COMP 432 Machine Learning

Convolutional Networks

Computer Science & Software Engineering
Concordia University, Fall 2021

Convolution and cross-correlation

- Standard operations, long used in compression and signal processing (DFT), computer vision ("edge detection") and image processing tools like Photoshop ("custom filter")

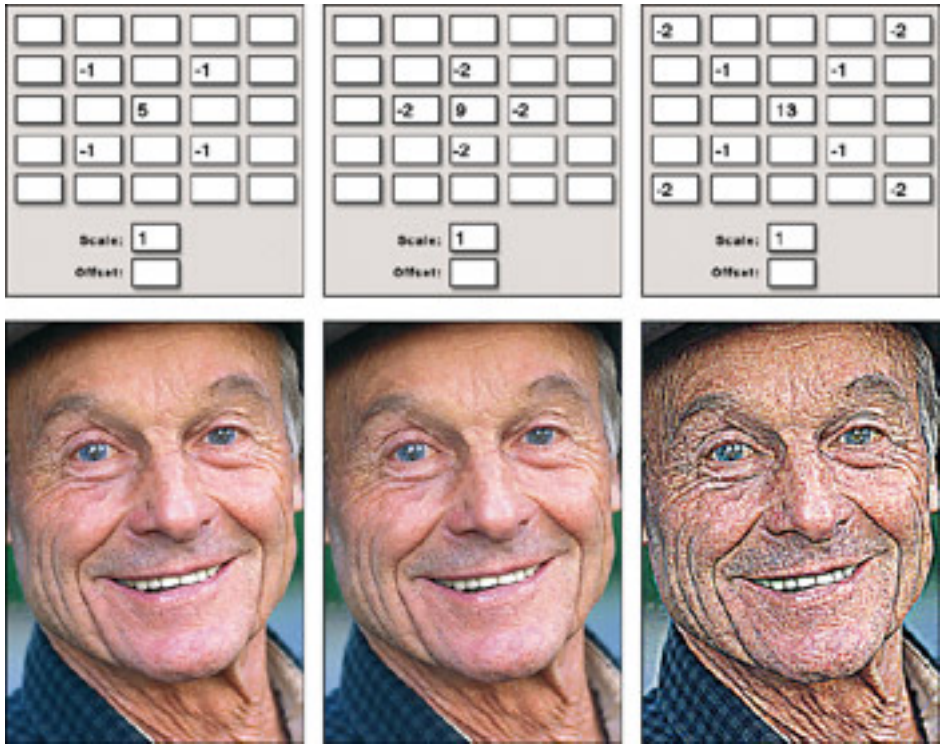


Image credit:
Photoshop CS2 Bible

Convolution layer

So convolution is a special case of matrix multiply!

Convolution is a special linear transformation of the input.
Convolution *layer* usually adds a non-linearity *after*.

"Convolutional networks are simply neural networks that use [discrete] convolution in place of general [linear transformation] in at least one of their layers." – Goodfellow et al (Deep Learning book)

Core idea for ML purposes:

- Many forms of input have a natural *spatial* or *temporal* arrangement (images, audio, etc.)
- For such inputs, it helps to *learn* a *shared* set of *local* "pattern detectors" that can be applied at every shift of the input

  "If it is useful to detect a pattern *here*, it is probably useful to also detect it over *there*!"

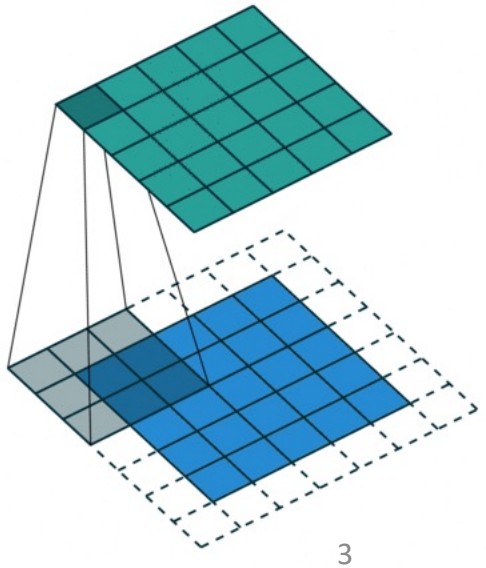


Image credit: Dumoulin & Visin

Convolution examples

Note to signal processing folks:
in neural networks, "convolution"
actually means ***cross-correlation!***

64x43 input

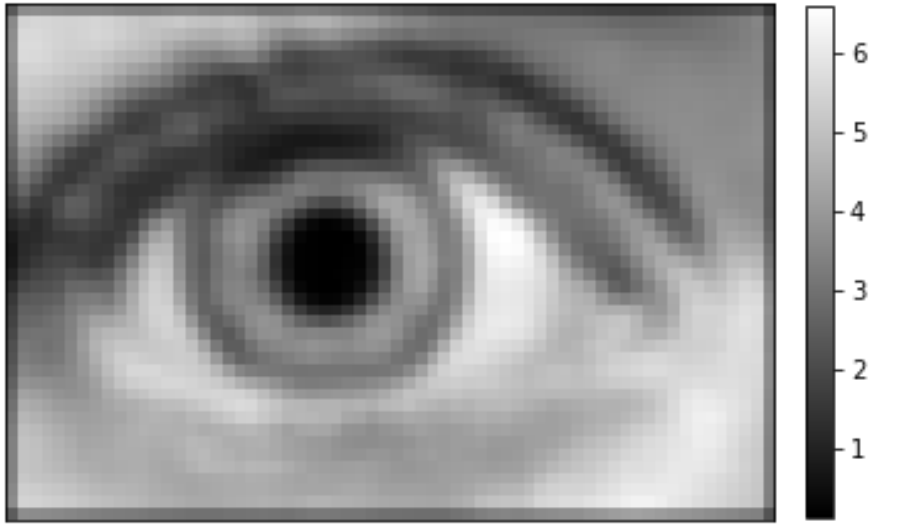
3x3 filter
(9 weights)

+1	+1	+1
+1	+1	+1
+1	+1	+1

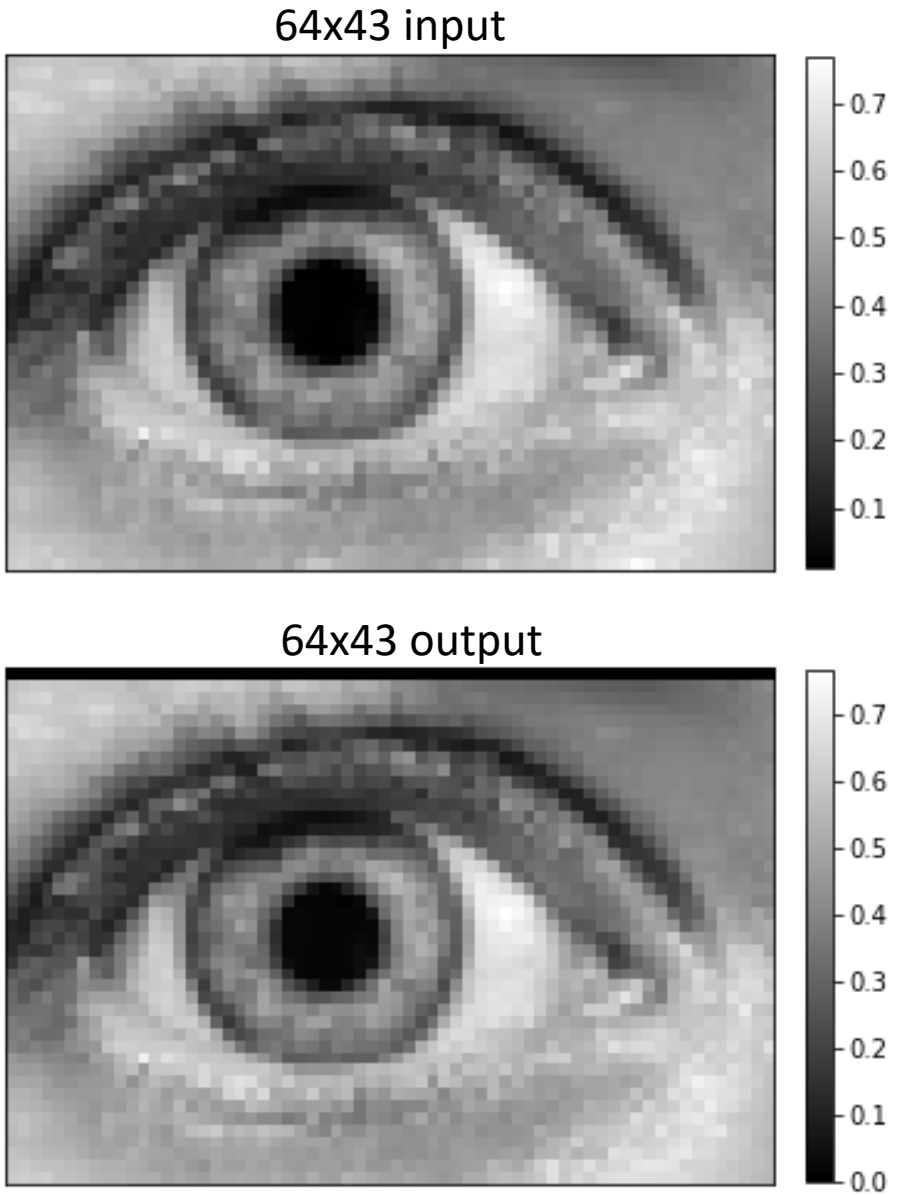
Conv2D

(with padding=1)

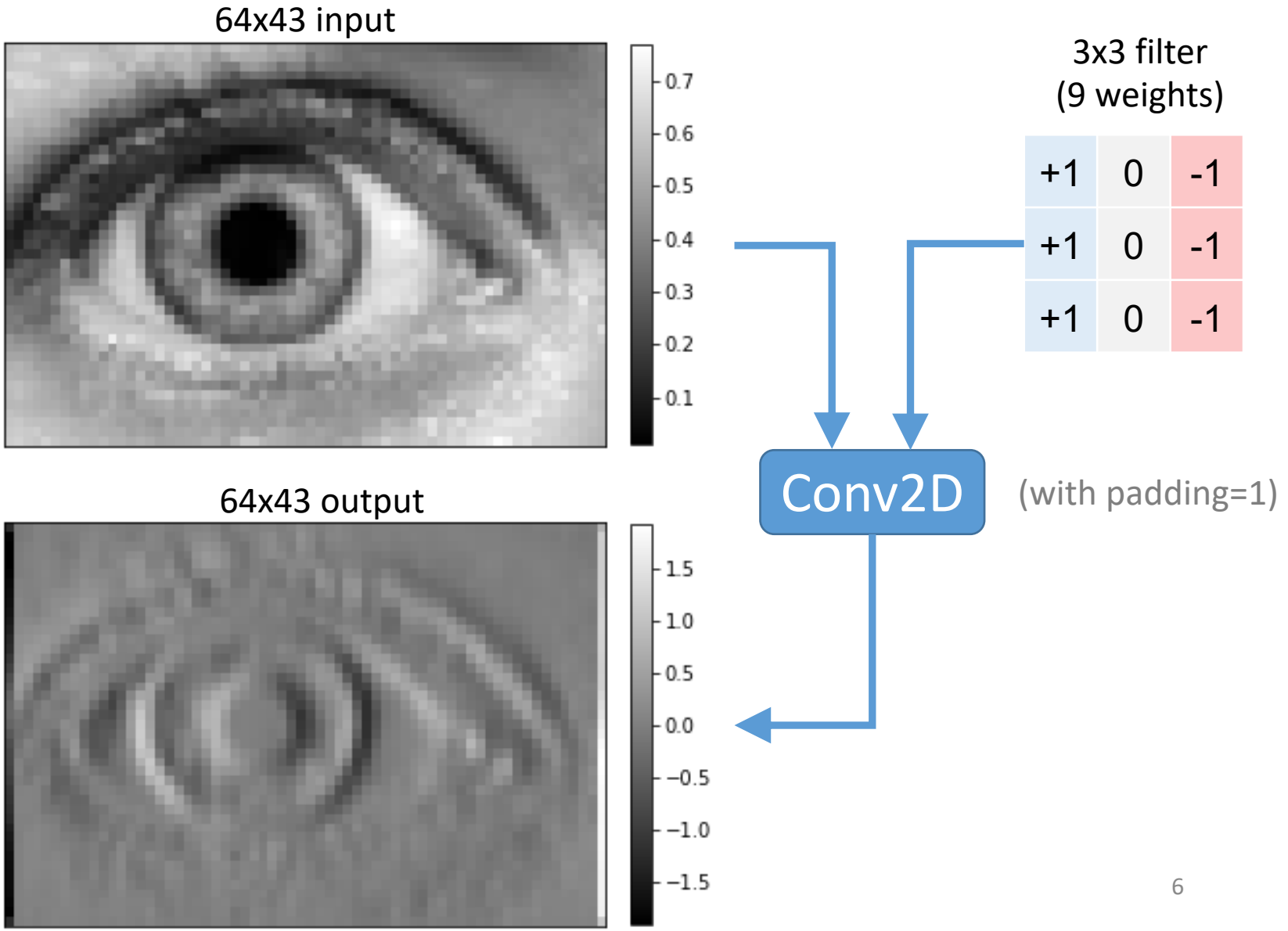
64x43 output



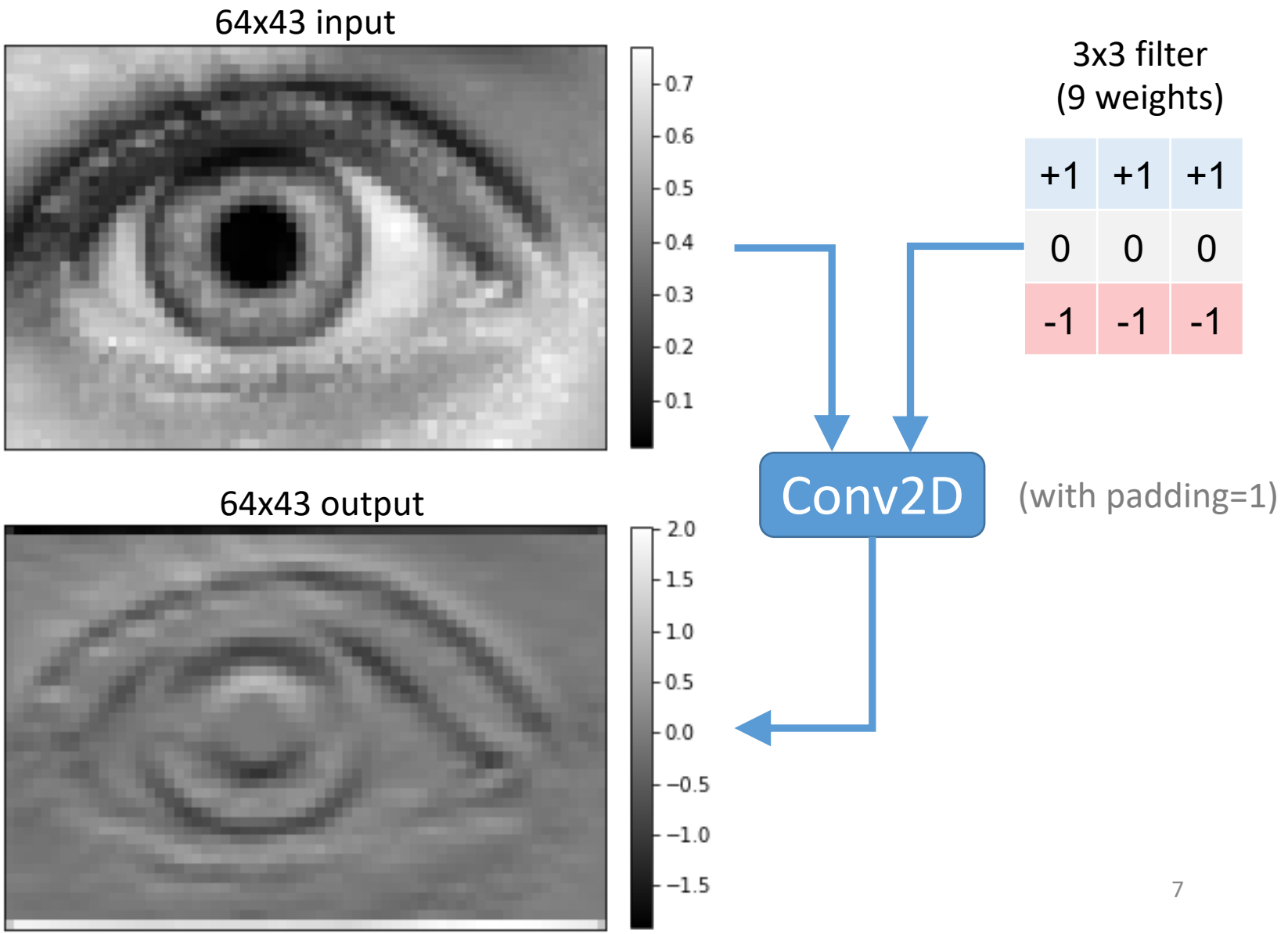
Convolution examples



Convolution examples



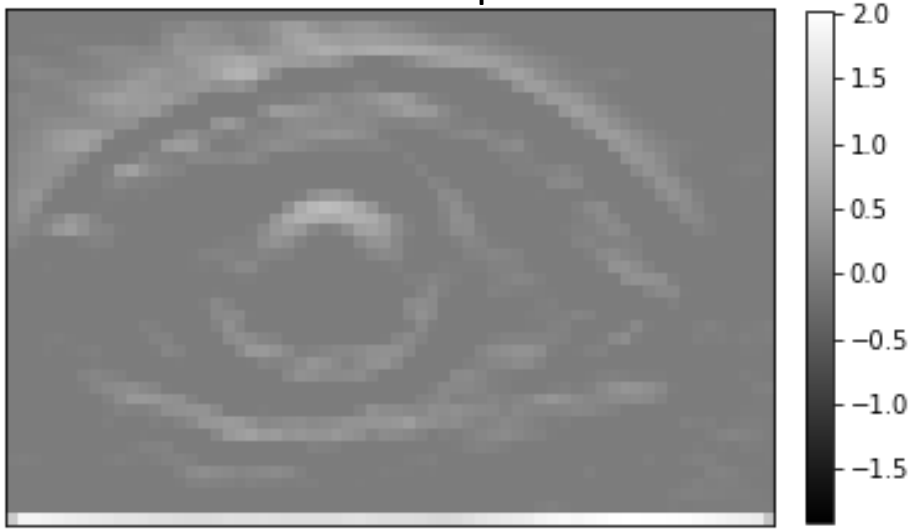
Convolution examples



Convolution layer with ReLU

64x43 input

64x43 output



3x3 filter
(9 weights)

+1	+1	+1
0	0	0
-1	-1	-1

x

w

Conv2D

(with padding=1)

a

"activation map"

ReLU

z

"feature map"

Why "feature map"? Because z serves as features x of next layer!

Convolution as linear transform

1D convolution w/ padding

$$a_{i'} = \sum_{i=1\ldots s} w_i x_{i'+i-p}$$

padding

filter size

2D convolution w/ padding

$$a_{i',j'} = \sum_{\substack{i=1\ldots s \\ j=1\ldots s}} w_{i,j} x_{i'+i-p,j'+j-p}$$

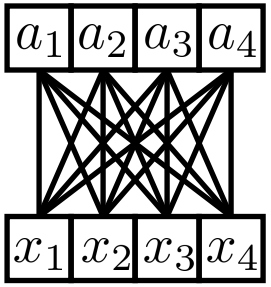
2D convolution in pure Python

```
def conv_py(x, w, p=0):
    """Convolution of (H,W) array x with (H,W) filter w with padding p."""
    xn, xm = x.shape
    wn, wm = w.shape
    an, am = (xn+2*p-wn+1, xm+2*p-wm+1)    # size of output feature map
    a = np.zeros((an, am), dtype=np.promote_types(x.dtype, w.dtype))
    for ai in range(an):                    # loop over output row ai
        for aj in range(am):                # loop over output col aj
            for wi in range(wn):            # loop over filter row wi
                for wj in range(wm):        # loop over filter col wj
                    xi = ai + wi - p        # calculate input row xi
                    xj = aj + wj - p        # calculate input col xj
                    xij = x[xi,xj] if  xi >= 0 and xi < xn \
                                   and xj >= 0 and xj < xm \
                                   else 0.0     # pad with zeros
                    a[ai,aj] += w[wi,wj] * xij  # contribute x[xi,xj] to a[ai,aj]
    return a
```

(see convolution_layer_demo.ipynb)

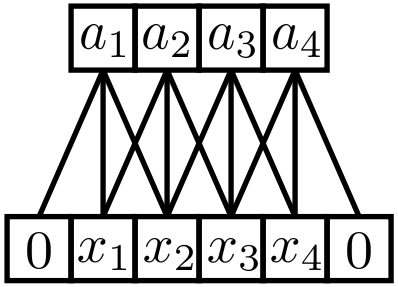
"Fully-connected" neural network layer

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$



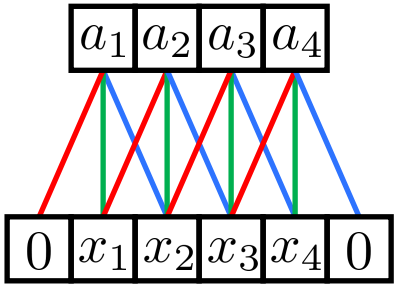
1D "locally-connected" layer $(s = 3, p = 1)$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & 0 & 0 & 0 \\ 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & 0 \\ 0 & 0 & w_{3,1} & w_{3,2} & w_{3,3} & 0 \\ 0 & 0 & 0 & w_{4,1} & w_{4,2} & w_{4,3} \end{bmatrix} \begin{bmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ 0 \end{bmatrix}$$



1D convolutional layer $(s = 3, p = 1)$

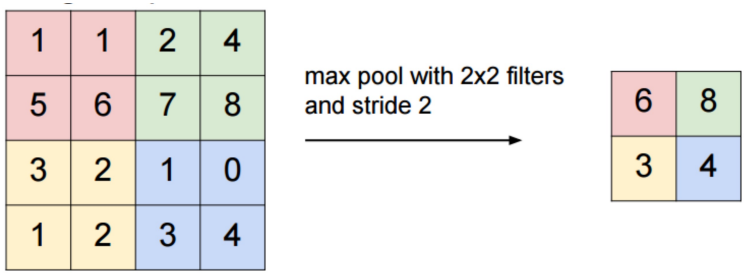
$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ 0 \end{bmatrix}$$



MUCH FEWER PARAMETERS!!

Strides and pooling

- Strides are used to reduce the size of feature maps by "summarizing" the contents of spatial regions.
  - e.g., stride=4 goes from 200x200 to 50x50
- Convolution and pooling both allow strides
- **Average-pooling:** take the average of each region
  - special case of convolution, where all weights equal.
- **Max-pooling:** take the max of each region
  - not special case of convolution; non-linear



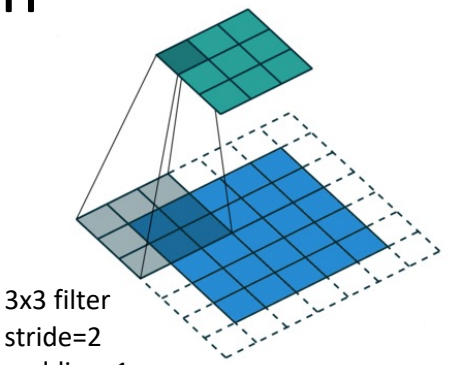
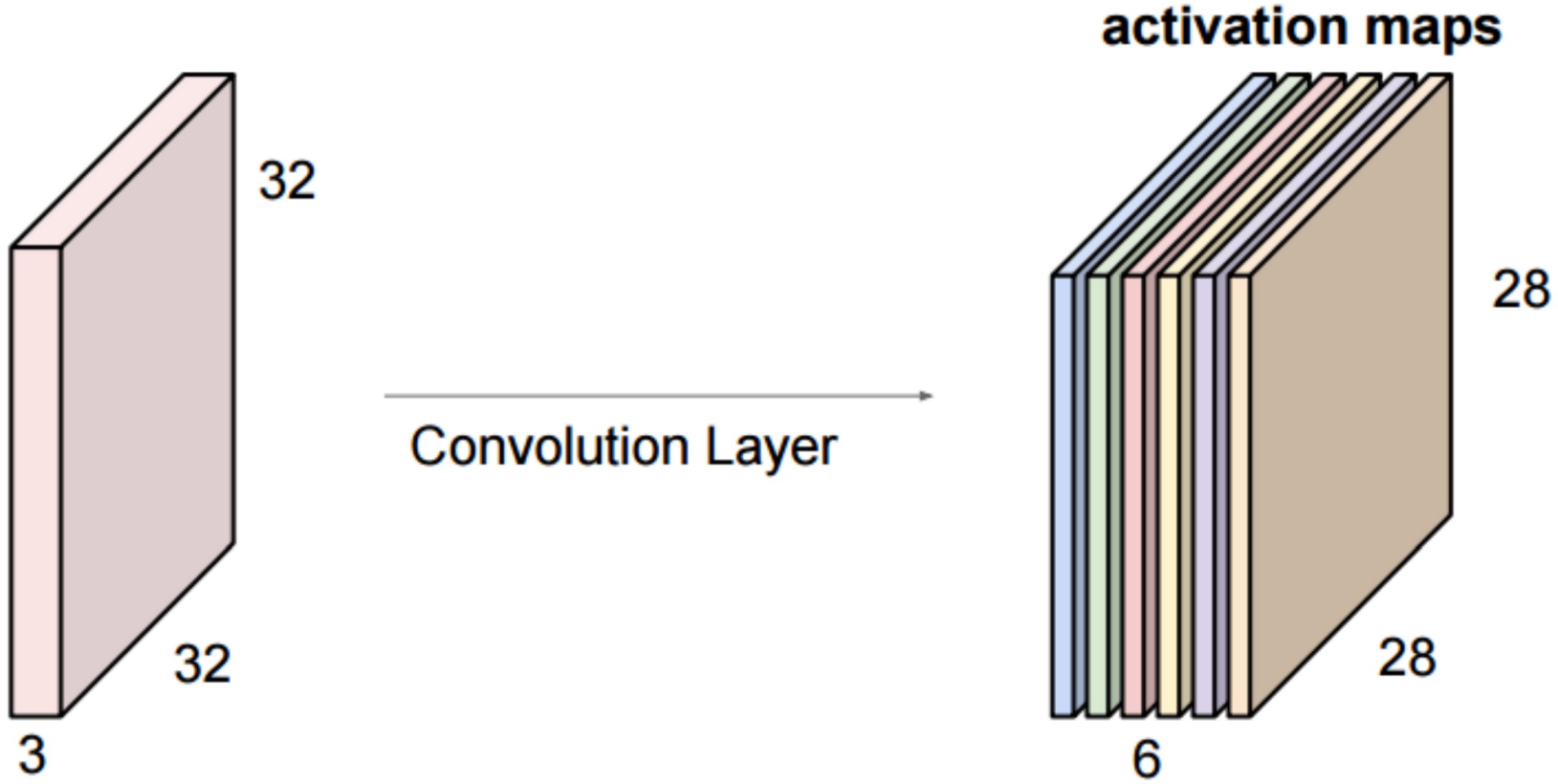


Image credit: Dumoulin & Visin

Typically multiple input channels and multiple output channels



6 filters => 6 output channels (and 6 activation maps)

Convolution in PyTorch

CLASS `torch.nn.Conv2d(`*`in_channels`*, *`out_channels`*, *`kernel_size`*, *`stride=1`*, *`padding=0`*, *`dilation=1`*, *`groups=1`*, *`bias=True`*, *`padding_mode='zeros'`*`)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Pooling in PyTorch

CLASS `torch.nn.MaxPool2d(`*`kernel_size`*`, `*`stride=None`*`, `*`padding=0`*`, `*`dilation=1`*`, `*`return_indices=False`*`, `*`ceil_mode=False`*`)` [SOURCE]

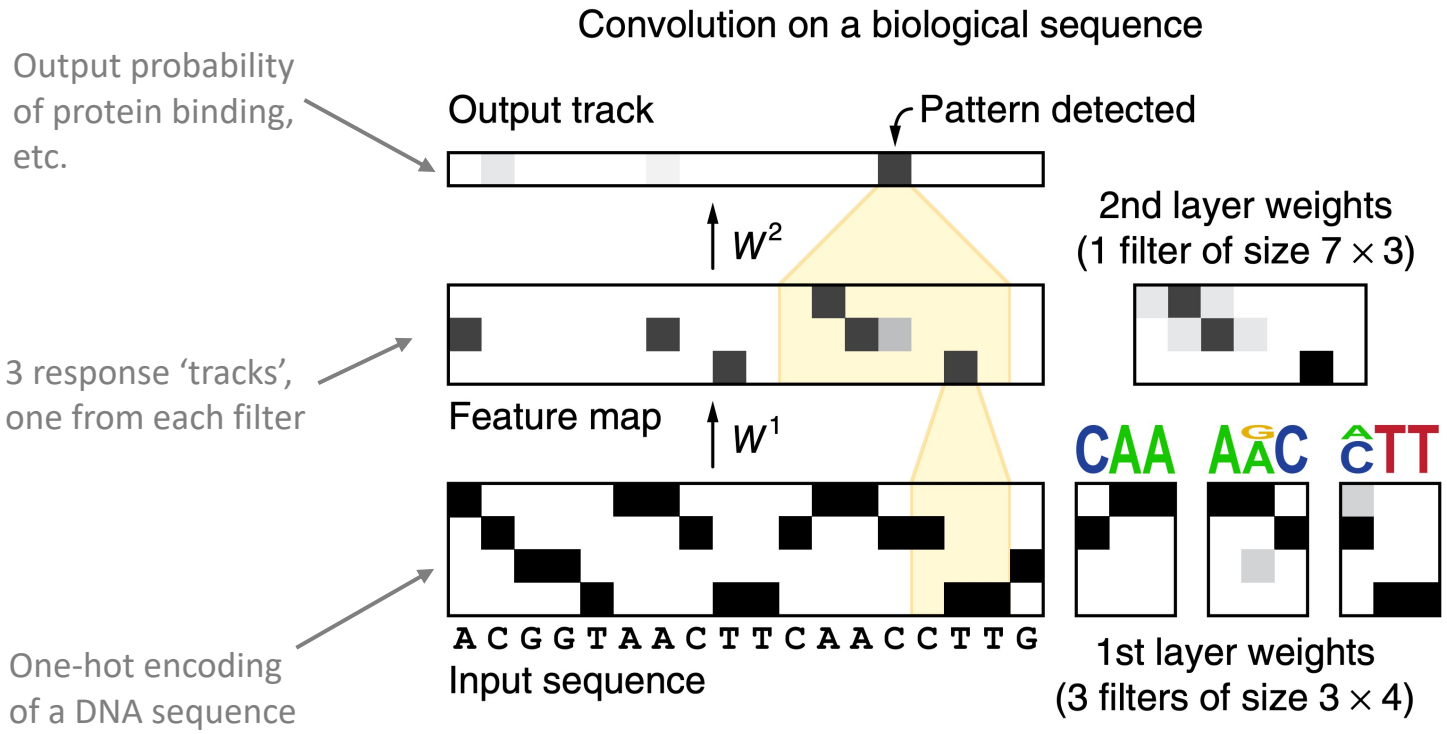
Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly zero-padded on both sides for `padding` number of points. `dilation` controls the spacing between the kernel points. It is harder to describe, but this link has a nice visualization of what `dilation` does.

Works as “local pattern detector” for continuous and discrete data



Learning hierarchies of features

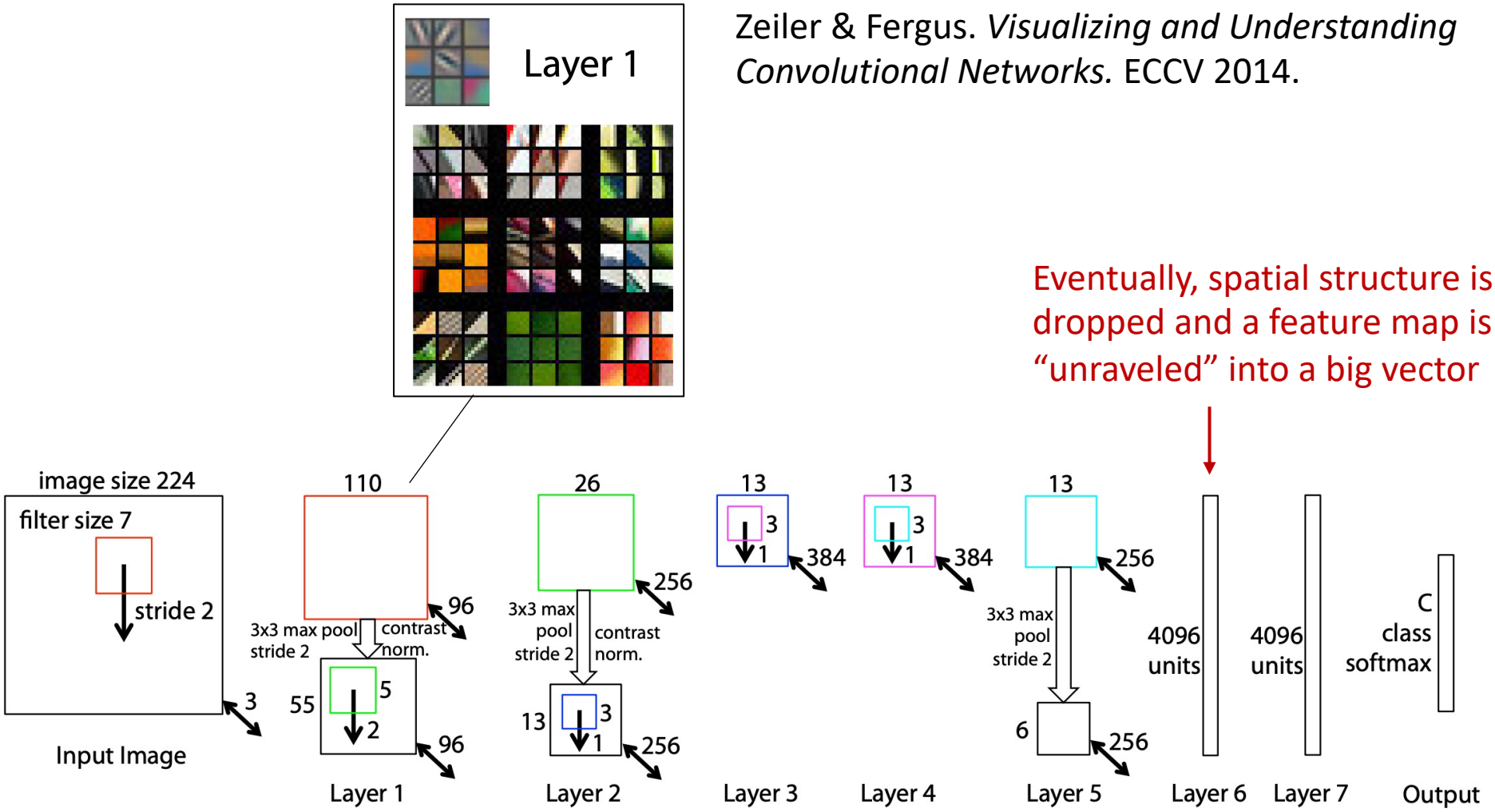
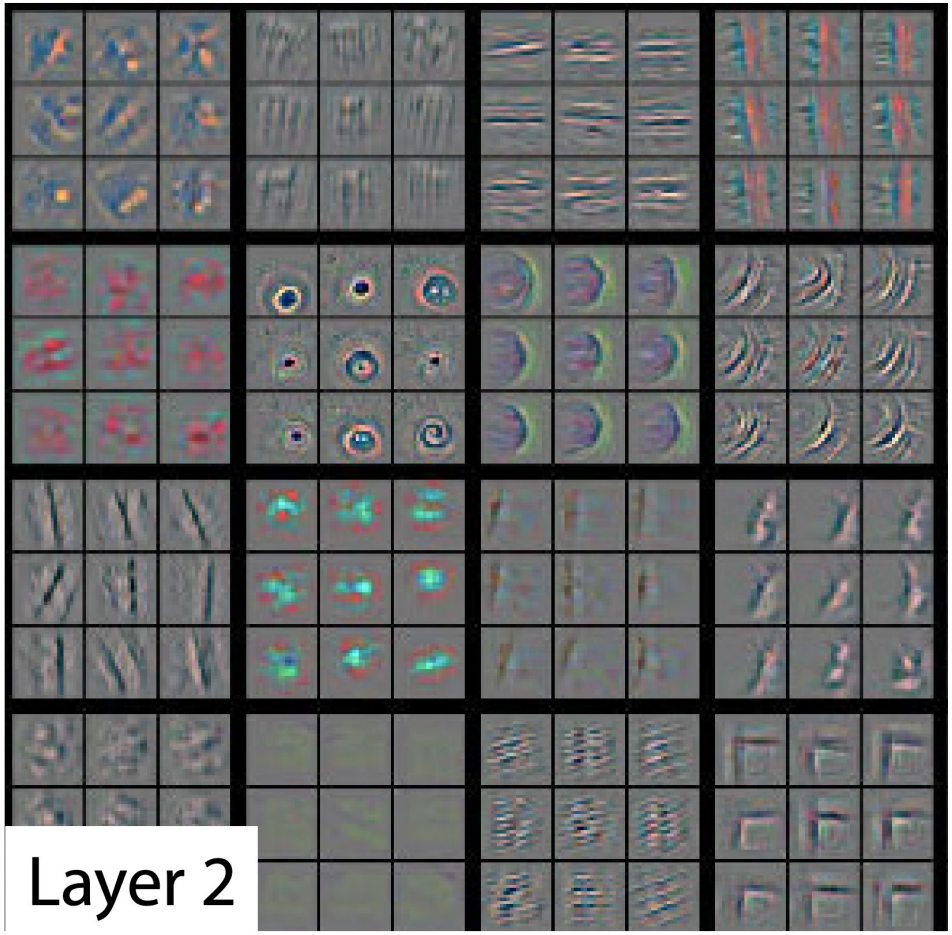
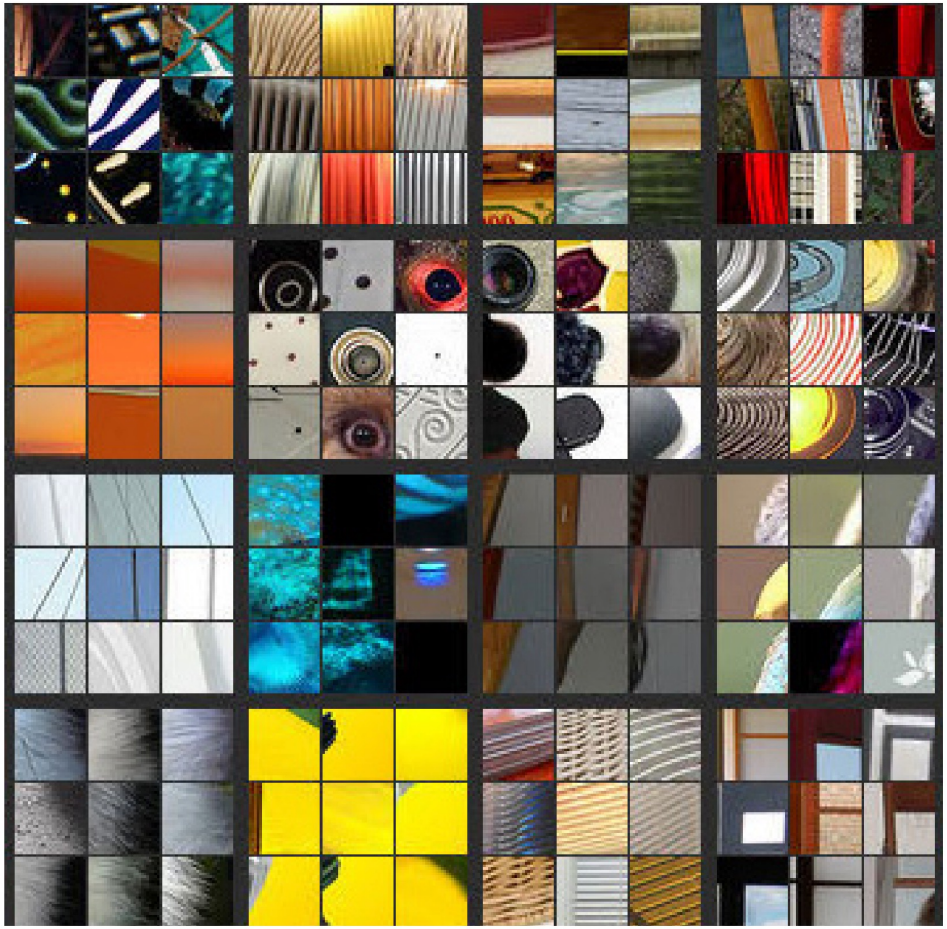


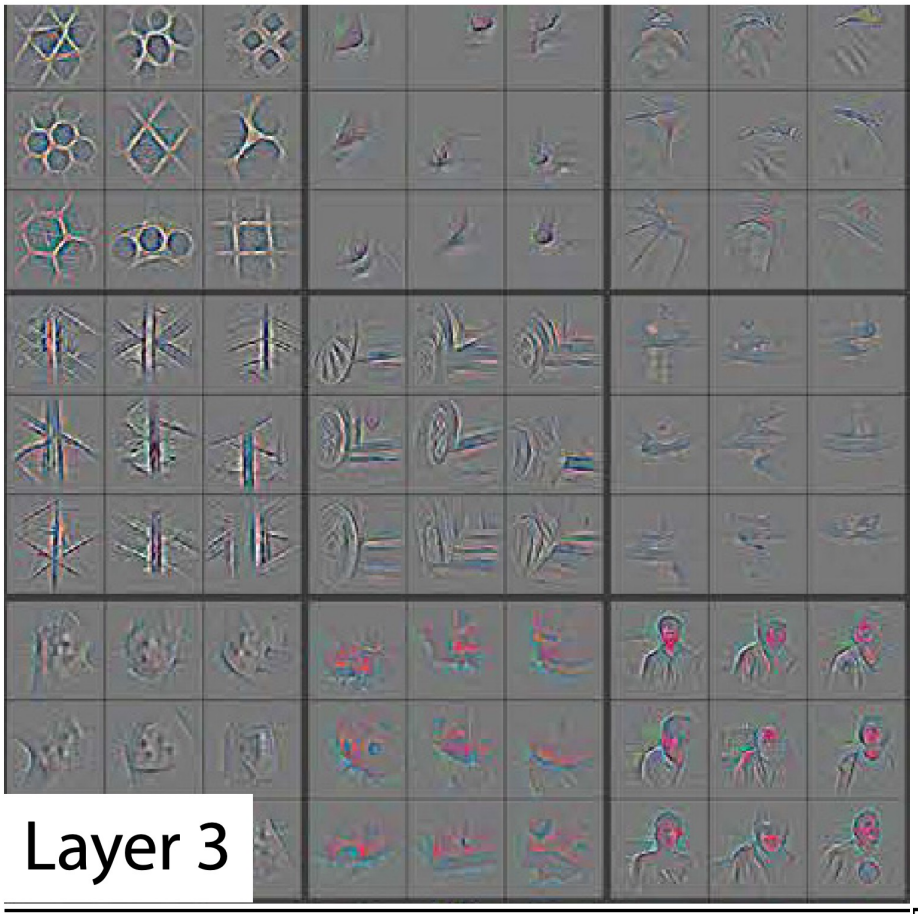
Fig. 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with

Learning hierarchies of features

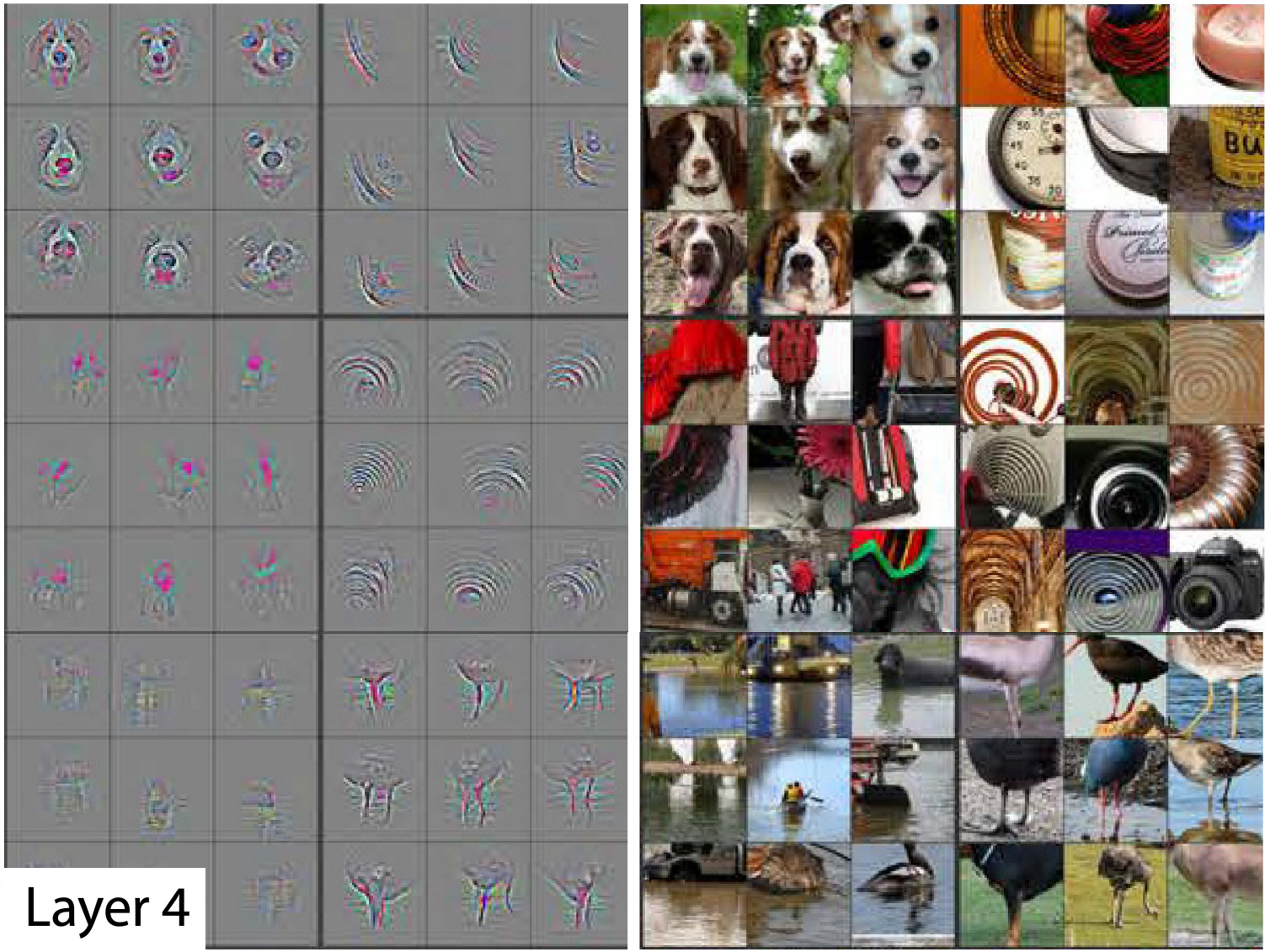




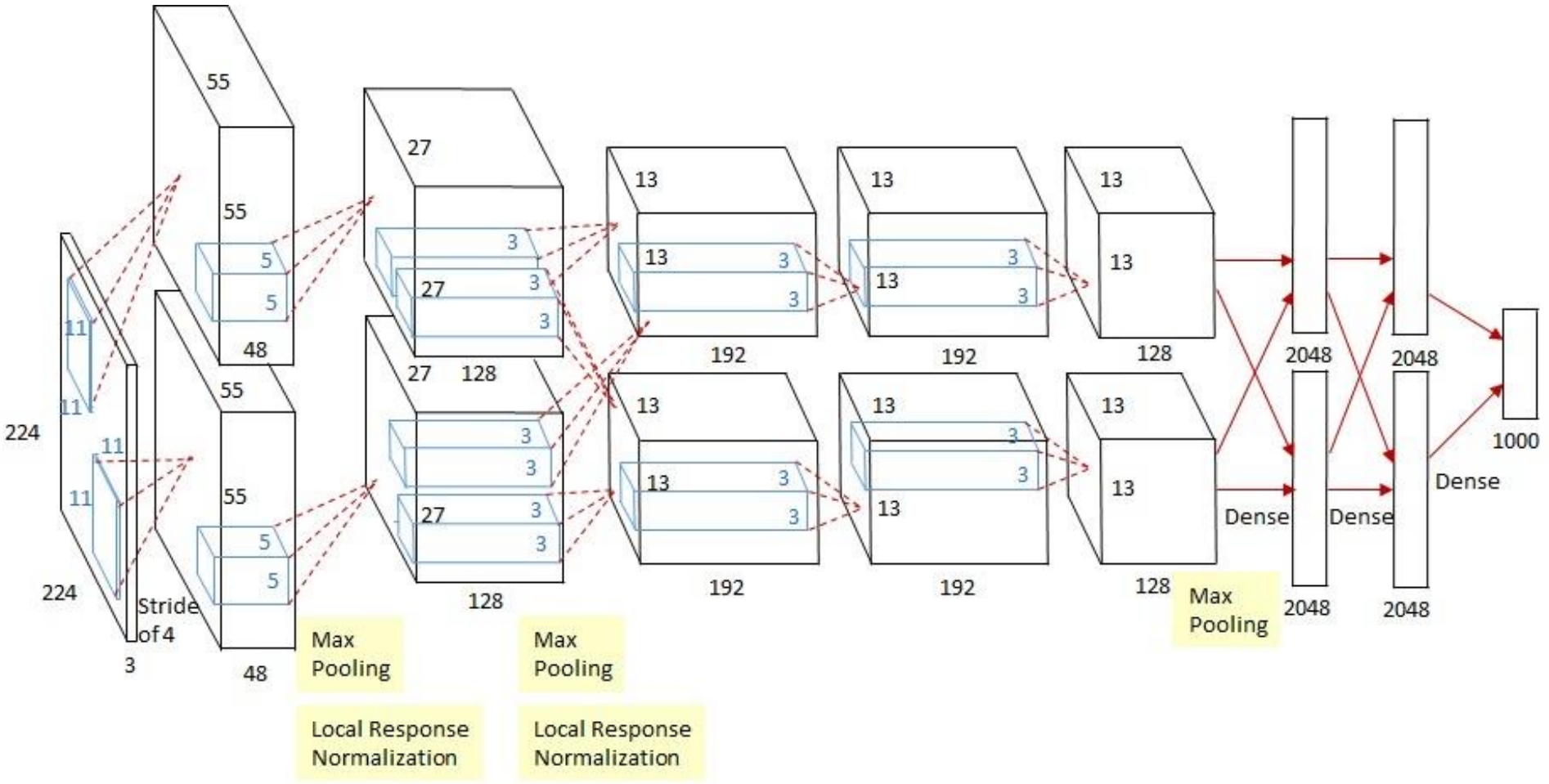
Learning hierarchies of features



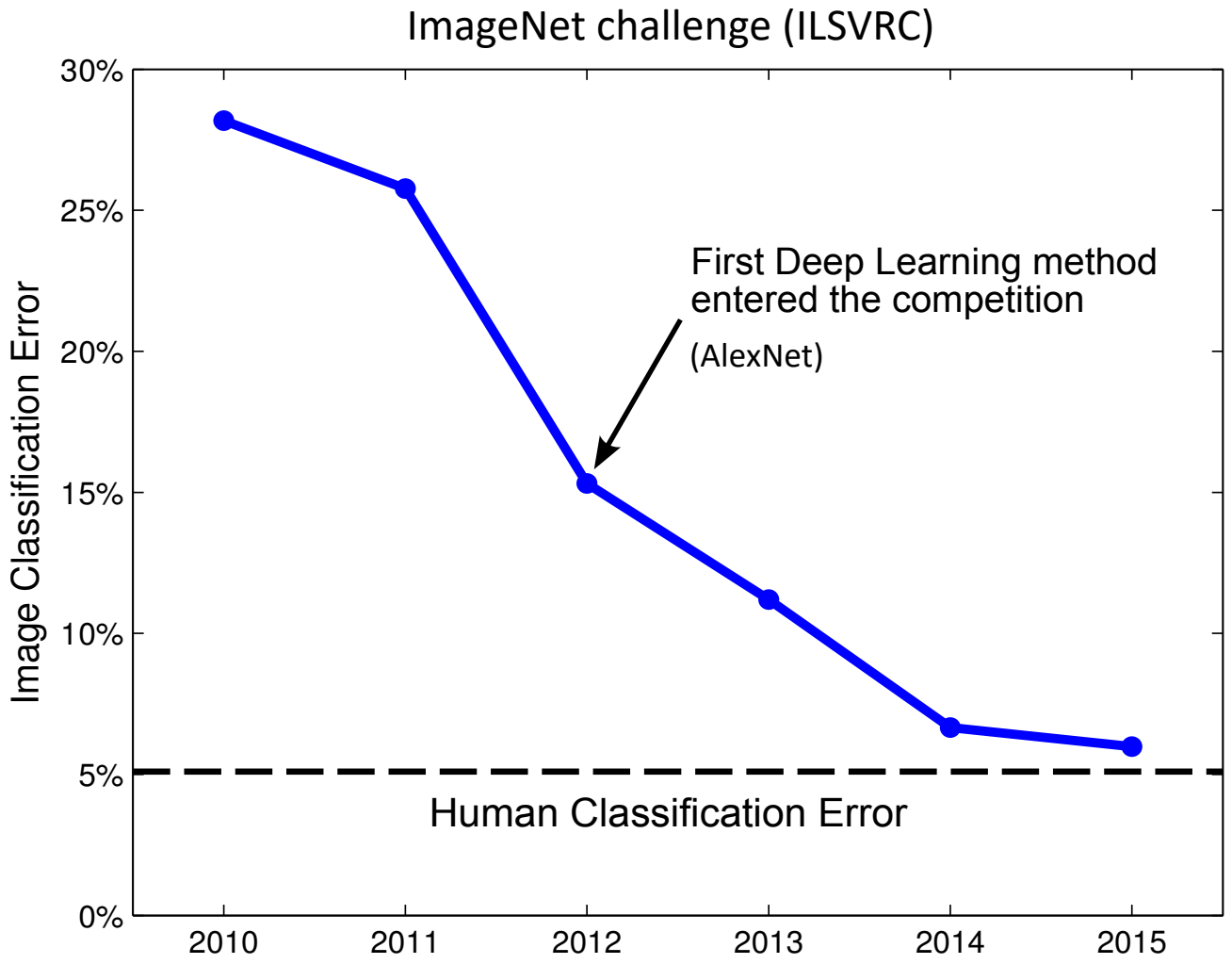
Learning hierarchies of features



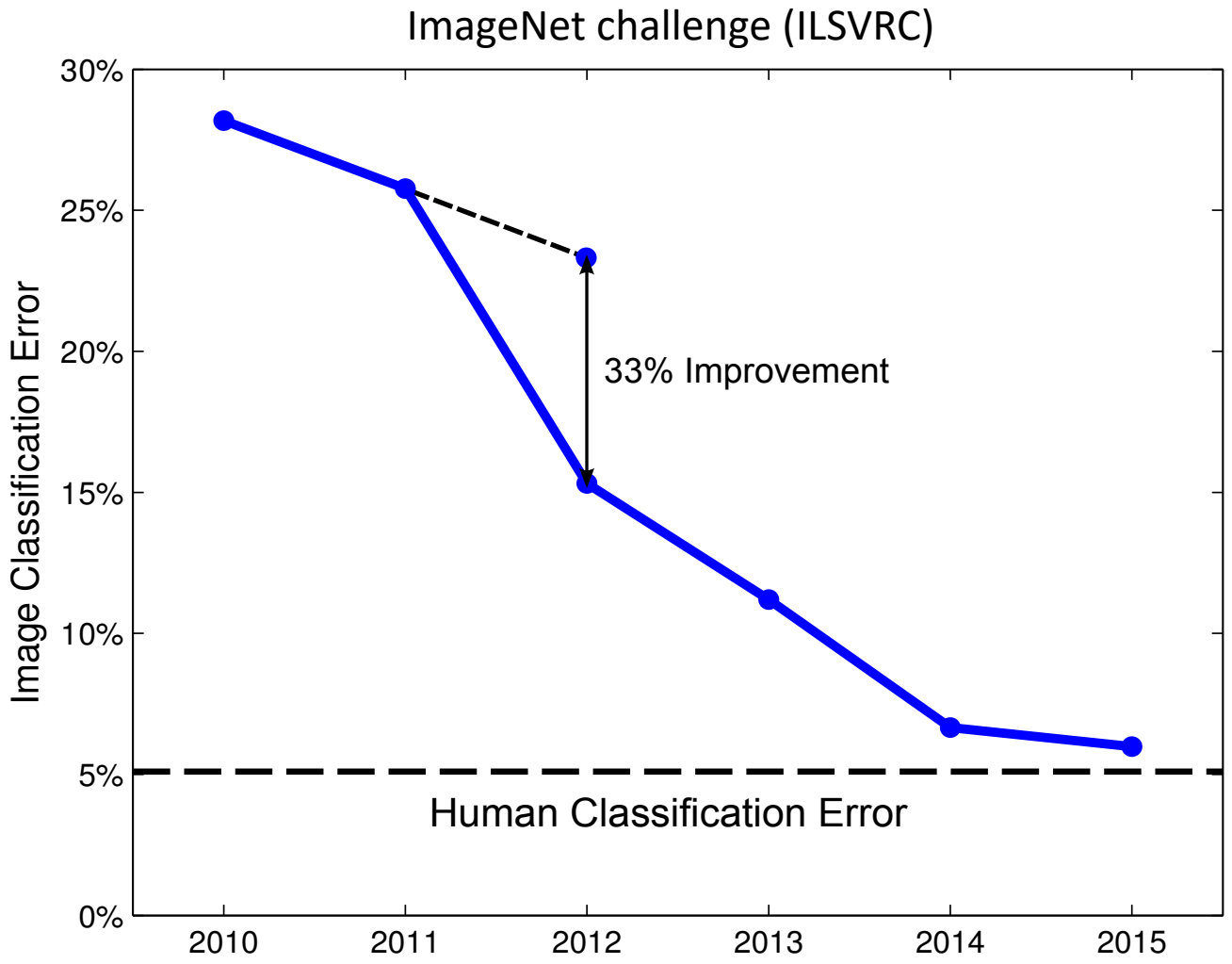
AlexNet: First breakthrough (2012)



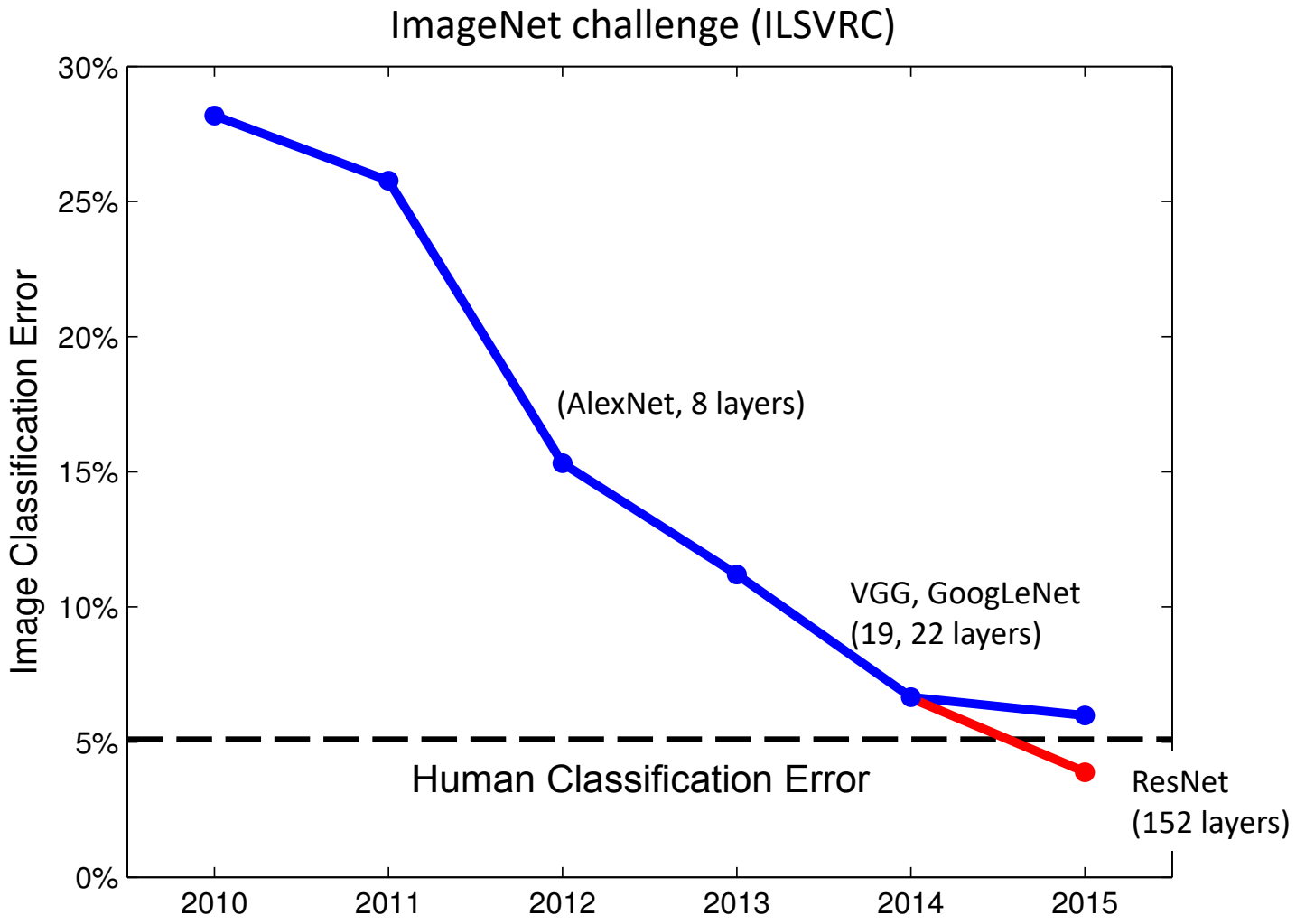
The moment CNNs became SOTA

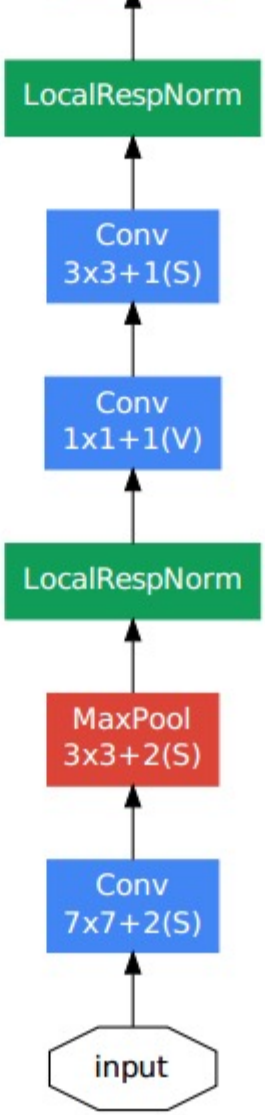


The moment CNNs became SOTA



The moment CNNs became SOTA





GoogLeNet architecture
http://arxiv.org/abs/1409.4842

PRML Readings

§5.5.6 Convolutional networks

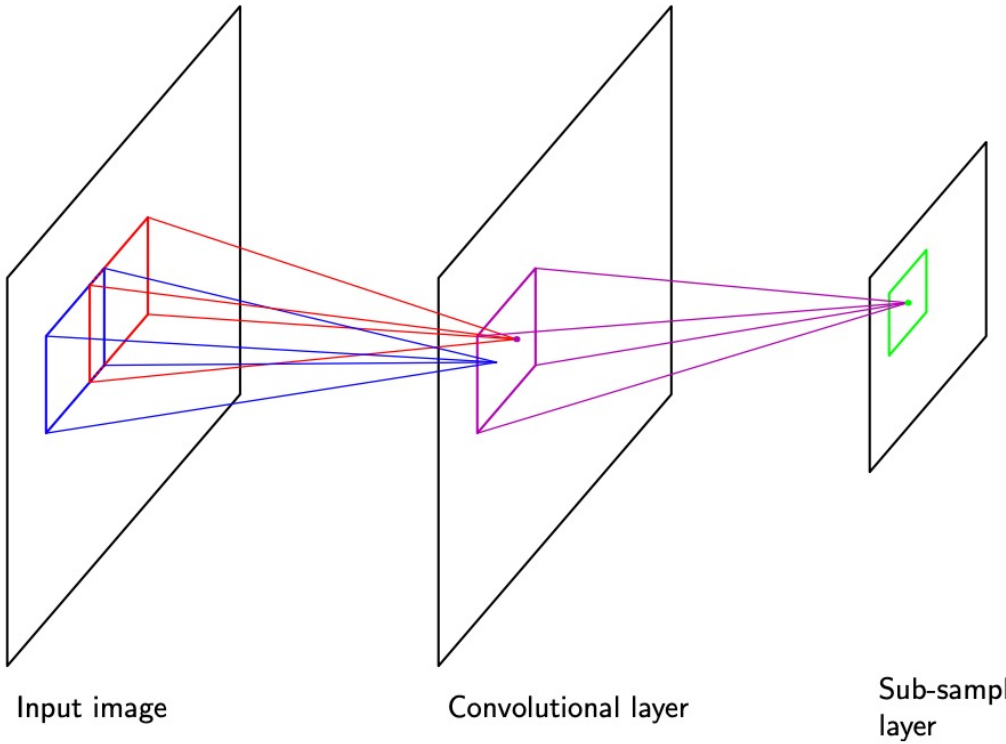


Figure 5.17 Diagram illustrating part of a convolutional neural network, showing a layer of convolutional units followed by a layer of subsampling units. Several successive pairs of such layers may be used.