Convolutional Neural Networks

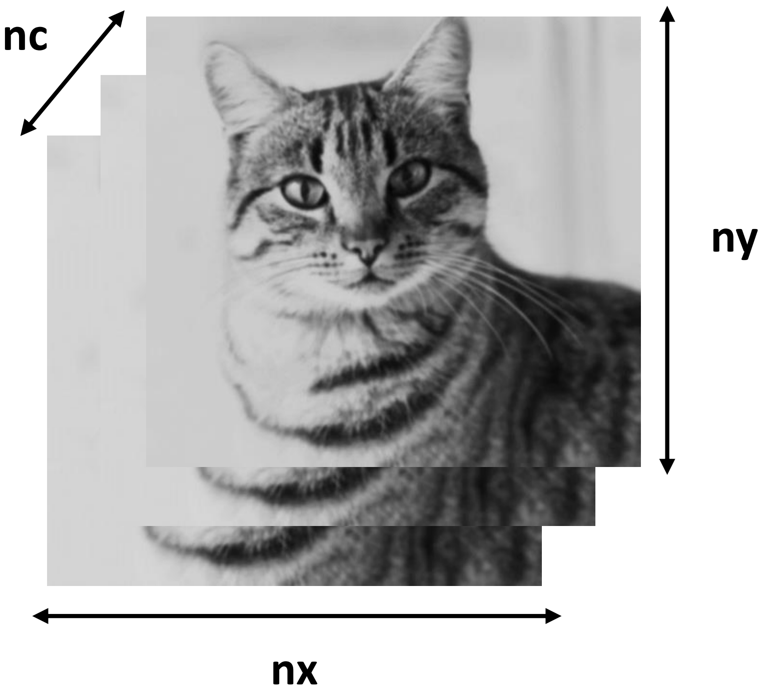
QSB 2018: Learning and Artificial intelligence – Tutorial session 3

Giulio Matteucci

Neural network architectures for computer **vision** tasks

Images are **high dimensional**!

representing them with pixel intensity **values** $\longrightarrow$ input $\mathbf{x} \in \boldsymbol{R}^n$ where **n** = nx x ny x nc … is large!



number of parameters (weights) grows quadratically with resolution

fully connected networks do not scale well to real world computer vision problems!

Can we exploit our prior knowledge about the the visual world to design a better architecture for vision?

Start from two considerations about natural visual input ….

1 visual **features are local** → natural images are made of **sparse**, **local** independent **components** …

because … visual **scenes** are **made of** (often) **repeated elements**

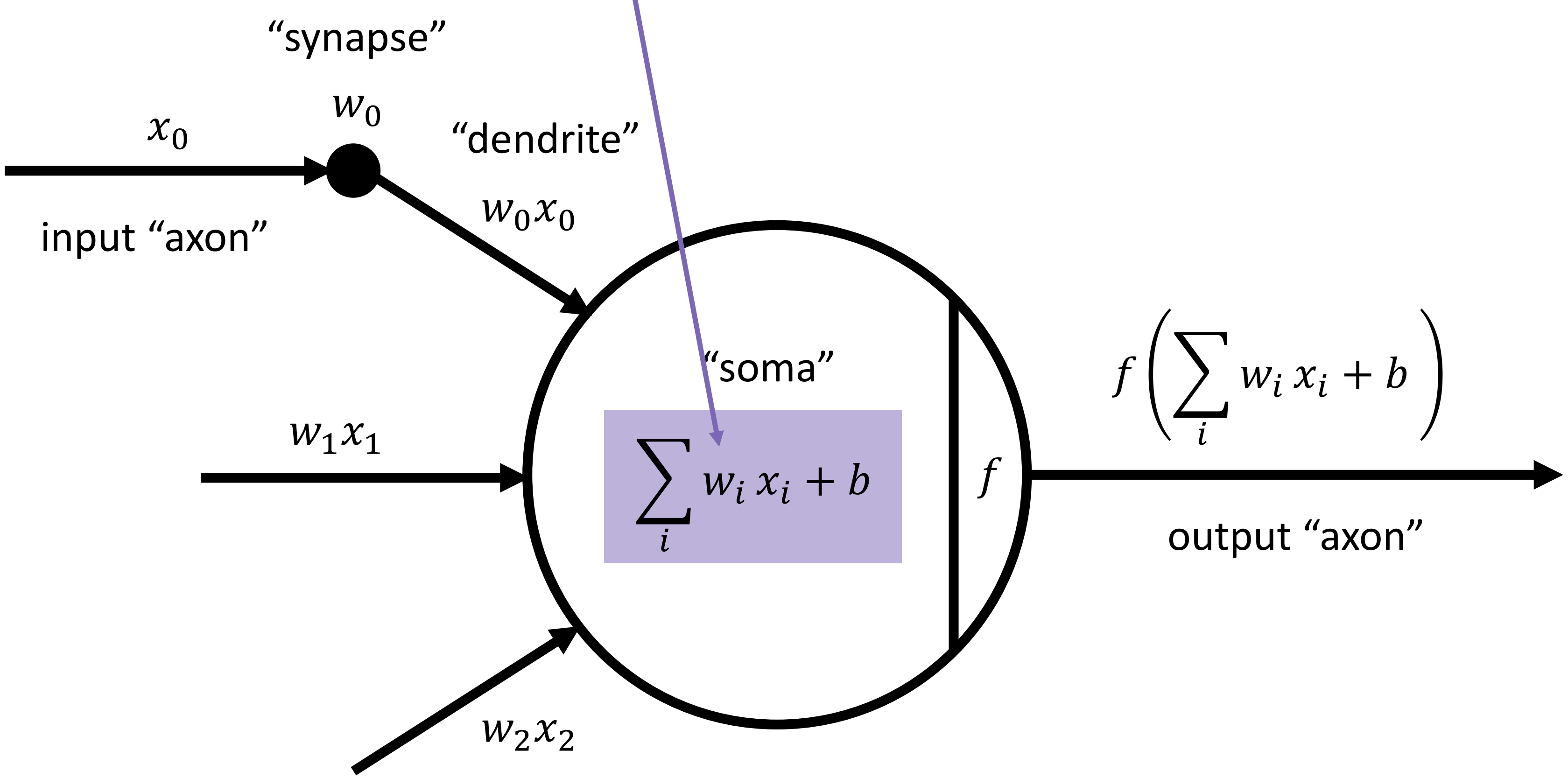
2 visual **features can show up everywhere** → natural image statistics is (approximately) **stationary across** visual **space**

because … visual objects undergo **identity preserving transformations** (e.g. translation)

Hyvärinen et al. "Natural Image Statistics", 2009 neurons as filters....

dot product measures similarity

neurons search for the pattern stored in their weighs in input

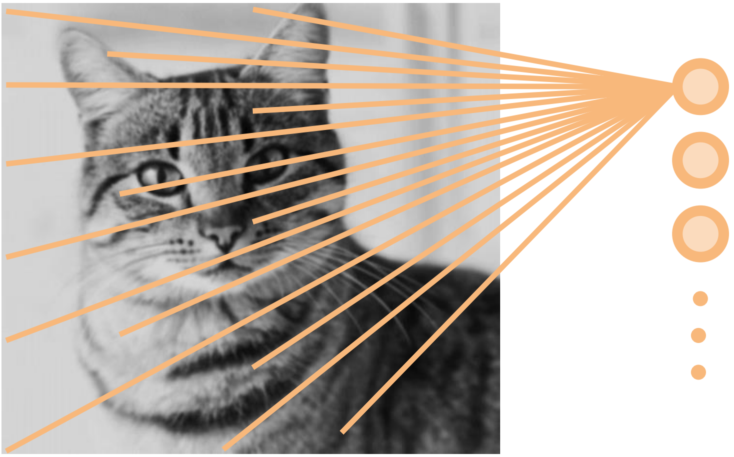


... **when input** vector is **similar** enough **to weight** vector: → **response** = preferred **feature is detected**

1 **learn** small **localized filters**:

... to do so let's keep spatial structure (i.e. do not flatten input)

fully connected units



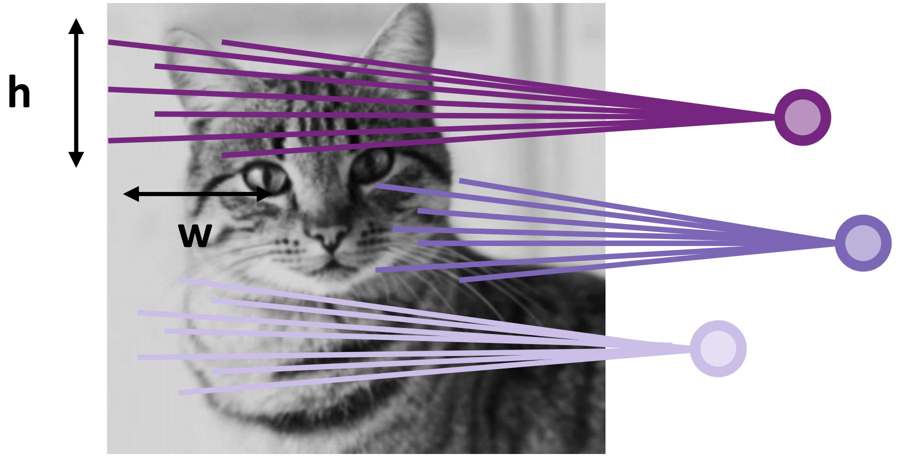
learning **global filters** for **local features** ✗

nx x ny **parameters** per hidden unit

Eg: nx=ny=200 ---> **40000** parameters per unit

Costly and inefficient!

locally connected units



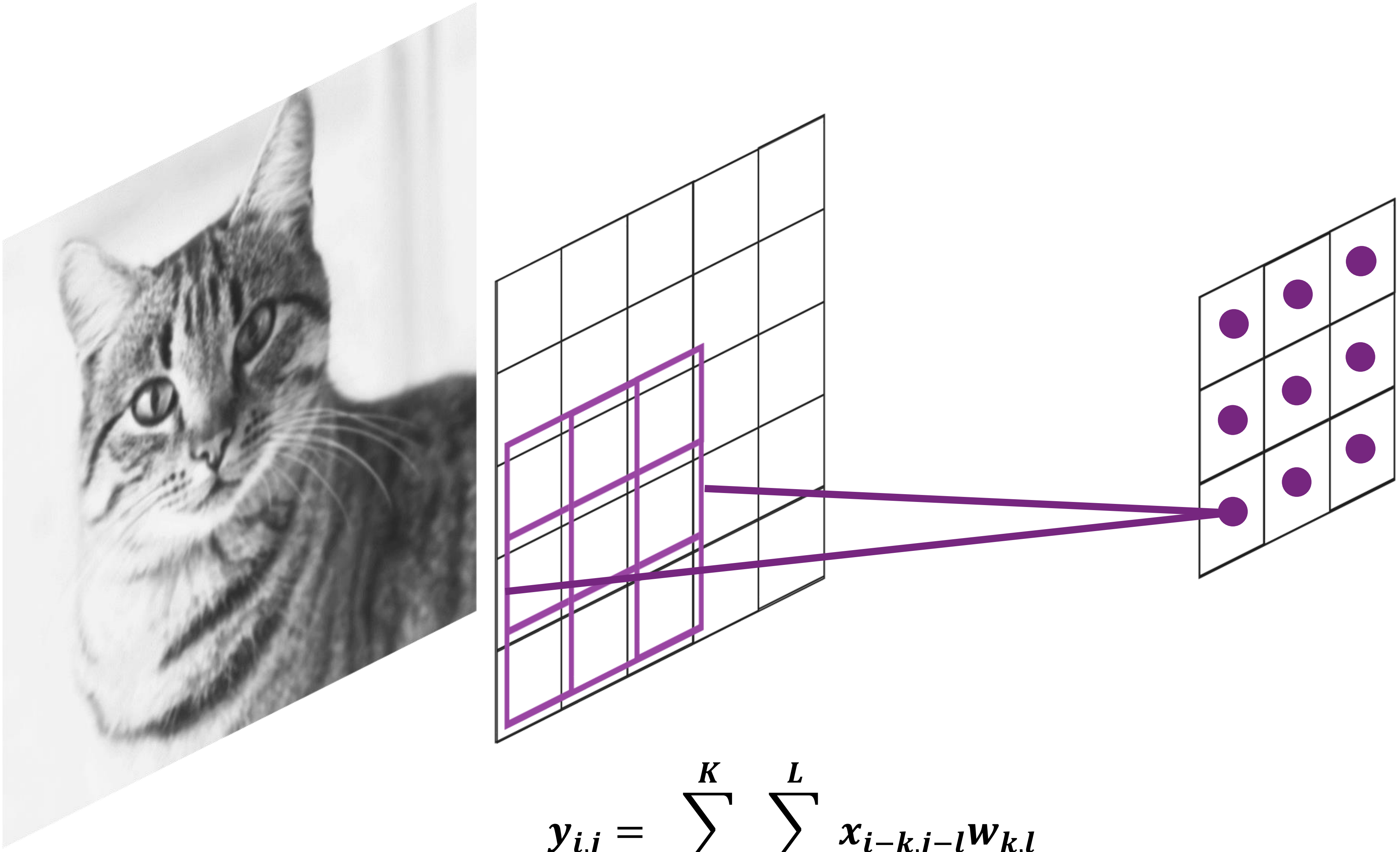
learning **local filters** for **local features** ✓

h x w **parameters** per hidden unit

Eg: h=w=4 ---> **16** parameters per unit

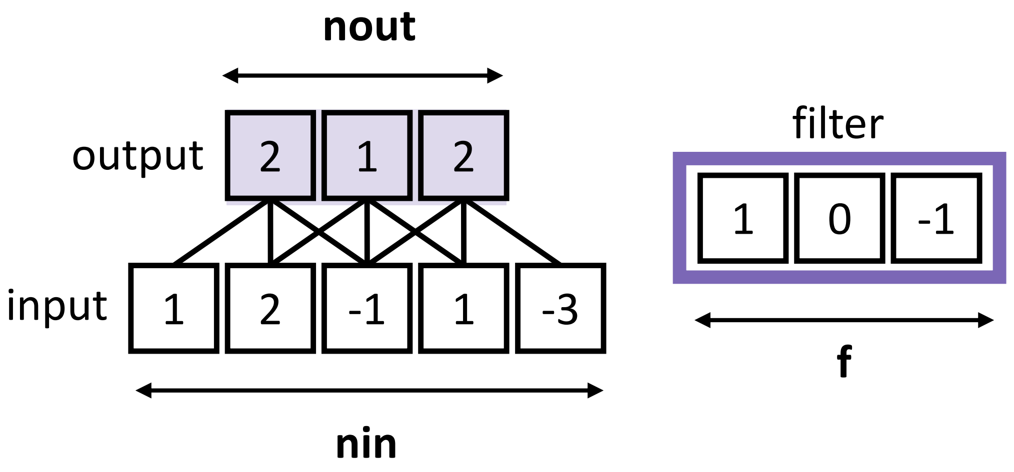
Cheap and efficient!

2 **reuse localized filters** unaltered **across** different part of **image** → **convolution operation**



$$y_{i,j} = \sum_{k=-K}^{K} \sum_{l=-L}^{L} x_{i-k,j-l} w_{k,l}$$

applying convolution
output naturally shrinks

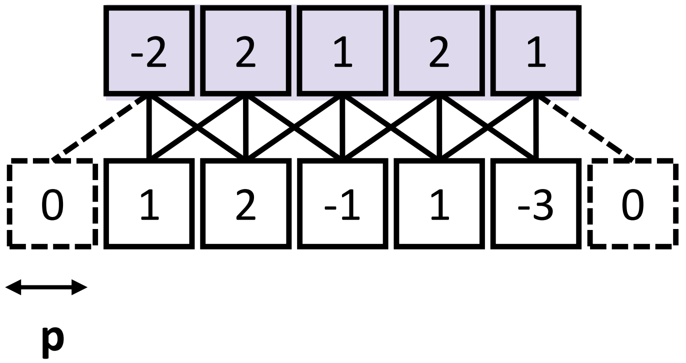


nout = (nin – f) +1

We can avoid this …

padded convolution

adding 0s at the
input border



nout = (nin + 2p – f) +1

p=0 | **nin≠nout** → called: “***valid***” convolution

p | **nin=nout** → called: “***same***” convolution when cascading multiple convolution operations useful to introduce: → **receptive field (RF)**

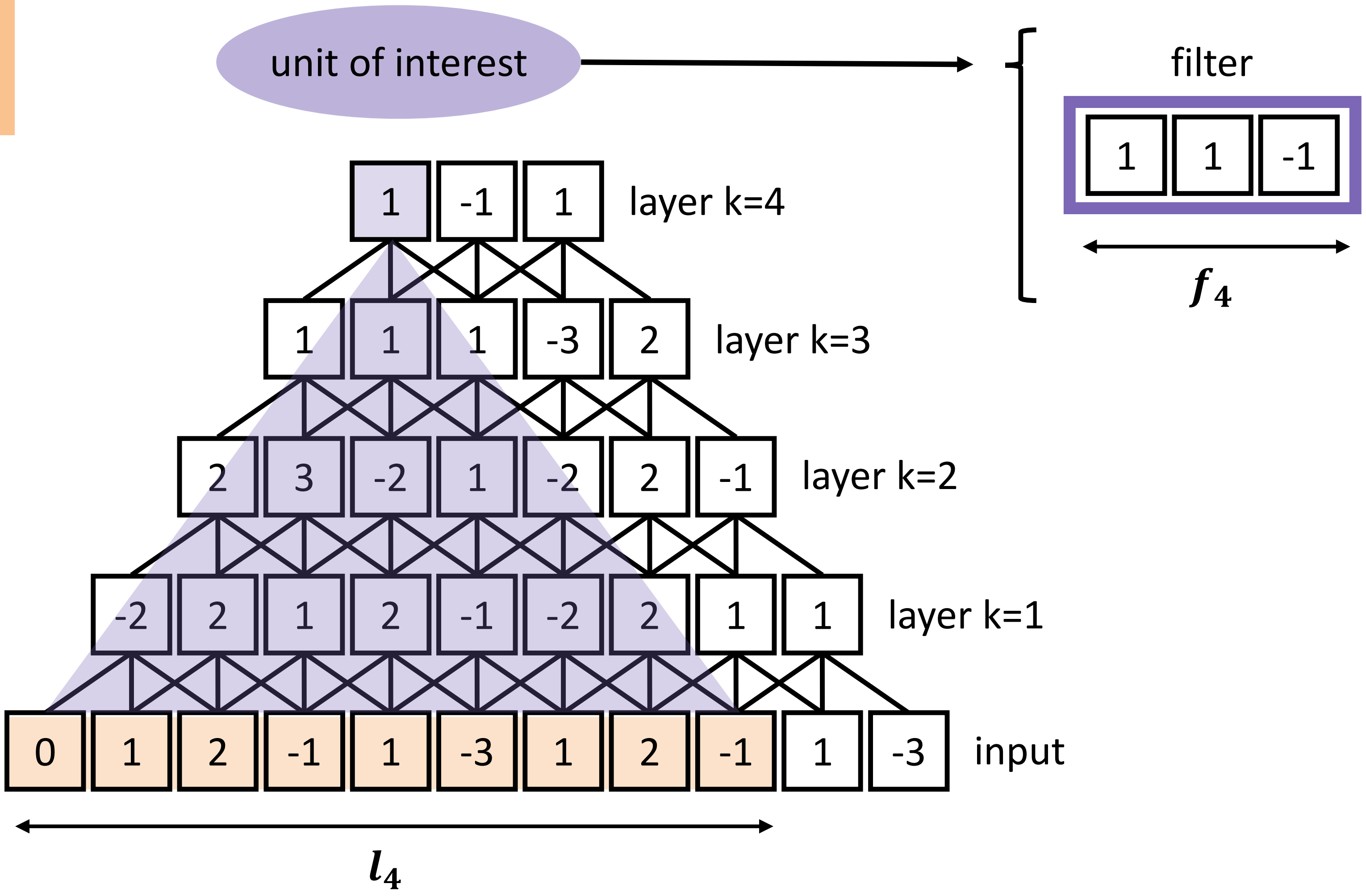
region of the **input space from which** a given neuron **receives information** from

- equal to filter size in first layer
- **grow** as **of f-1 each** next **layer**

for the kth one, recursively:

$$l_k = l_{k-1} + (f_k - 1)$$

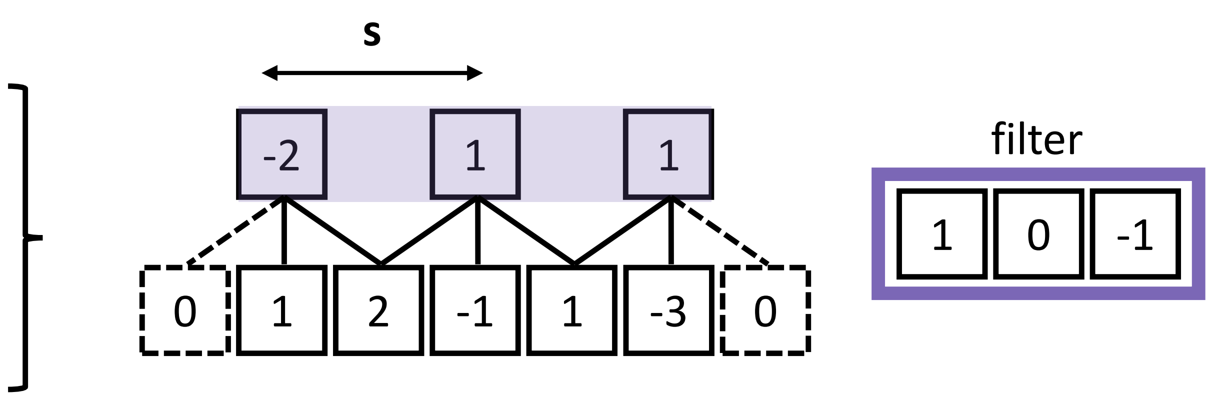
with: $l_1 = f_k$



modern CNNs use very small filters (e.g. 3x3) → to develop selectivity for meaningful pattern **we need larger RF**!

we may want to make them grow faster …

strided convolution → change the "step" of filter displacement



in this way **RF size can grow faster**:

$$l_k = l_{k-1} + (f_k - 1) \prod_{i=1}^{k-1} s_i$$

strided convolution also **act as a downsampling** greatly reducing output size

…. considering stride (and padding) the output size will be:

$$\mathbf{nout} = \frac{\mathbf{(nin + 2p - f) + 1}}{s}$$

with

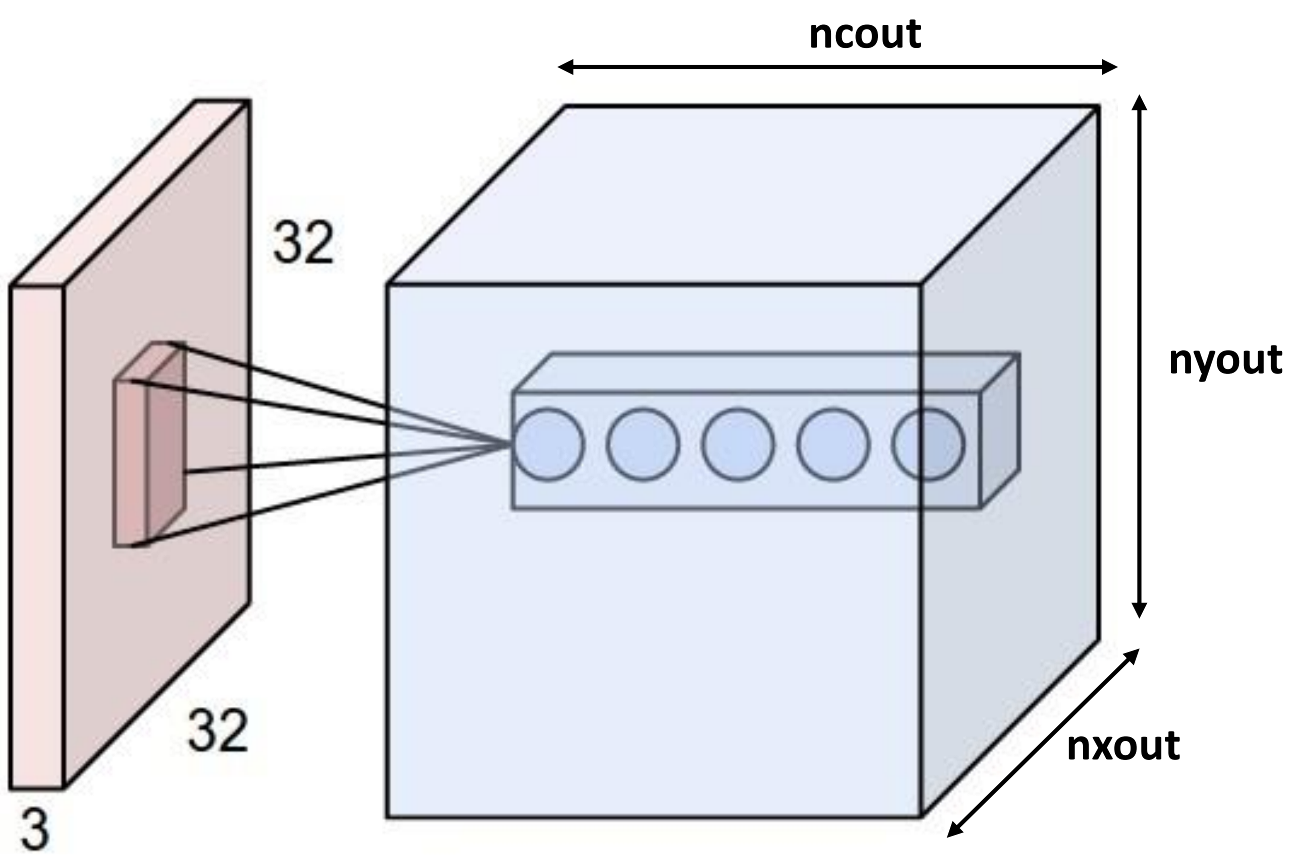
s = stride

p = padding

f = filter dimension

we are learning **multiple filters**, acting **on all input channels** together… filter output will form a ***"feature map"***

convolutional layer → **stack** different **feature maps on the third dimension** (as different channels):



size of output volume:

$$\textbf{nxout} = \frac{(\textbf{nxin} + 2\textbf{p} - \textbf{fx}) + 1}{s}$$

$$\textbf{nyout} = \frac{(\textbf{nyin} + 2\textbf{p} - \textbf{fy}) + 1}{s}$$

ncout = nf

with

nf = number of filters

fx,fy= filters size

nxin,nyin= input size

s = stride

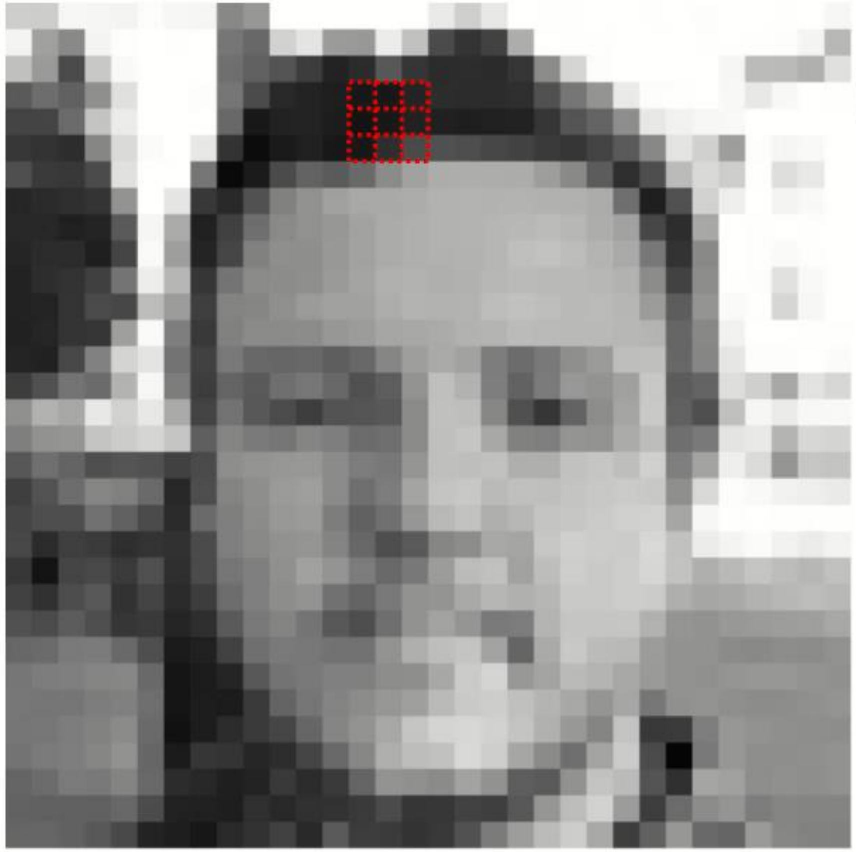
p = padding

Karpathy 2016

example of convolution of an **edge detecting filter**:

Sobel filter

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

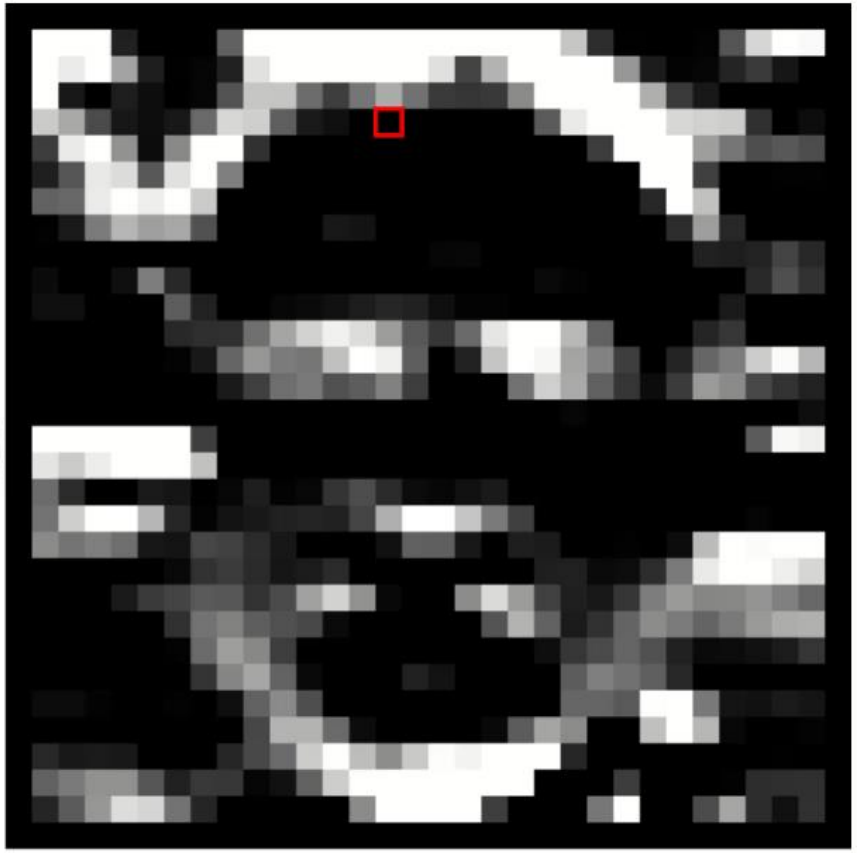


input image

$$\left(24 \times 1 + 25 \times 2 + 36 \times 1 + 20 \times 0 + 25 \times 0 + 34 \times 0 + 24 \times -1 + 46 \times -2 + 75 \times -1\right) = -81$$

kernel:

top sobel ▾



output image

from setosa.io

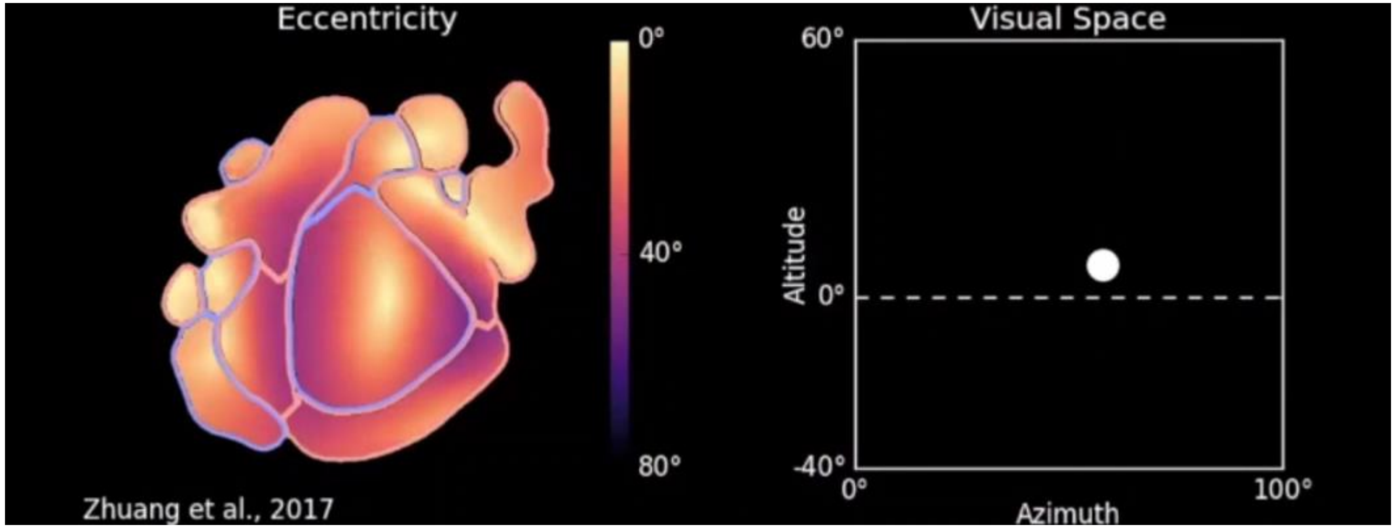
solving FCN **bad scaling**:

learning convolutional filters to enforce …

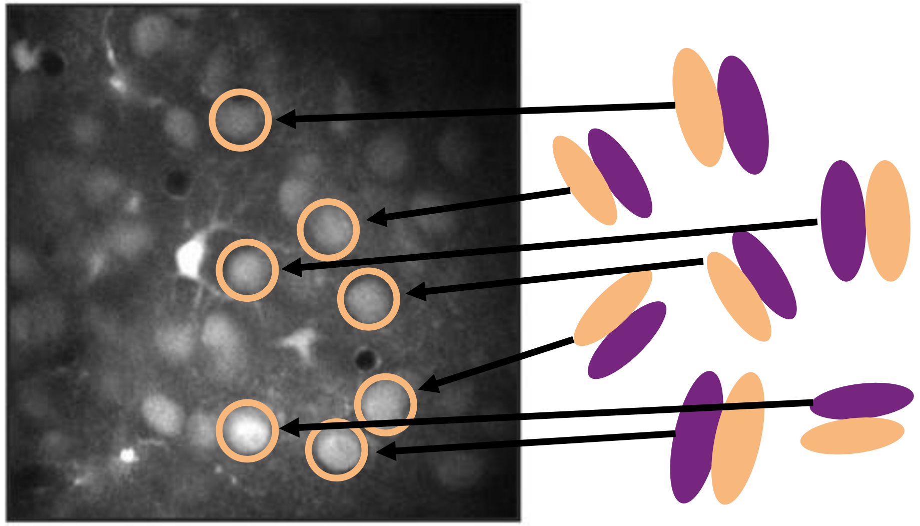
1. **sparsity of connections** → neuron connected to a small region of input only (***localized receptive field***)
2. **parameter sharing** → whole input space tiled with RF re-using the same parameters (***feature maps***)

reminiscent of how visual information is represented across the **brain** surface

retinotopic maps



localized feature detectors

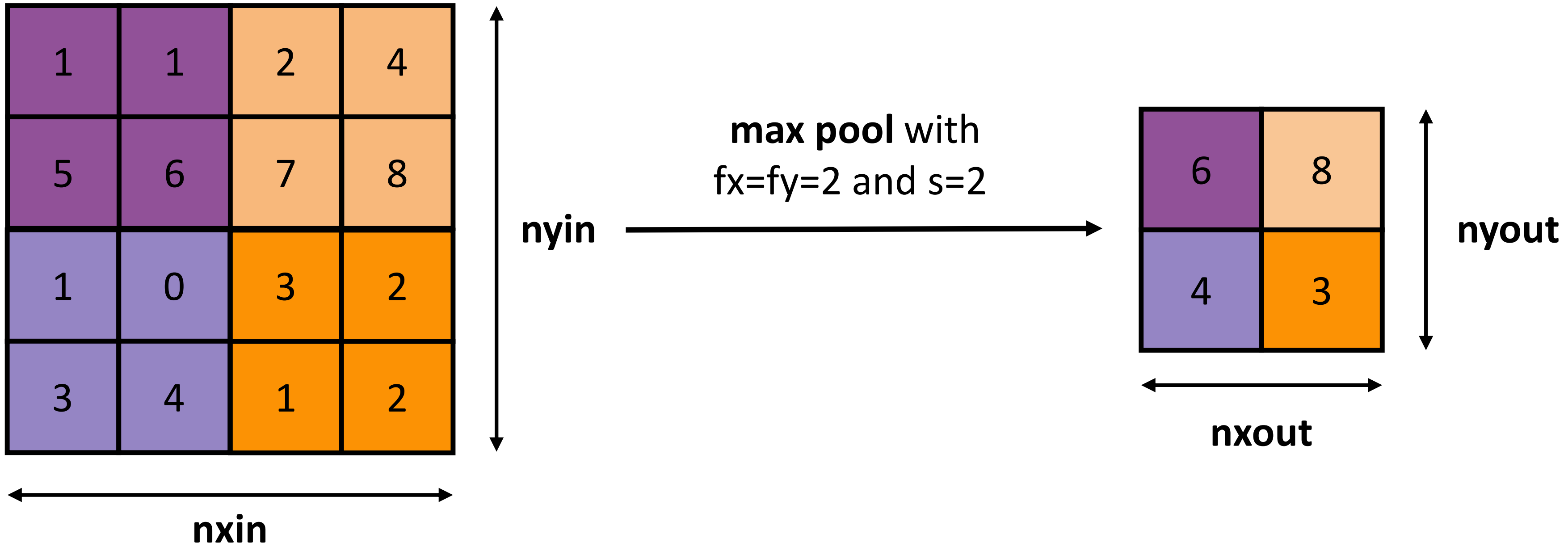


thinking to 2 ... we may want to **hardwire** some amount of **translation tolerance** in our network!

pooling operation

convolutionally **apply** a "**max**" **filter** to the input ⟷ nonlinear blur and down sampling "replacing" a subregions with their max value

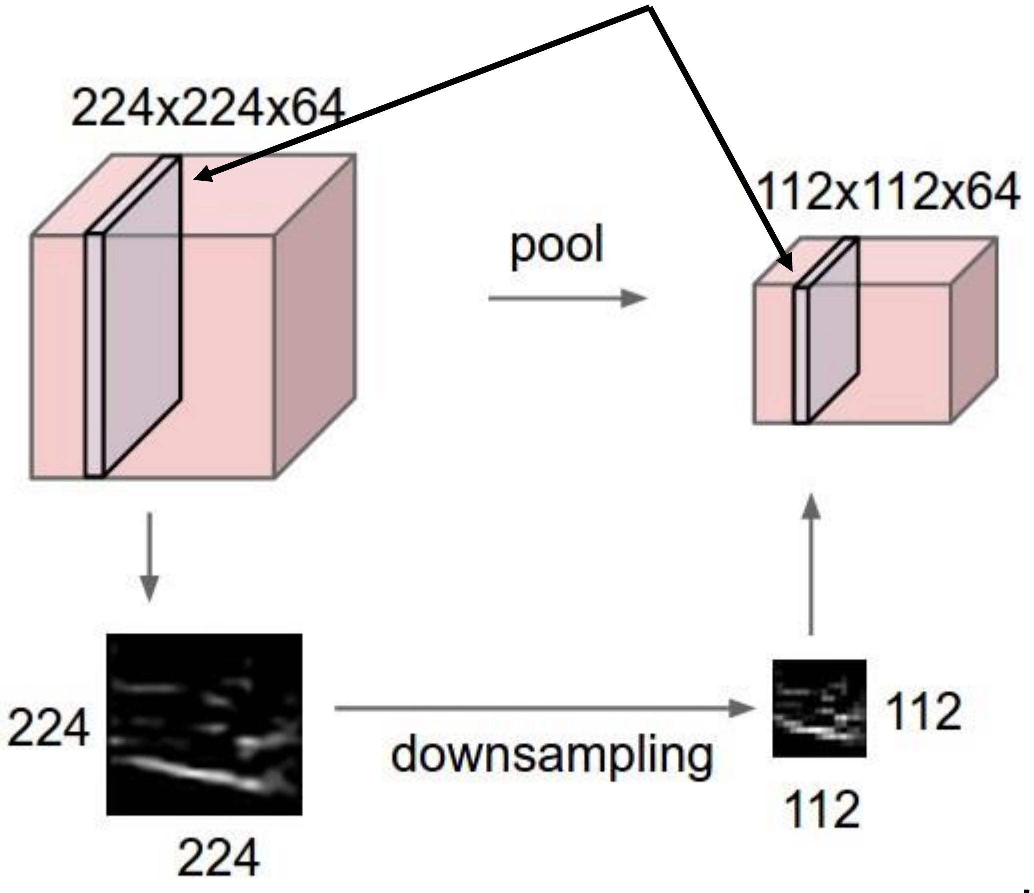
$$y_{i,j} = max\{pool_{\,i,j}\}$$ with $$pool_{\,i,j} = \left[x_{i-k,j-l}\right]$$ **with k=1,..fy and l=1,..fx**



usually done with stride s=fx=fy to have **non-overlapping subregions**

pooling operation will be applied to convolutional layer volumes **independently to each feature map** …

individual feature map



number of **parameters reduced** by 75%

- **less** computationally **expansive**
- **less** likely to **overfit**

dimension of output volume:

$$\text{nyout} = \frac{(\text{nyin} + 2p - \text{fx}) + 1}{s} \qquad \text{nxout} = \frac{(\text{nxin} + 2p - \text{fy}) + 1}{s}$$

ncout = nf   but since usually p=0, s=2 and fx=fy=2 …

$$\mathbf{nyout} = \frac{\mathbf{nyin}}{\mathbf{2}} \qquad \mathbf{nxout} = \frac{\mathbf{nxin}}{\mathbf{2}}$$

also for RF size calculation old formula still holds

Karpathy 2016

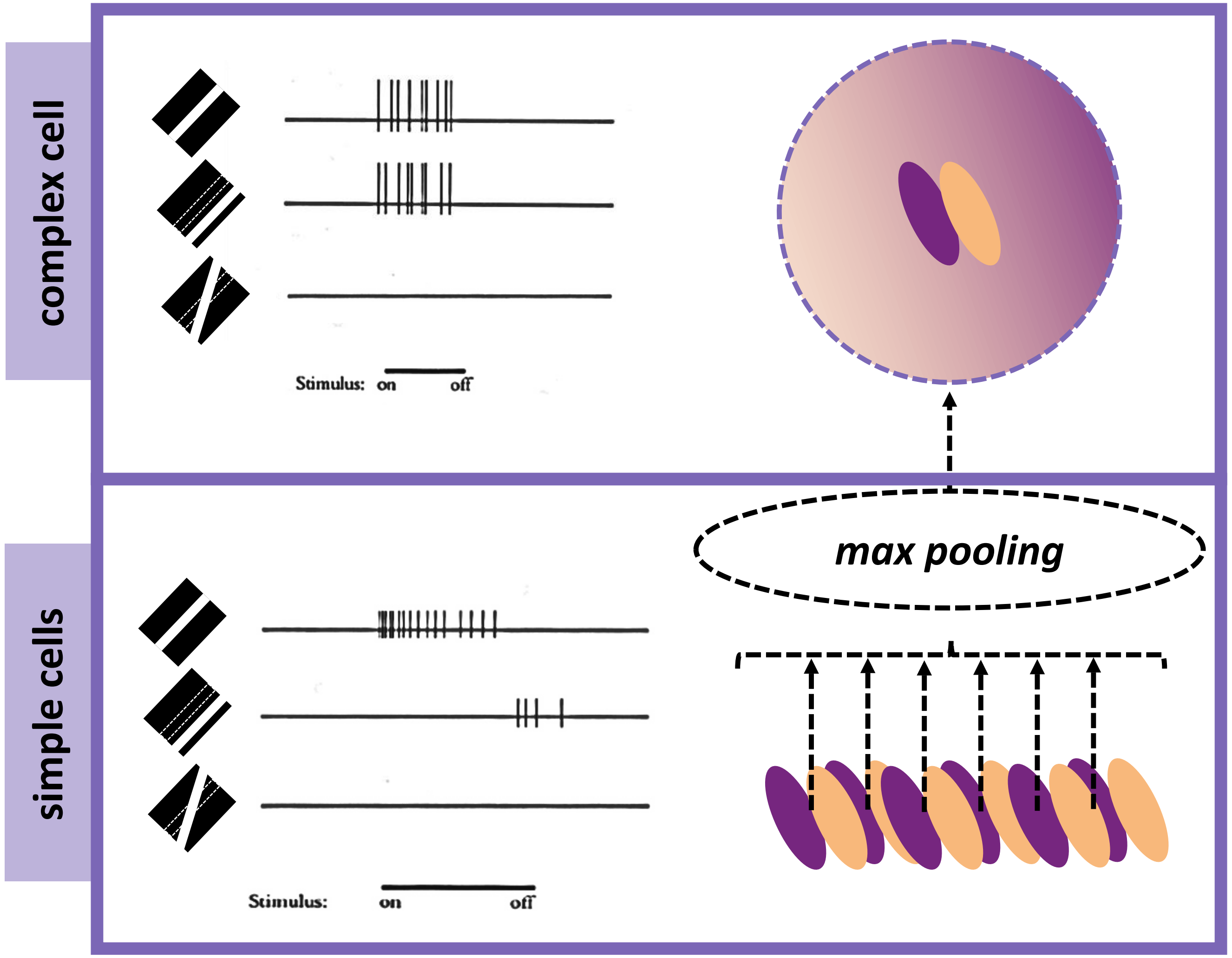
max-like **pooling** computation → underlie **transformation tolerance build up** observed through the primate shape processing stream …

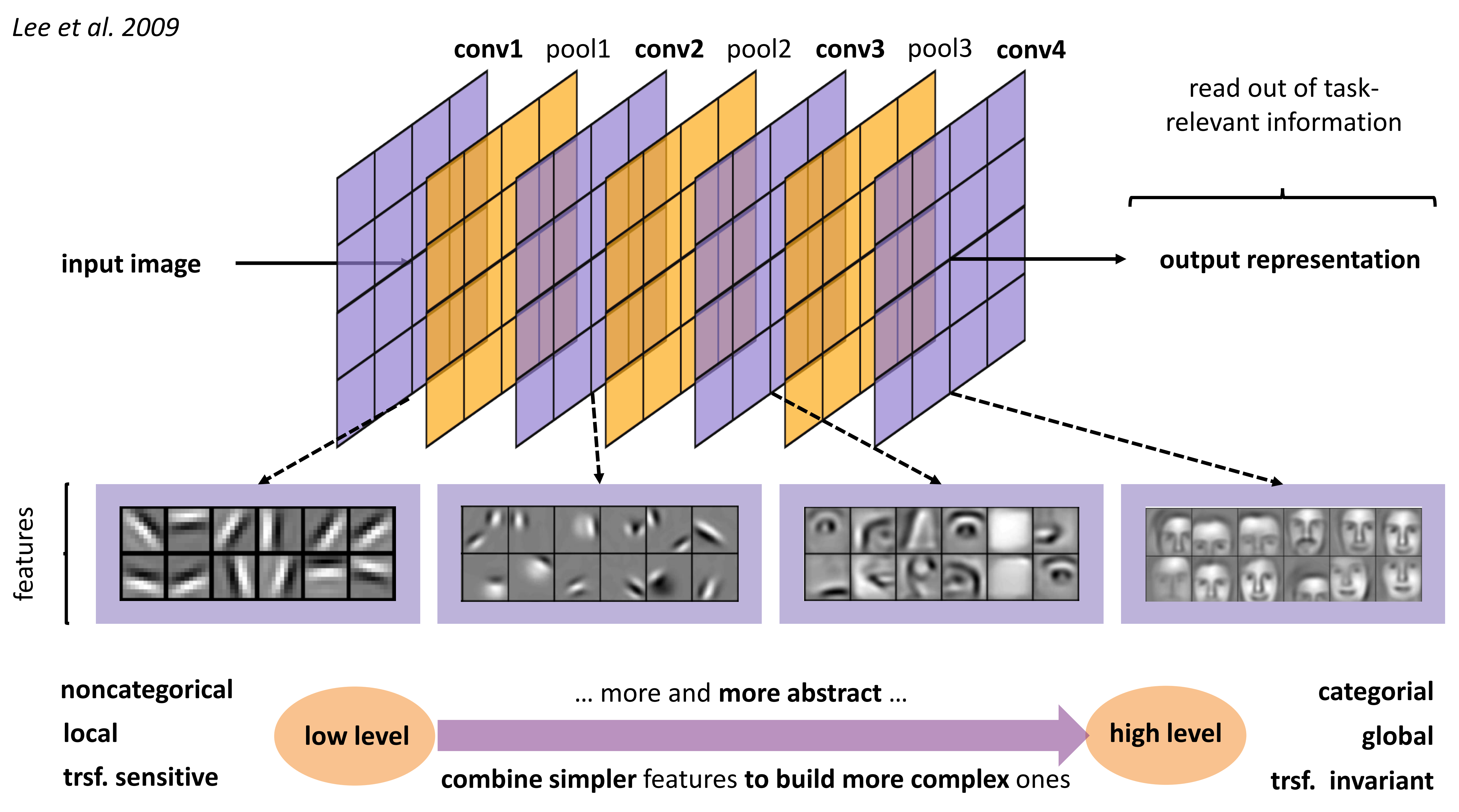
a classical example …

V1 simple &
complex cells

position tolerant
oriented edge
detector neuron

position selective
oriented edge
detector neuron




Lee et al. 2009
conv1
pool1
conv2
pool2
conv3
pool3
conv4
read out of task-relevant information
input image
output representation
features
noncategorical
local
trsf. sensitive
low level
... more and more abstract ...
combine simpler features to build more complex ones
high level
categorial
global
trsf. invariant

we can consider stacks of convolutional layers as visual **feature extractors …**

Features learned in solving one supervised task **can** frequently **be useful in different contexts.**

No need to learn every **feature from scratch for new tasks**!

transfer learning → **re-use the first N-layers** of a network with **pre-trained** weights (on **different task**)

… how far in depth push N? ... depends on how distant task domain involved are!

close domains { face recognition & emotion recognition

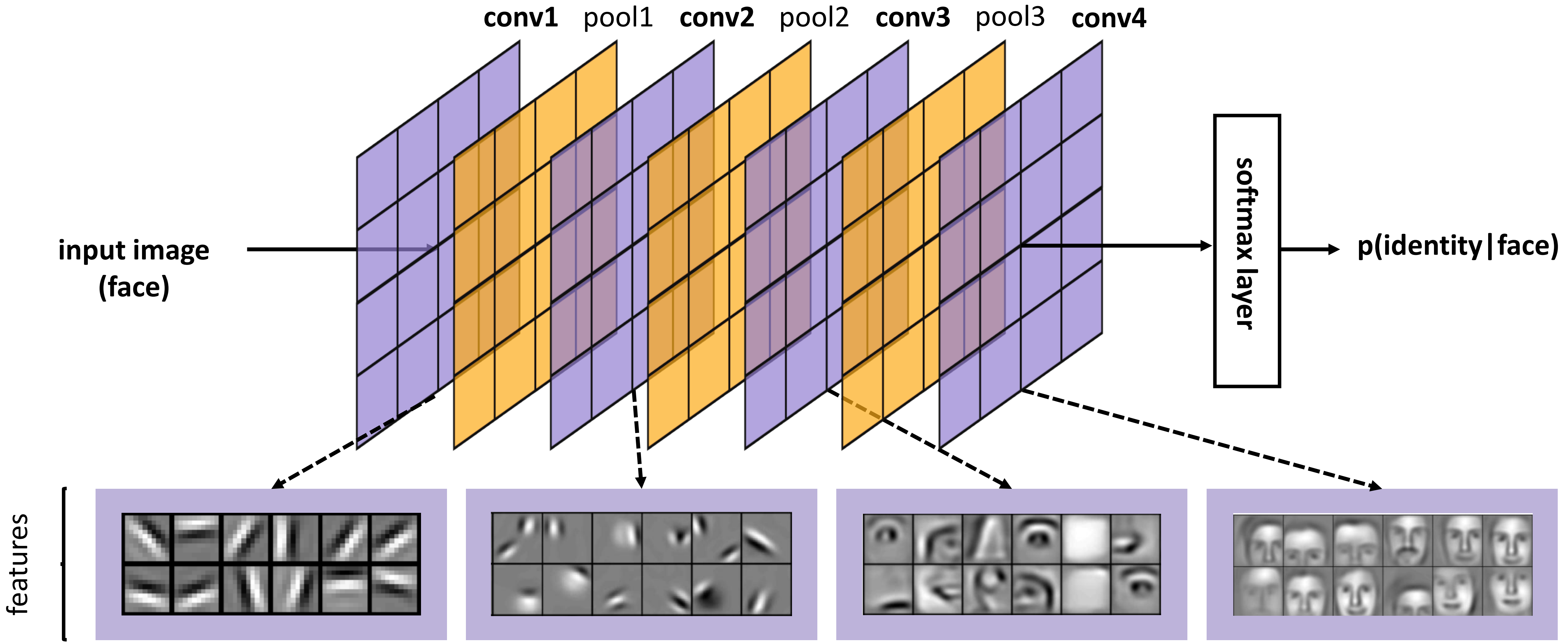
far domains { face recognition & satellite image classification

common high-level features → **high N**

only low-level features in common → **low N**

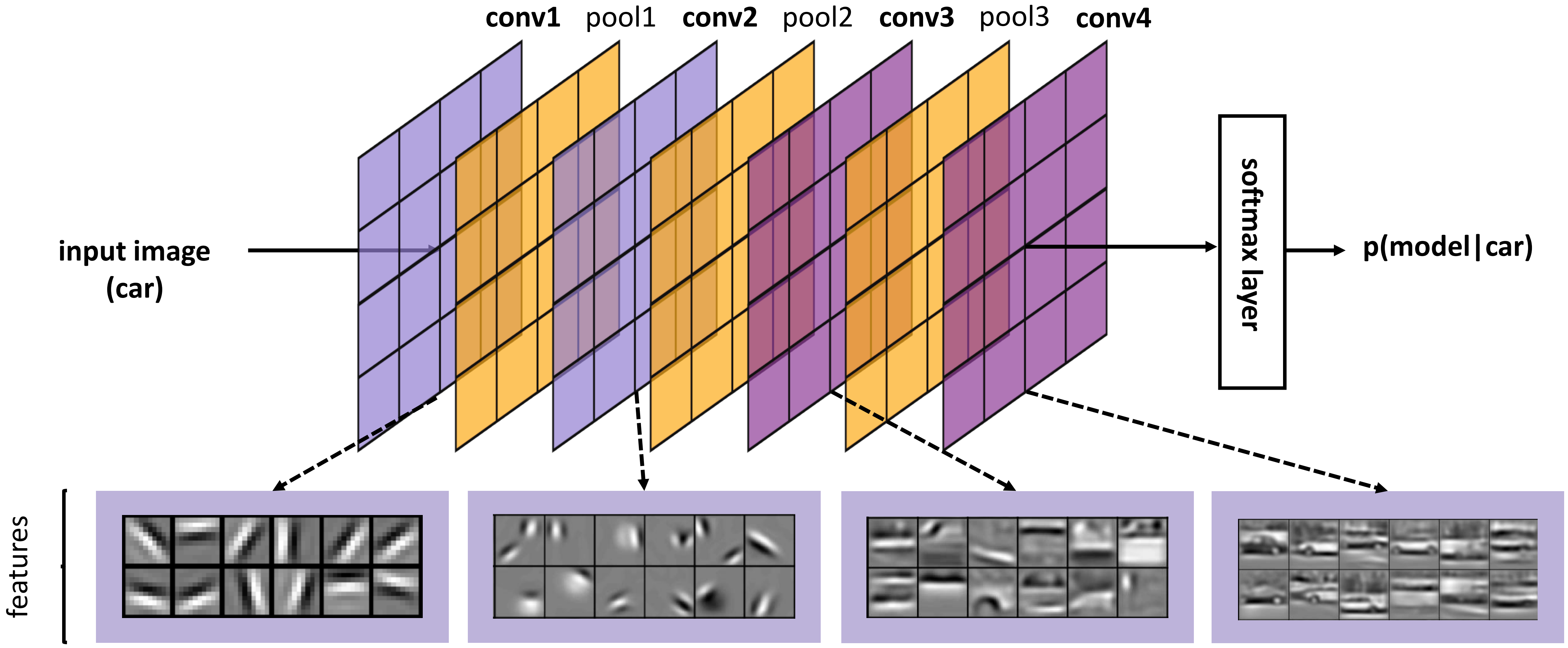
extends applicability of **deep learning in the *small data* regime** imagine to **start with** a trained **face recognition** system → now you want a **car model recognition** one

high level features will be **poorly trasferable** (too domain specific): → **strip away last layers!**



you are left with **a general purpose middle-level feature extractor** → ontop of that stick some new conv layers and a new softmax output

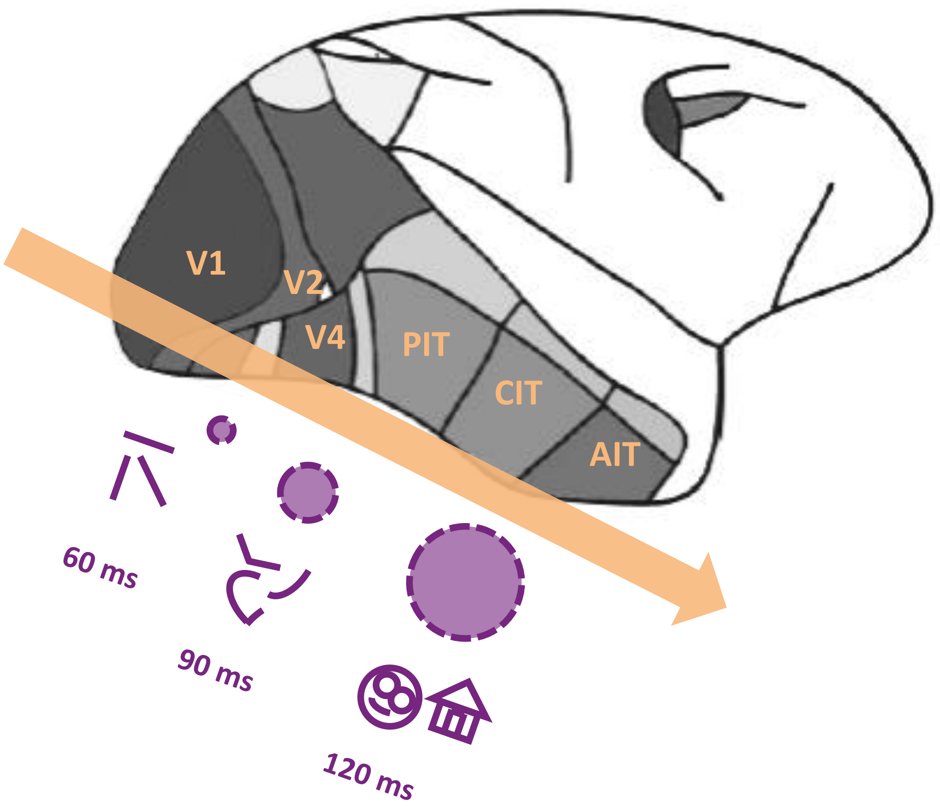
with training (much less) you will build **new car-specific** high-level **features and** a **working classifier**



hierarchical structure of CNNs layers (and features)

may be interpreted **as reflecting the compositionality of the visual world** (objects are made of parts and subpart etc…)

reminiscent of anatomical and functional hierarchy of visual pathways:



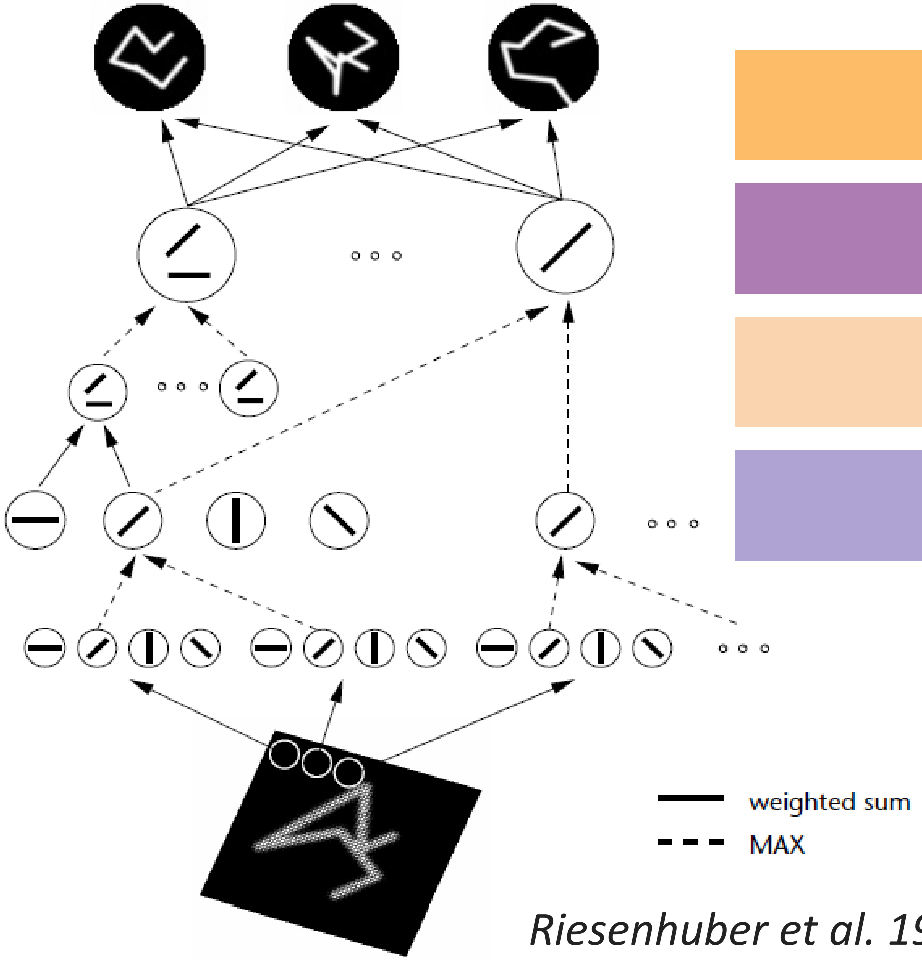
Huberman et al. 2011

ventral stream

- **response latency** increase
- **RF size** increase
- **tuning complexity** increase
- **transformation tolerance** increase
- **linear decodability** increase

this kind **of hierarchical brain processing** of visual shape information has **been modelled throughout the years** (80’, 90’) …

… from Fukushima’s ***Neocognitron*** to Poggio’s ***HMAX* model**



Riesenhuber et al. 1999

C2

S2

C1

S1

S: shape **selectivity build-up** (**AND**-like operations)

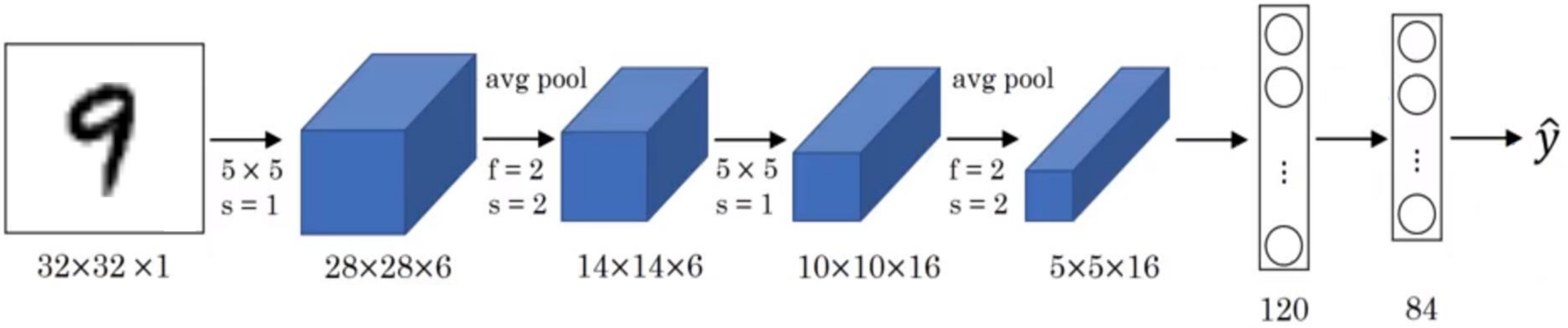
C: transformation **tolerance build-up** (**OR**-like operations)

biologically-derived ideas instantiated by these models
inspired the birth of modern CNNs architectures …

… first of which was Yan LeCun’s ***LeNet*** (‘98) → **first successful convnet** (handwritten digit recognition)

- first applying **stack of conv and pool layers** followed by fc ones
- **shallow: 2 conv layers** interleaved with pooling

- **conv** filter **size 5x5** (p=0 ↔ “valid”, s=1)
- **pooling** filter **size 2x2** (p=0, **s=2**)
- $\mathbf{60 \cdot 10^3}$ **parameters** (small)

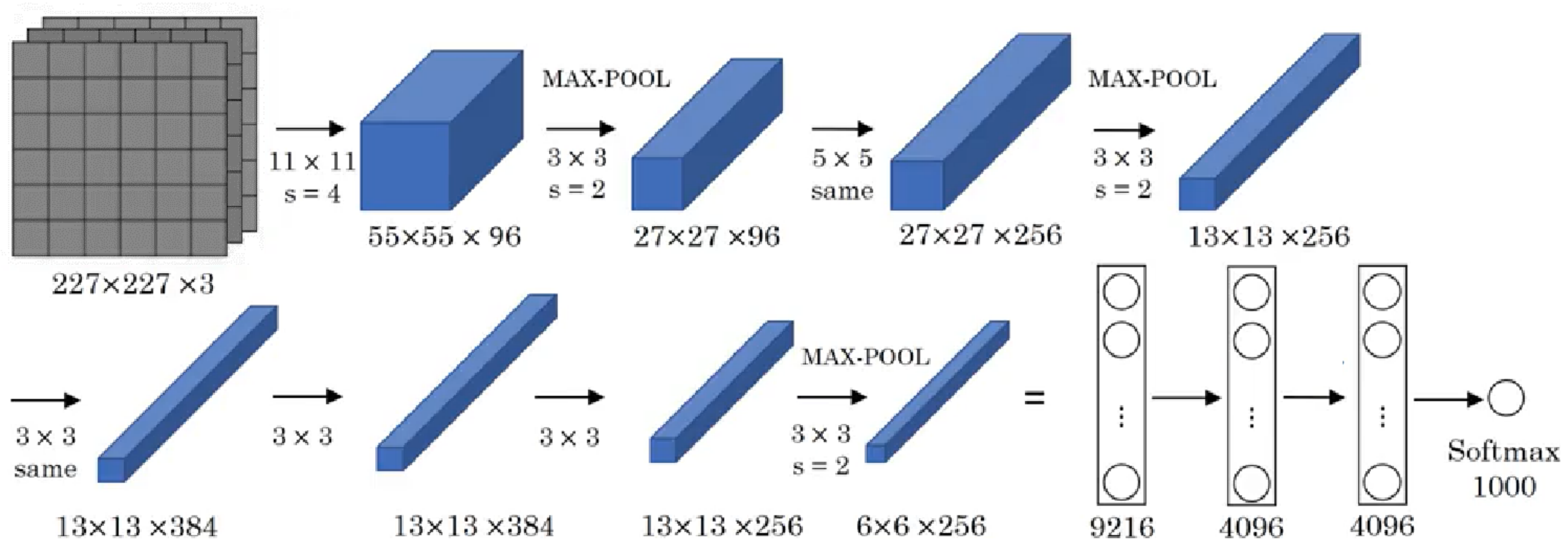


Ng. 2017

AlexNet **popularized CNNs** significantly outperforming competitors in ILSVRC 2012 (top 5 error to 16% from 26%)

- **avoid vanishing gradients**: first to **use ReLU** activations instead of sigmoid for conv layers
- **improve training**: used **dropout**, **data augmentation** and SGD with **momentum**
- **deep**: **5 conv** layers (not always interleaved by pooling ones) followed by fc

- variable filter size, stride and padding
- $60 \cdot 10^6$ **parameters** (bigger)



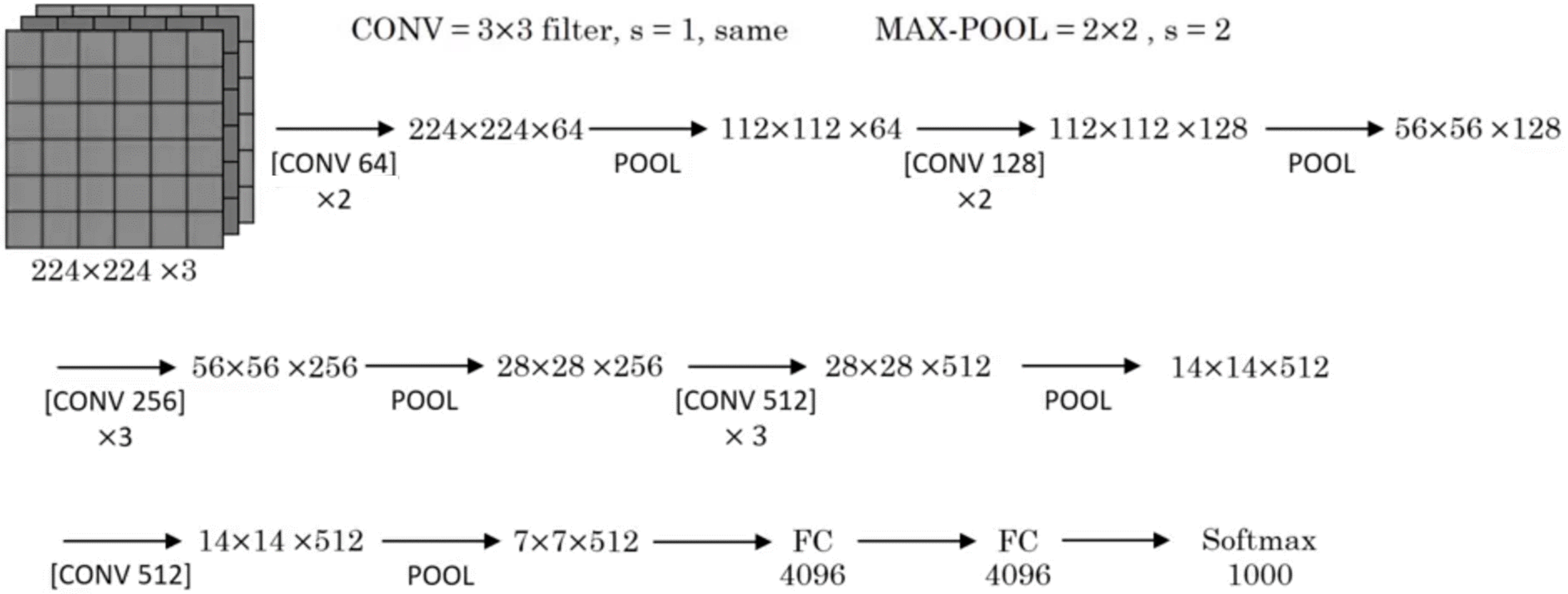
Ng. 2017

VGG16 **showed** that the **depth** of the network **is a critical** component performance (second place at ILSVRC 2014)

- **deeper**: **13 conv** layers (5 “blocks” of conv layers + pooling) + 3 fc
- **reducing filter size to increase depth** pays off
- **homogeneous**: only **3x3** conv **filters** (p=1 ↔ “same”, s=1) + 2x2 pooling (p=0, s=2)

feature map size ↓ (pool)
number of feature ↑ (conv)

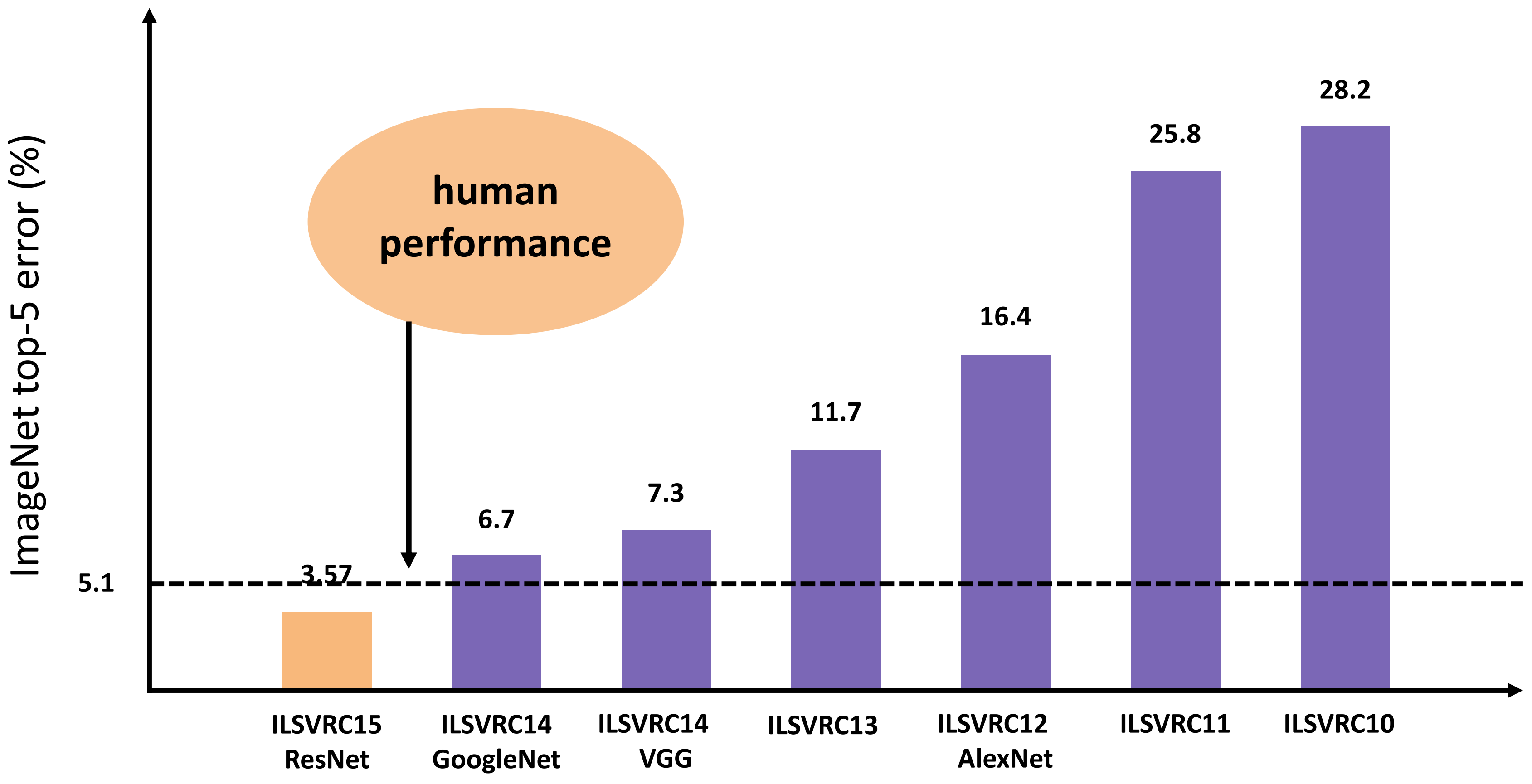
- $\mathbf{138 \cdot 10^6}$ **parameters** (big, but pretrained model **available for plug and play use** in Keras API)



Ng. 2017

more **recently** → **Deeper architectures** matched or **surpassed human performance in many domains** …

… from image classification to face recognition and CAPTCHAs



… **however** their behaviour is **surprisingly brittle**!

imperceptible (purposely crafted) **perturbation** of input may produce **huge change in output** class **probability**

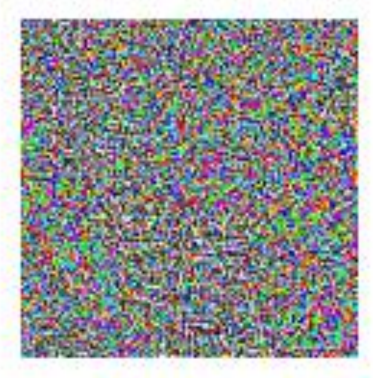
adversarial attack — add small **perturbation computed** as a function of parameters **to fool** the **network** $\xrightarrow{\text{e.g.}}$ *fast gradient sign method*

original example

correctly **classified** image

panda with 57.7% confidence

$+ .007 \times$



structured attack image

$=$

gibbon with 99.9% confidence

adversarial example

confidently **misclassified** image

Goodfellow. 2014

x $\qquad \varepsilon \cdot sign(\nabla_x J(\theta, x, y))$ $\qquad \tilde{x}$

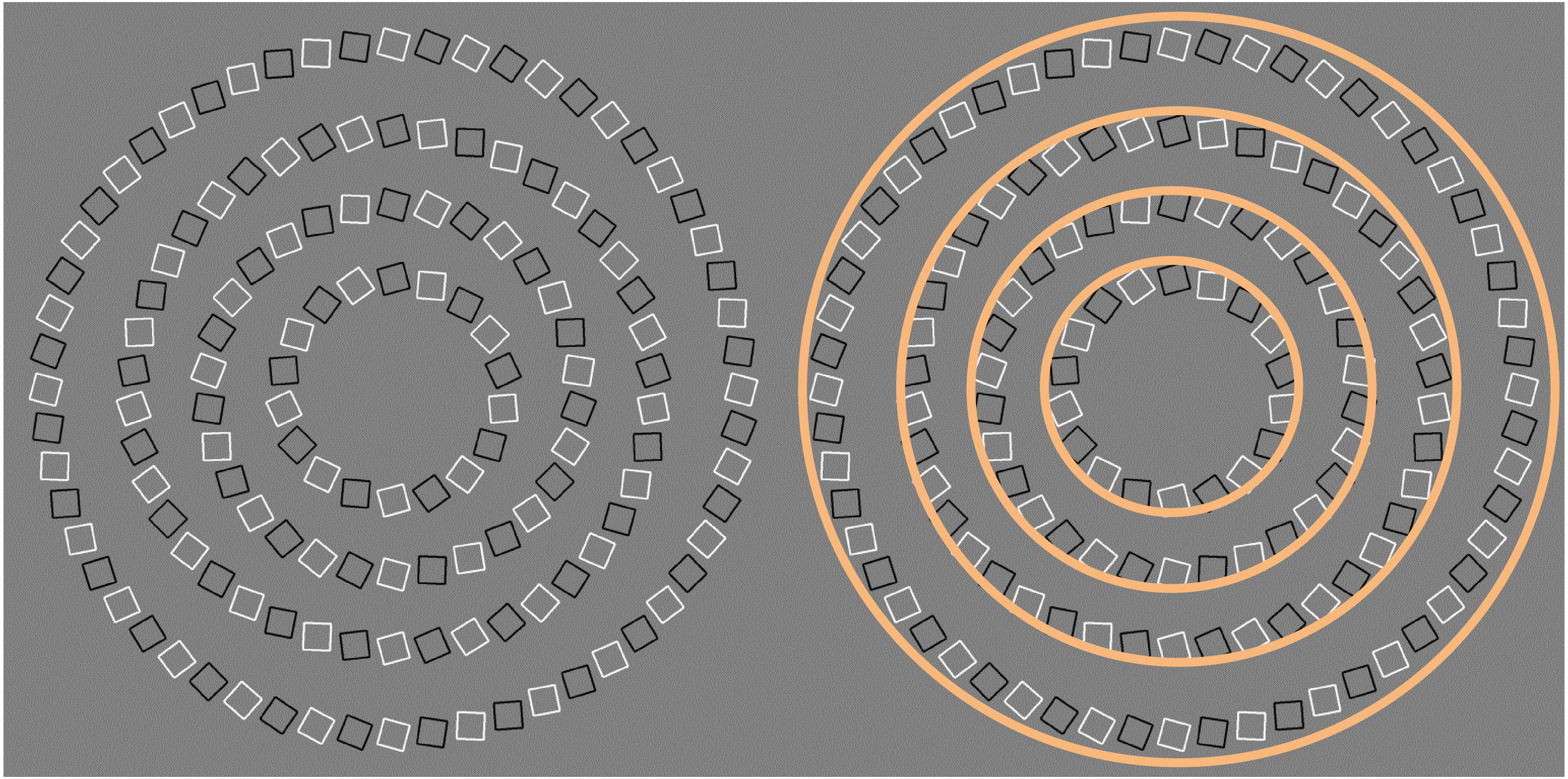
many other kinds of adversarial attacks **exist** (additive patterns, transformation/deformations)

still a lot to do to improve robustness/generalization capacity … ⟶ **far from human** in this sense!

however ...

you are sensitive to "adversarial attacks" too! (even if different kind)

e.g. → *Pinna's Illusion*

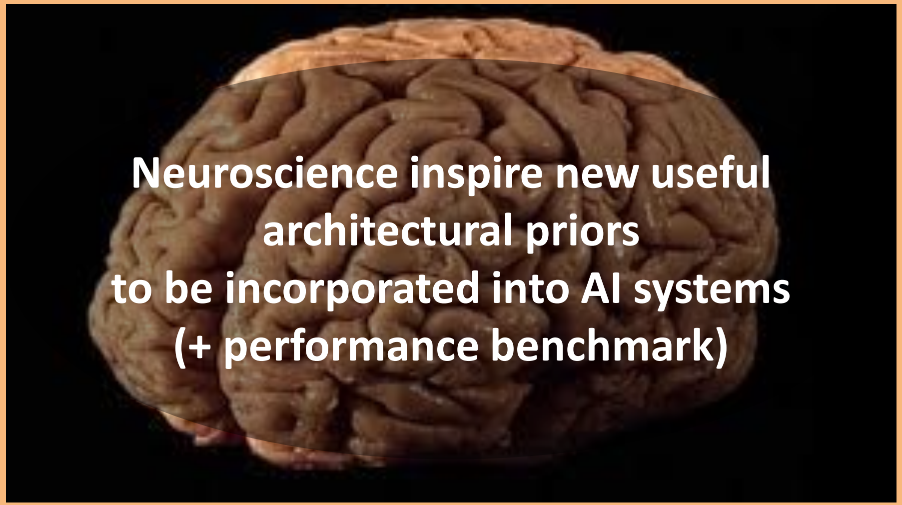


CNNs success all about a specific **inductive bias** (prior) → an image processing system should be **local and translationally invariant**

enforced through an **architectural choices** (conv. weight sharing/pooling)

architectural priors

- ***necessary good*** (C. Manning) → enable to **learn more** from less data, **faster**, and at a higher level of abstraction
- ***necessary evil*** (Y. LeCun) → **assumptions may be wrong** for at least some portion of the data (better learn than hardwire)




Thank you!