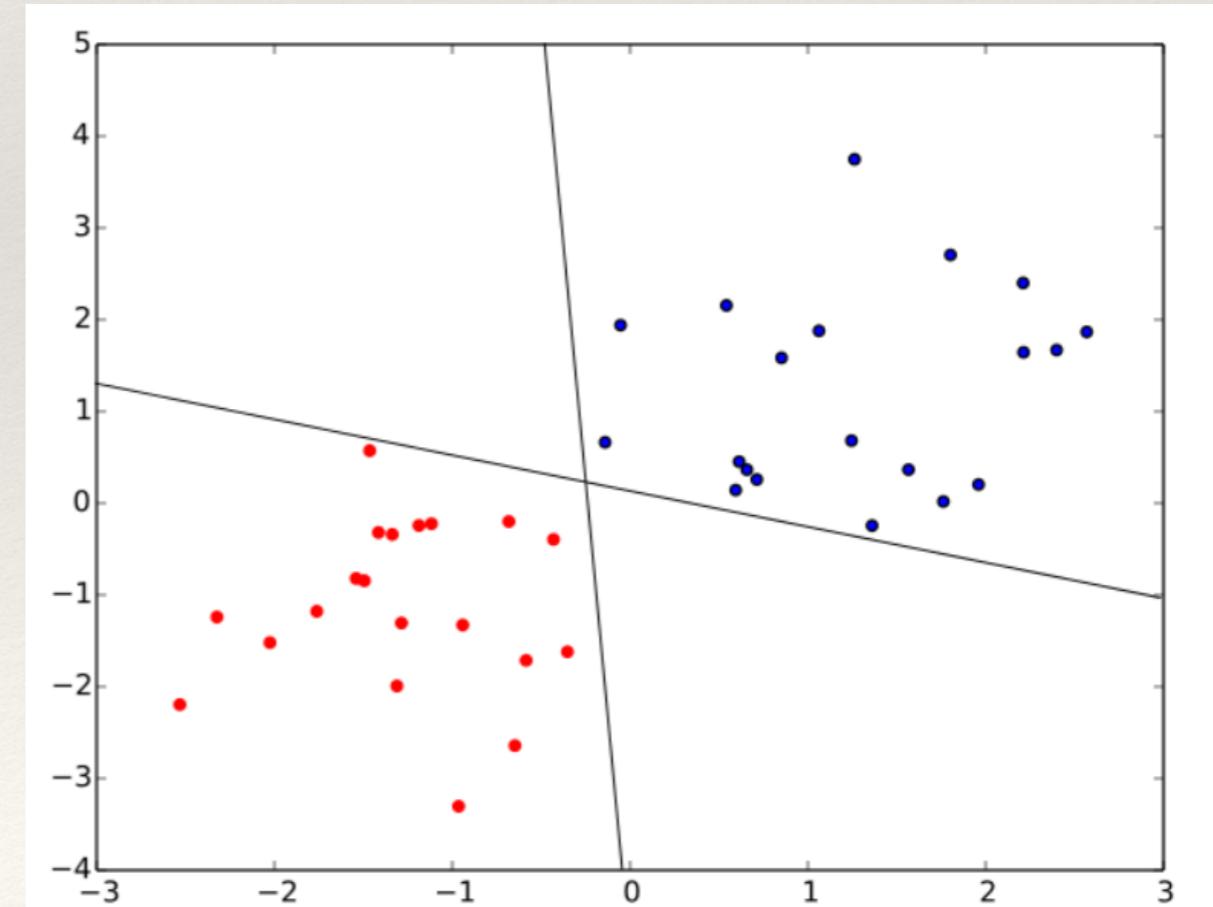
SVMs attempt to learn the support vectors which maximize the margin between classes.

A perceptron does not.

Both of these perceptron classifiers are equivalent.

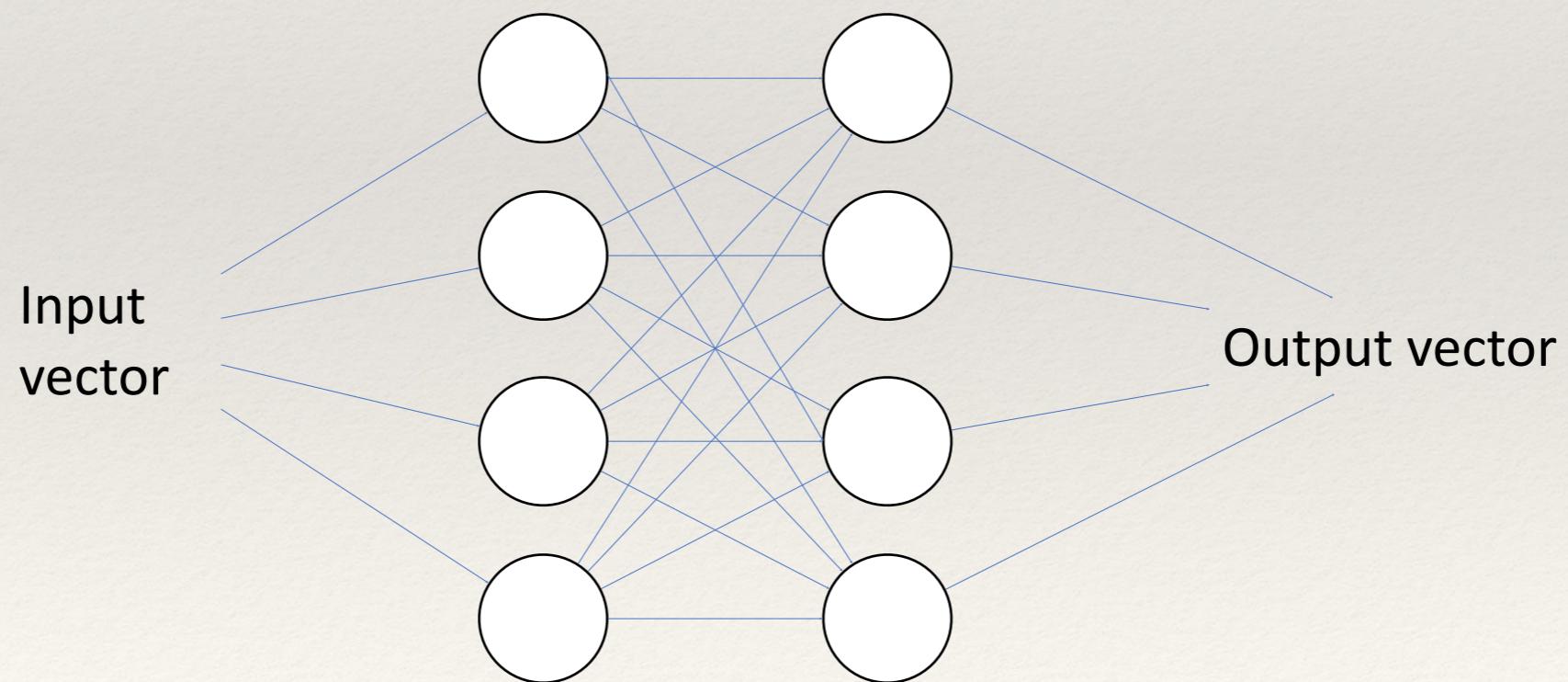
'Perceptron of optimal stability' is used in SVM:

Perceptron
+ optimal stability
+ kernel trick
= *foundations of SVM*



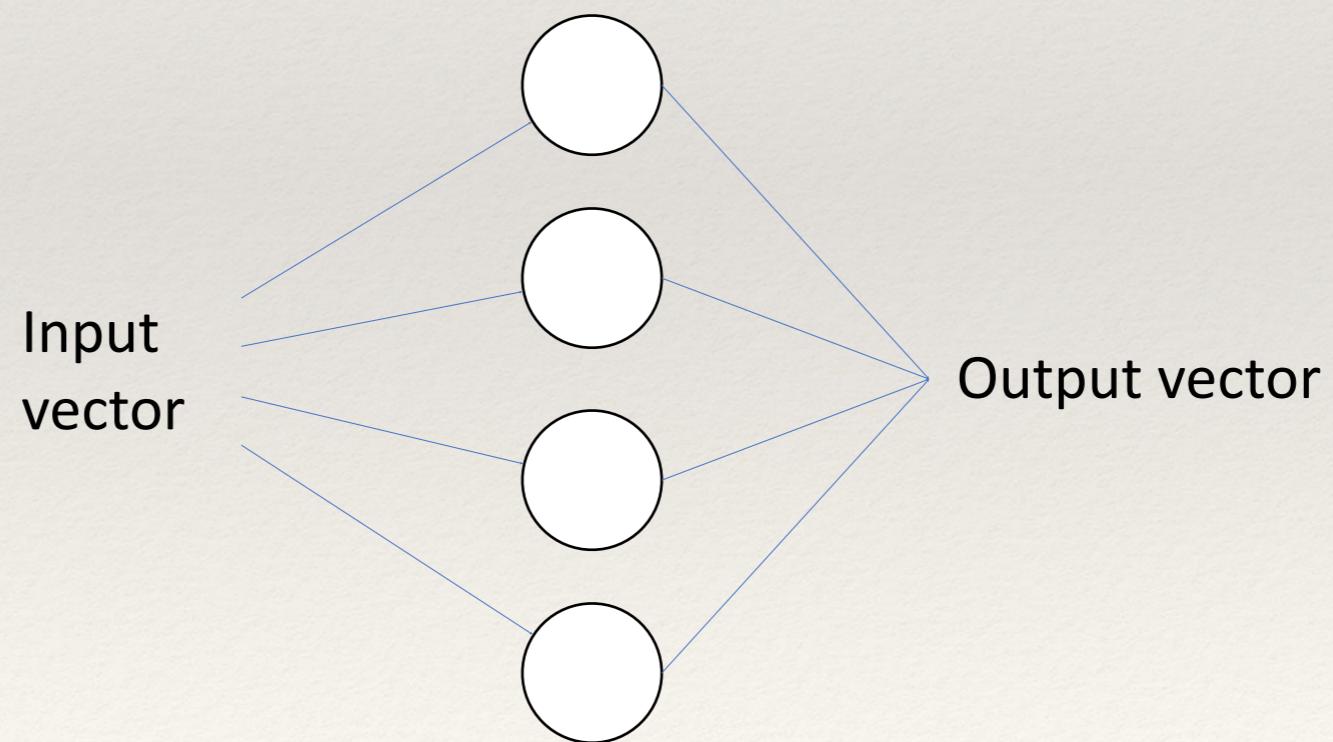
Does anyone pass along the weight without an activation function?

No – this is linear chaining.



Does anyone pass along the weight without an activation function?

No – this is linear chaining.



Are there other activation functions?

Yes, many.

As long as:

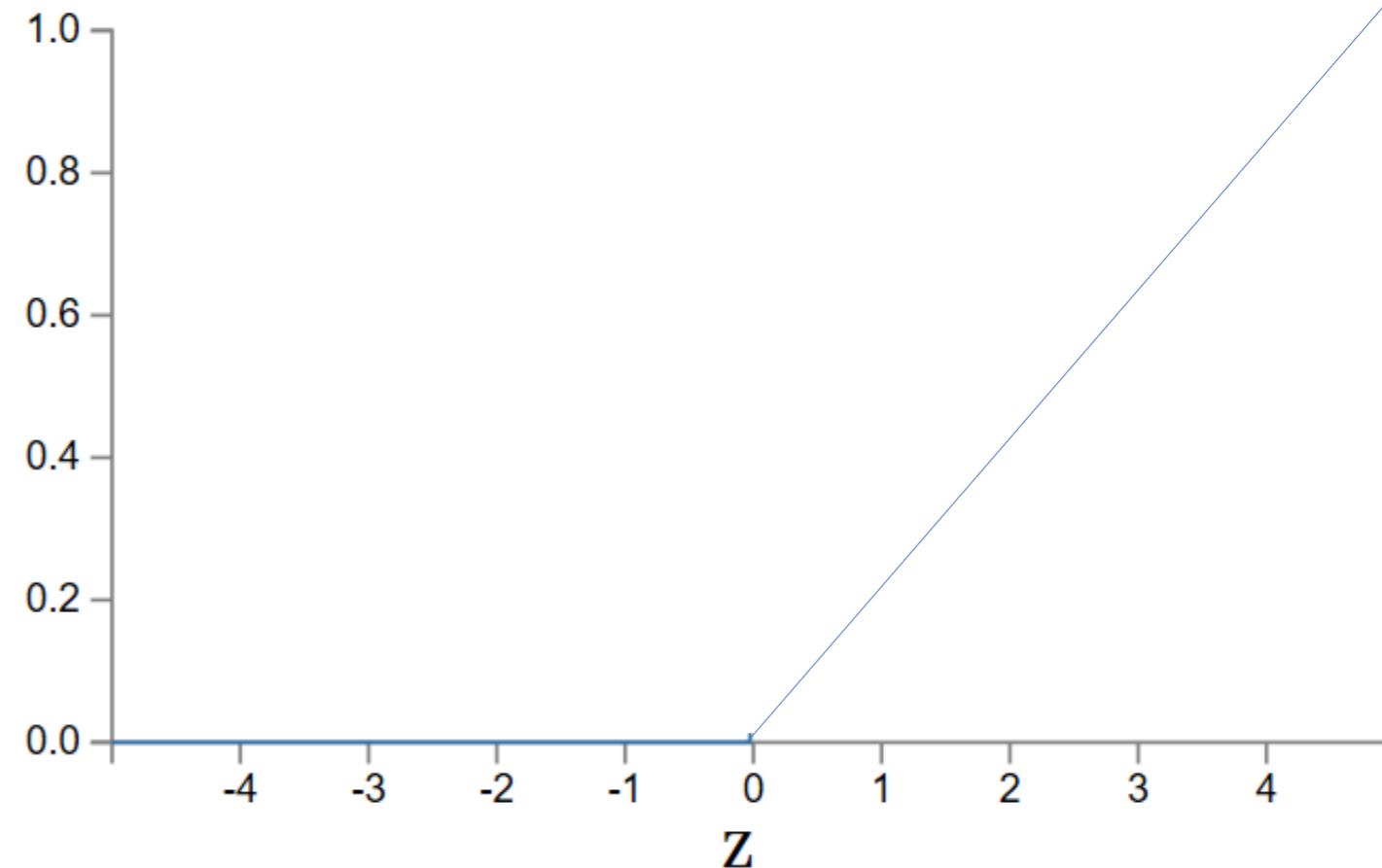
- Activation function $s(z)$ is well-defined as $z \rightarrow -\infty$ and $z \rightarrow \infty$
- These limits are different

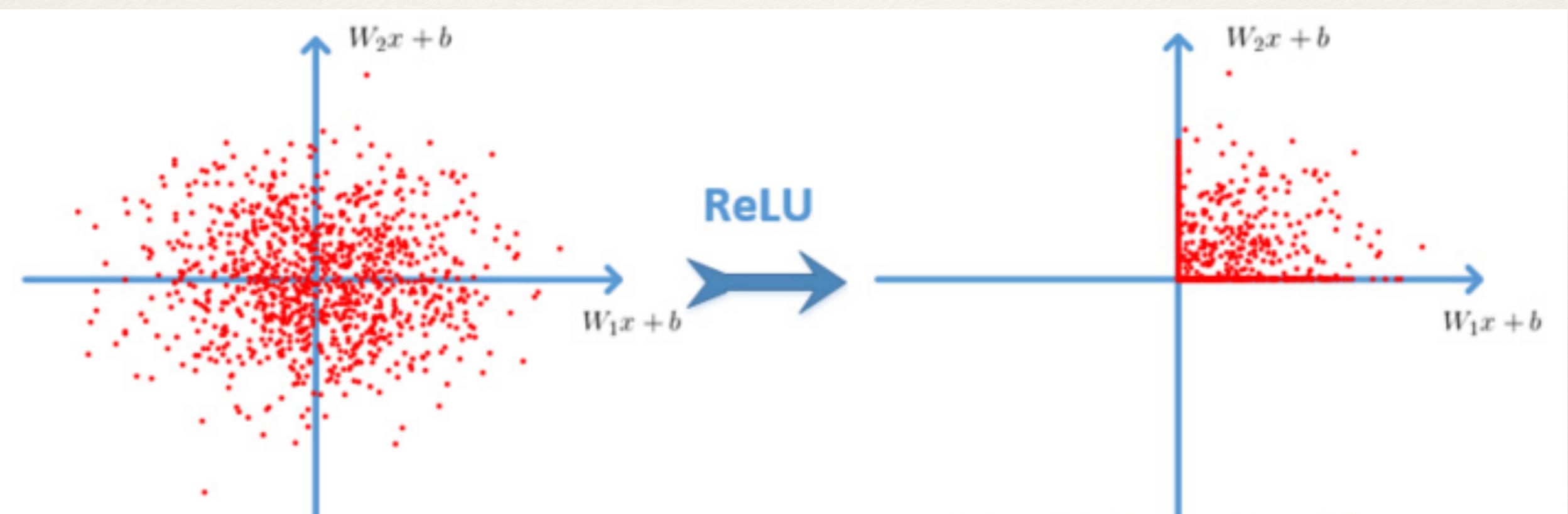
Then we can make a step! [Think visual proof]
It can be shown that it is universal for function approximation.

Activation functions: Rectified Linear Unit

$$f(x) = \max(0, x)$$

❖ ReLU

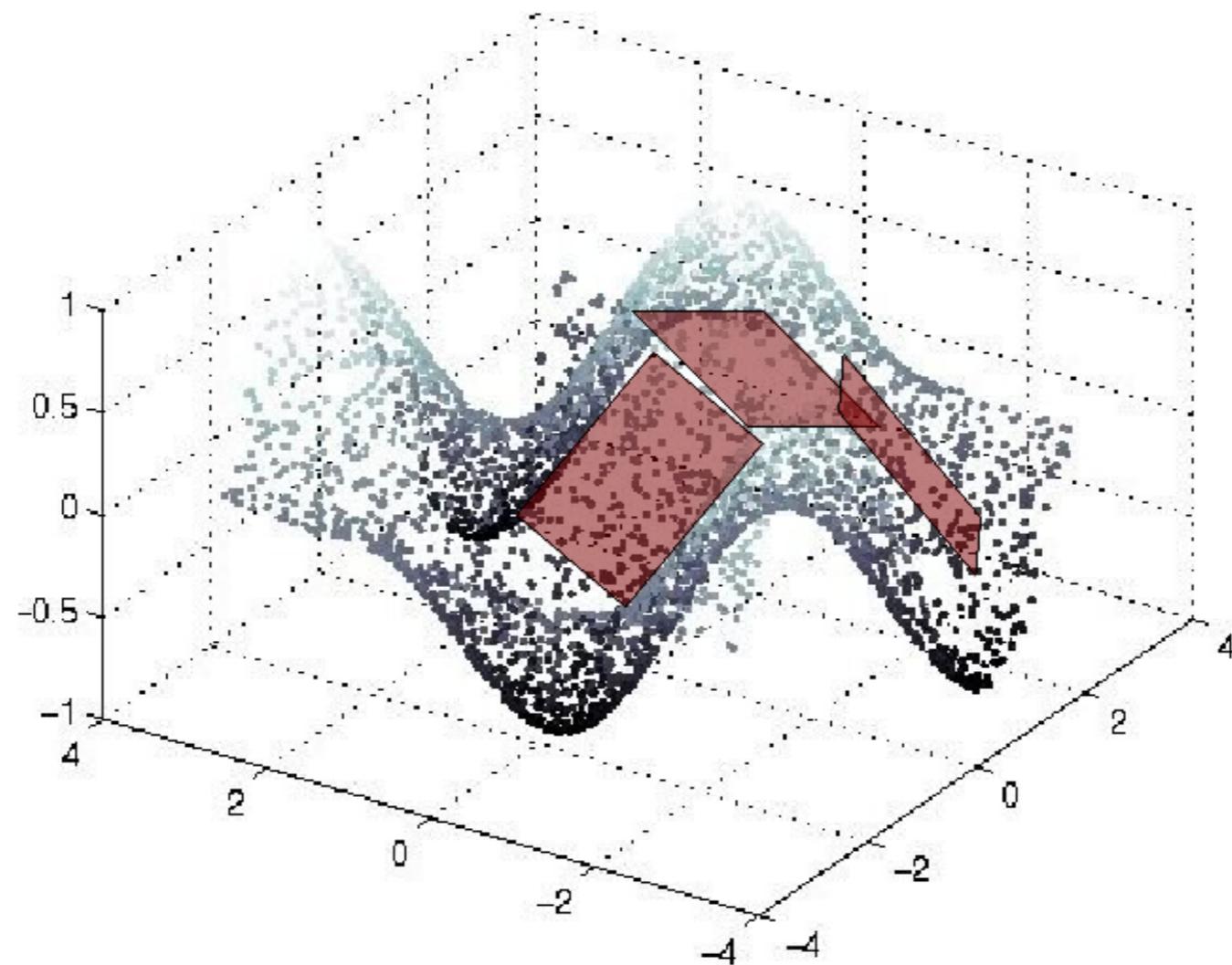




Rectified Linear Unit

Question: What do ReLU layers accomplish?

Answer: Piece-wise linear tiling: mapping is locally linear.



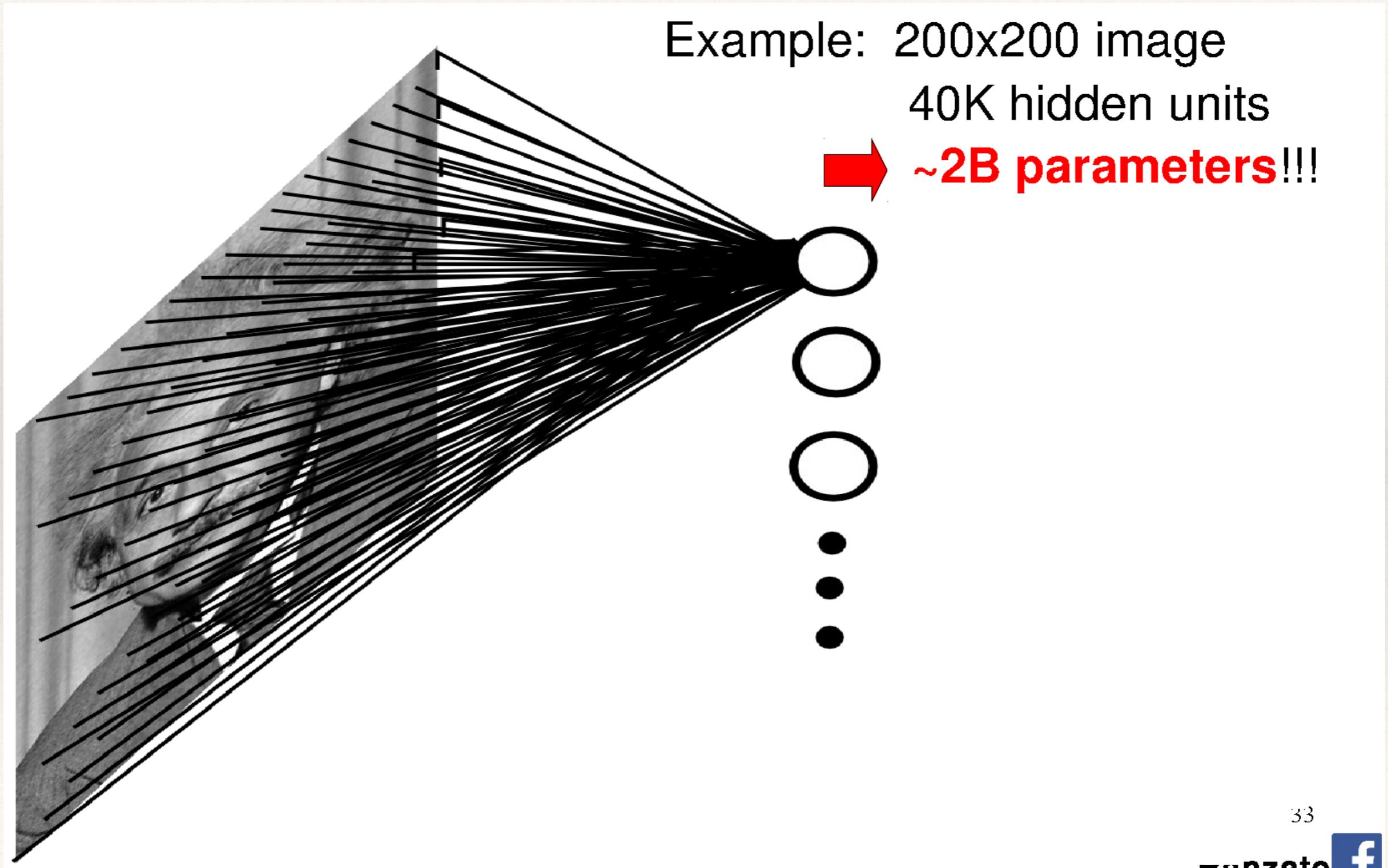
Goals

Build a classifier which is more powerful at representing complex functions *and* more suited to the learning problem.

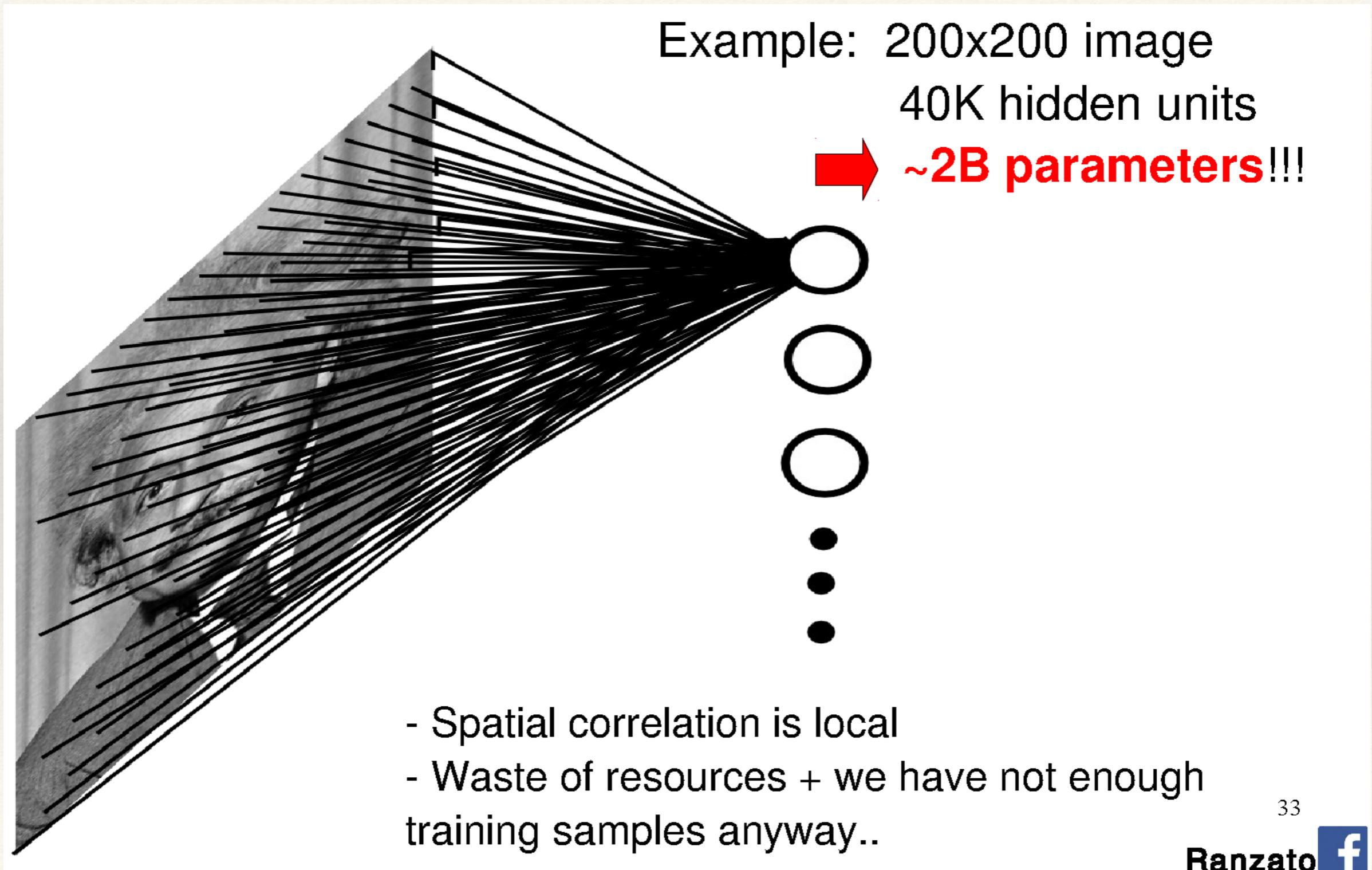
What does this mean?

1. Assume that the *underlying data generating function* relies on a composition of factors.
2. Learn a feature representation that is specific to the dataset.

Images as input to neural networks

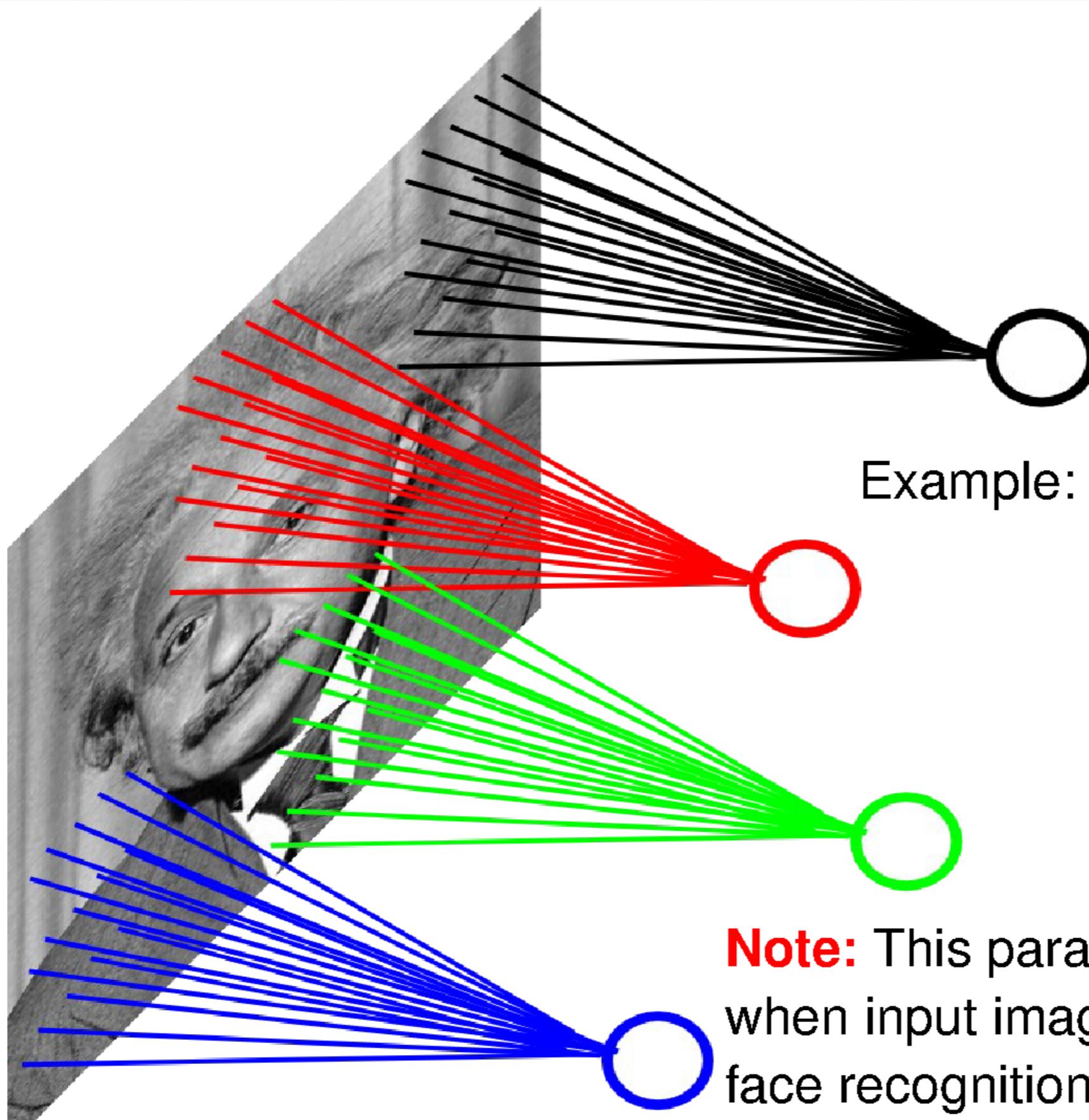


Images as input to neural networks



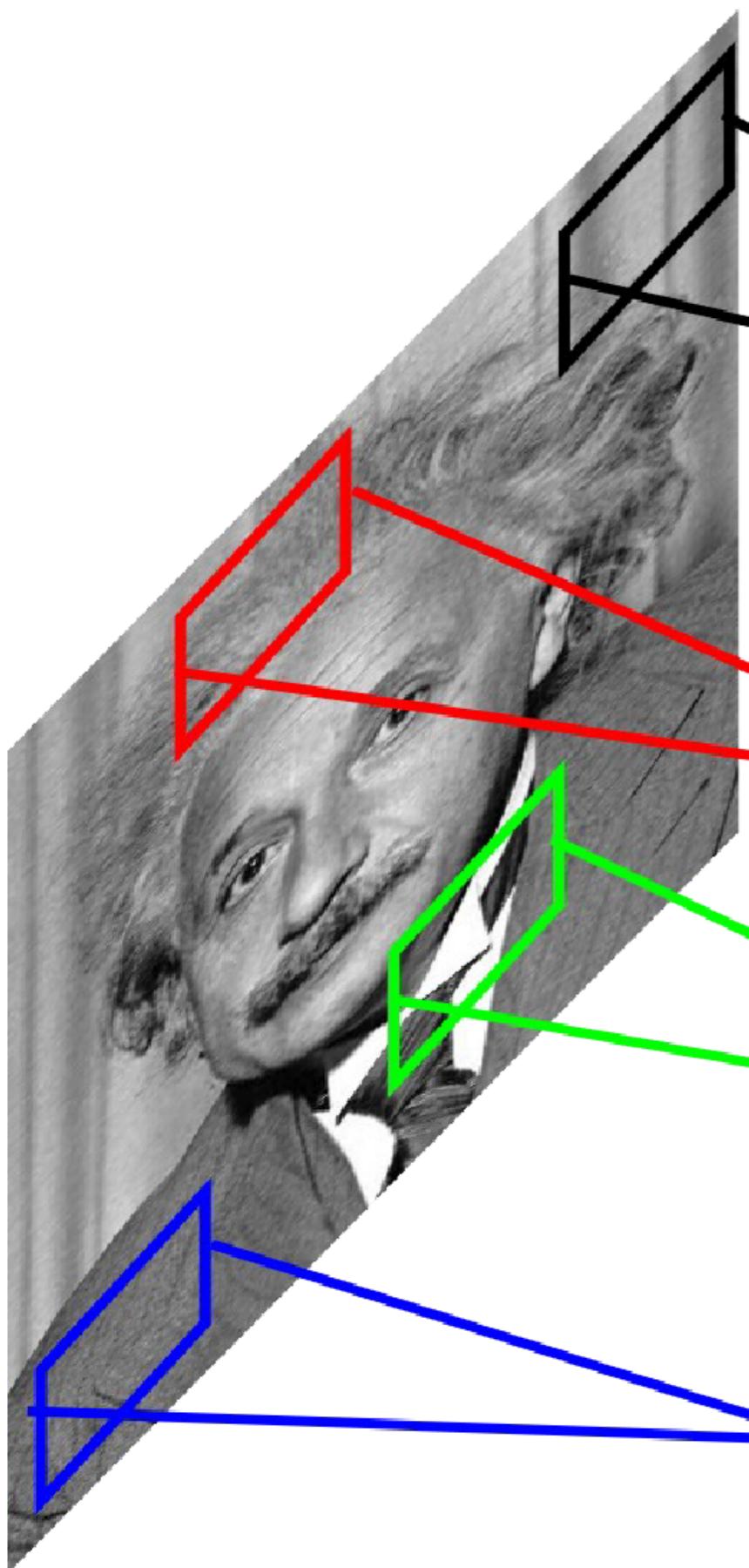
Motivation

- ❖ Sparse interactions – *receptive fields*
 - ❖ Assume that in an image, we care about 'local neighborhoods' only for a given neural network layer.
 - ❖ Composition of layers will expand local -> global.



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good
when input image is registered (e.g.,
face recognition).



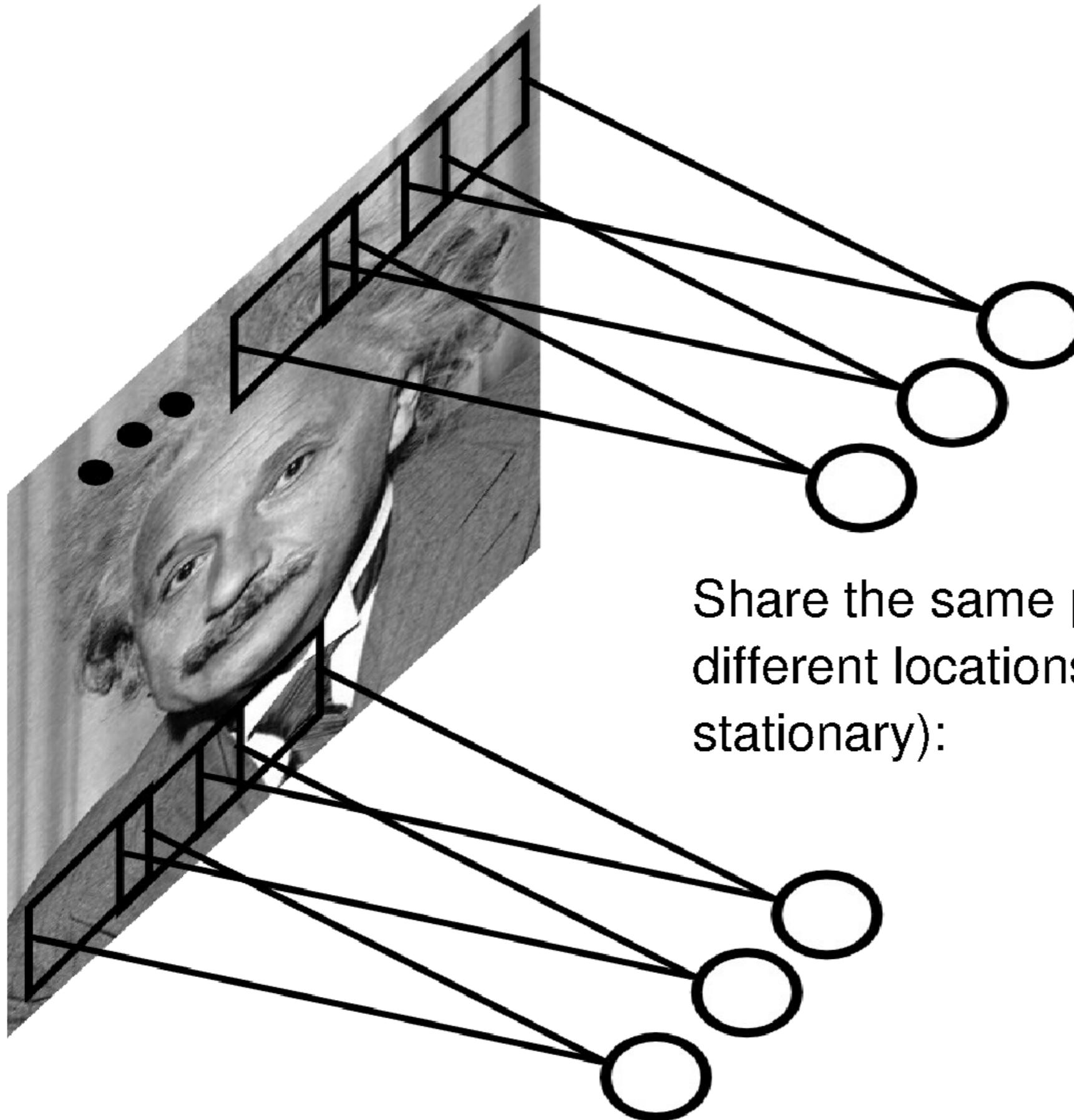
STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Motivation

- ❖ Sparse interactions – *receptive fields*
 - ❖ Assume that in an image, we care about 'local neighborhoods' only for a given neural network layer.
 - ❖ Composition of layers will expand local -> global.
- ❖ Parameter sharing
 - ❖ 'Tied weights' – use same weights for more than one perceptron in the neural network.
 - ❖ Leads to *equivariant representation*
 - ❖ If input changes (e.g., translates), then output changes similarly



Share the same parameters across
different locations (assuming input is
stationary):

Filtering remainder: Correlation (rotated convolution)

$$f[\cdot, \cdot] \quad \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$I[.,.]$

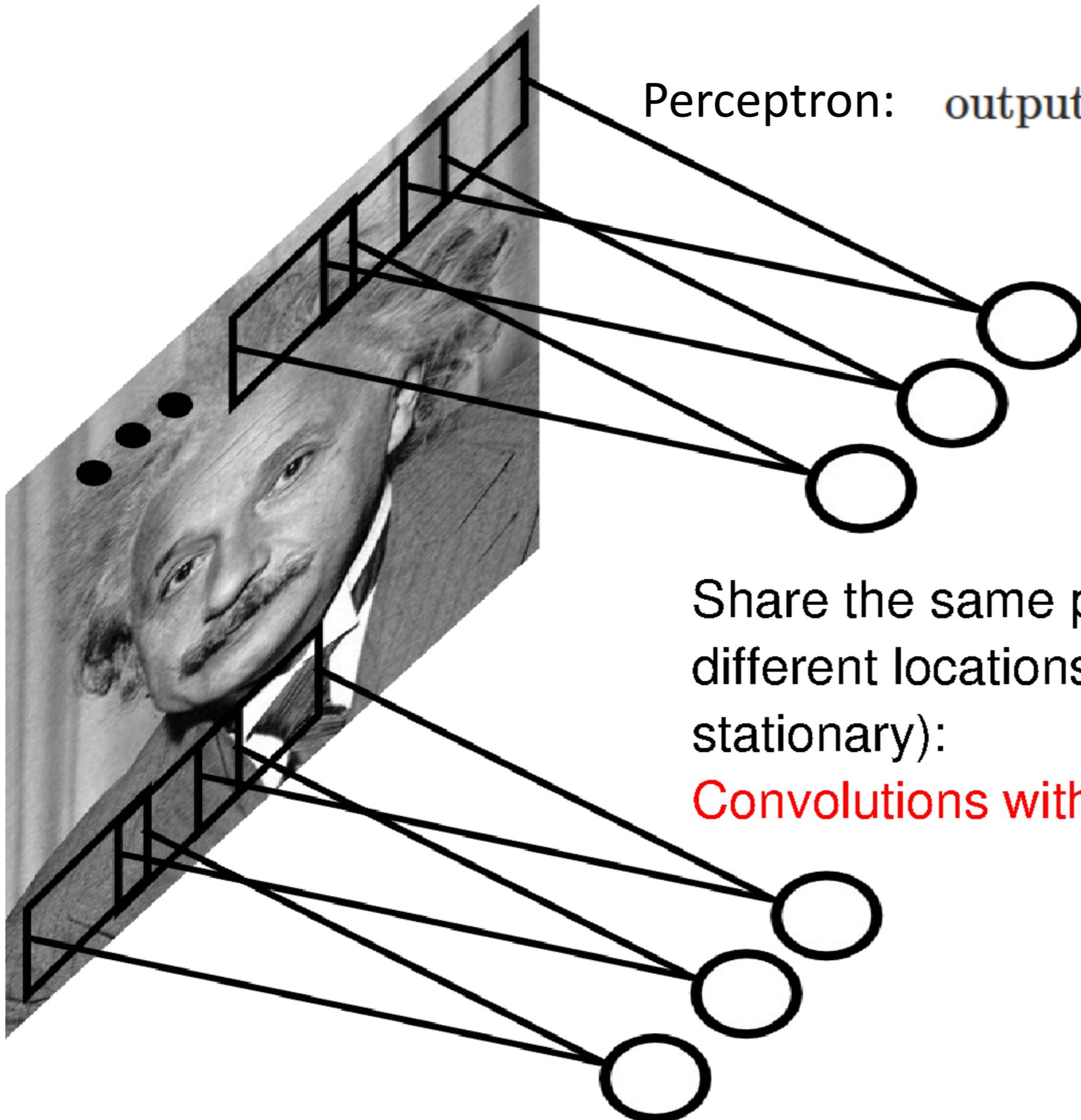
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m, n] = \sum_{k,l} f[k, l] I[m + k, n + l]$$

Convolutional Layer



Perceptron:

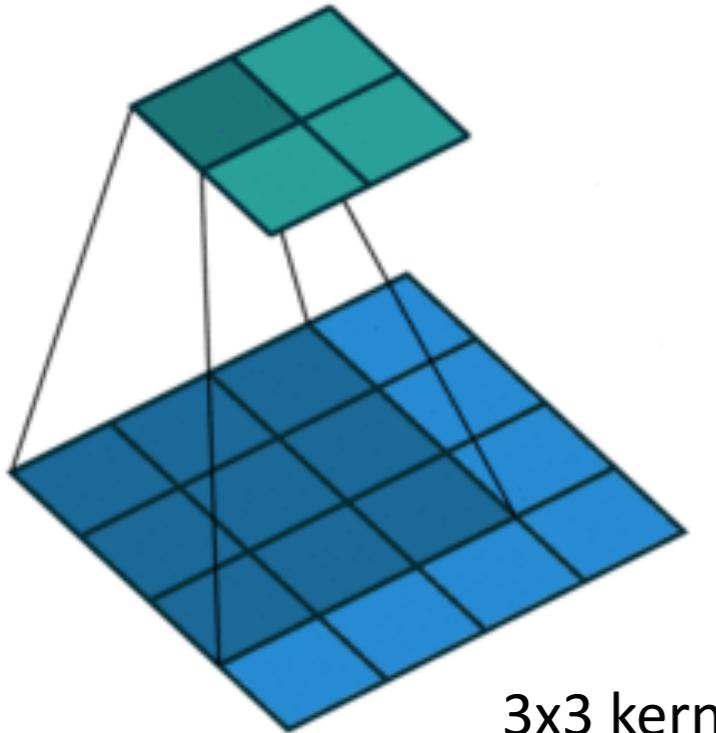
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j;$$

This is convolution!

Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels

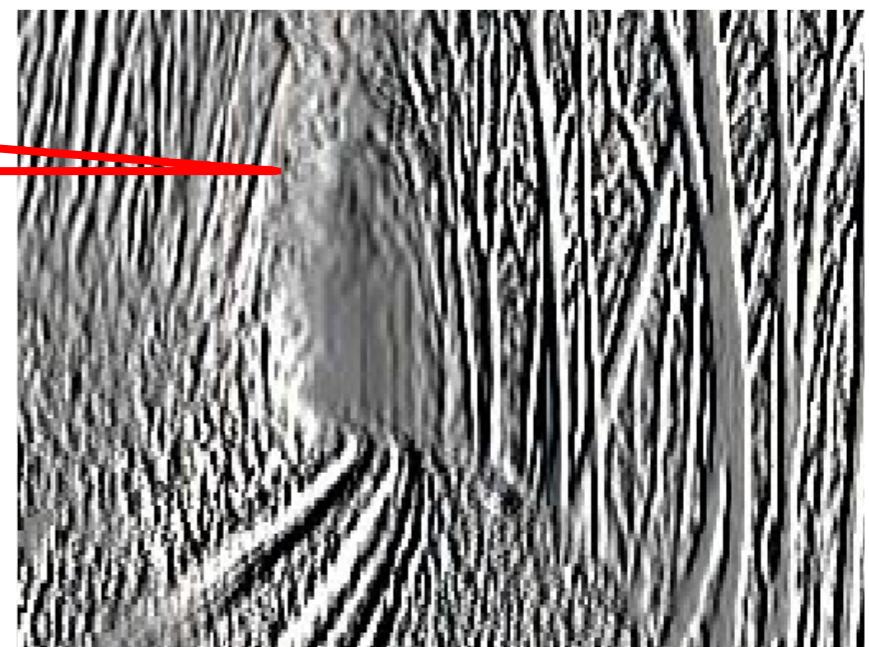


Convolution

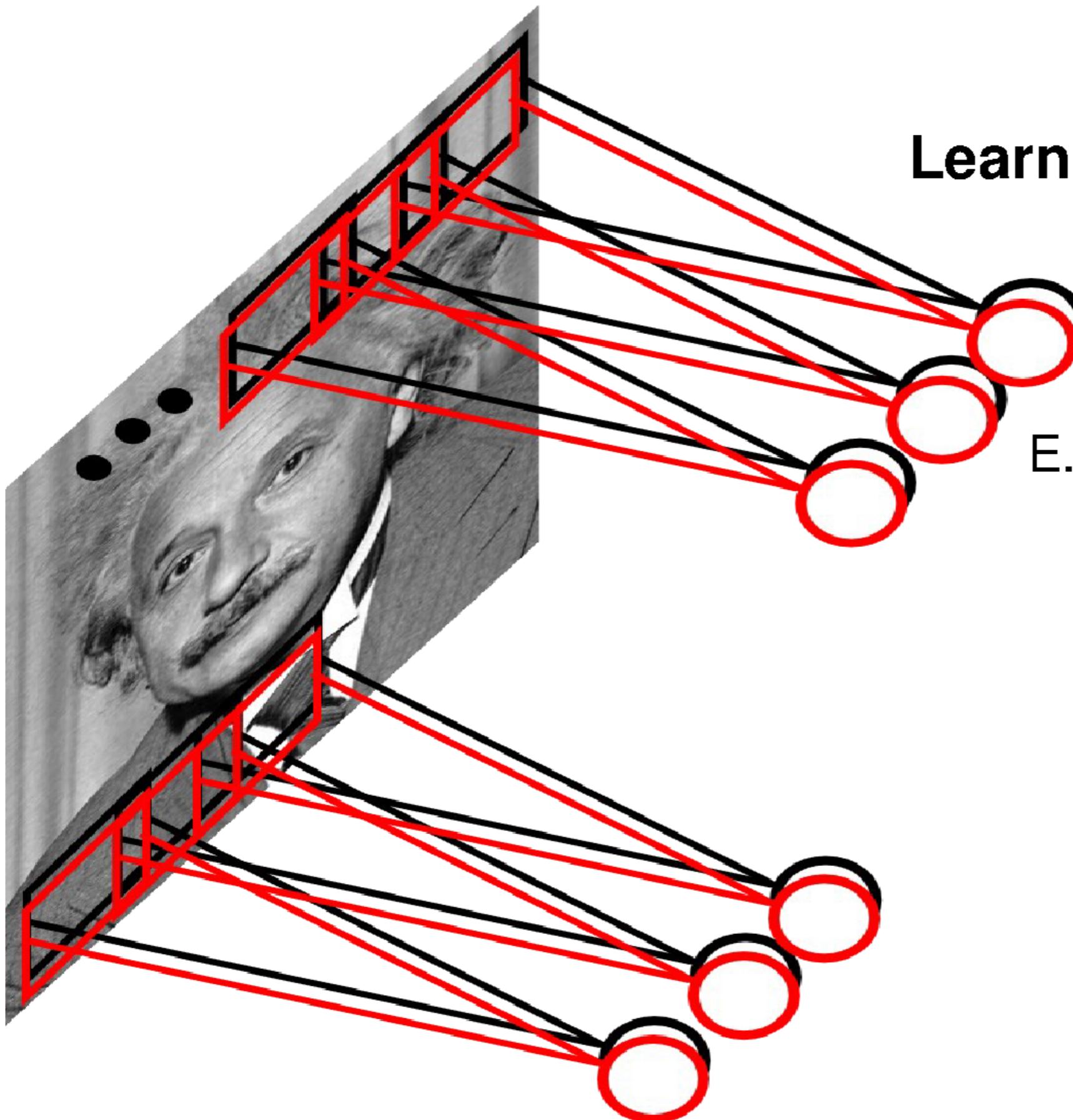


$$\ast \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$

Shared weights



Convolutional Layer

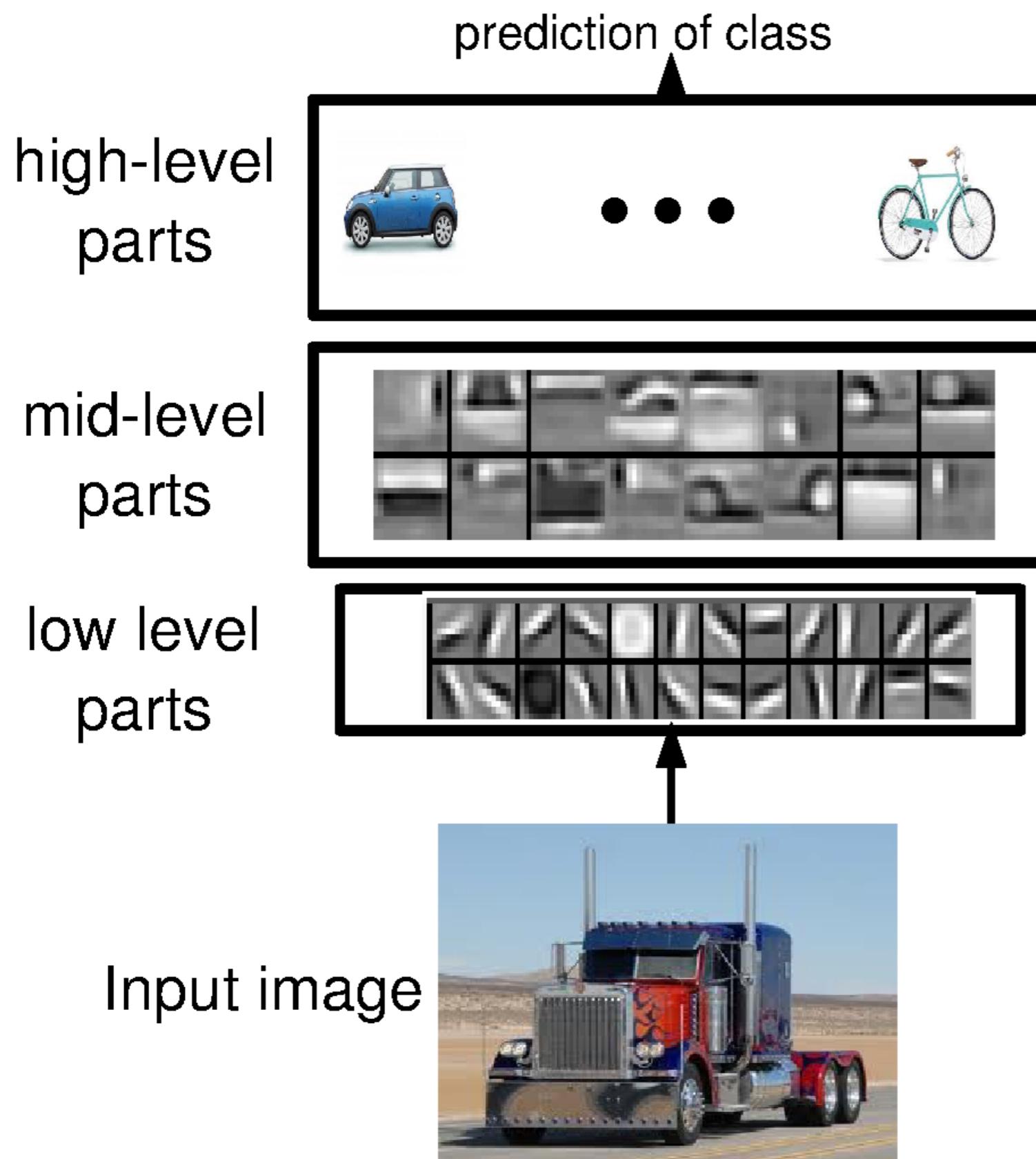


Learn multiple filters.

Filter = 'local' perceptron.
Also called *kernel*.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Interpretation



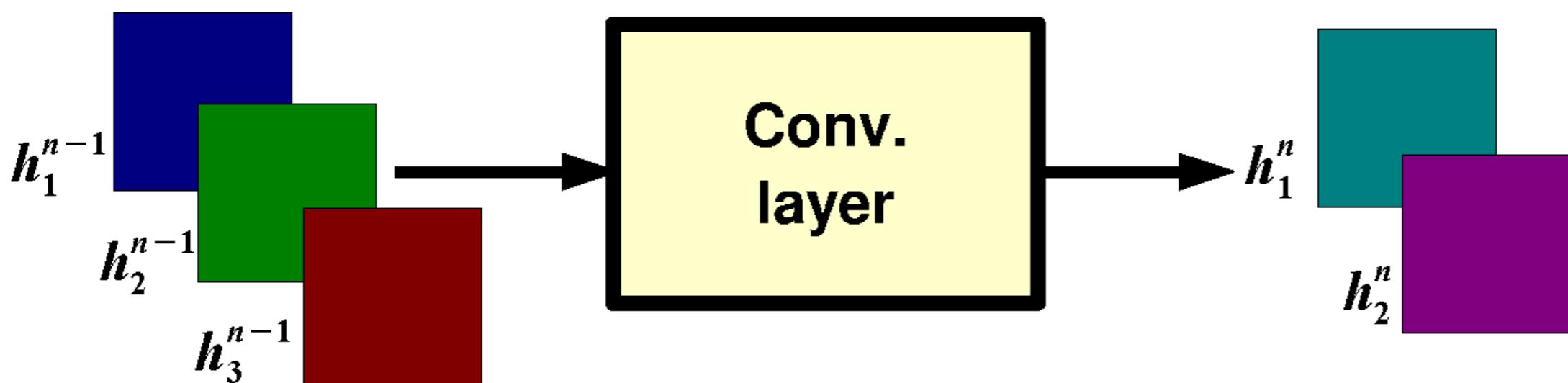
- distributed representations
- feature sharing
- compositionality

Convolutional Layer

$$h_j^n = \max \left(0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n \right)$$

output feature map input feature map kernel

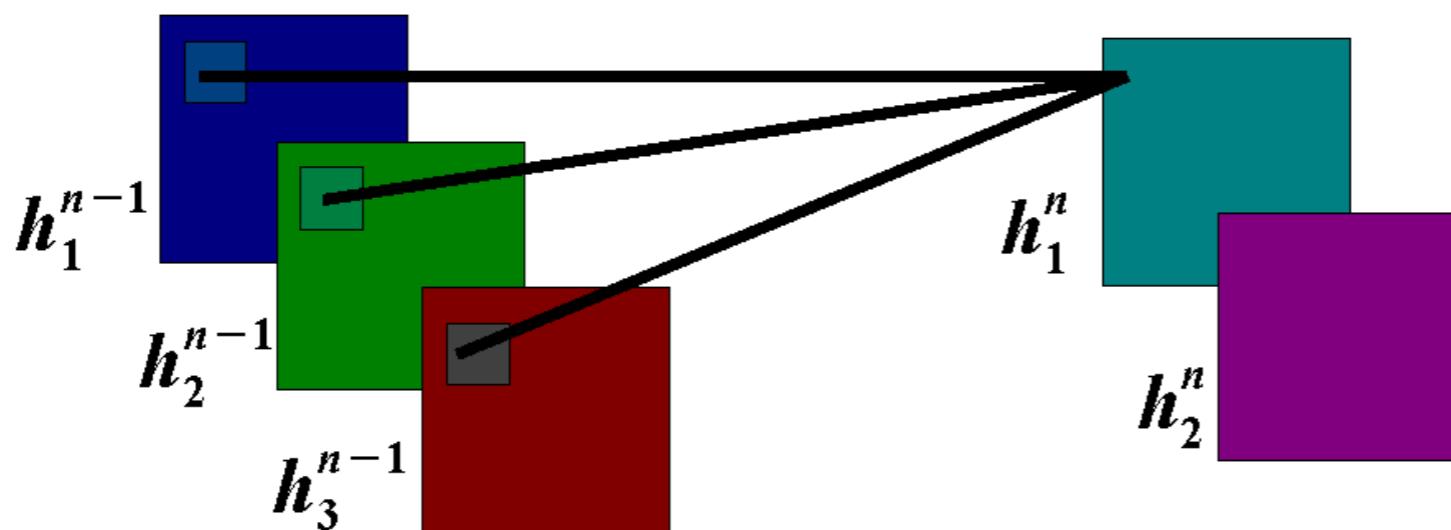
n = layer number
 K = kernel size
 j = # channels (input) or # filters (depth)



Convolutional Layer

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n)$$

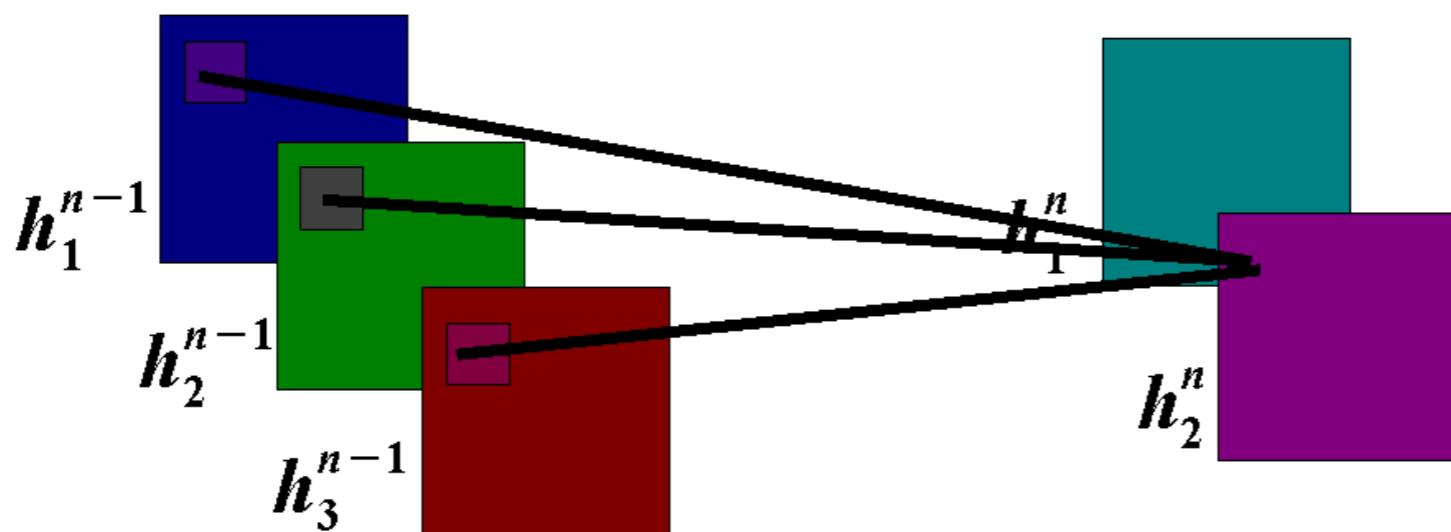
output feature map input feature map kernel



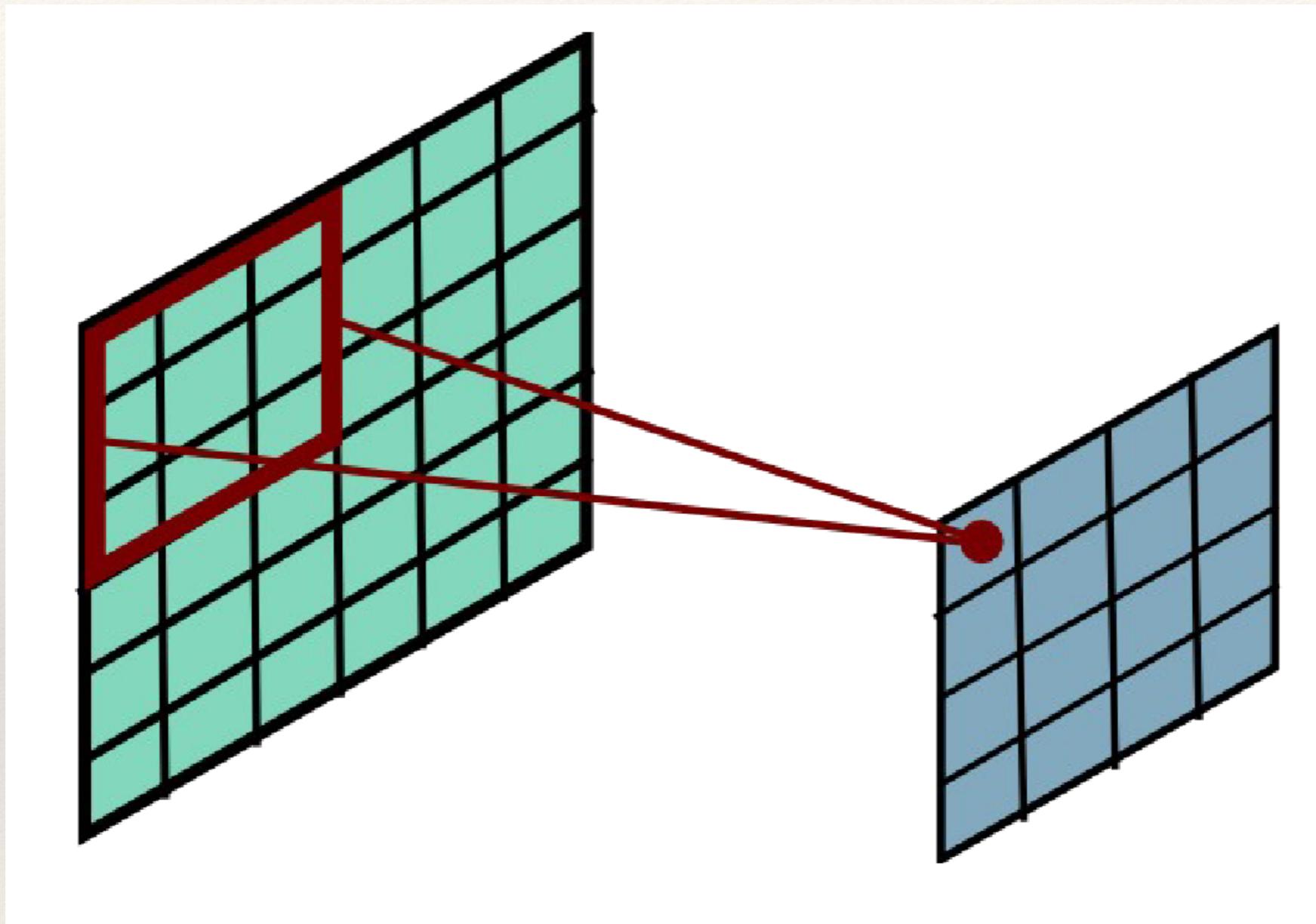
Convolutional Layer

$$h_j^n = \max(0, \sum_{k=1}^K h_k^{n-1} * w_{kj}^n)$$

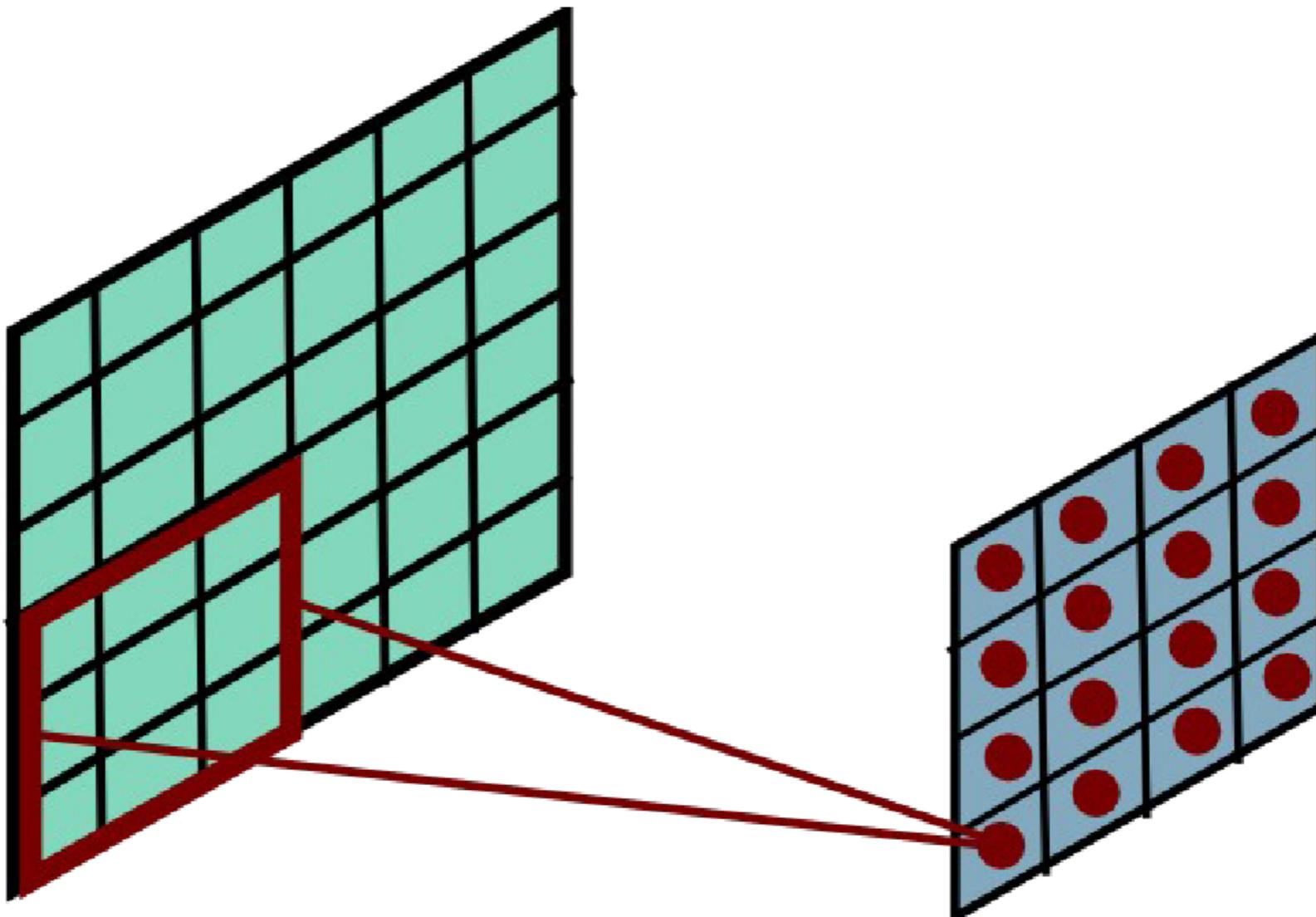
output feature map input feature map kernel



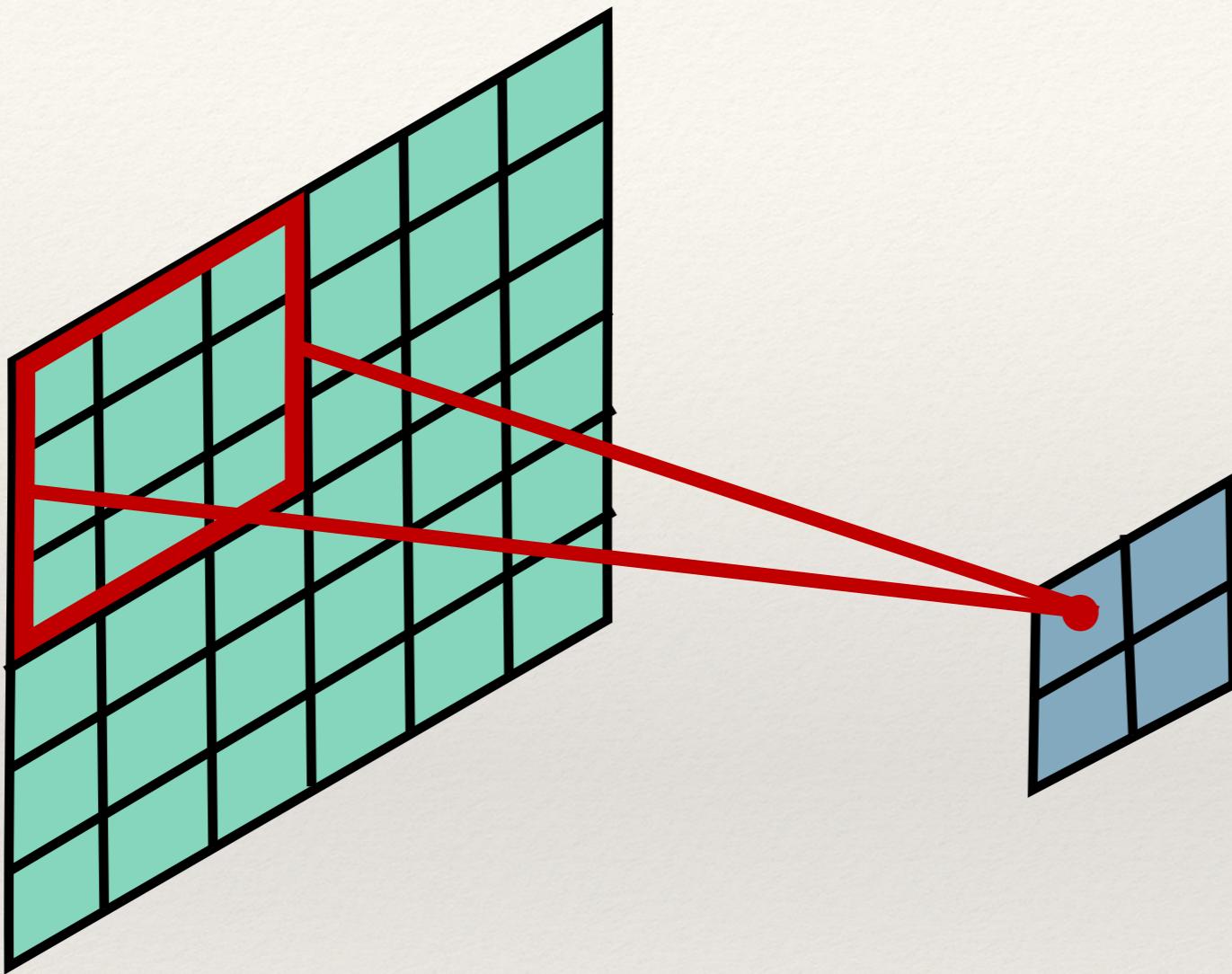
Stride = 1



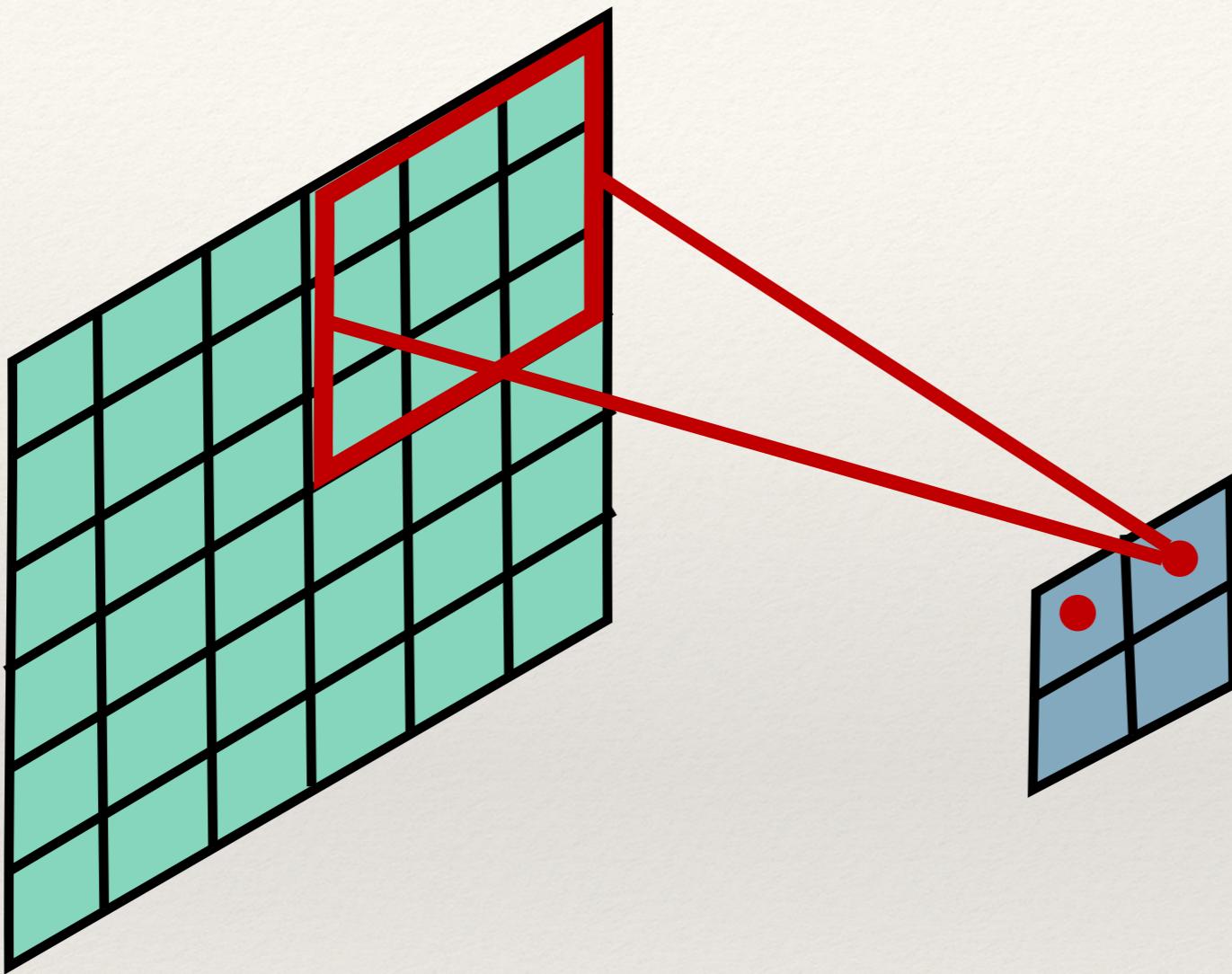
Stride = 1



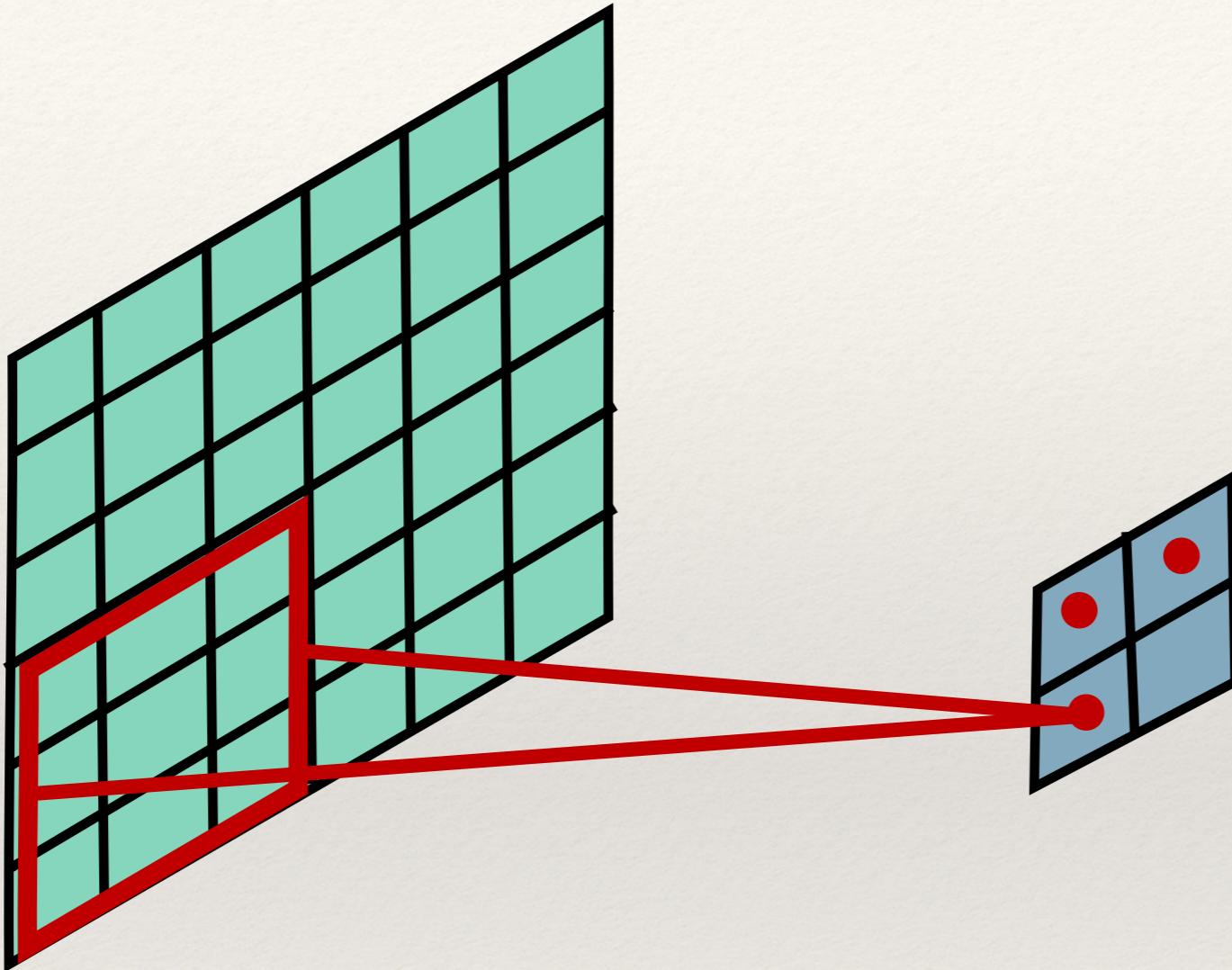
Stride = 3



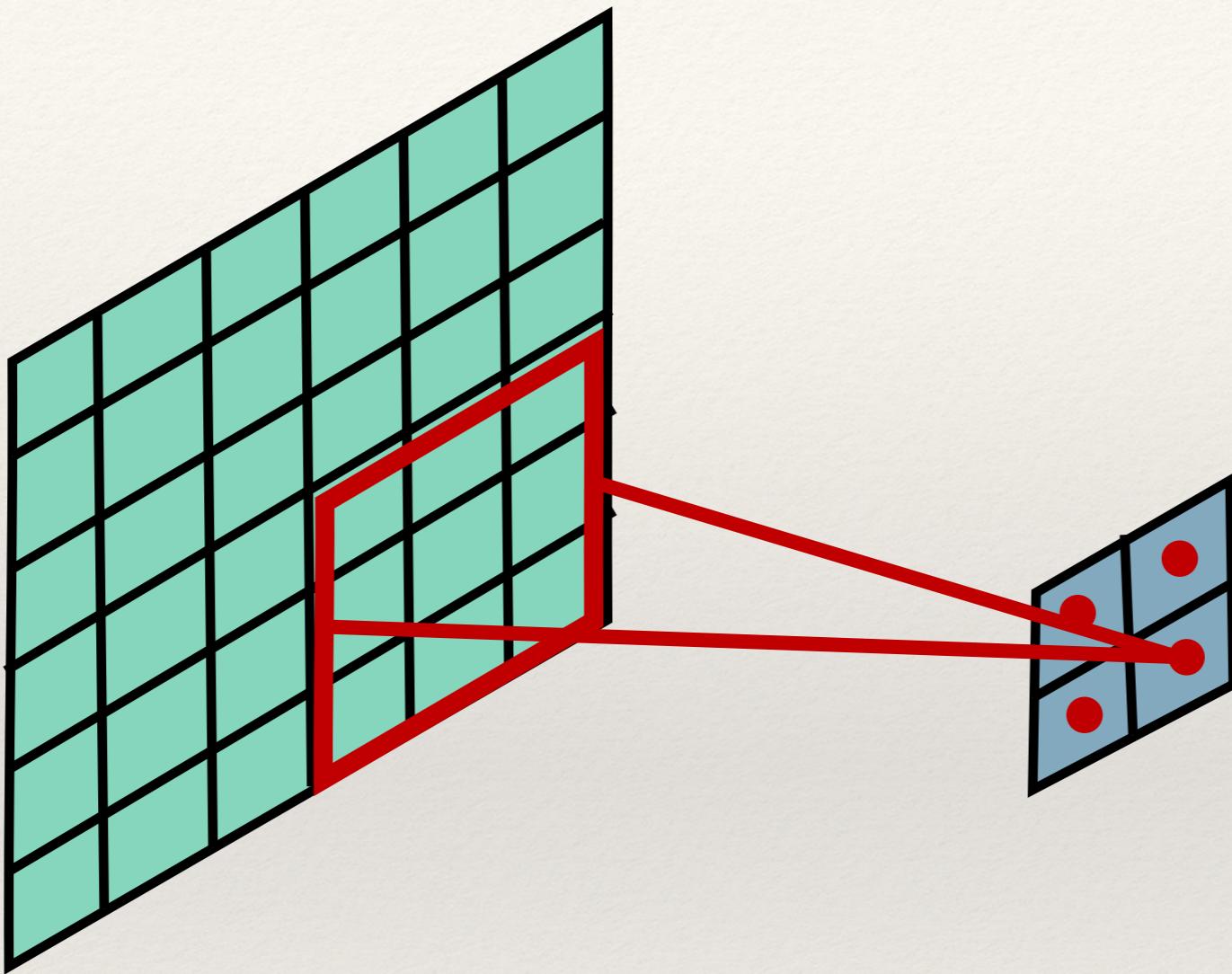
Stride = 3



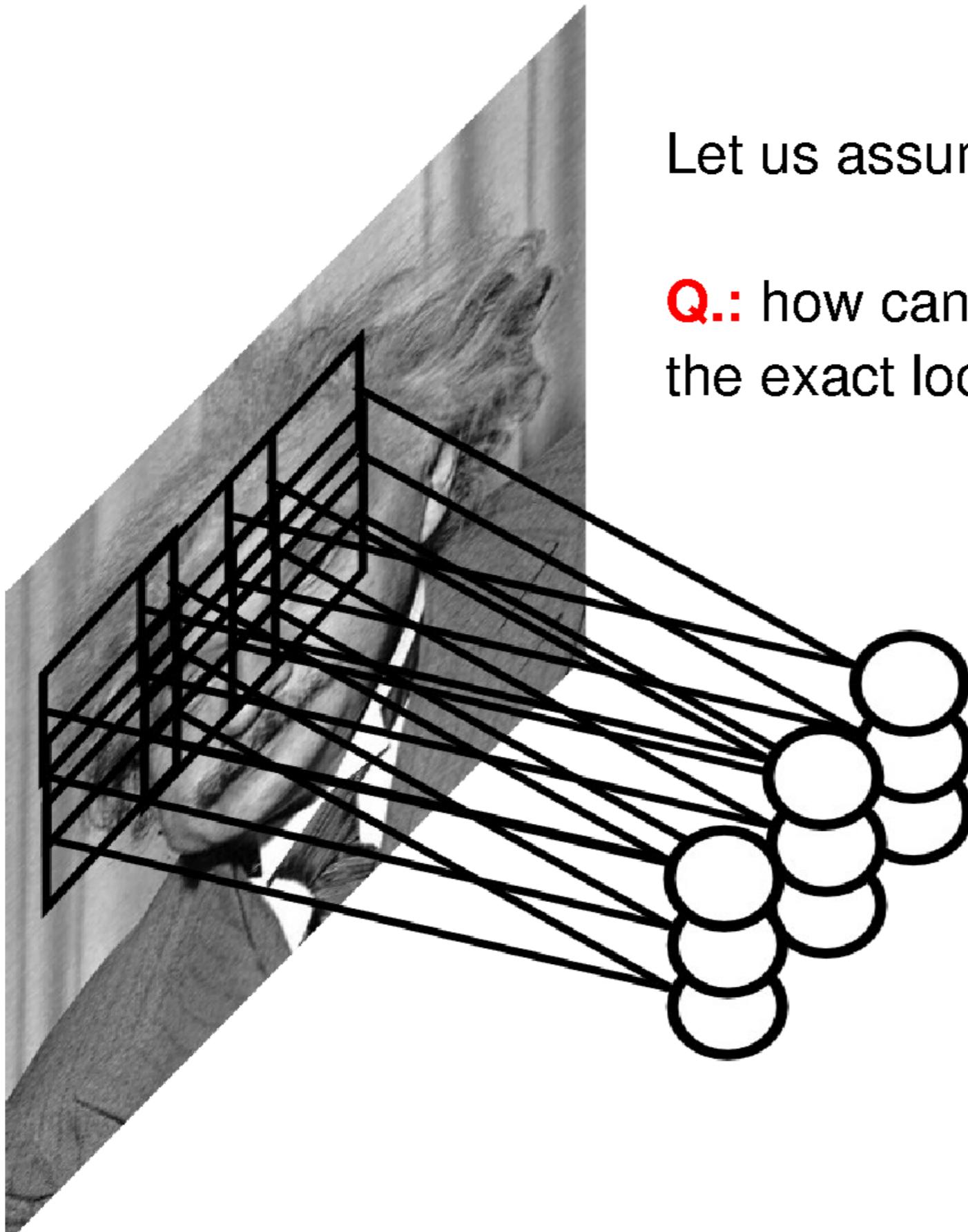
Stride = 3



Stride = 3



Pooling Layer

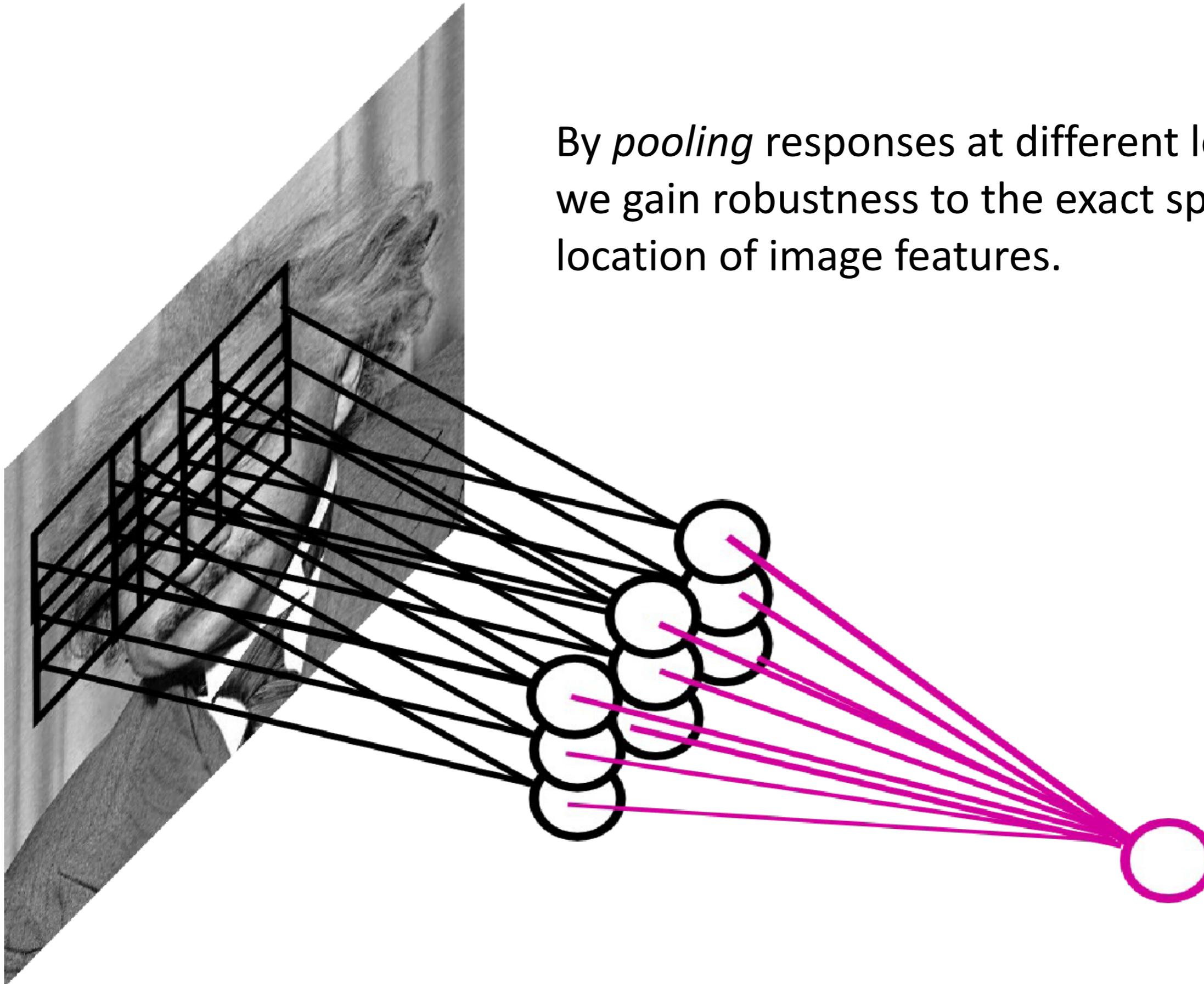


Let us assume filter is an “eye” detector.

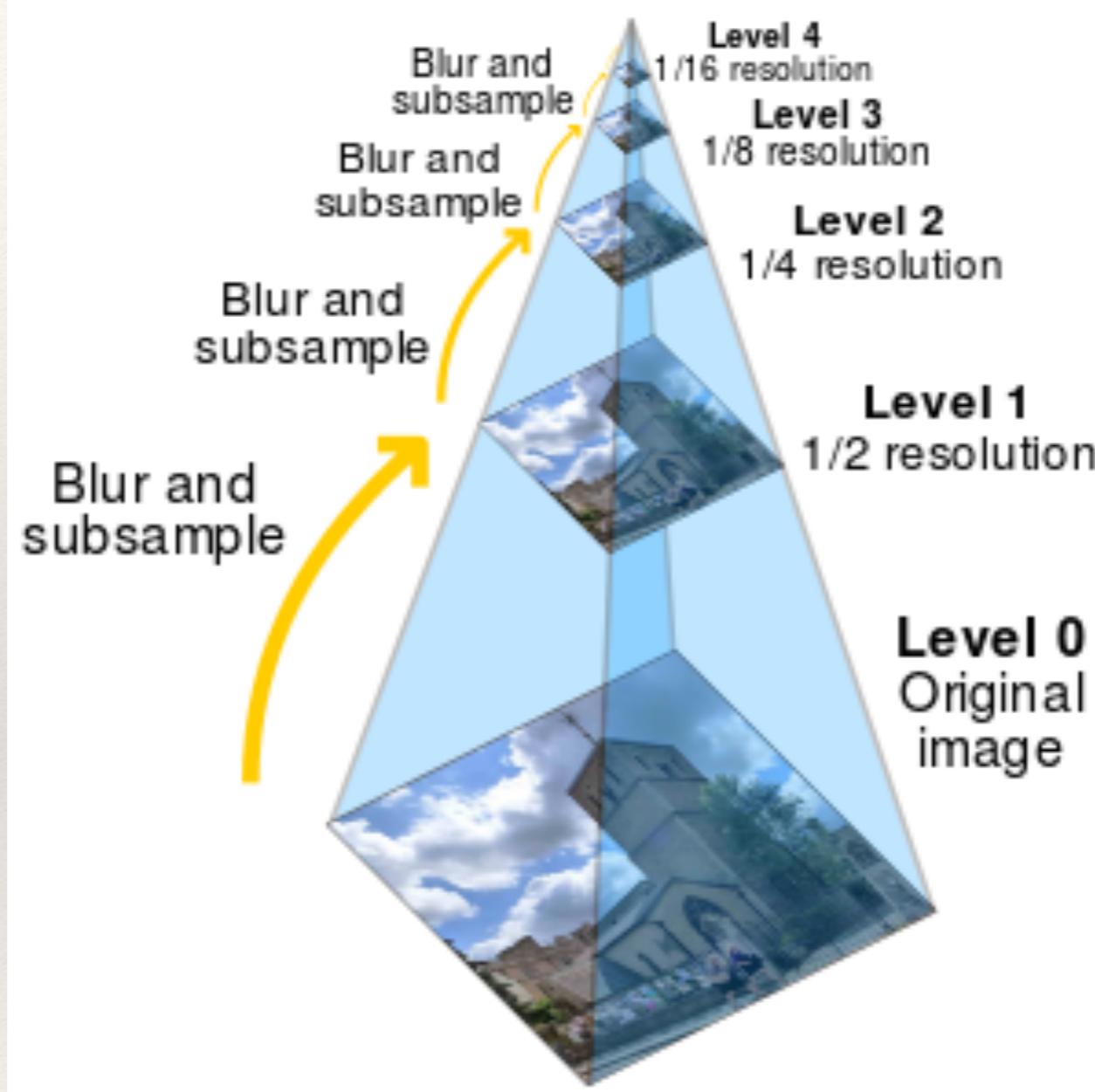
Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer

By *pooling* responses at different locations, we gain robustness to the exact spatial location of image features.

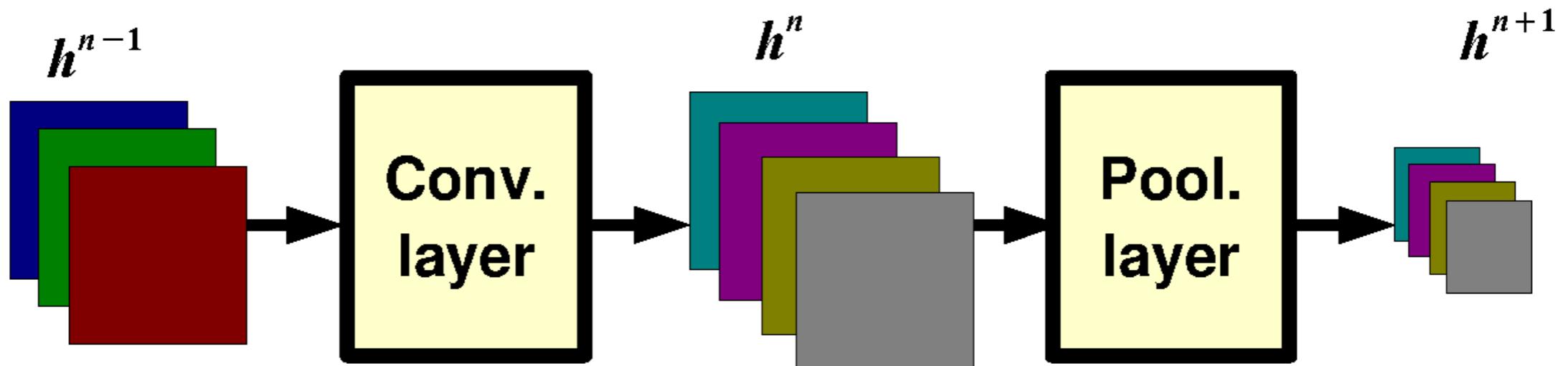


Pooling is similar to downsampling



...except sometimes we don't want to blur,
as other functions might be better for classification.

Pooling Layer: Receptive Field Size



Pooling Layer: Examples

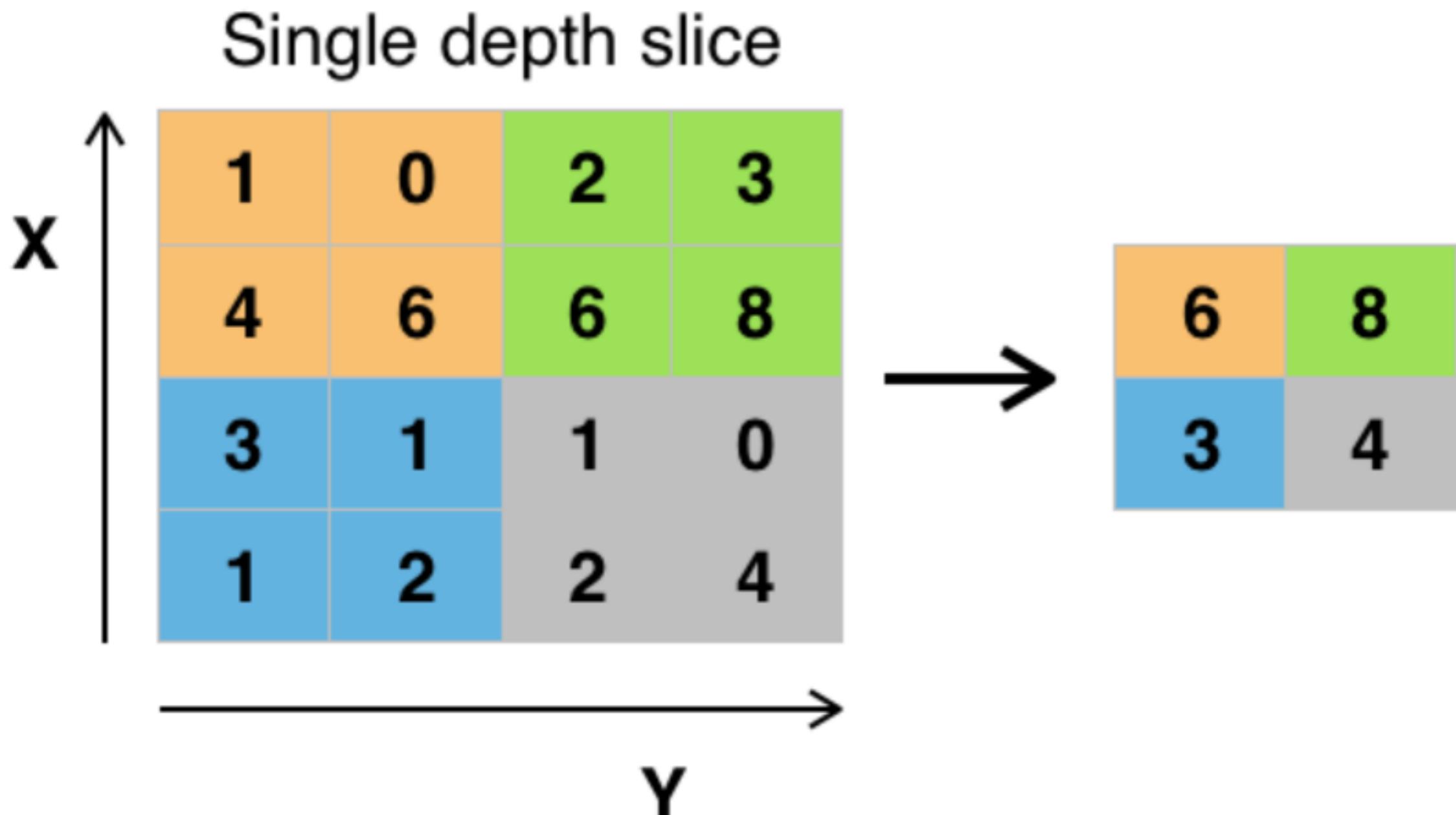
Max-pooling:

$$h_j^n(x, y) = \max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Max pooling



Pooling Layer: Examples

Max-pooling:

$$h_j^n(x, y) = \max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x, y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

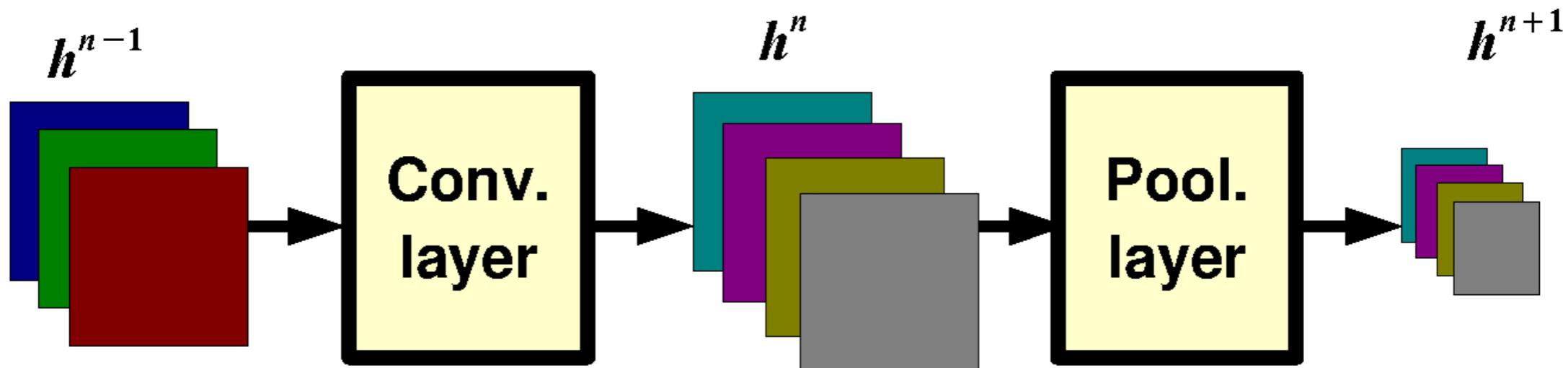
L2-pooling:

$$h_j^n(x, y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

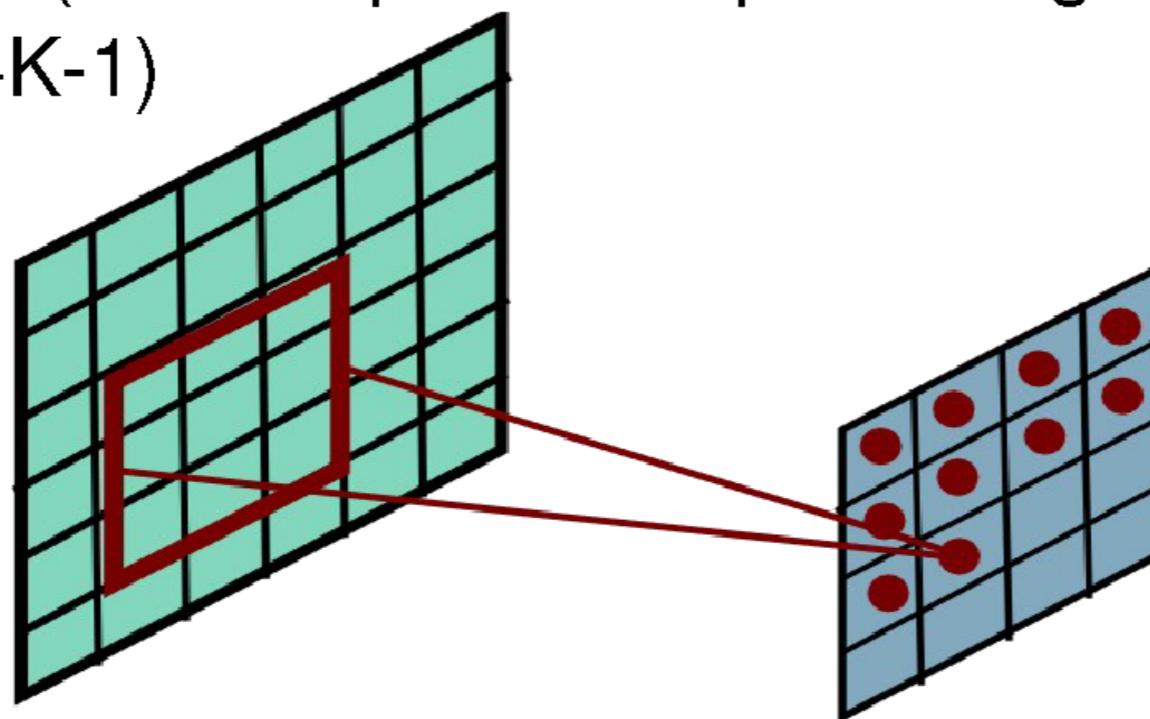
L2-pooling over features:

$$h_j^n(x, y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

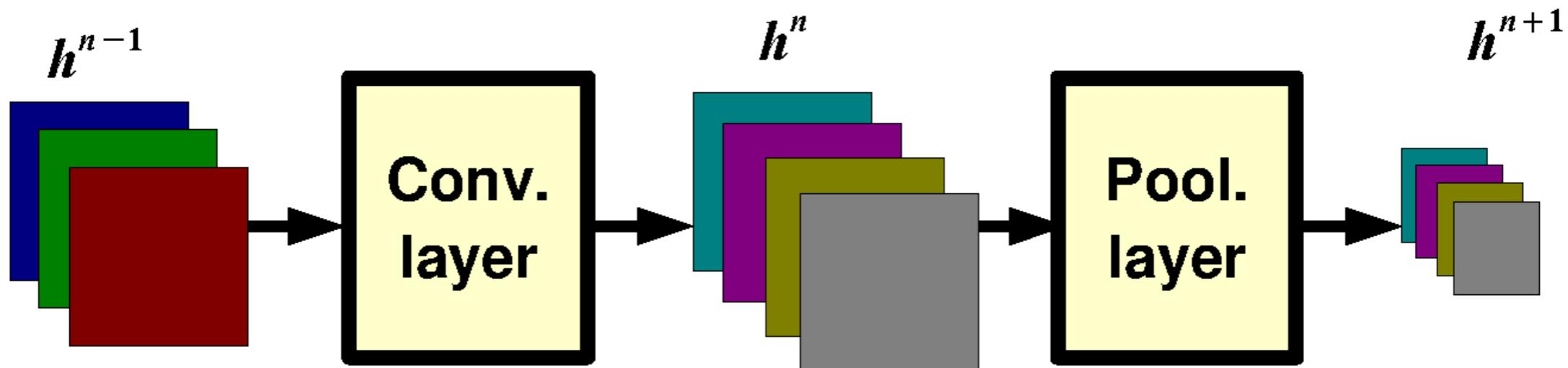
Pooling Layer: Receptive Field Size



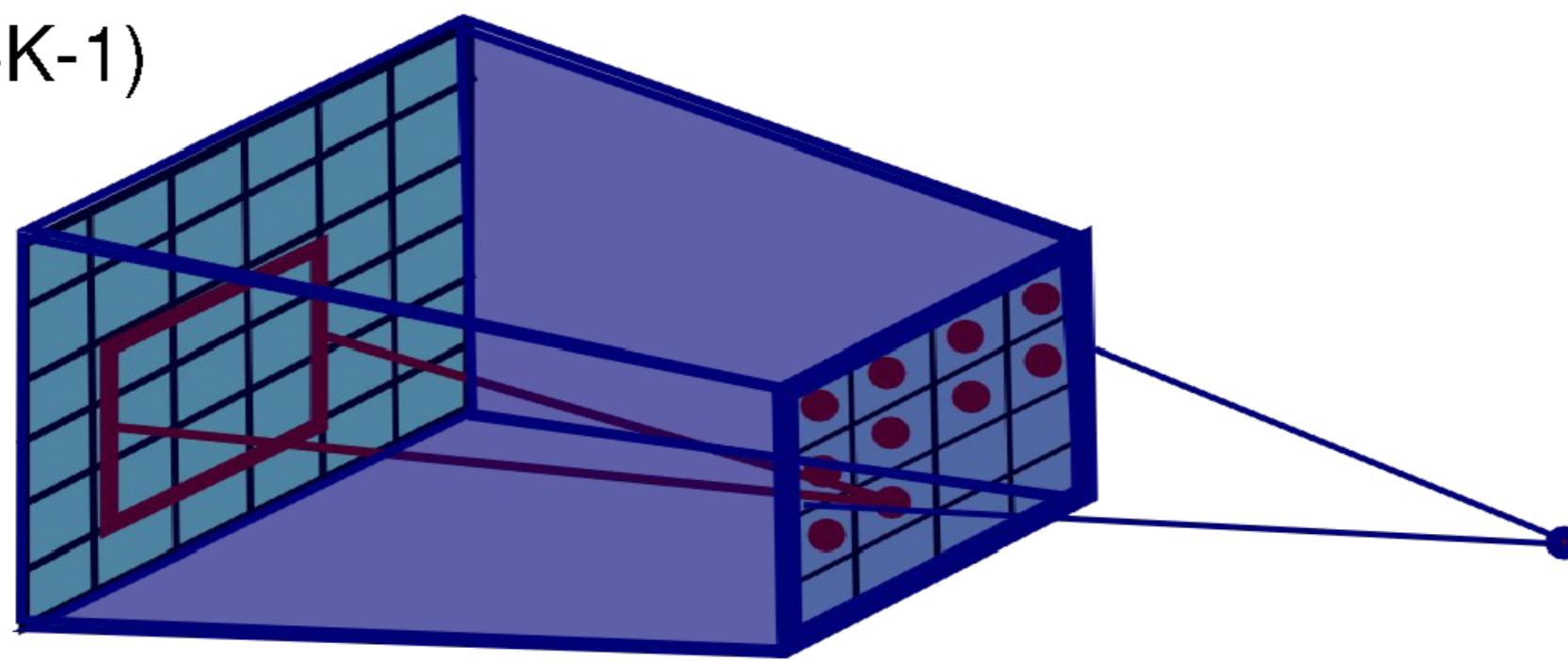
If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$



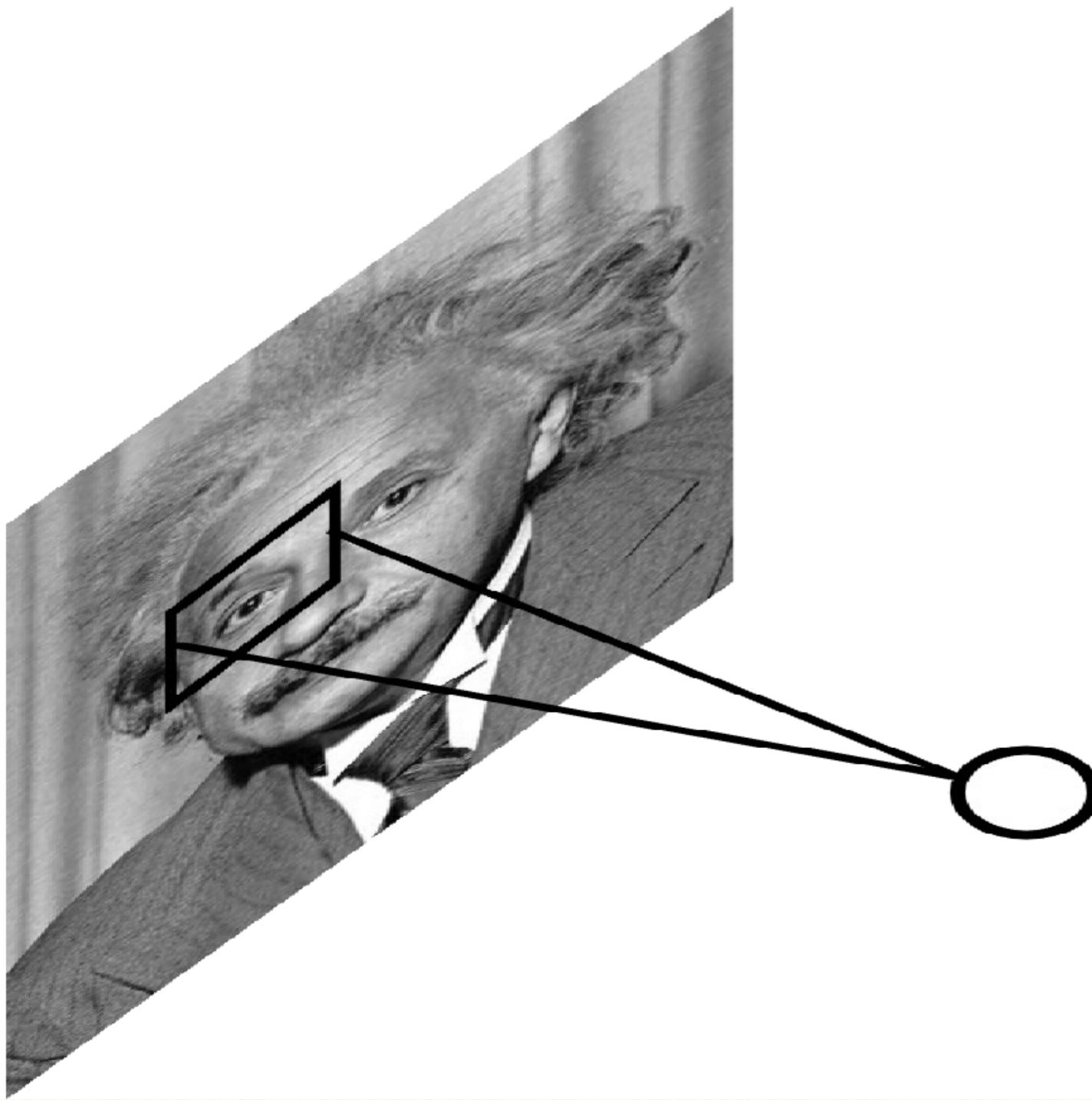
Pooling Layer: Receptive Field Size



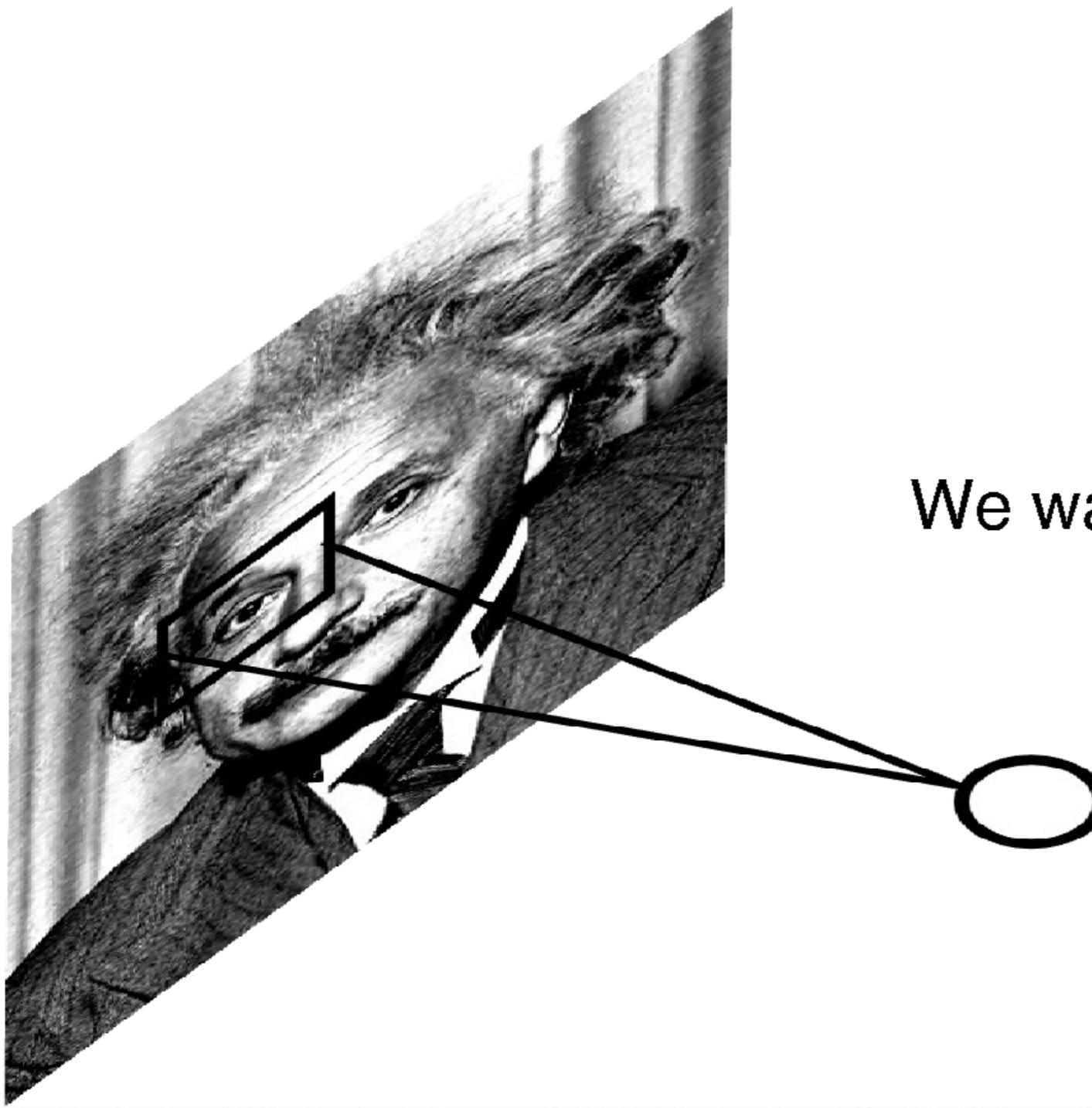
If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$



Local Contrast Normalization



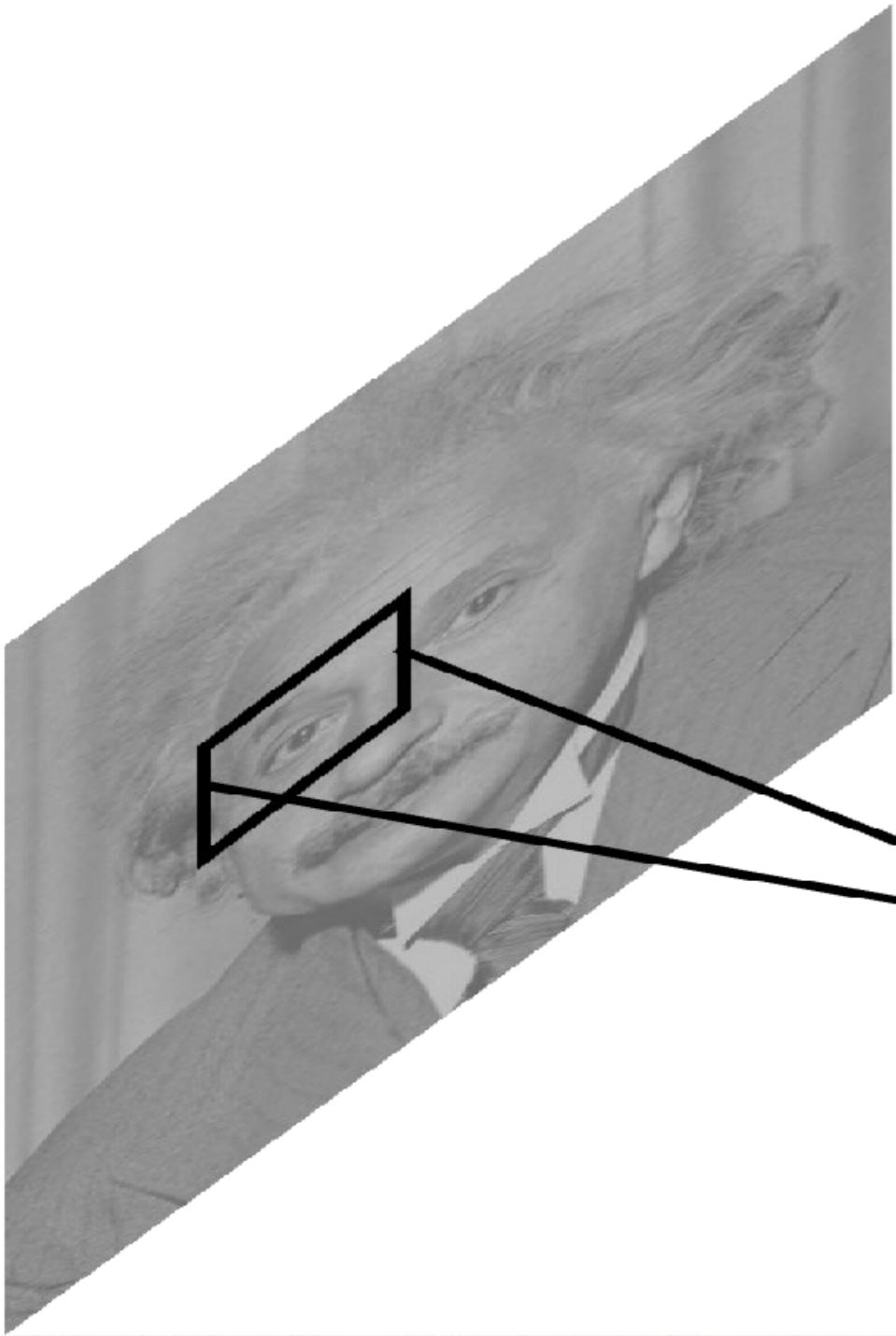
Local Contrast Normalization



We want the same response.

Local Contrast Normalization

$$h^{i+1}(x, y) = \frac{h^i(x, y) - m^i(N(x, y))}{\sigma^i(N(x, y))}$$



$N(x,y)$ = model pixel values in window as a normal distribution

m = mean

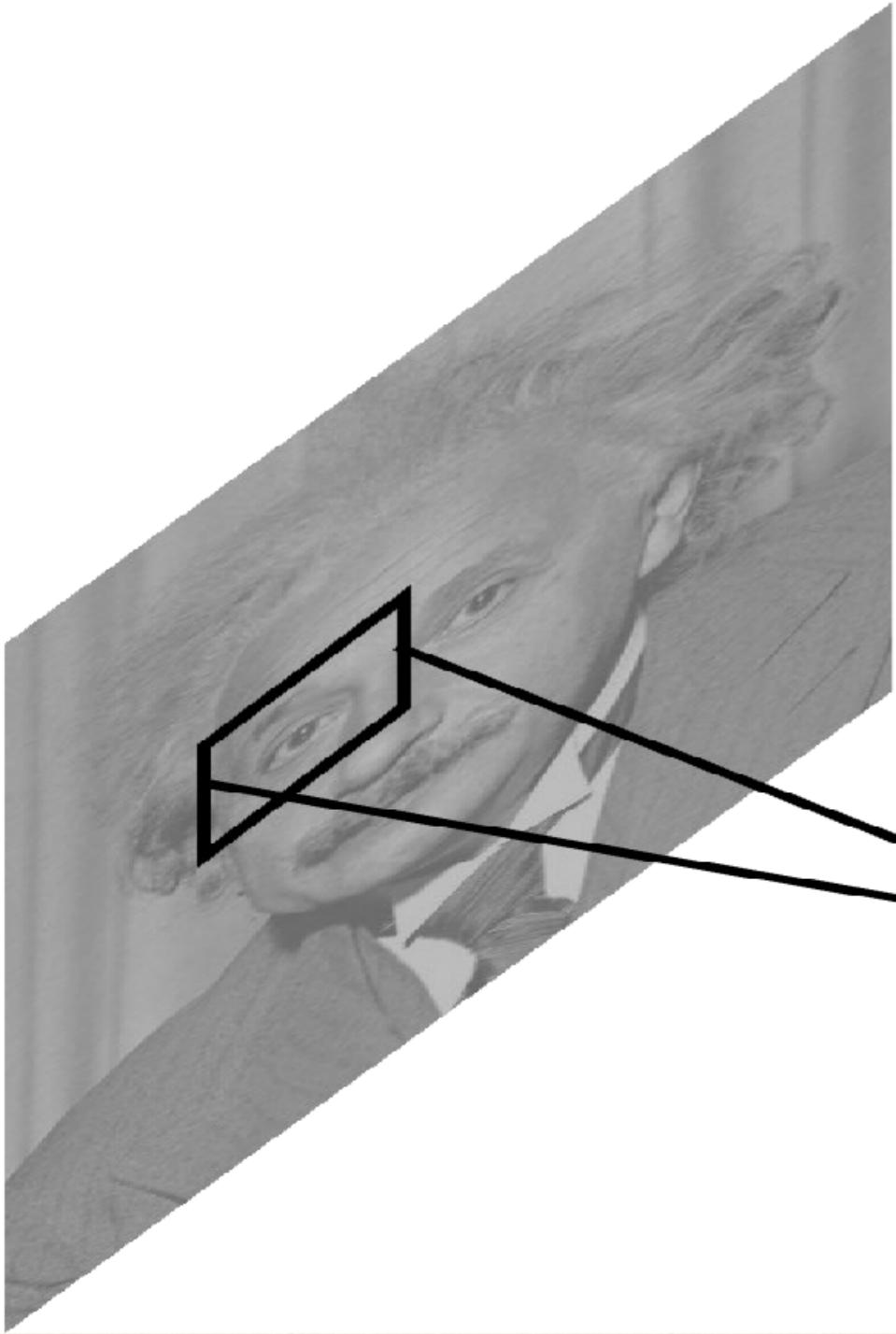
σ = variance

Note: computational cost is negligible w.r.t. conv. layer.

70

Local Contrast Normalization

$$h^{i+1}(x, y) = \frac{h^i(x, y) - m^i(N(x, y))}{\sigma^i(N(x, y))}$$



Performed also across features
and in the higher layers..

Effects:

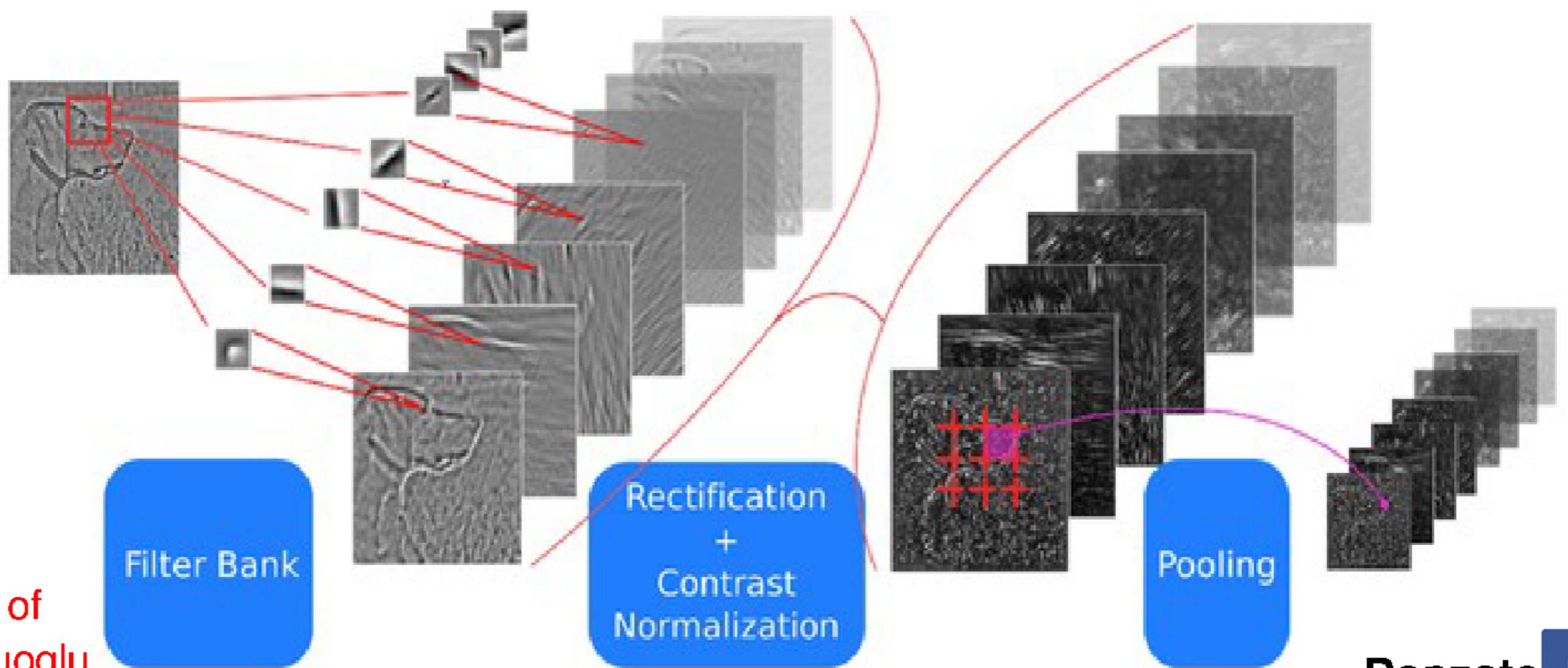
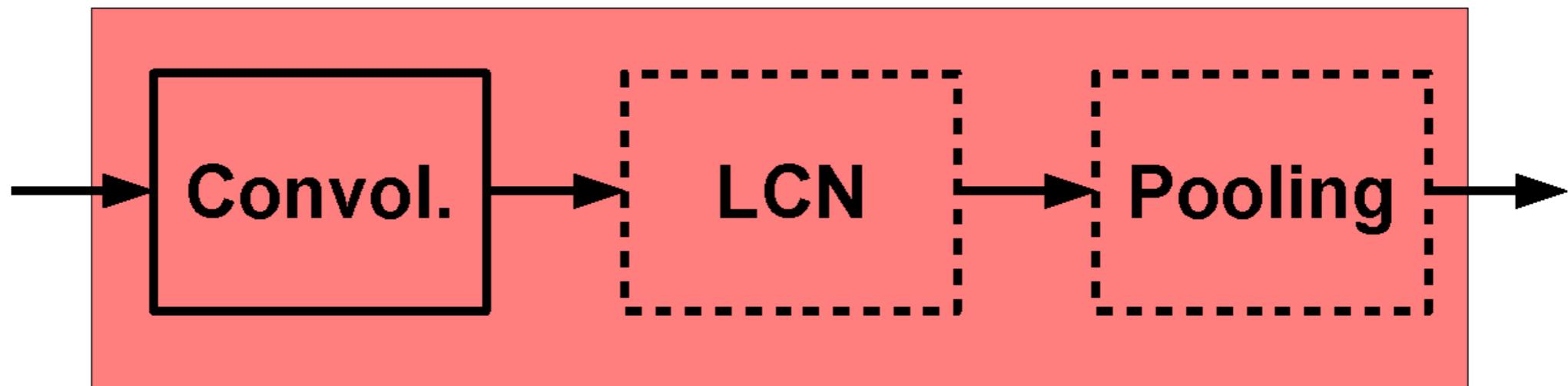
- improves invariance
- improves optimization
- increases sparsity

Note: computational cost is negligible w.r.t. conv. layer.

70

ConvNets: Typical Stage

One stage (zoom)

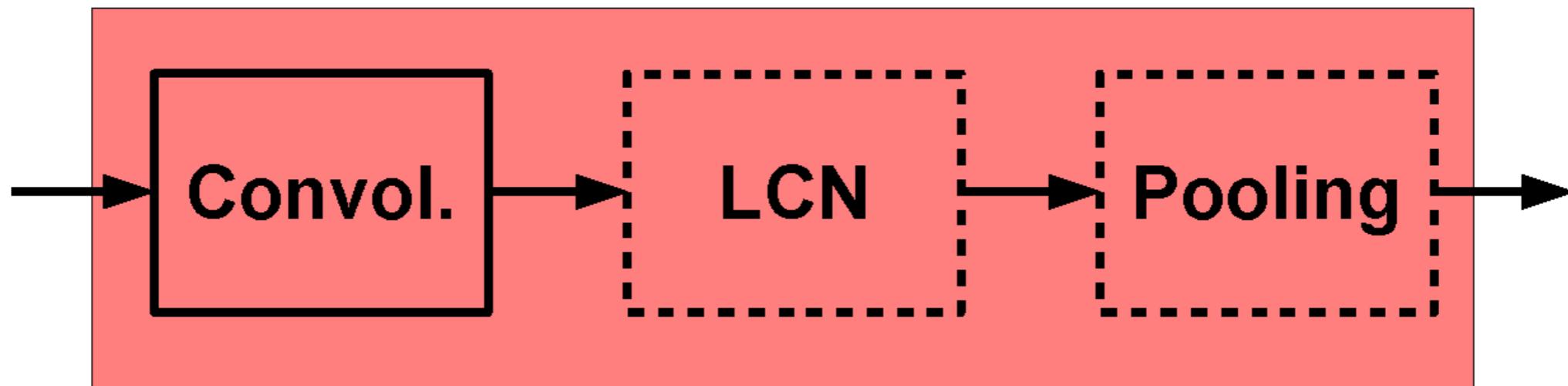


courtesy of
K. Kavukcuoglu

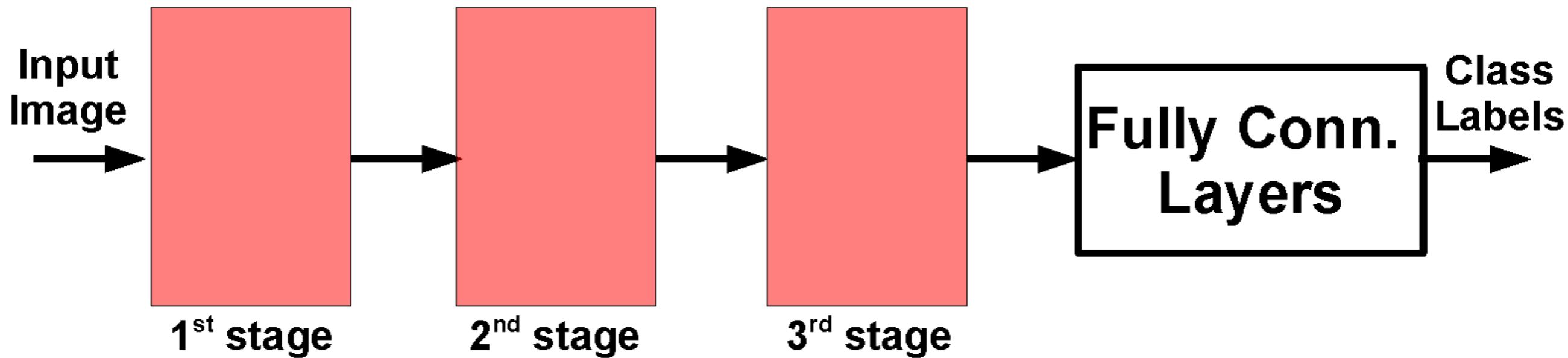
Ranzato

ConvNets: Typical Architecture

One stage (zoom)

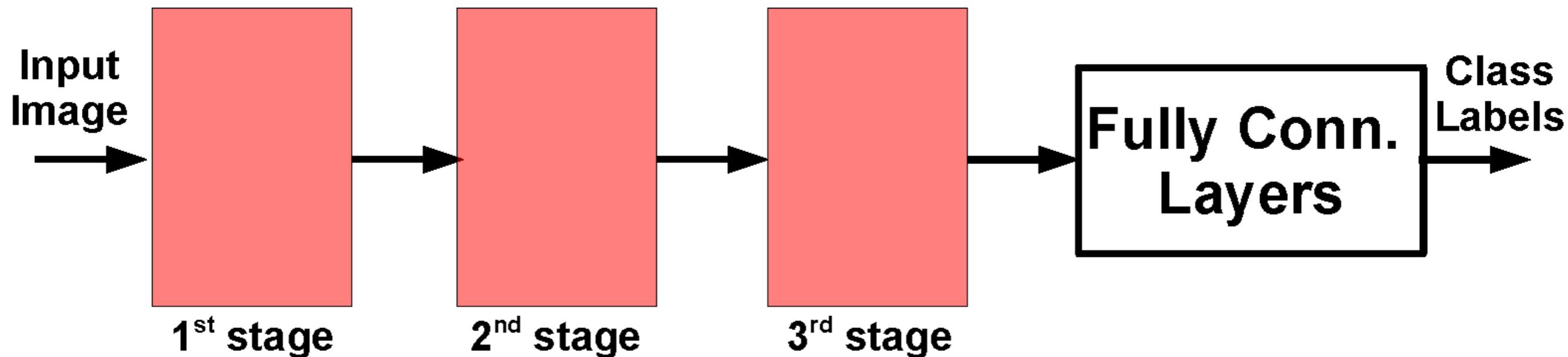


Whole system



ConvNets: Typical Architecture

Whole system



Conceptually similar to:

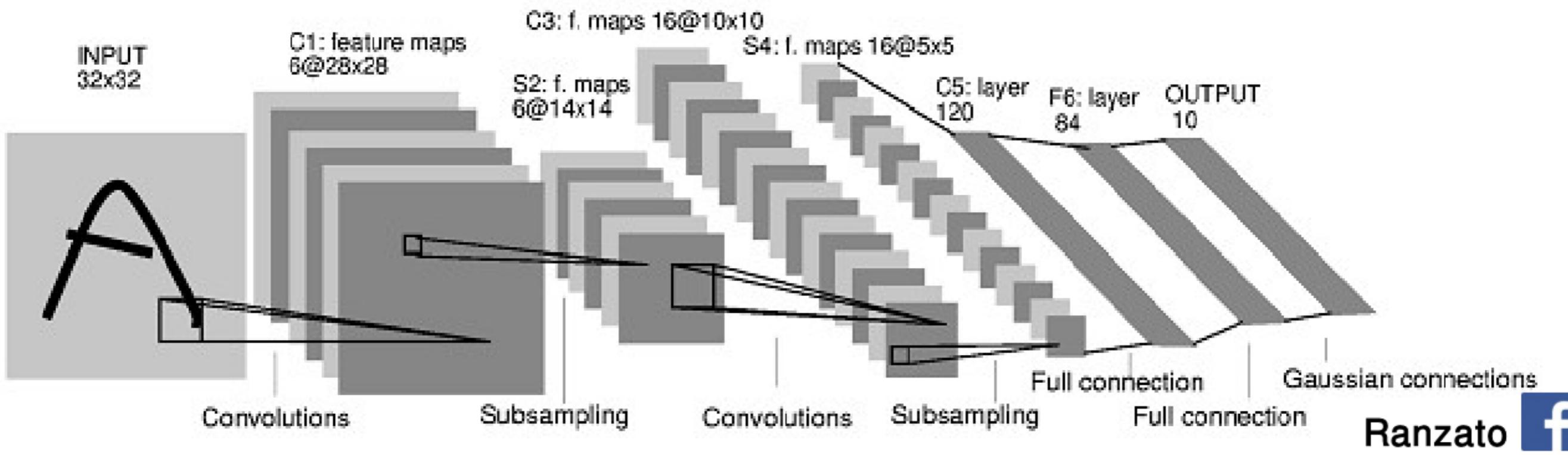
SIFT → K-Means → Pyramid Pooling → SVM

Lazebnik et al. "...Spatial Pyramid Matching..." CVPR 2006

SIFT → Fisher Vect. → Pooling → SVM

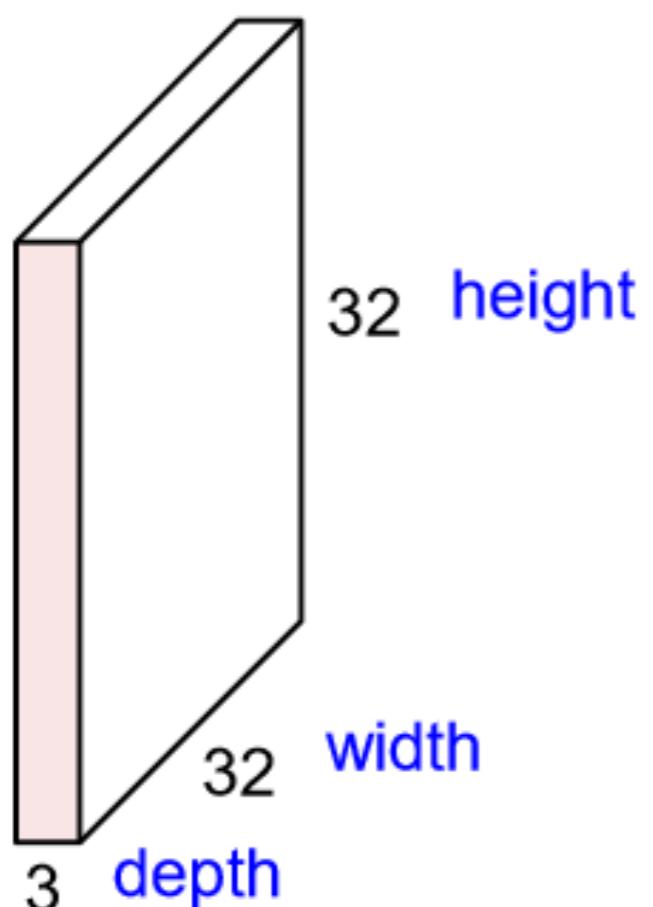
Sanchez et al. "Image classification with F.V.: Theory and practice" IJCV 2012

Yann LeCun's MNIST CNN architecture



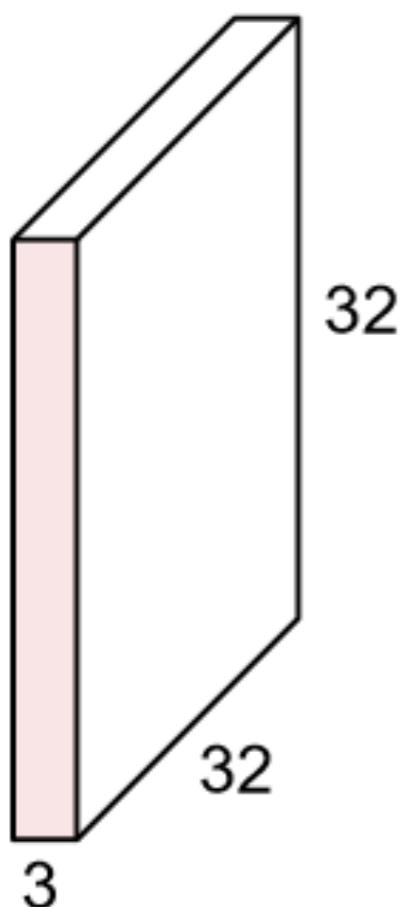
Convolutions: More detail

32x32x3 image



Convolutions: More detail

32x32x3 image

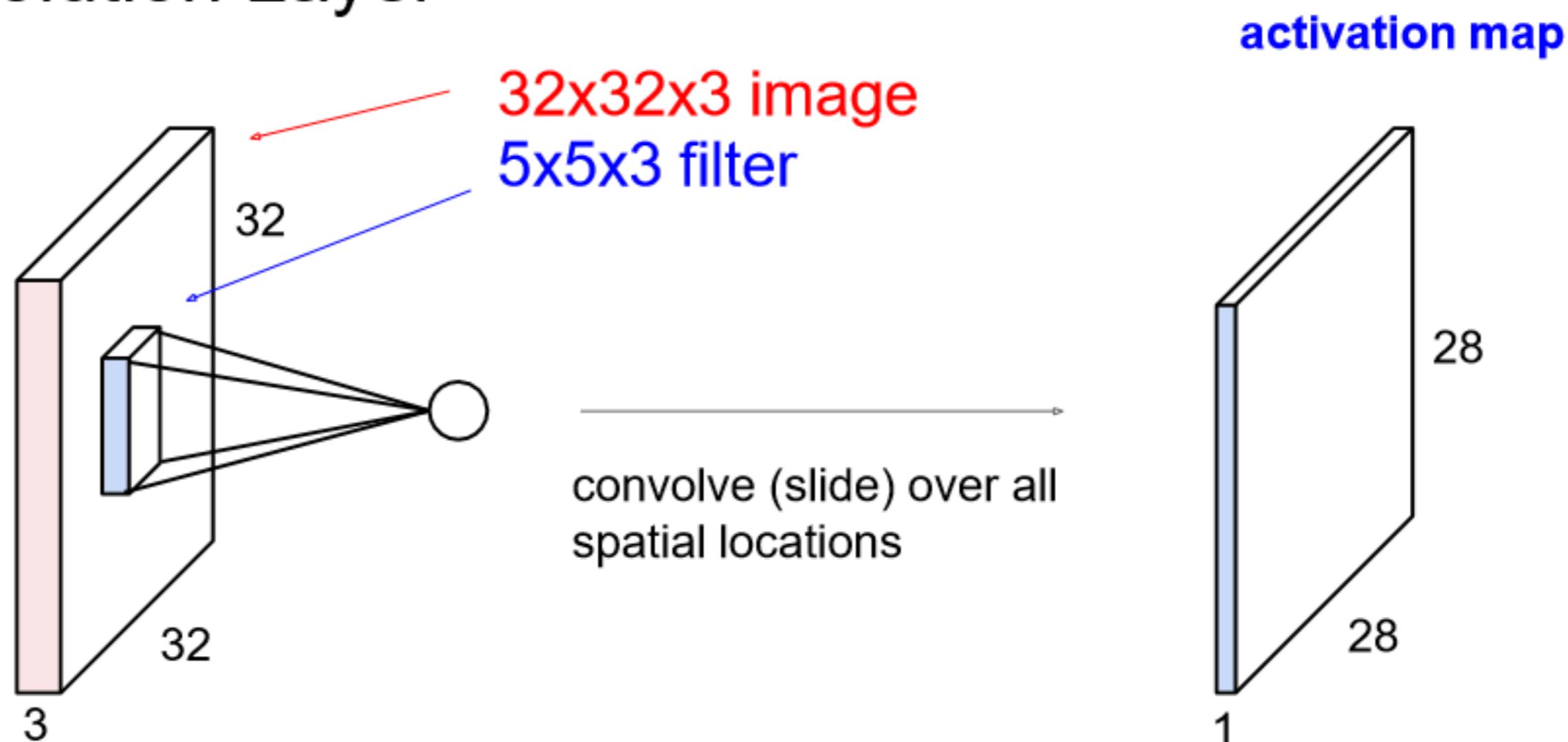


5x5x3 filter



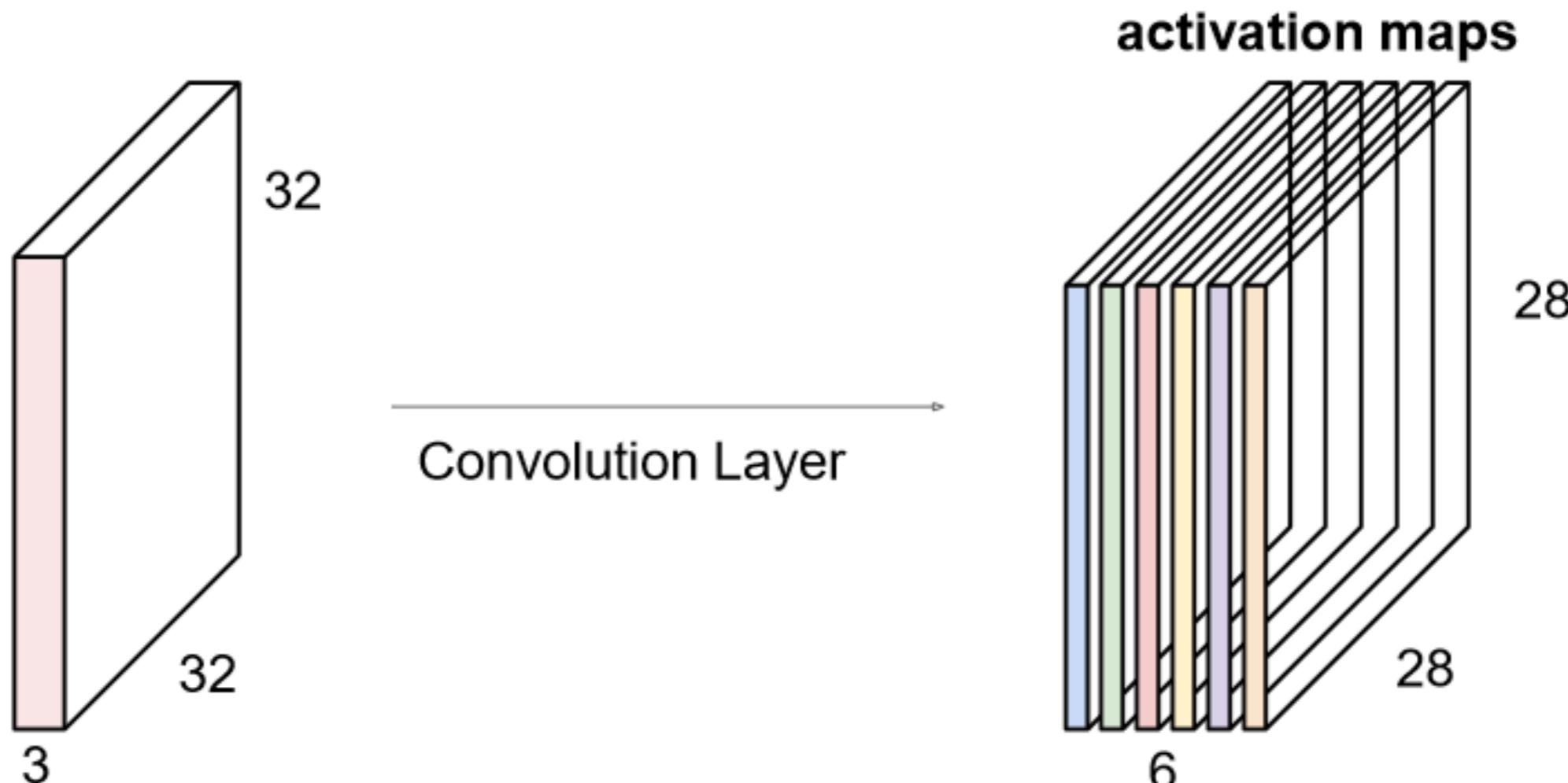
Convolutions: More detail

Convolution Layer



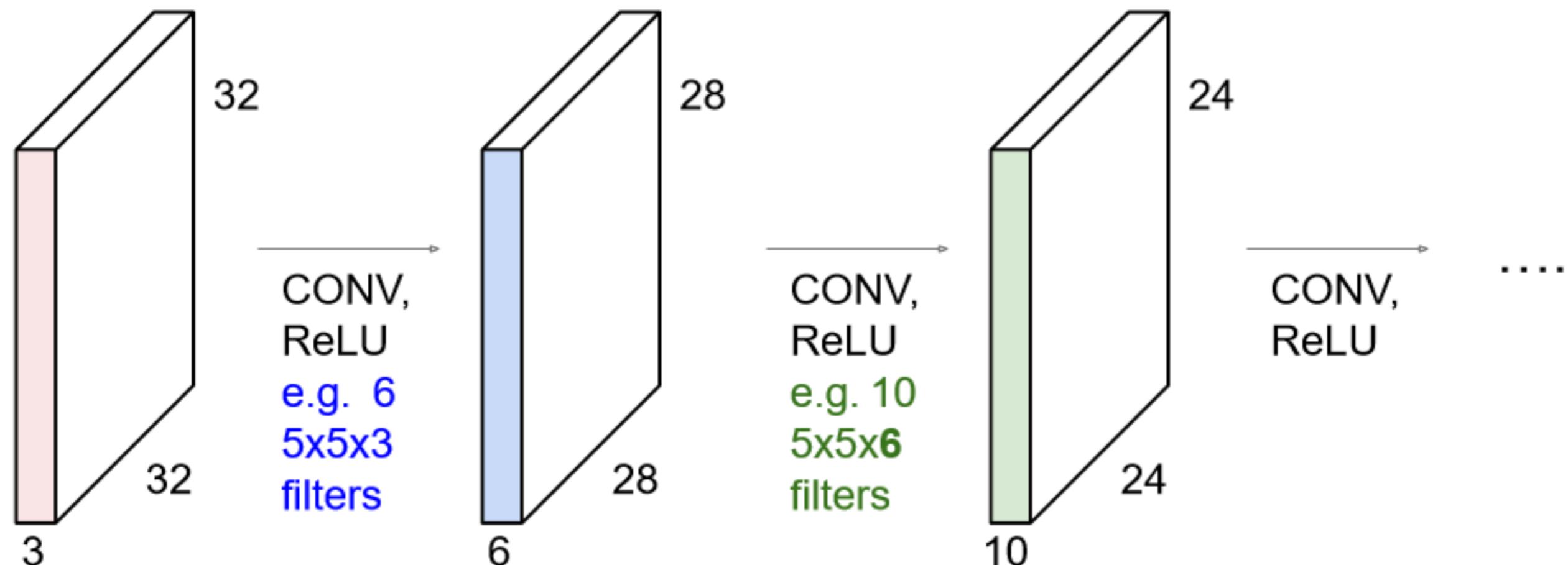
Convolutions: More detail

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

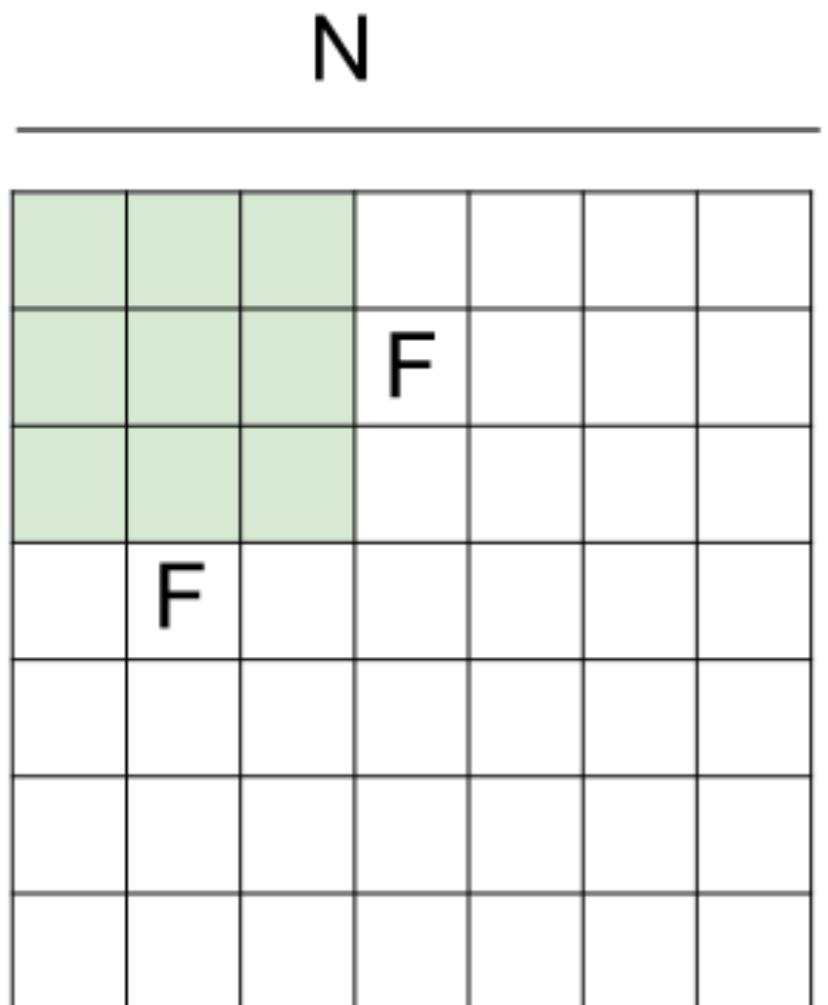


We stack these up to get a “new image” of size $28 \times 28 \times 6$!

Convolutions: More detail



Convolutions: More detail



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

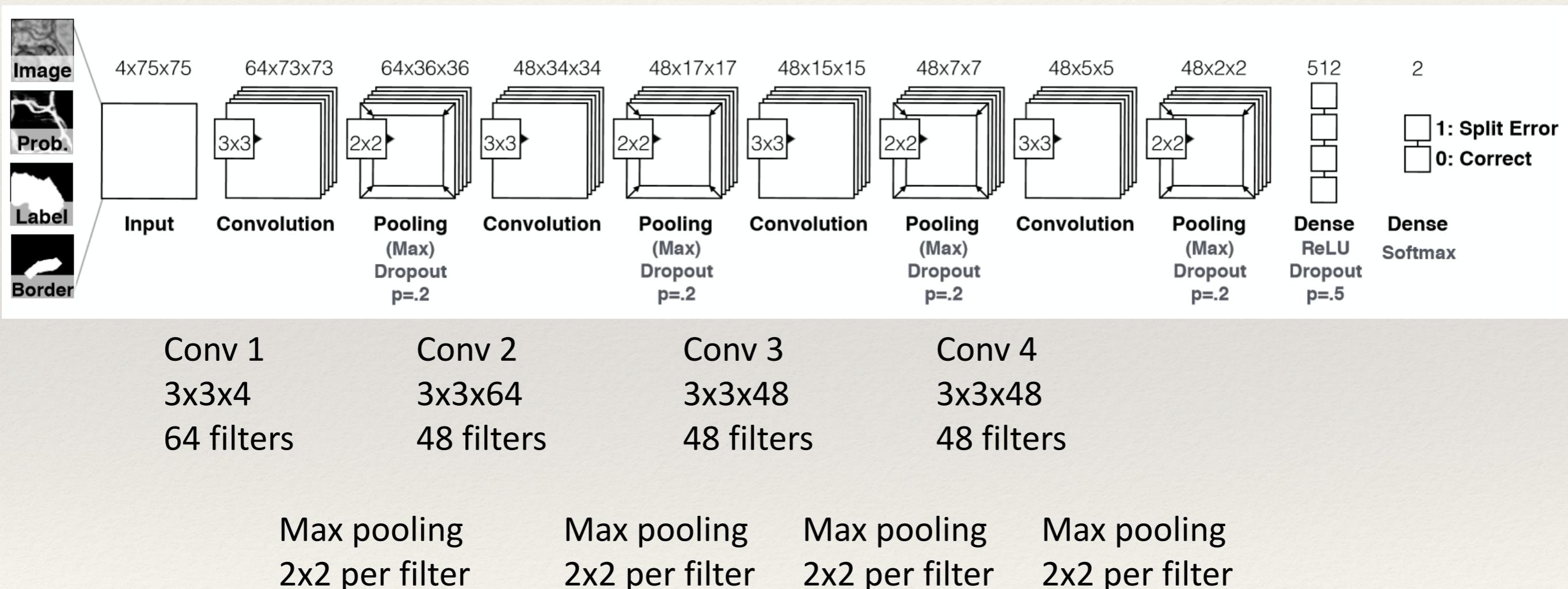
N

Our connectomics diagram

Auto-generated from network declaration by *nolearn* (for Lasagne / Theano)

Input

75x75x4



Reading architecture diagrams

Layers

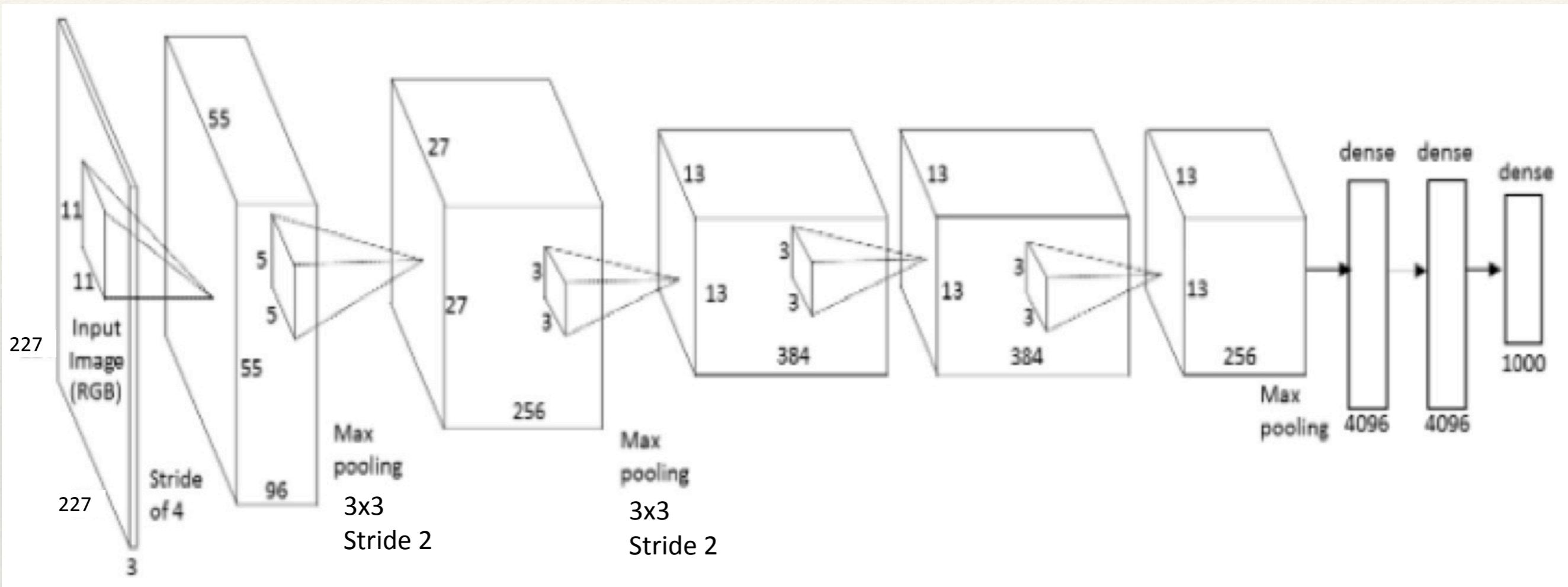
- Kernel sizes
- Strides
- # channels
- # kernels
- Max pooling

params	AlexNet	FLOPs
4M	FC 1000	4M
16M	FC 4096 / ReLU	16M
37M	FC 4096 / ReLU	37M
	Max Pool 3x3s2	
442K	Conv 3x3s1, 256 / ReLU	74M
1.3M	Conv 3x3s1, 384 / ReLU	112M
884K	Conv 3x3s1, 384 / ReLU	149M
	Max Pool 3x3s2	
	Local Response Norm	
307K	Conv 5x5s1, 256 / ReLU	223M
	Max Pool 3x3s2	
	Local Response Norm	
35K	Conv 11x11s4, 96 / ReLU	105M

AlexNet diagram (simplified)

Input size

227 x 227 x 3



Conv 1

$11 \times 11 \times 3$
Stride 4
96 filters

Conv 2

$5 \times 5 \times 96$
Stride 1
256 filters

Conv 3

$3 \times 3 \times 256$
Stride 1
384 filters

Conv 4

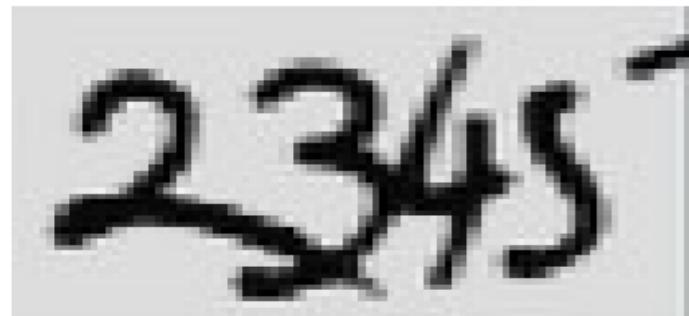
$3 \times 3 \times 192$
Stride 1
384 filters

Conv 4

$3 \times 3 \times 192$
Stride 1
256 filters

CONV NETS: EXAMPLES

- OCR / House number & Traffic sign classification



Ciresan et al. "MCDNN for image classification" CVPR 2012

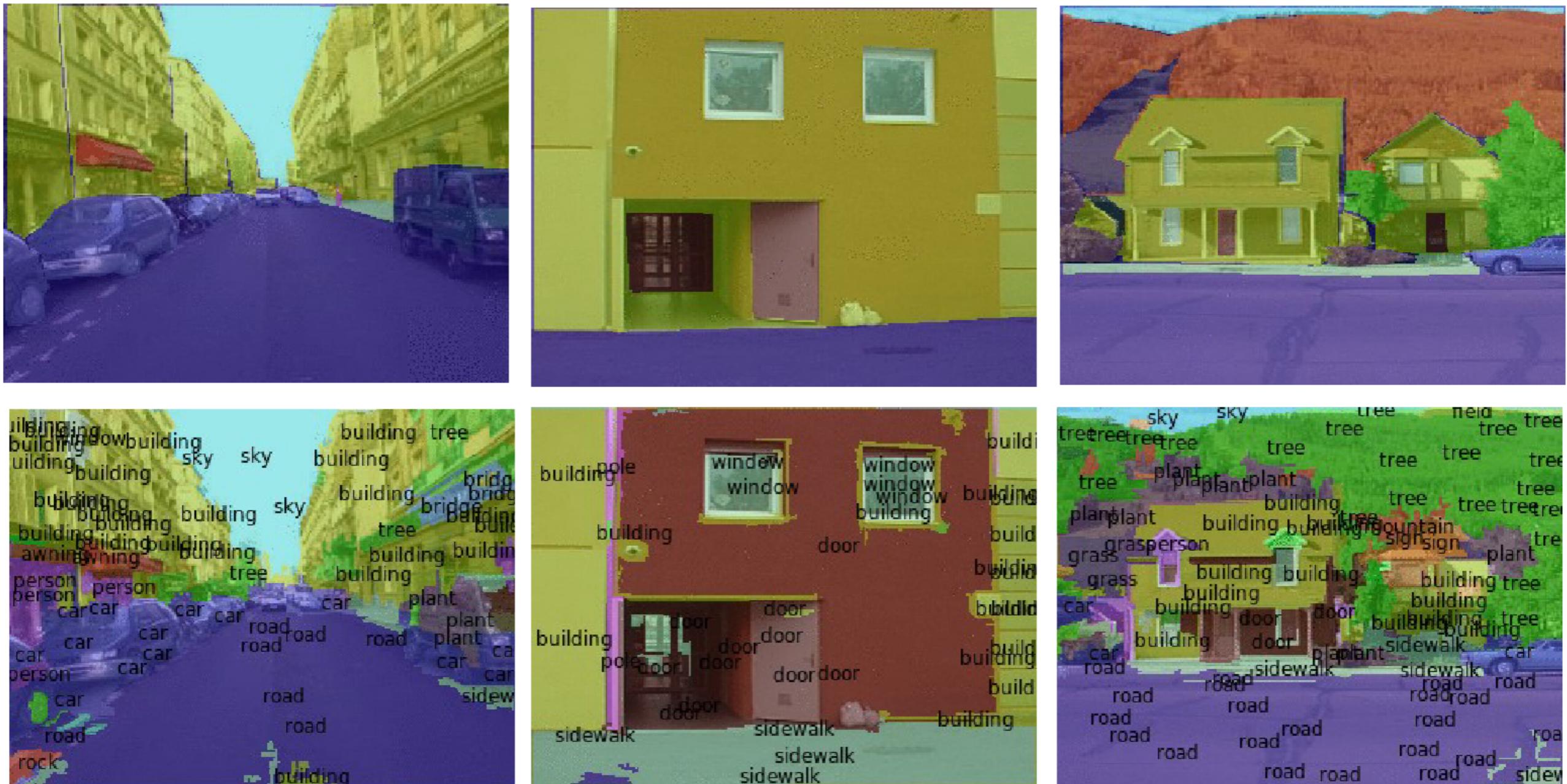
Wan et al. "Regularization of neural networks using dropconnect" ICML 2013

82

Jaderberg et al. "Synthetic data and ANN for natural scene text recognition" arXiv 2014

CONV NETS: EXAMPLES

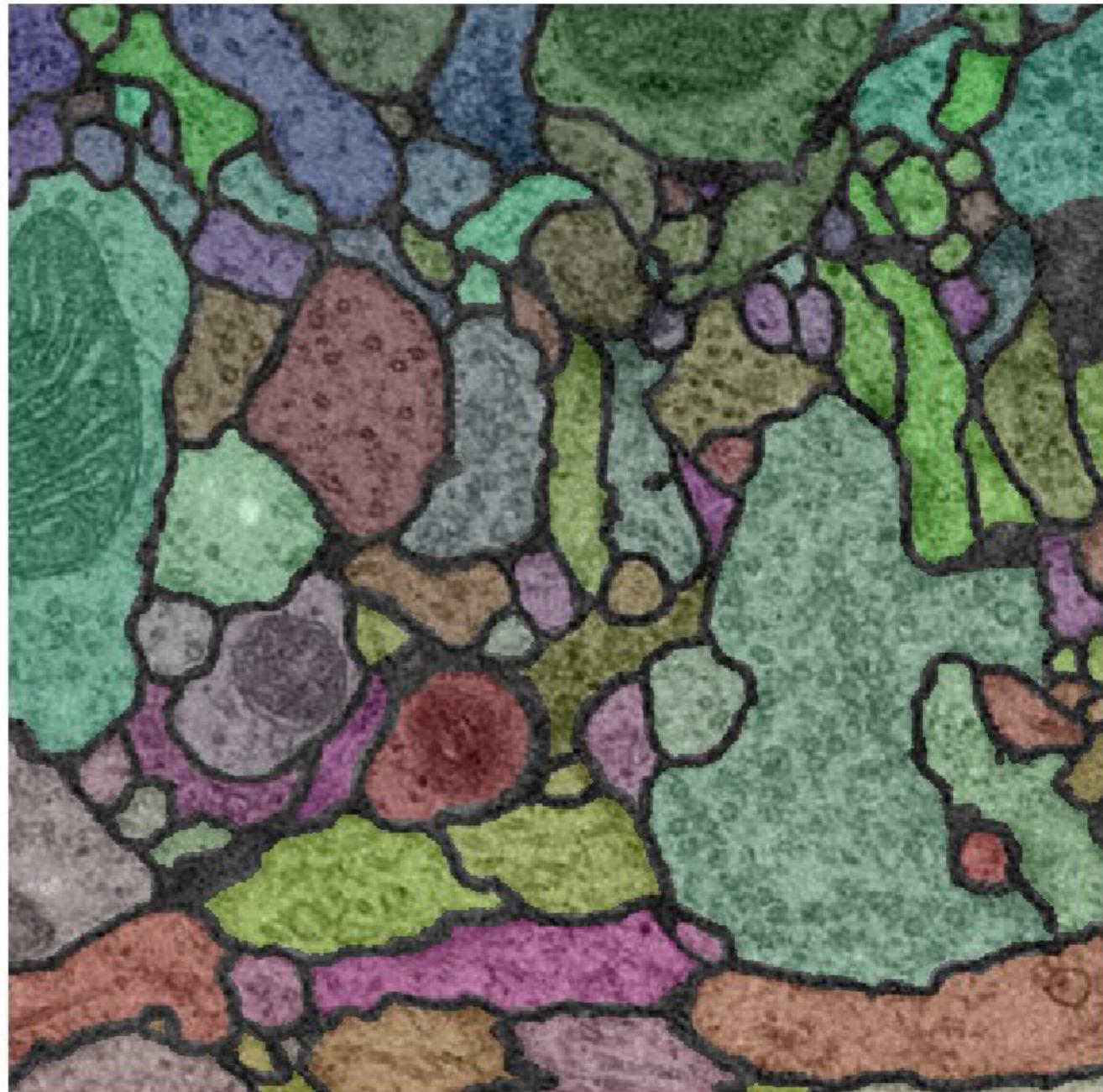
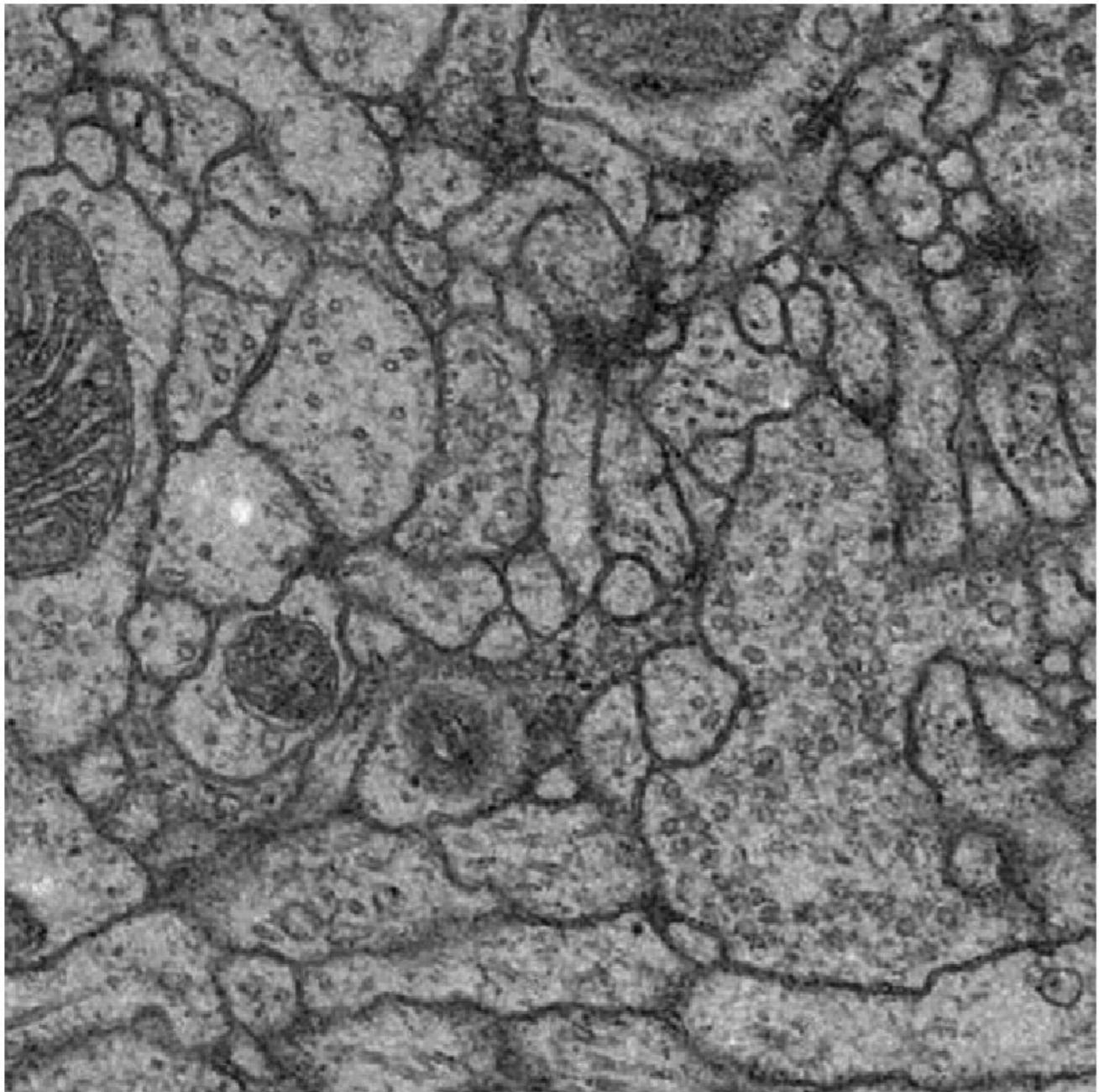
- Scene Parsing



Farabet et al. "Learning hierarchical features for scene labeling" PAMI 2013
Pinheiro et al. "Recurrent CNN for scene parsing" arxiv 2013

CONV NETS: EXAMPLES

- Segmentation 3D volumetric images



Ciresan et al. "DNN segment neuronal membranes..." NIPS 2012

Turaga et al. "Maximin learning of image segmentation" NIPS 2009

CONV NETS: EXAMPLES

- Action recognition from videos



Taylor et al. "Convolutional learning of spatio-temporal features" ECCV 2010

Karpathy et al. "Large-scale video classification with CNNs" CVPR 2014

Simonyan et al. "Two-stream CNNs for action recognition in videos" arXiv 2014

CONV NETS: EXAMPLES

- Robotics



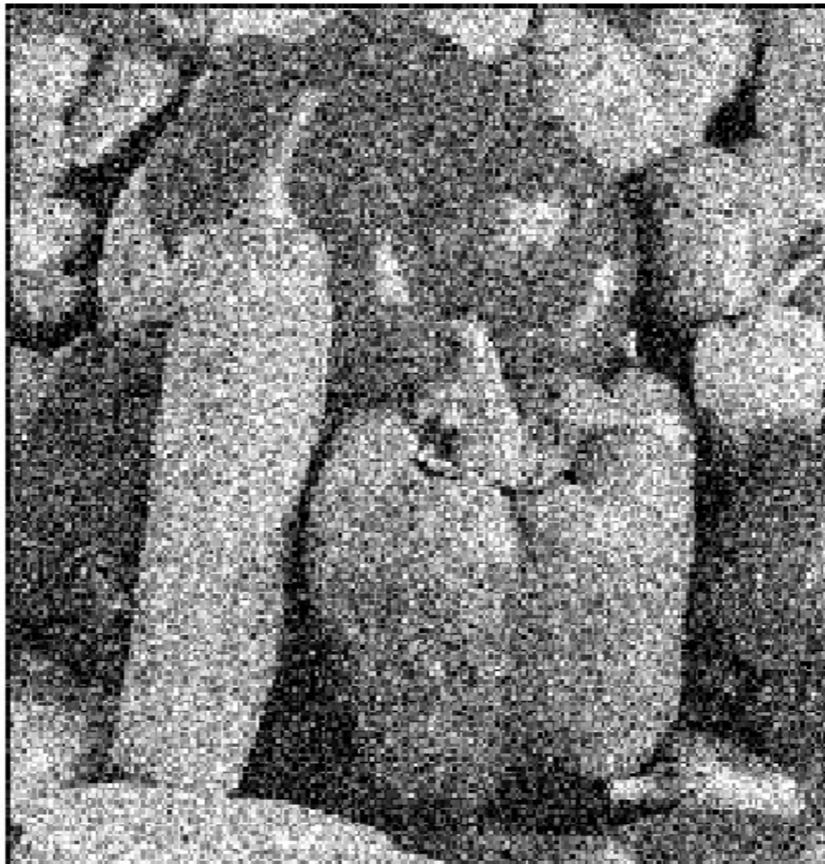
CONV NETS: EXAMPLES

- Denoising

original



noised

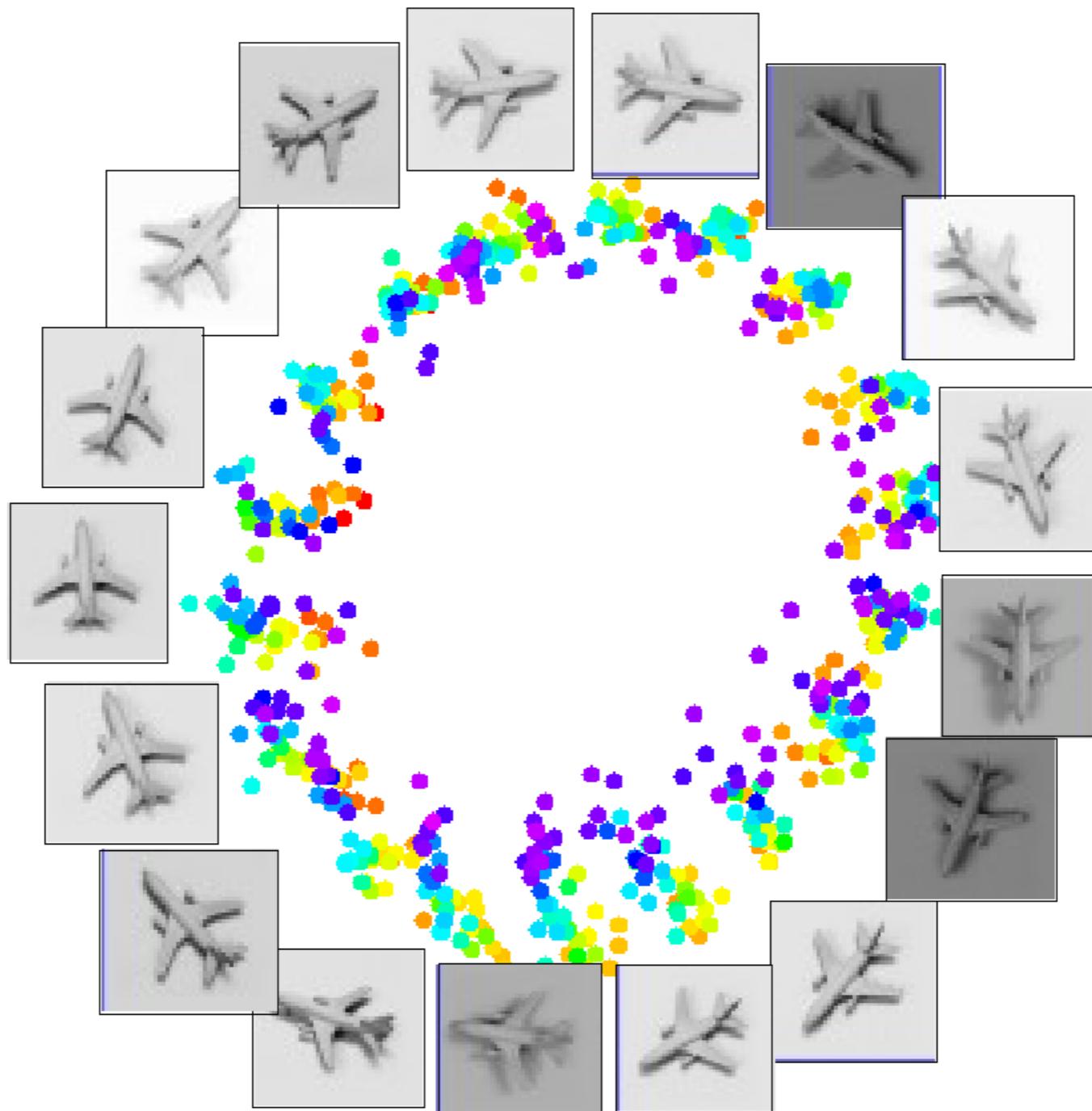


denoised



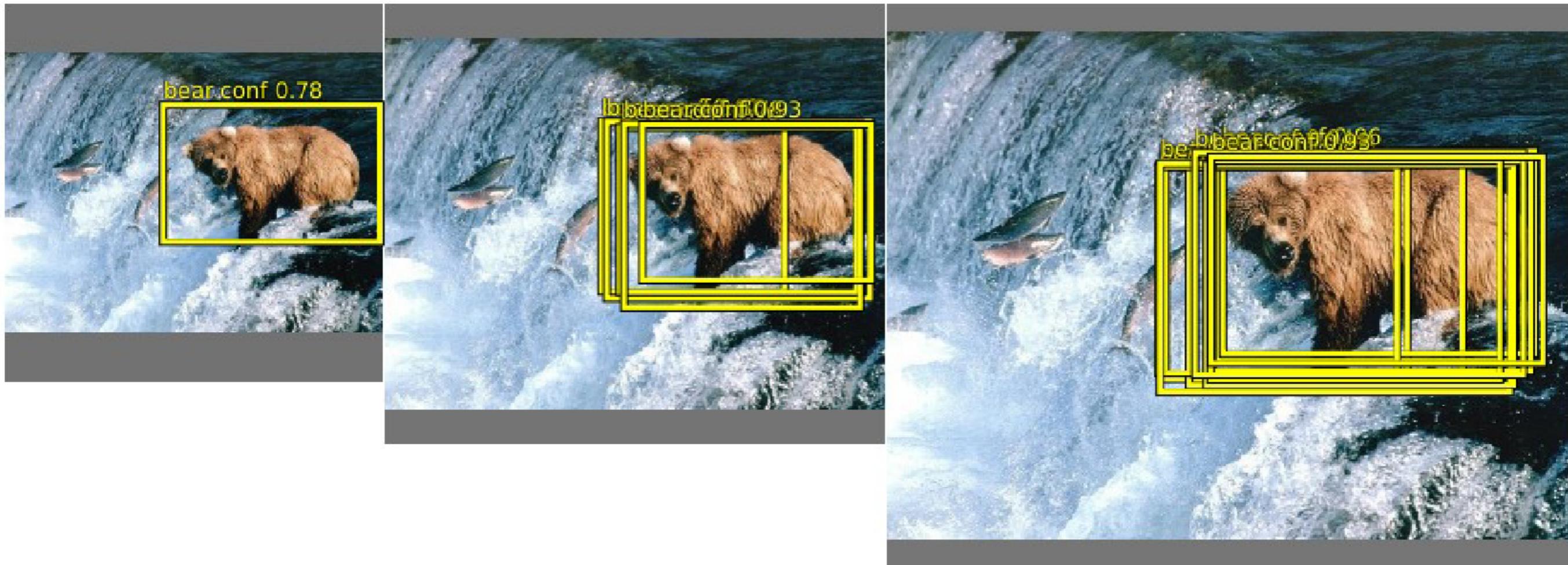
CONV NETS: EXAMPLES

- Dimensionality reduction / learning embeddings



CONV NETS: EXAMPLES

- Object detection



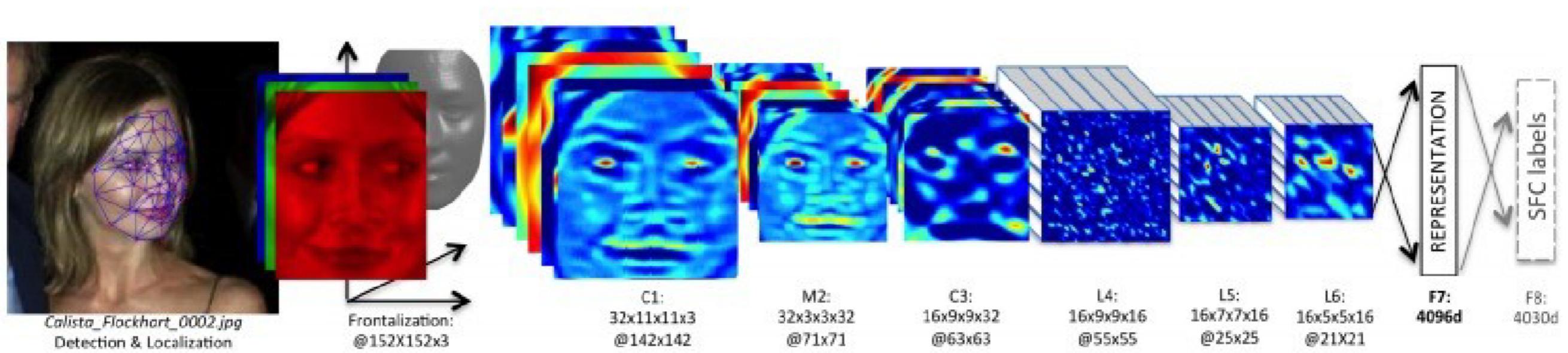
Sermanet et al. “OverFeat: Integrated recognition, localization, ...” arxiv 2013

Girshick et al. “Rich feature hierarchies for accurate object detection...” arxiv 2013 91

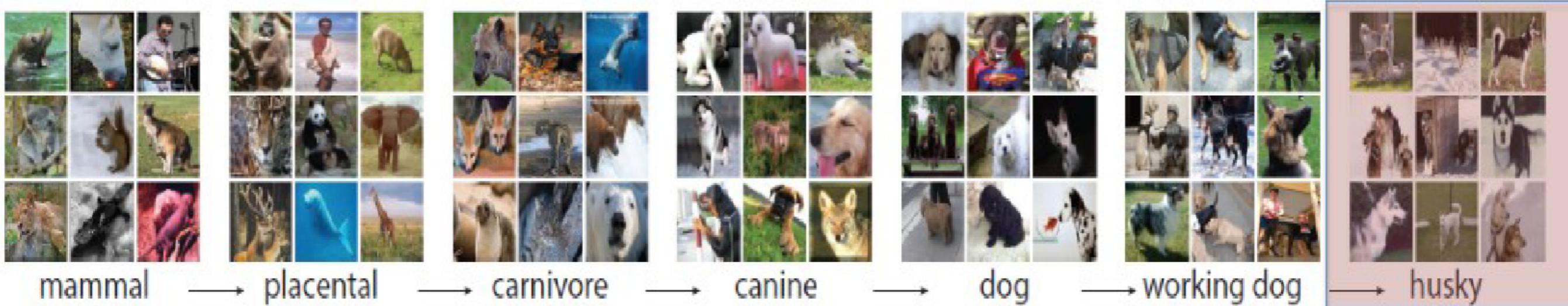
Szegedy et al. “DNN for object detection” NIPS 2013

CONV NETS: EXAMPLES

- Face Verification & Identification



Dataset: ImageNet 2012



- S: (n) [Eskimo dog](#), [husky](#) (breed of heavy-coated Arctic sled dog)
 - *direct hypernym / inherited hypernym / sister term*
- S: (n) [working dog](#) (any of several breeds of usually large powerful dogs bred to work as draft animals and guard and guide dogs)
- S: (n) [dog](#), [domestic dog](#), [Canis familiaris](#) (a member of the genus *Canis* (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) "the dog barked all night"
 - S: (n) [canine](#), [canid](#) (any of various fissiped mammals with nonretractile claws and typically long muzzles)
 - S: (n) [carnivore](#) (a terrestrial or aquatic flesh-eating mammal) "terrestrial carnivores have four or five clawed digits on each limb"
 - S: (n) [placental](#), [placental mammal](#), [eutherian](#), [eutherian mammal](#) (mammals having a placenta; all mammals except monotremes and marsupials)
 - S: (n) [mammal](#), [mammalian](#) (any warm-blooded vertebrate having the skin more or less covered with hair; young are born alive except for the small subclass of monotremes and nourished with milk)
 - S: (n) [vertebrate](#), [craniate](#) (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain enclosed in a skull or cranium)
 - S: (n) [chordate](#) (any animal of the phylum Chordata having a notochord or spinal column)
 - S: (n) [animal](#), [animate being](#), [beast](#), [brute](#), [creature](#), [fauna](#) (a living organism characterized by voluntary movement)
 - S: (n) [organism](#), [being](#) (a living thing that has (or can develop) the ability to act or function independently)
 - S: (n) [living thing](#), [animate thing](#) (a living (or once living) entity)
 - S: (n) [whole](#), [unit](#) (an assemblage of parts that is regarded as a single entity) "how big is that part compared to the whole?"; "the team is a unit"
 - S: (n) [object](#), [physical object](#) (a tangible and visible entity; an entity that can cast a shadow) "it was full of rackets, balls and other objects"
 - S: (n) [physical entity](#) (an entity that has physical existence)
 - S: (n) [entity](#) (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))



mite

mite
black widow
cockroach
tick
starfish

container ship

container ship
lifeboat
amphibian
fireboat
drilling platform

motor scooter

motor scooter
go-kart
moped
bumper car
golfcart

leopard

leopard
jaguar
cheetah
snow leopard
Egyptian cat



grille

convertible
grille
pickup
beach wagon
fire engine

mushroom

agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers

cherry

dalmatian
grape
elderberry
ffordshire bullterrier
currant

Madagascar cat

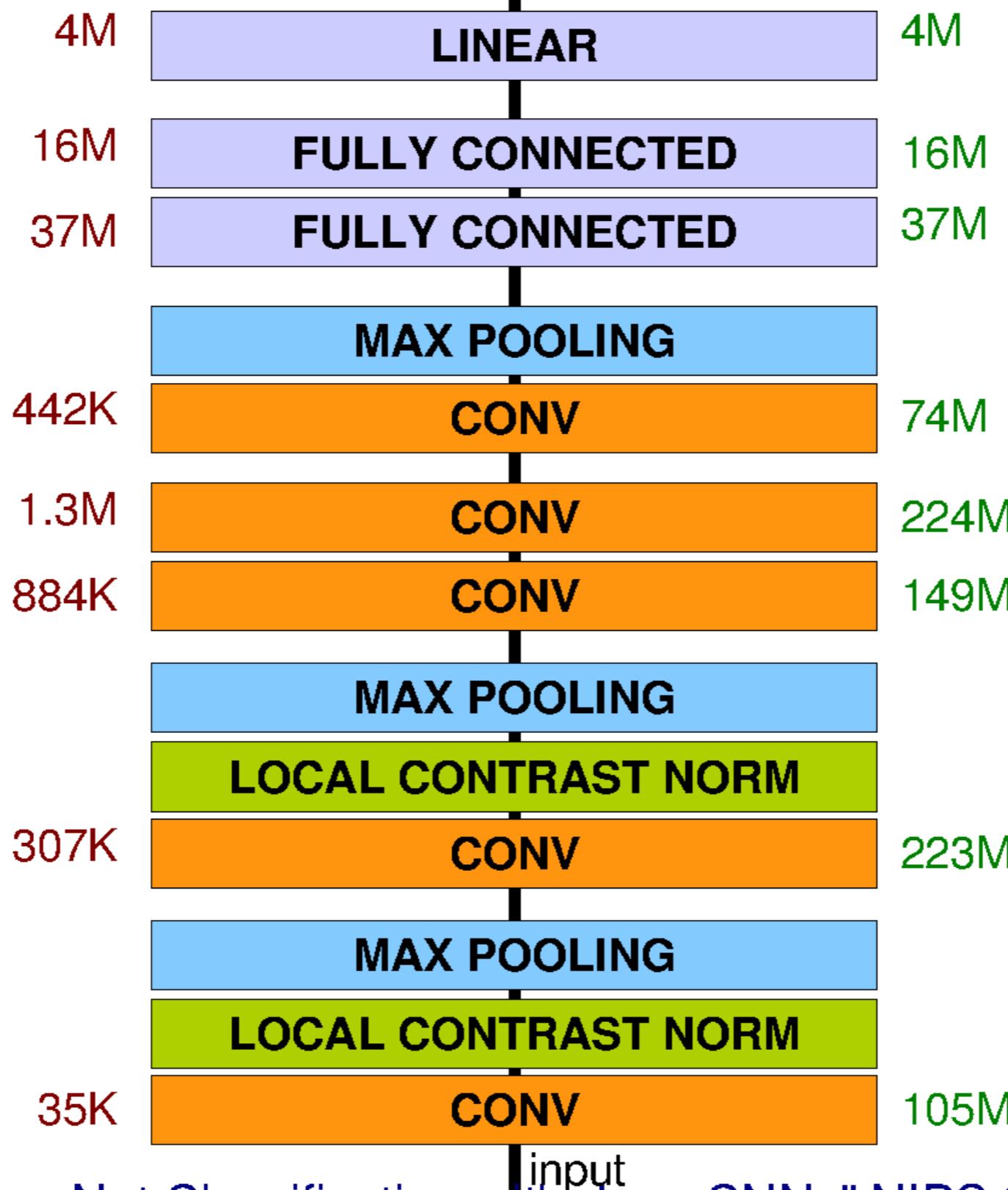
squirrel monkey
spider monkey
titi
indri
howler monkey

Architecture for Classification

Total nr. params: 60M

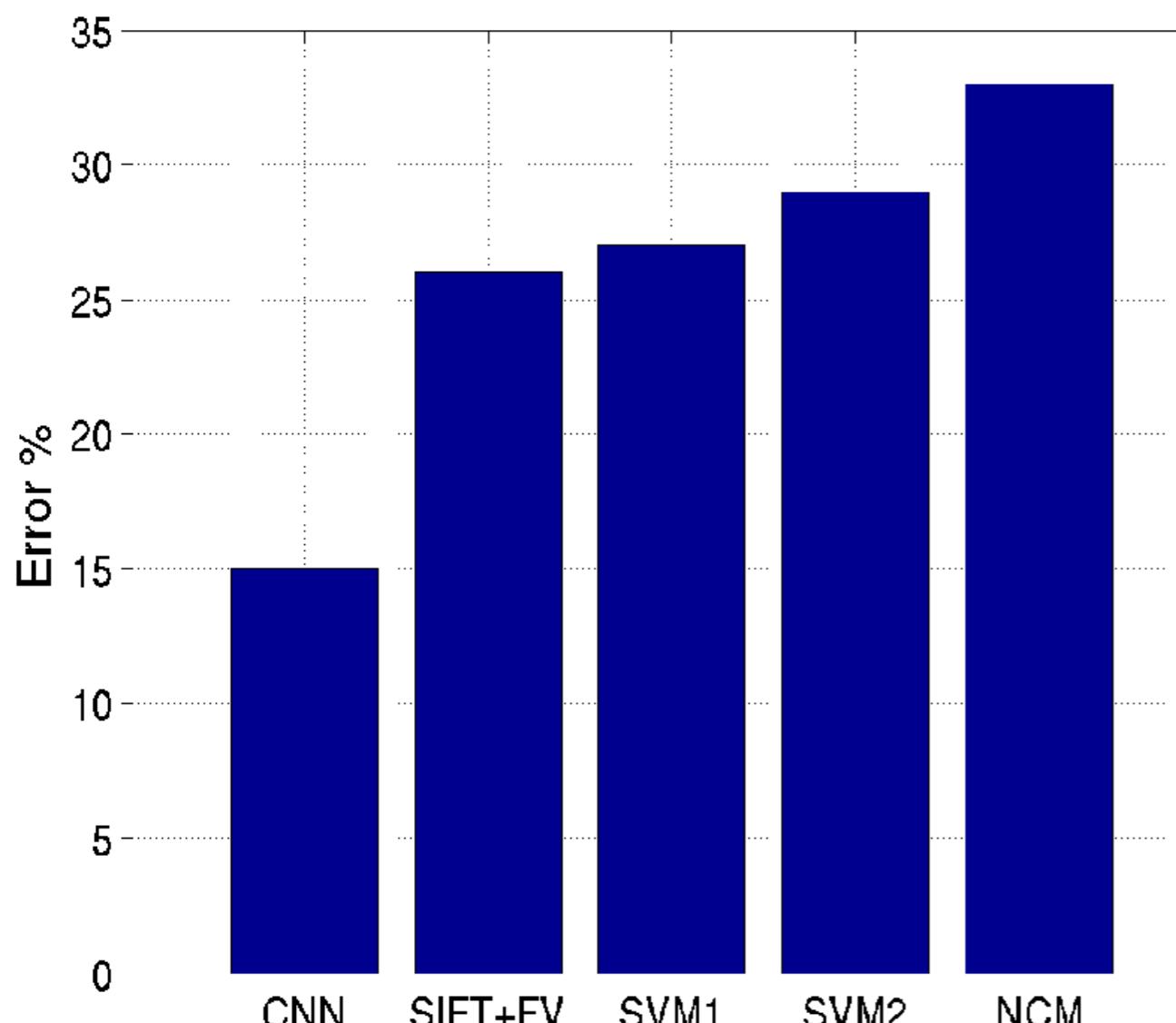
category
prediction

Total nr. flops: 832M

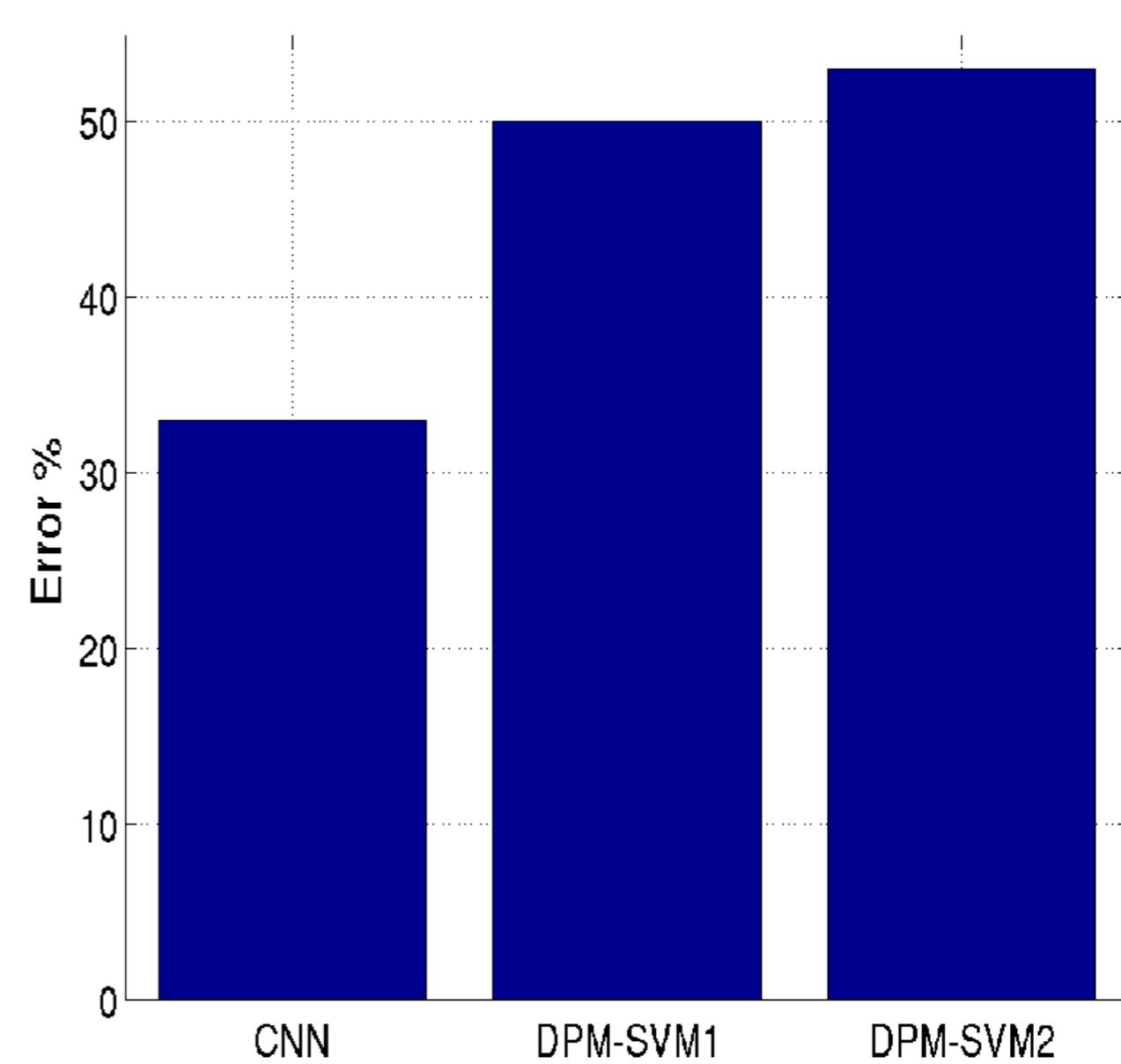


Results: ILSVRC 2012

TASK 1 - CLASSIFICATION



TASK 2 - DETECTION



Wait, why isn't it called a correlation neural network?

It could be.

Deep learning libraries actually implement correlation.

Correlation relates to convolution via a 180deg rotation of the kernel. When we *learn* kernels, we could easily learn them flipped.

Associative property of convolution ends up not being important to our application, so we just ignore it.

[p.323, Goodfellow]

More ConvNet explanations

- <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>