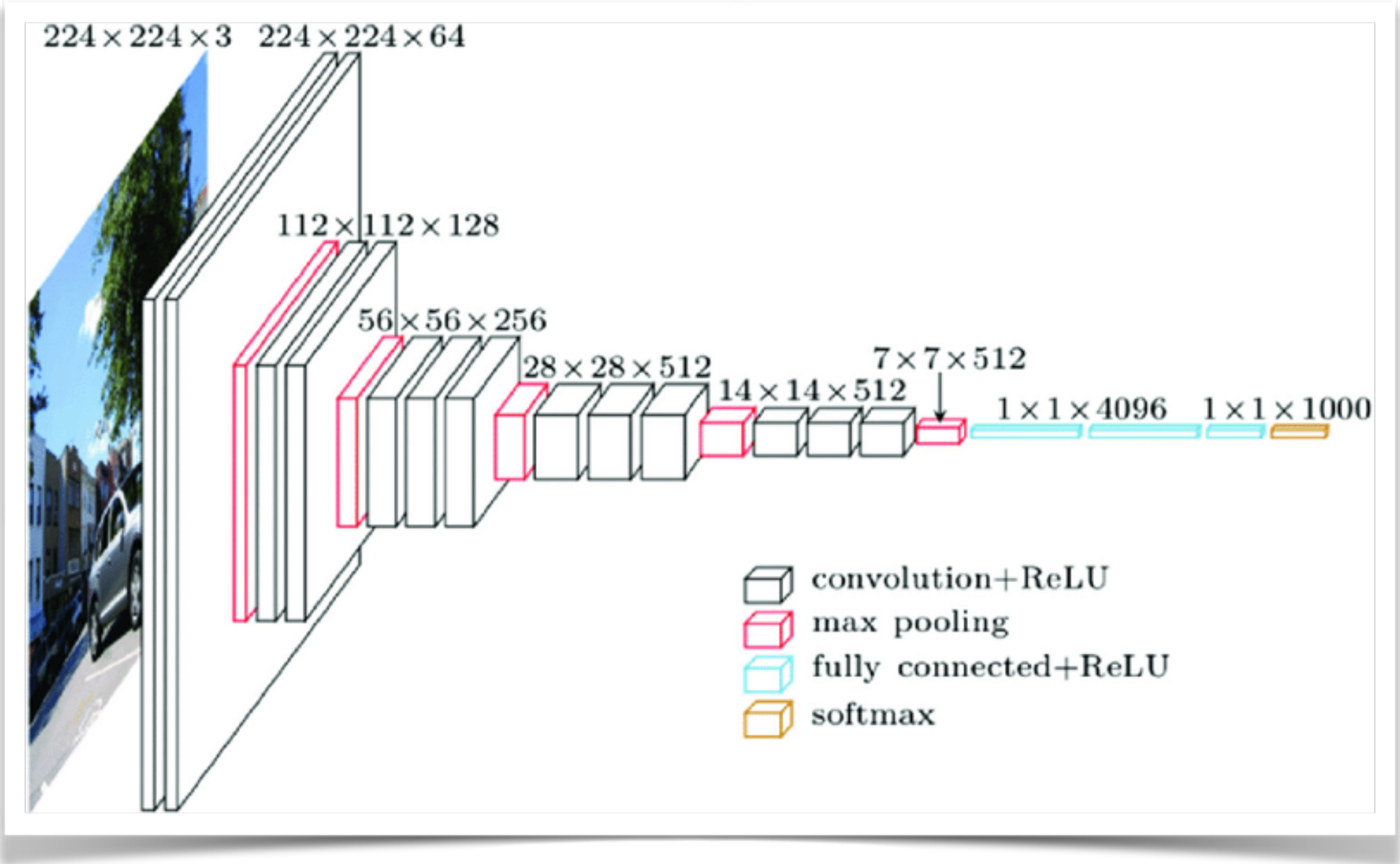
Deep Learning (BEV033DLE)
Lecture 6
Convolutional Neural Networks

Alexander Shekhovtsov

Czech Technical University in Prague

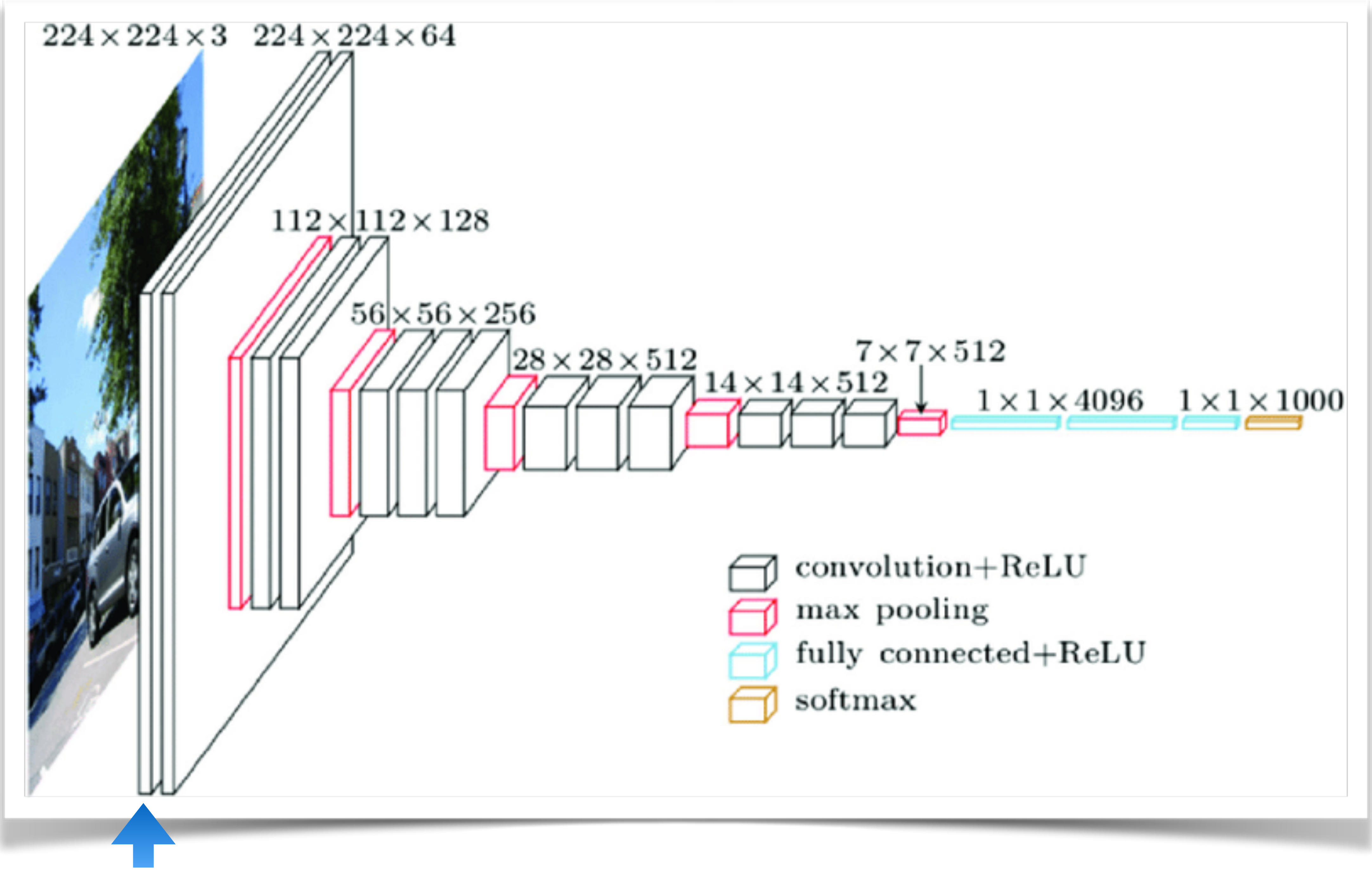
- Introduction, CNN for Classification
  - Correlation filters, translation equivariance, convolution and cross-correlation
  - Multi-channel, stride, 1x1
  - pooling, receptive field
- More CNNs
  - dilation, transposed
- Hierarchy of Parts, Visual Cortex

Classification CNN



- We'll see what this is
- Design principles
- Everything about convolutions in more detail

Classification CNN



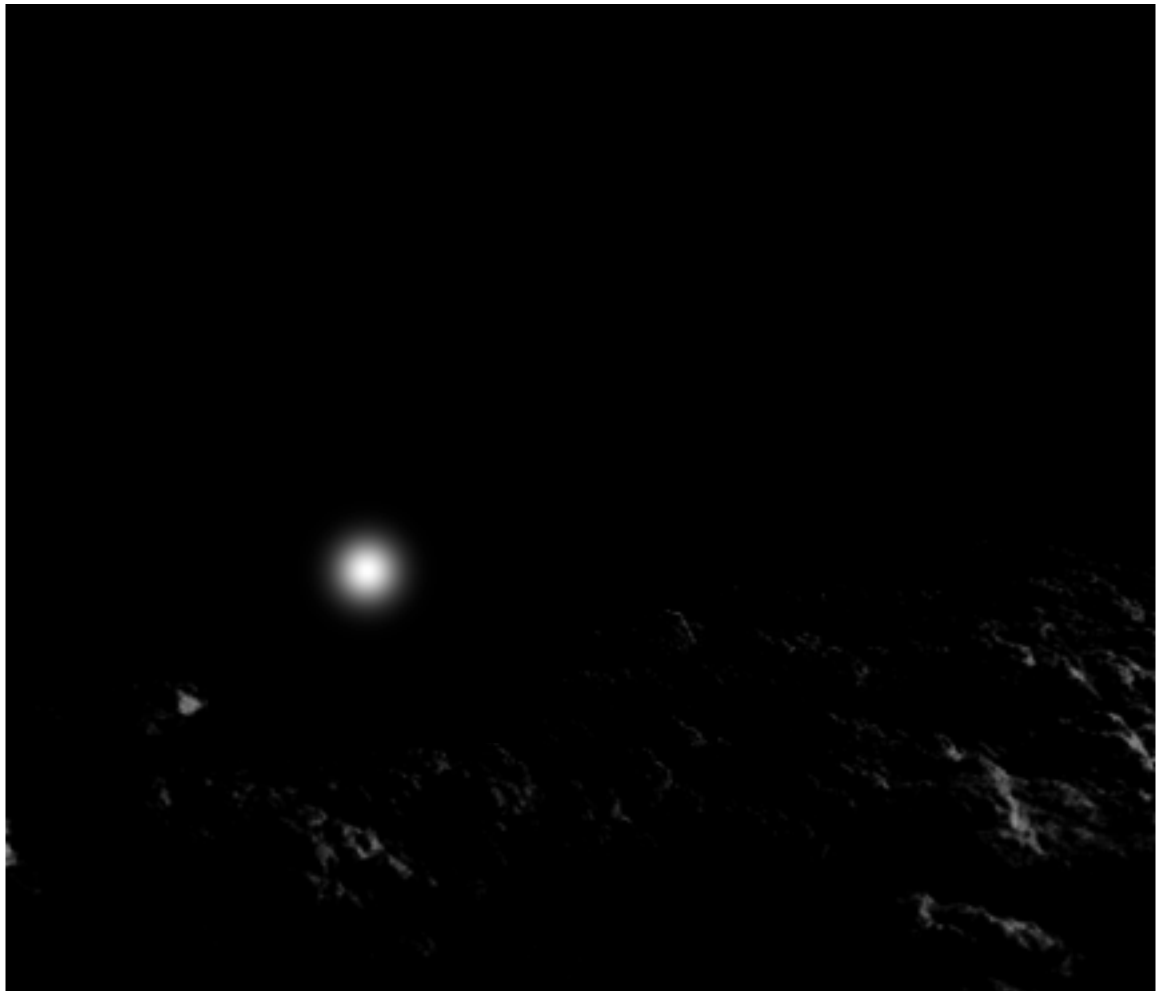
- We'll see what this is
- Design principles
- Everything about convolutions in more detail

Introduction

Template

Image

Response of the correlation filter

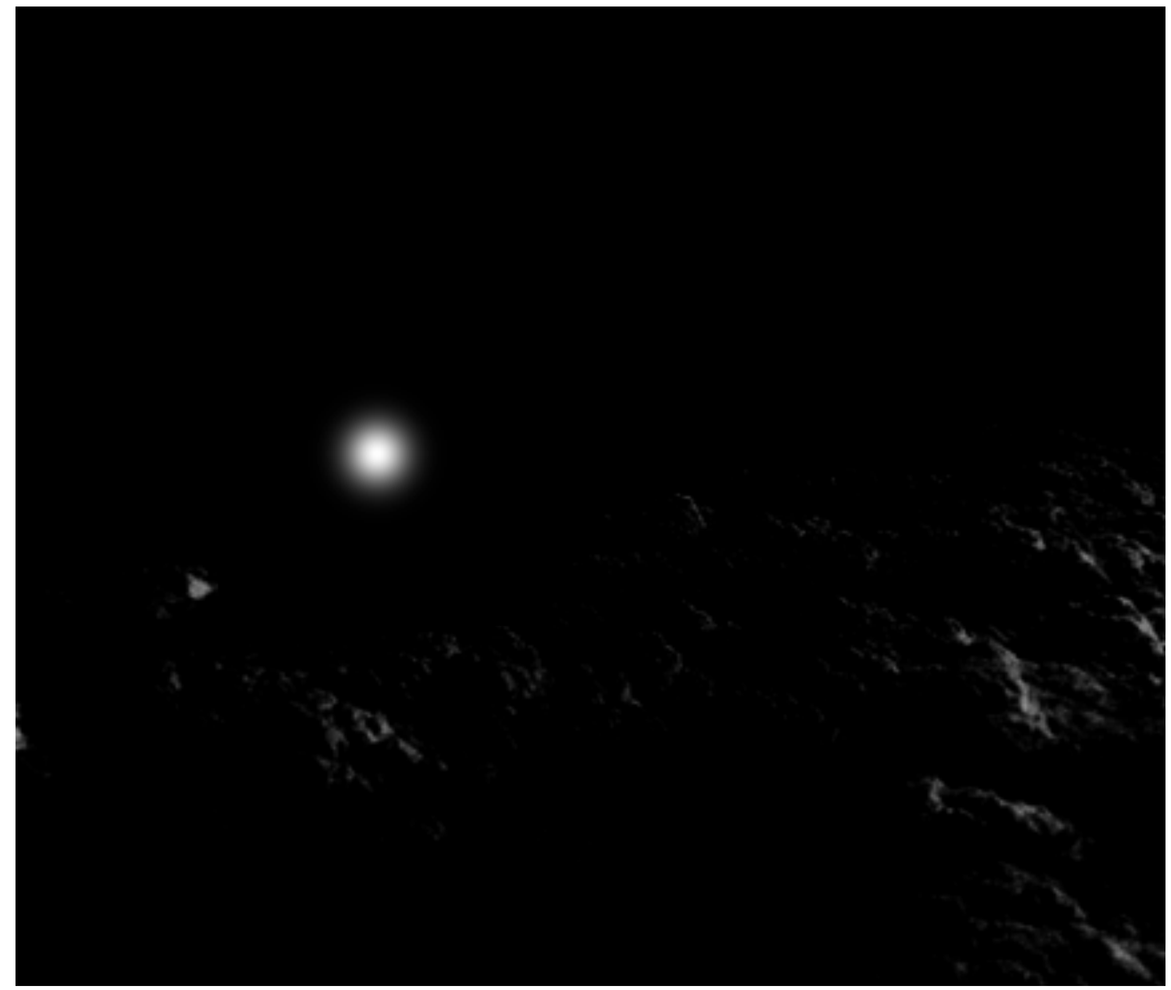


Introduction

Template

Image

Response of the correlation filter



- Translational equivariance idea: when the input shifts, the output shifts
  - Would be hard to achieve if the image was given as a general vector — we are using 2D grid structure and require that all locations are treated equally

Basics

- Convolution and Correlation (1D)
  - Convolution $y = w * x$: $y_j = \sum_{k=-h}^{h} w_k x_{j-k} = \sum_{k=-h}^{h} w_{-k} x_{j+k}$
  - Cross-correlation $y = w \star x$: $y_j = \sum_{k=-h}^{h} w_k x_{j+k}$

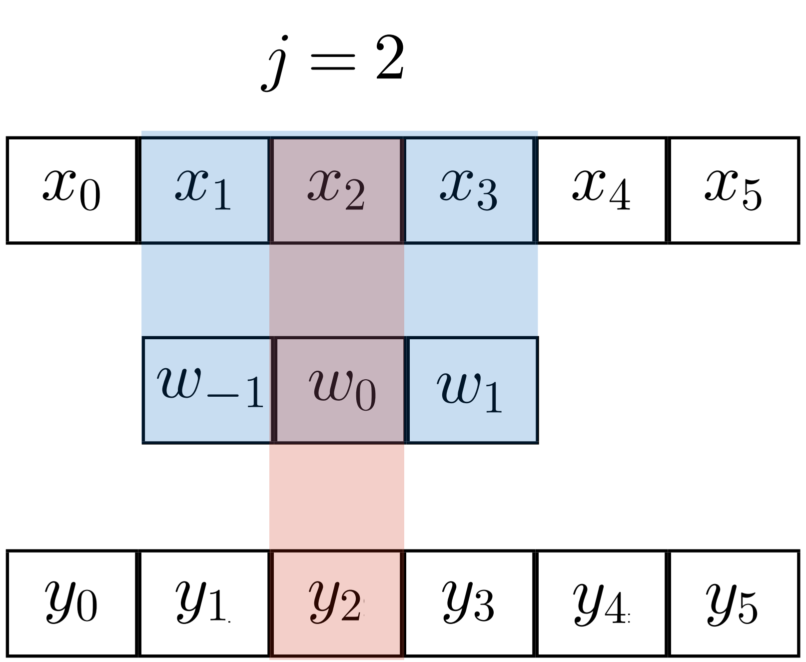
flip of the weight matrix

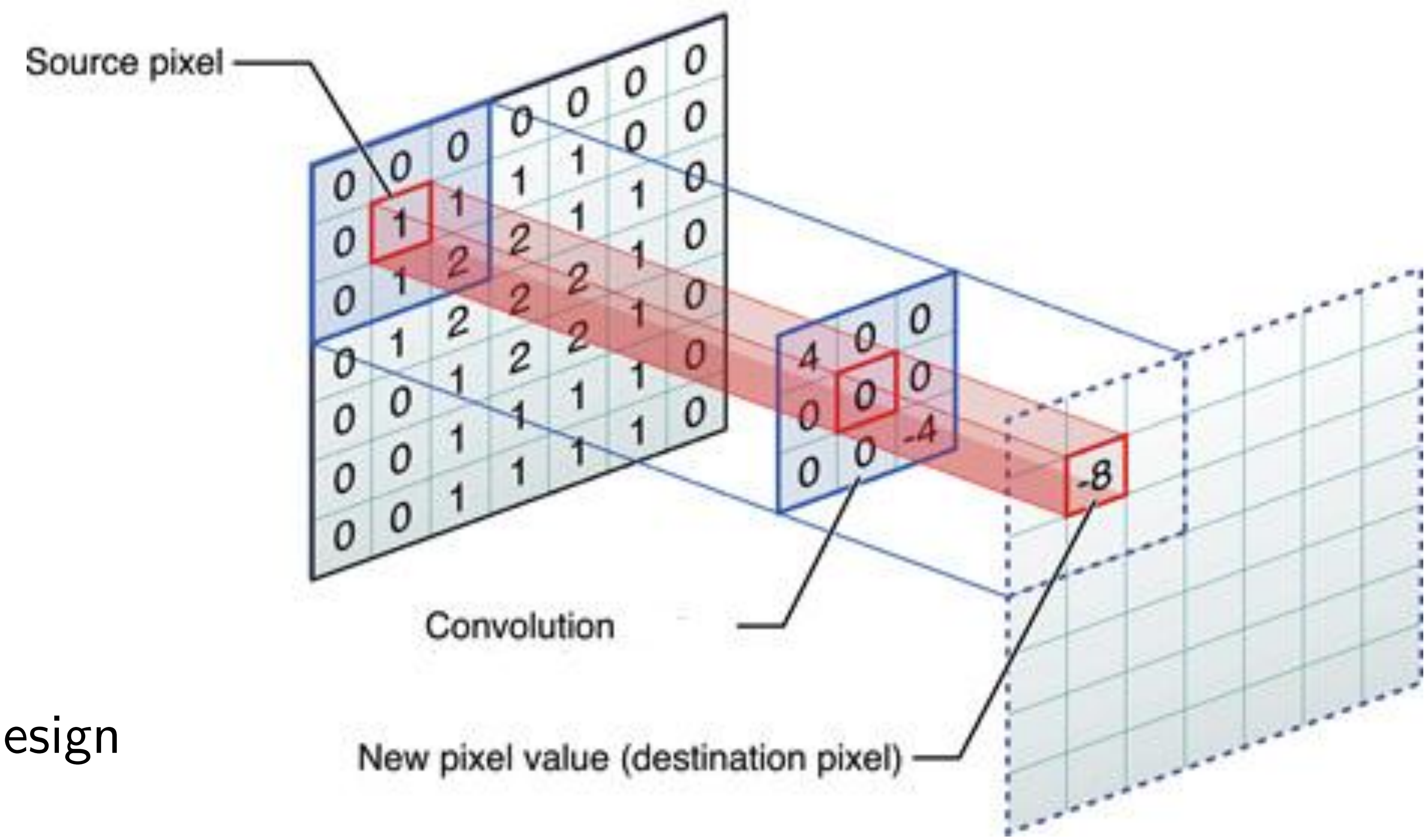
output

weights kernel

input

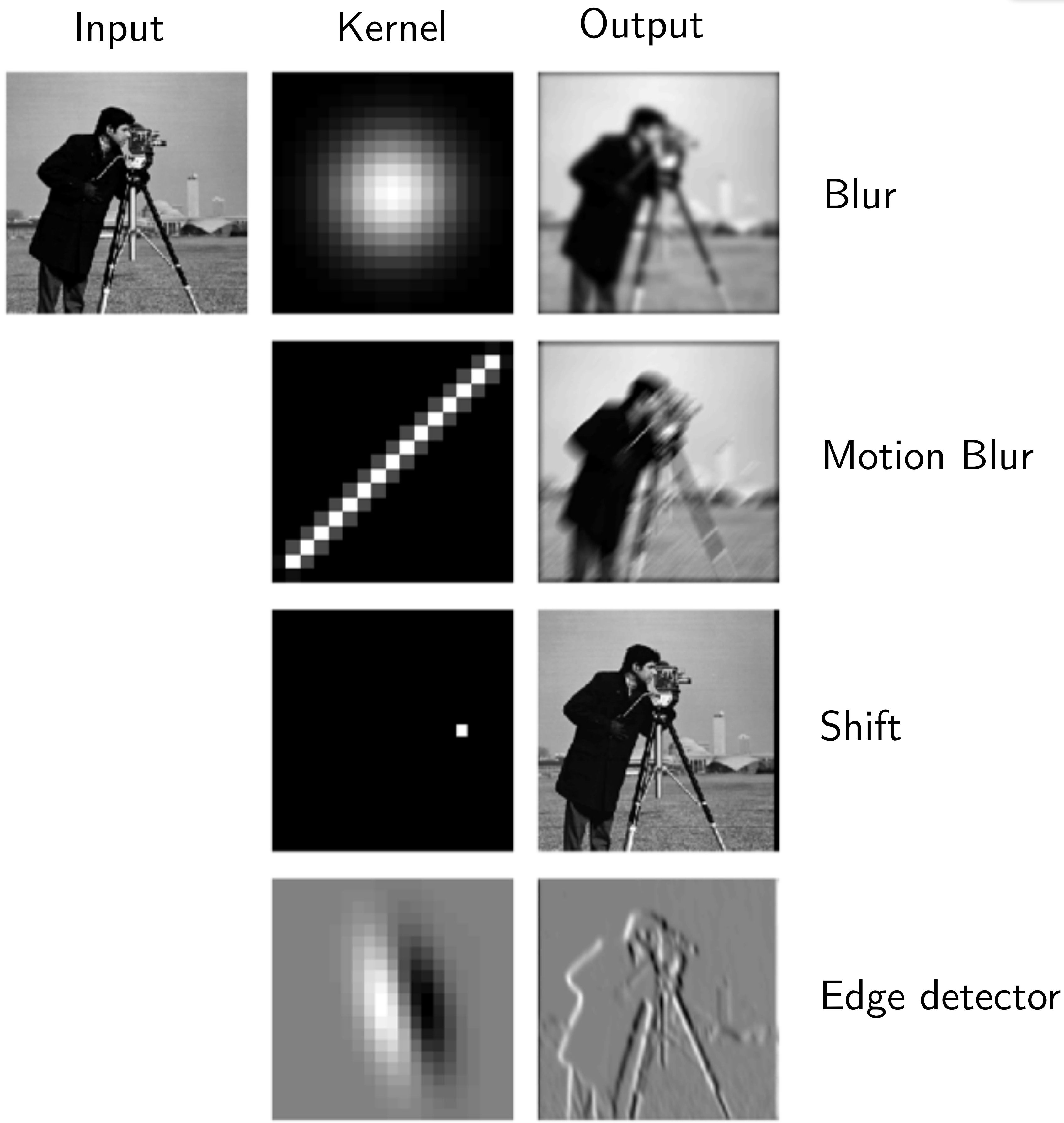
Easily convertible, more convenient to consider cross-correlation in Deep Learning





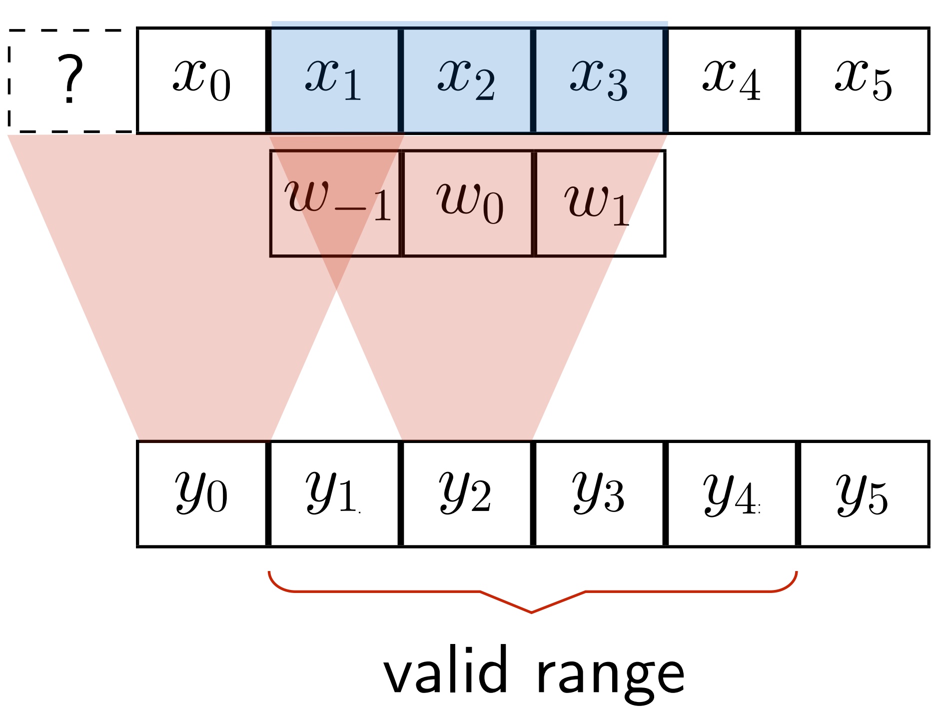
- Translation equivariance by design

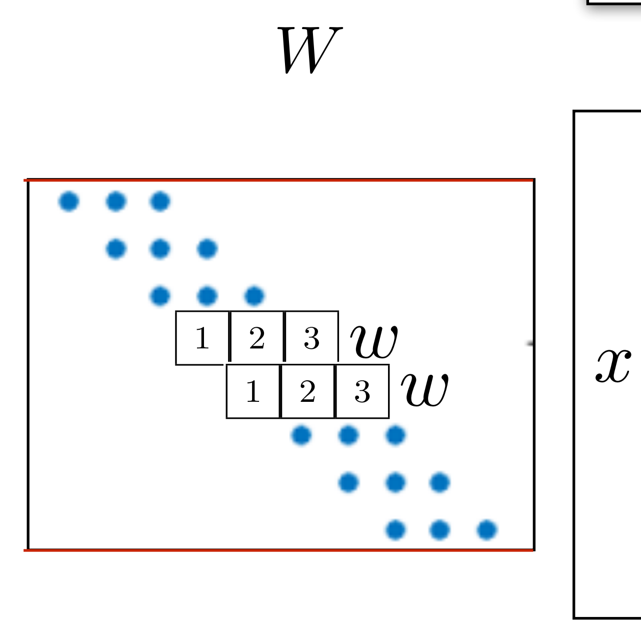
Examples



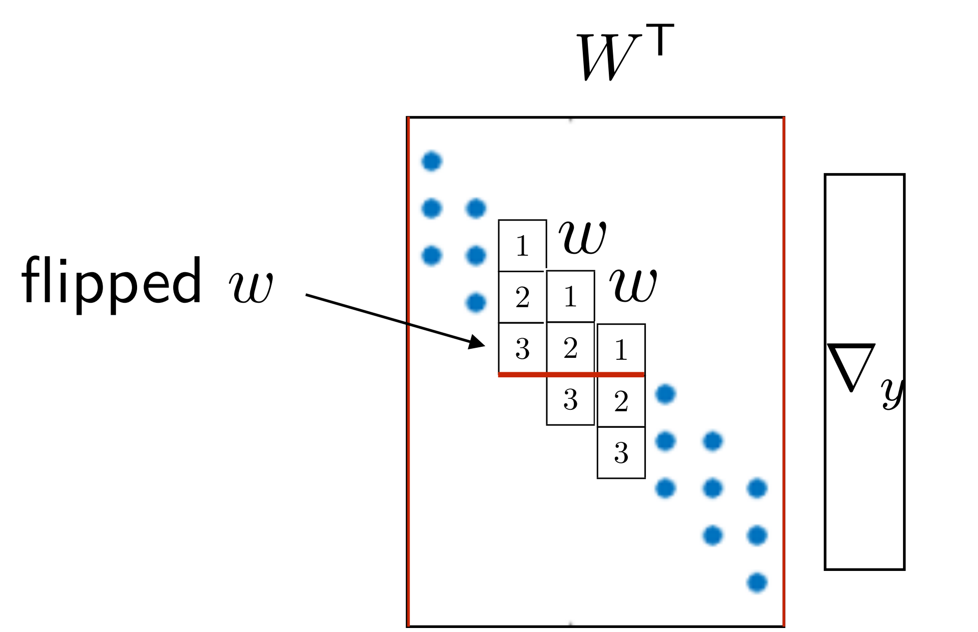
Properties

- Cross-Correlation:
  - $y_i = \sum_{k=-h}^{h} w_k x_{i+k}$
- As matrix-vector product: $y_i = \sum_j w_{j-i} x_j = \sum_j W_{i,j} x_j$
  - Relation: $j = i + k \Rightarrow k = j - i$
  - **Compact representation** of certain linear transforms
  - Everything that applies to linear transforms applies to convolution and cross-correlation
- **Valid** range for i:
  $0 \leq j \leq n \Rightarrow 0 \leq i - h,\ i + h \leq n \Rightarrow h \leq i \leq n - h.$
  - Optionally may **pad** input with zeros to obtain **same** range as unpadded input





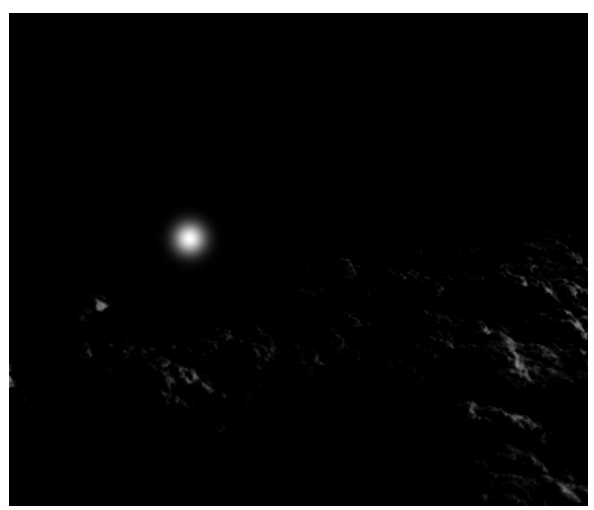
Example:
known rule for backprop:



=convolution (with padding)

Properties

- As a binary operation $y = w * x$
  - Everything that applies to linear operators, eg. associativity: $u * (w * x) = (u * w) * x$
  - Commutativity for convolutions: $w * x = x * w$:
    $\sum_k w_k x_{i-k} = \sum_j x_j w_{i-j}$
  - No commutativity for cross-correlation. But $u \star w \star x = w \star u \star x$
- Examples:
  - edge_filter(blur(image)) = blur(edge_filter(image)) = blur(edge_filter)(image)
  - filter(translation(image)) = translation(filter(image))
    equivariance w.r.t. translation

When the image shifts, the output shifts

Great prior knowledge for learning

($\star$) Can you show equivariance of convolution to sub-pixel displacements?

- In fact, linearity + translation-equivariance = convolution

Backprop

- New notation for the gradient:
  - $\text{đ}y_i \equiv \frac{dL}{dy_i}$ (previously denoted with ∇_{y_i})

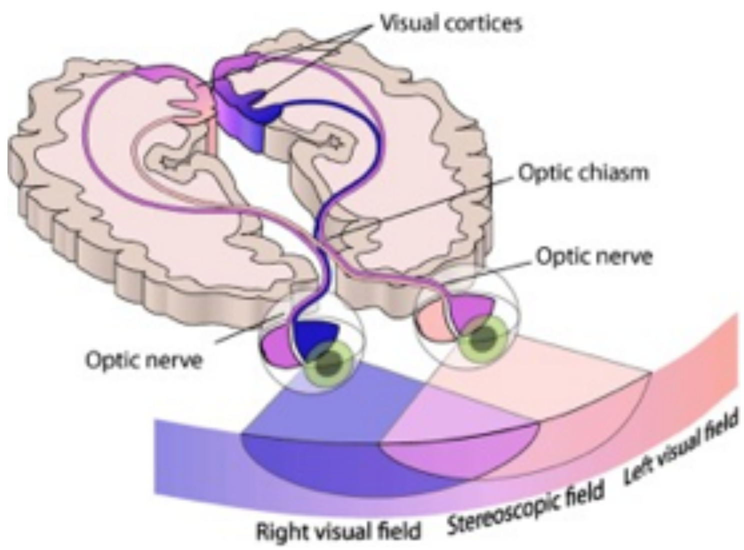
- Backprop of the cross-correlation $y = w \star x$ is convolution:
  - $y_i = \sum_k w_k x_{i+k} = \sum_j w_{j-i} x_j$
  - $\text{đ}x_j := \sum_i \frac{\partial y_i}{\partial x_j} \text{đ}y_i = \sum_i \frac{\partial}{\partial x_j} \Big(\sum_{j'} w_{j'-i} x_{j'} \Big) \text{đ}y_i = \sum_i w_{j-i} \text{đ}y_i = (\text{đ}y * w)_j = (w * \text{đ}y)_j$

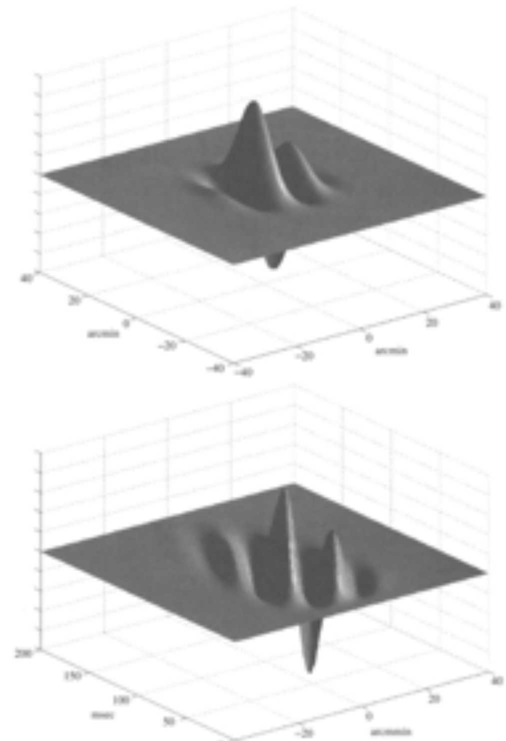
- Backprop of convolution $y = w * x$ is cross-correlation:
  - $y_i = \sum_k w_k x_{i-k} = \sum_j w_{i-j} x_j$
  - $\text{đ}x_j := \sum_i \frac{\partial y_i}{\partial x_j} \text{đ}y_i = \sum_i w_{i-j} \text{đ}y_i = (\text{đ}y \star w)_j$

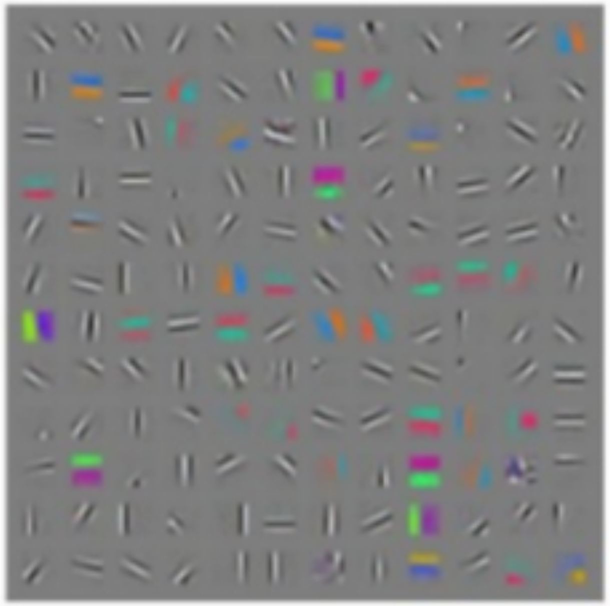
First Convolutional Layer

- Large filters are not very useful:
  - think of viewpoint changes, object deformations, variations within a category
  - small filters capture elementary features according to statistics of natural images

V1 simple cell responses —closely modeled by→ Gabor Filters (designed)

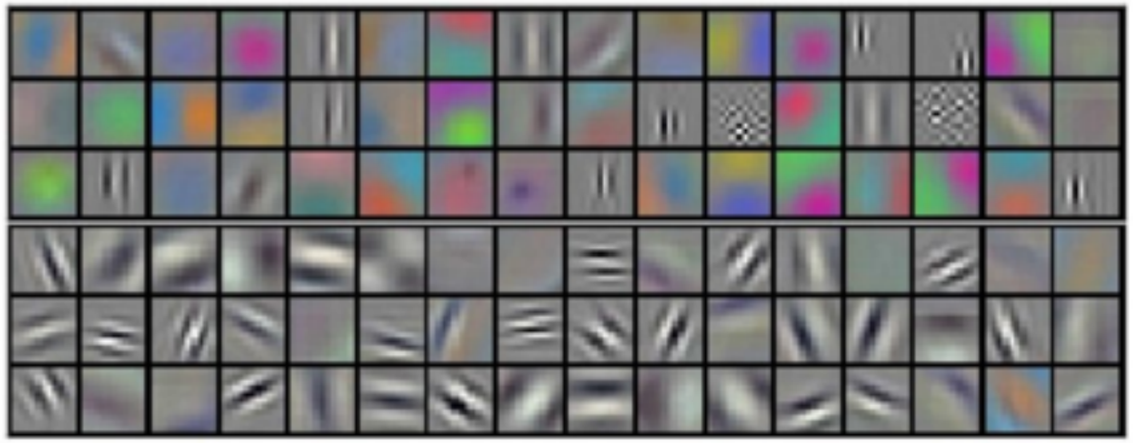






PCA of Image Patches
(natural image statistics)

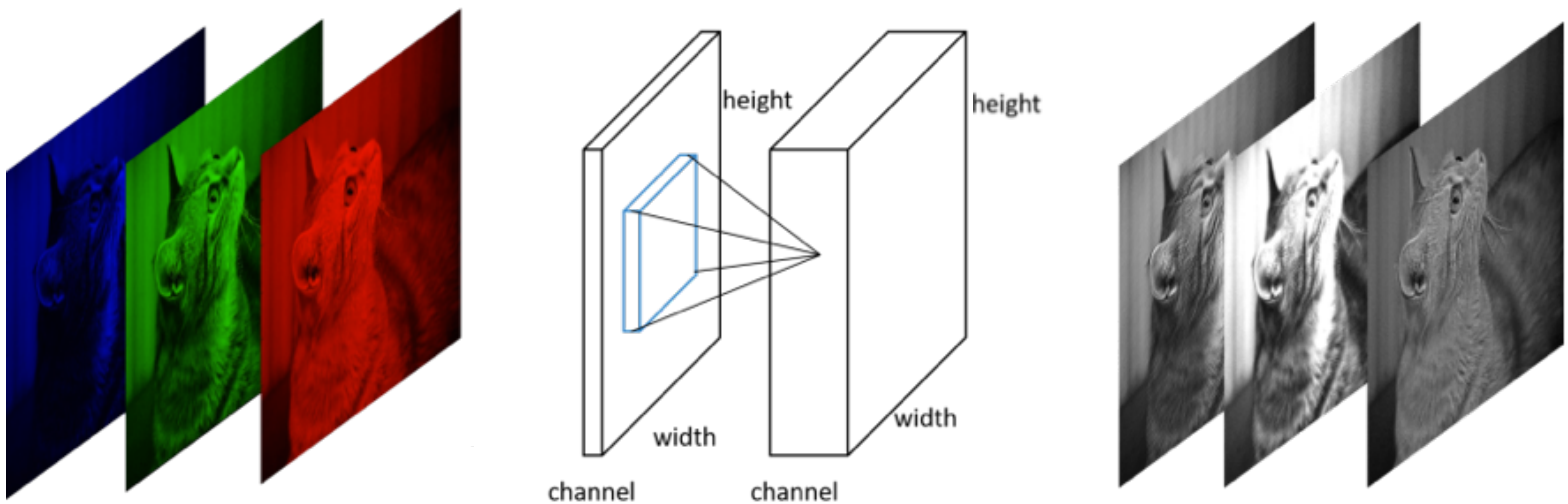
CNN first layer filters (learned)



Multi-Channel Convolution

- We just had:
  - color input images -> convolution kernel needs to have 3 channels
  - stack of filters -> channels of the output

feature map



- Multi-channel cross-correlation:

$$\sum_{c} \sum_{\Delta i} \sum_{\Delta j} x_{c,i+\Delta_i,j+\Delta j}\; w_{o,c,\Delta i,\Delta j} = y_{o,i,j}$$

input channel — filter spatial dimensions — output channel

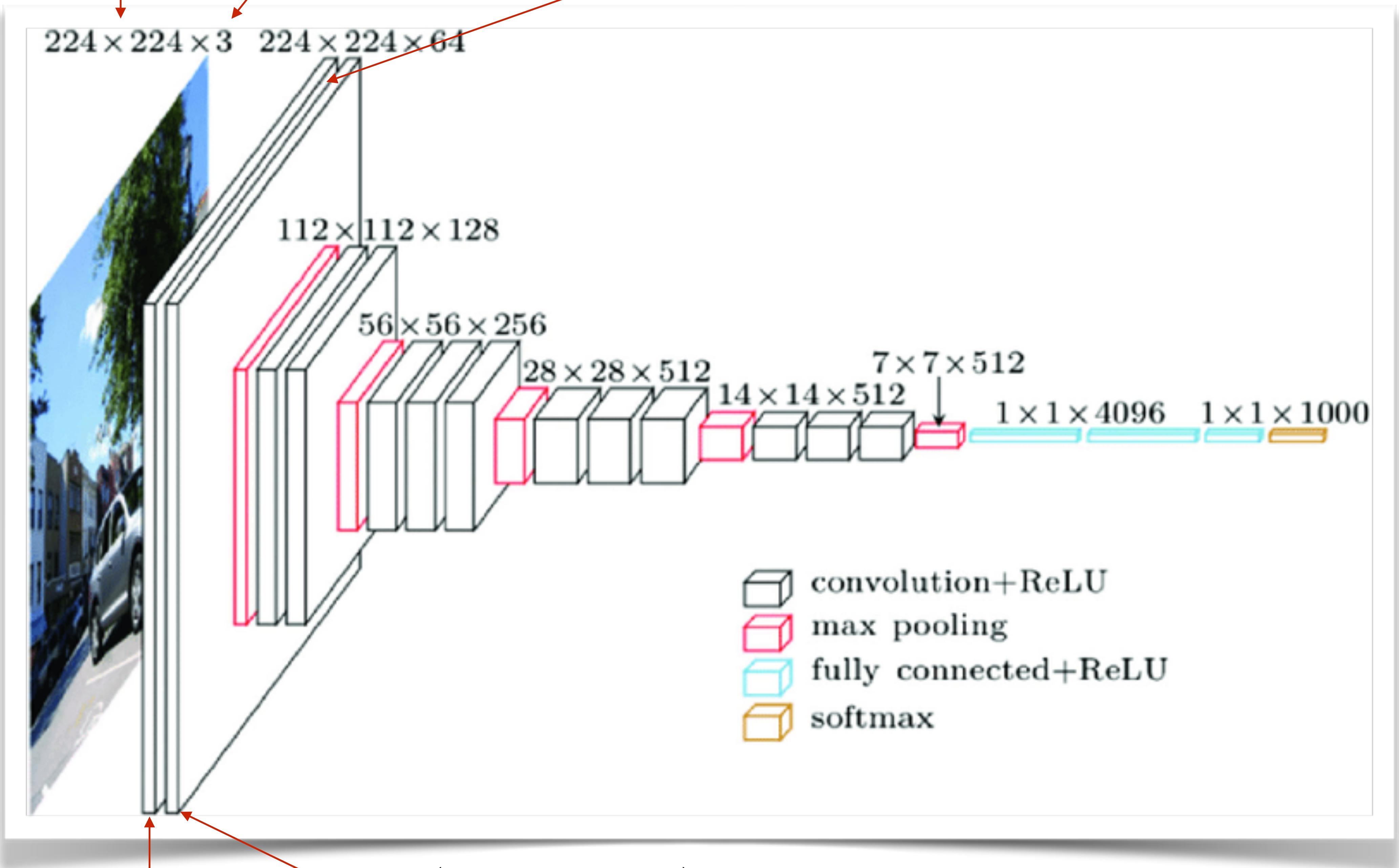
- input is 3D tensor, weight is 4D tensor, output is 3D tensor
- Essentially: a cross-correlation on spatial dims and fully connected on channel dims

Classification CNN

Spatial size of the input image

channels

feature maps



$\mathrm{conv}(3{\times}3,\, 64{\to}64)$

Result of $\mathrm{conv}(K{\times}K,\, 3{\to}64)$ followed by ReLU

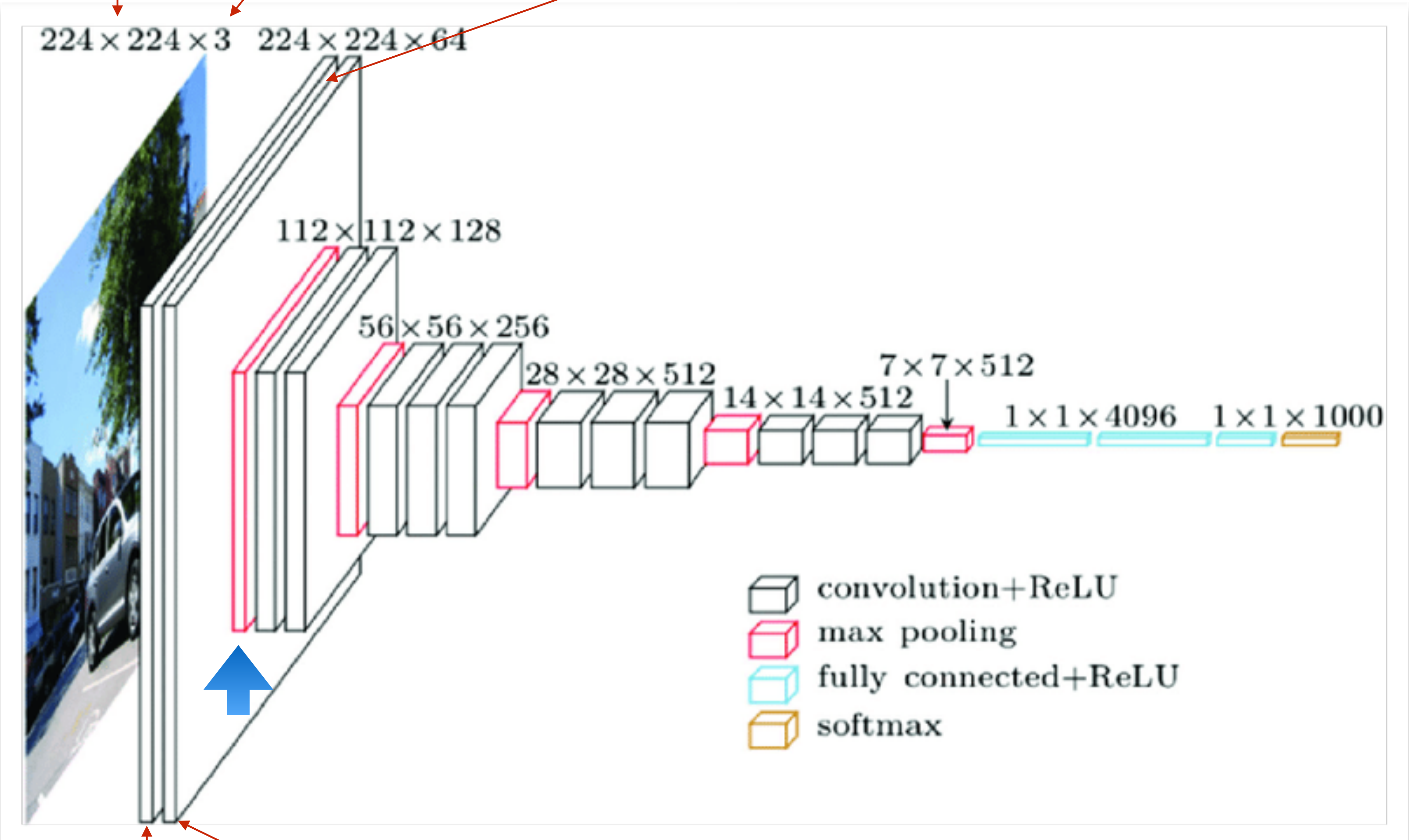
- Eventually want to classify -> need to reduce spatial dimensions

Classification CNN

Spatial size of the input image

channels

feature maps



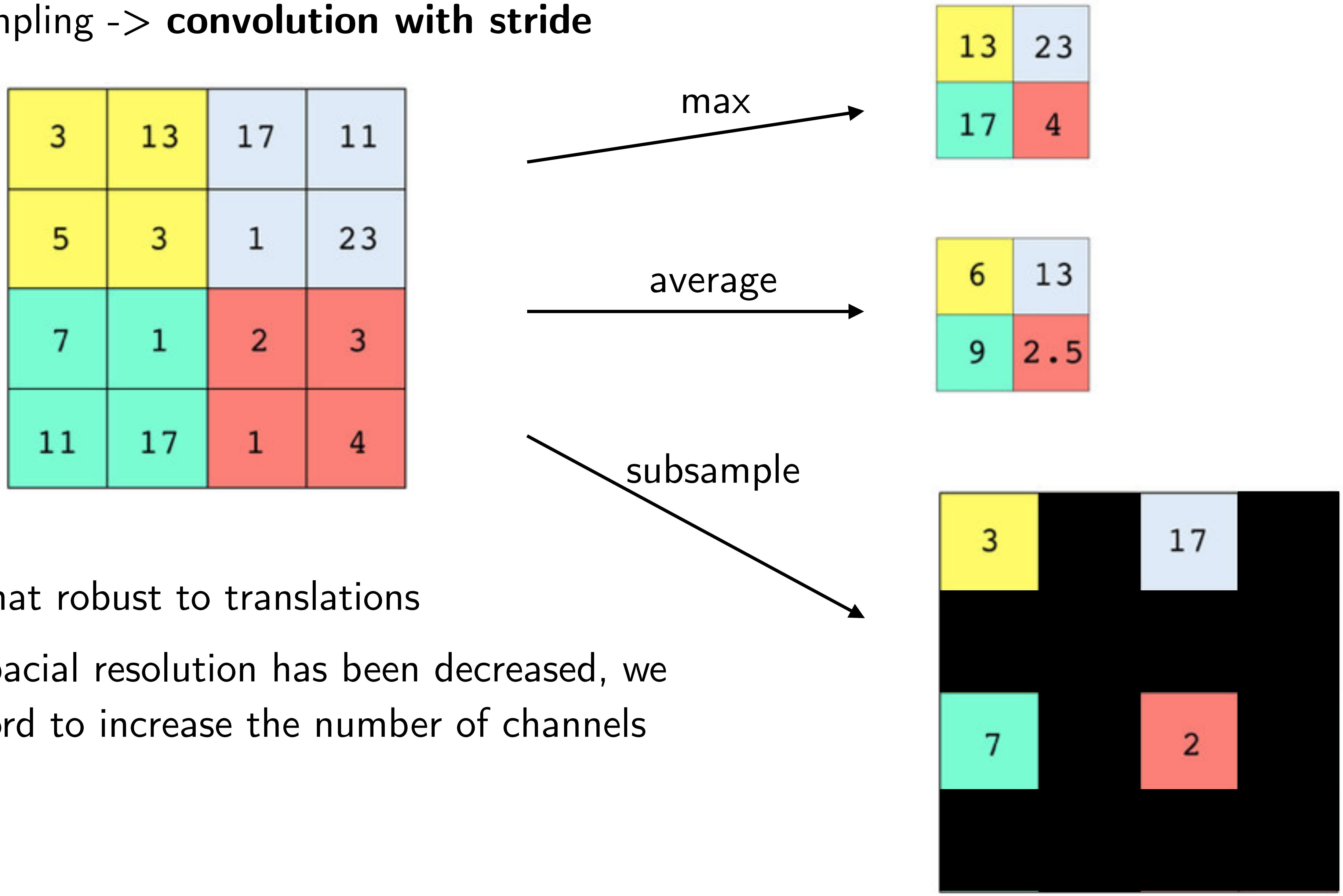
conv(3×3, $64\rightarrow64$)

Result of $\mathrm{conv}(K\times K, 3\rightarrow64)$ followed by ReLU

- Eventually want to classify -> need to reduce spatial dimensions

Pooling

- Following approaches are used to reduce the spatial resolution:
  - **max pooling**
  - **average pooling**
  - subsampling -> **convolution with stride**



- Somewhat robust to translations
- Once spacial resolution has been decreased, we can afford to increase the number of channels

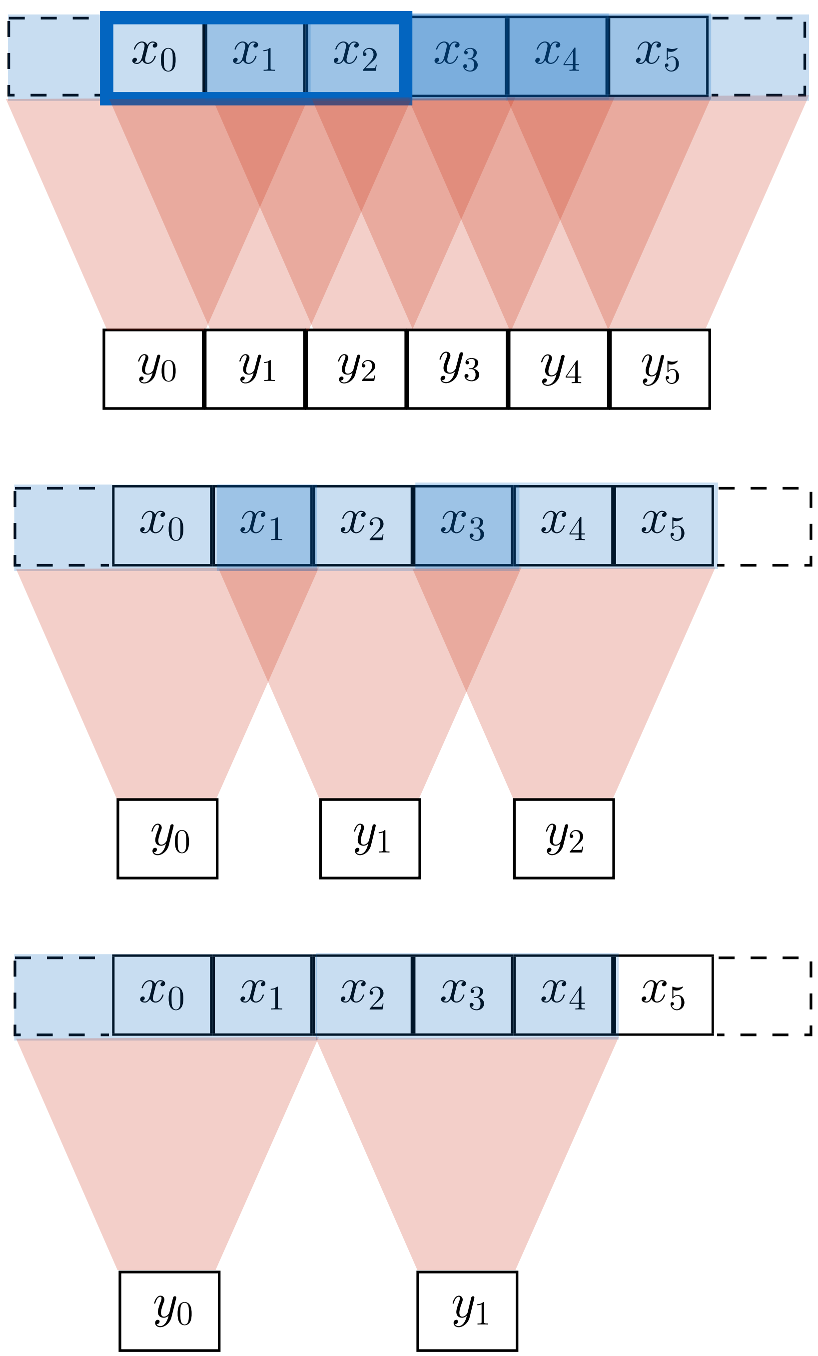
Convolution with Stride

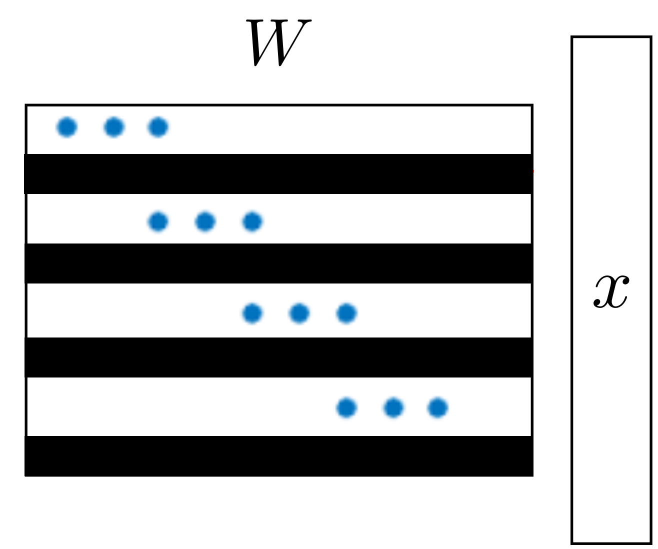
- Full convolution + subsampling is equivalent to calculating the result at the required locations only, stepping with a **stride**

Stride 1

Stride 2

Strid 3

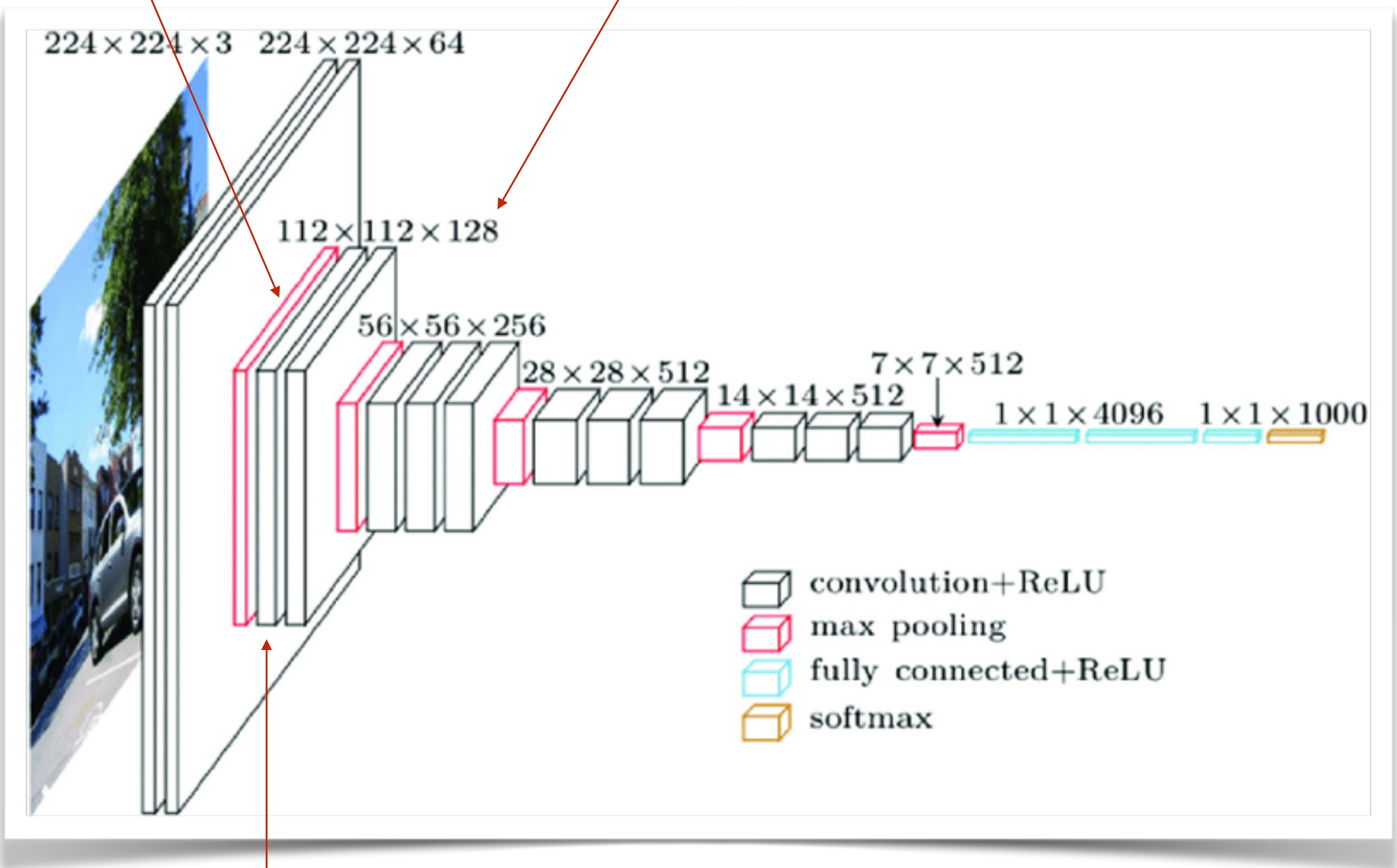




All variants and detail: [Dumoulin, Visin (2018): A guide to convolution arithmetic for deep learning]

Classification CNN

Reduced spatial size can afford more channels

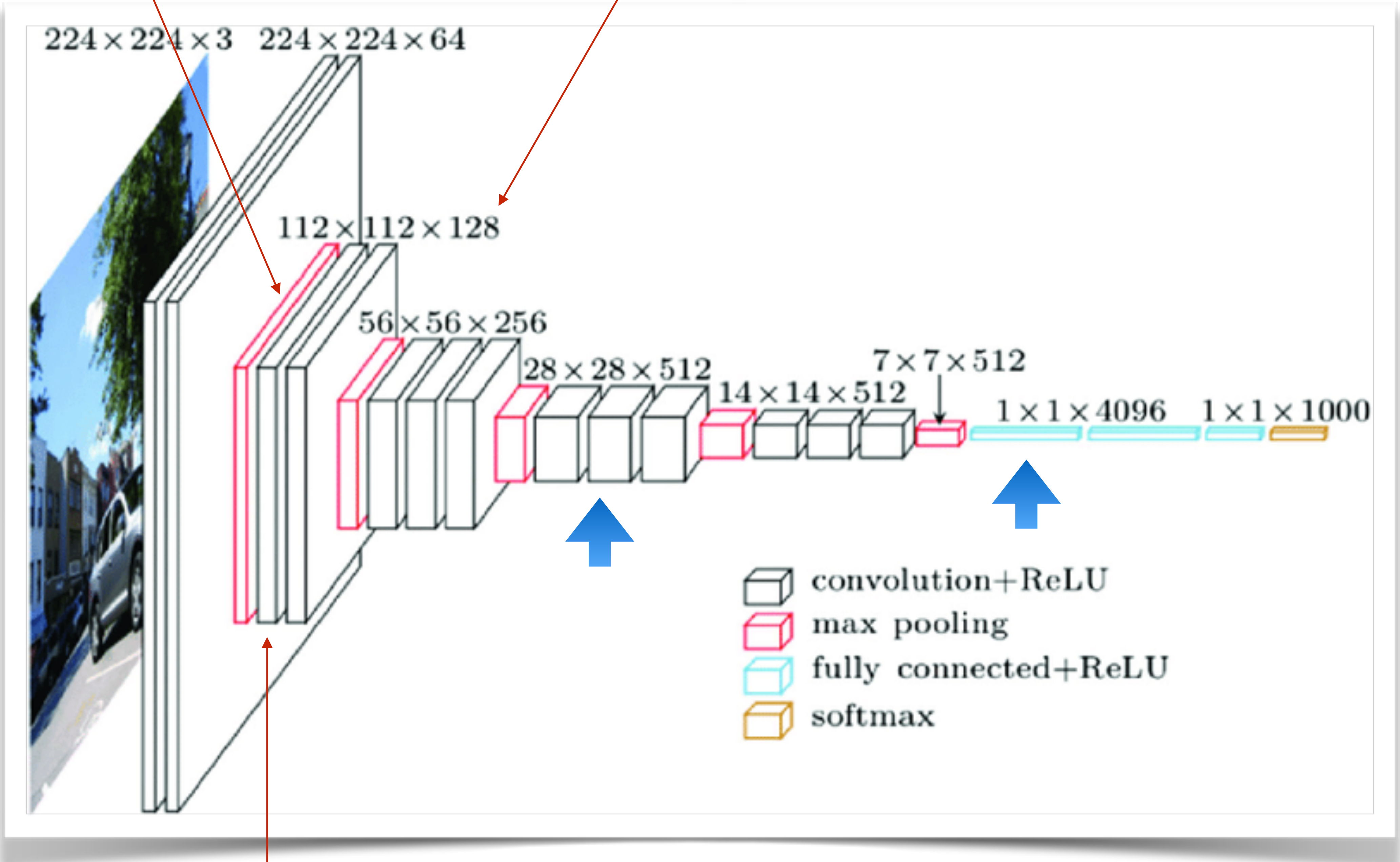


$\mathrm{conv}(3\times3, 64\rightarrow128)$

- Combining convolutions and spatial pooling increases units receptive field

Classification CNN

Reduced spatial size can afford more channels

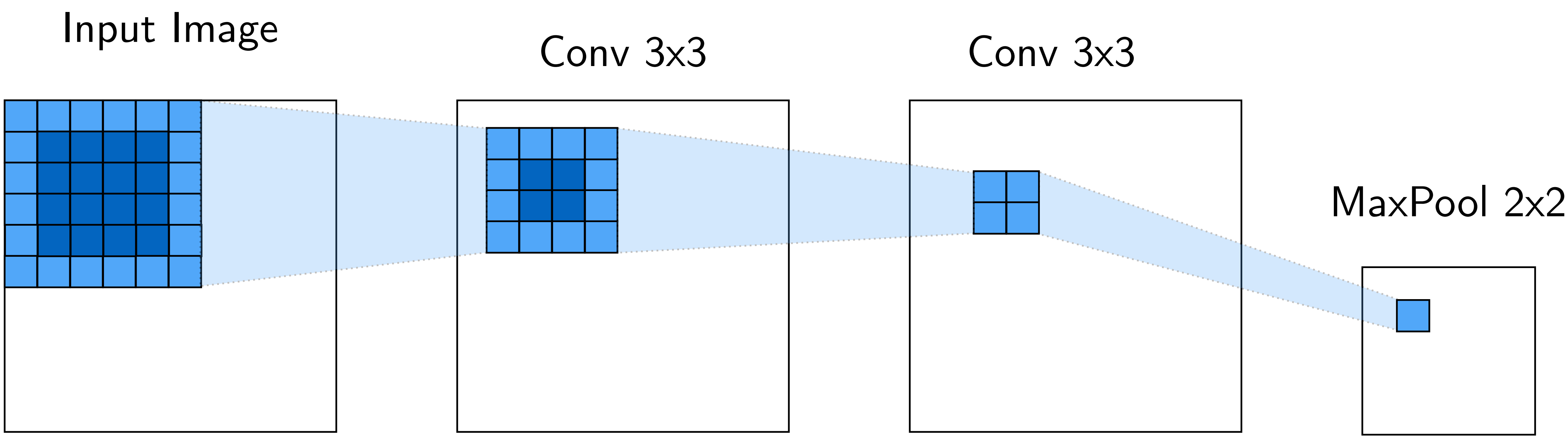


$\mathrm{conv}(3{\times}3,\, 64{\rightarrow}128)$

- Combining convolutions and spatial pooling increases units receptive field

Receptive Field

✦ Receptive Filed = pixels in the input which contribute to the specific output



✦ Small convolutions are not sufficient to building up the receptive field. Example:

- Want to classify images of size 256x256
- Each 3x3 convolution increases the receptive filed by 2 pixels
- Would take 128 convolutional layers

✦ Need pooling / strides / larger filters

Weight Kernel Sizes

- With pooling we reduced the size of feature maps. What about filter kernels?
  - First layer: $(7 \times 7,\ 3 \to 64) \approx 10^3$ – can afford large filter size
  - Second layer: $(3 \times 3,\ 64 \to 64) \approx 3 \cdot 10^4$ – small filter size preferable
  - Layers with more channels: $(3 \times 3,\ 256 \to 256) \approx 5 \cdot 10^5$ – become expensive
- Need further efficient parametrization techniques
  - Depth-wise separable convolutions:
    $\mathrm{conv}(K \times K, 1 \to 1)$ composed with $(1 \times 1, C \to C)$
  - More general:
    $\mathrm{conv}(K \times K, S \to S)$ composed with $(1 \times 1, C \to S)$, $S < C$

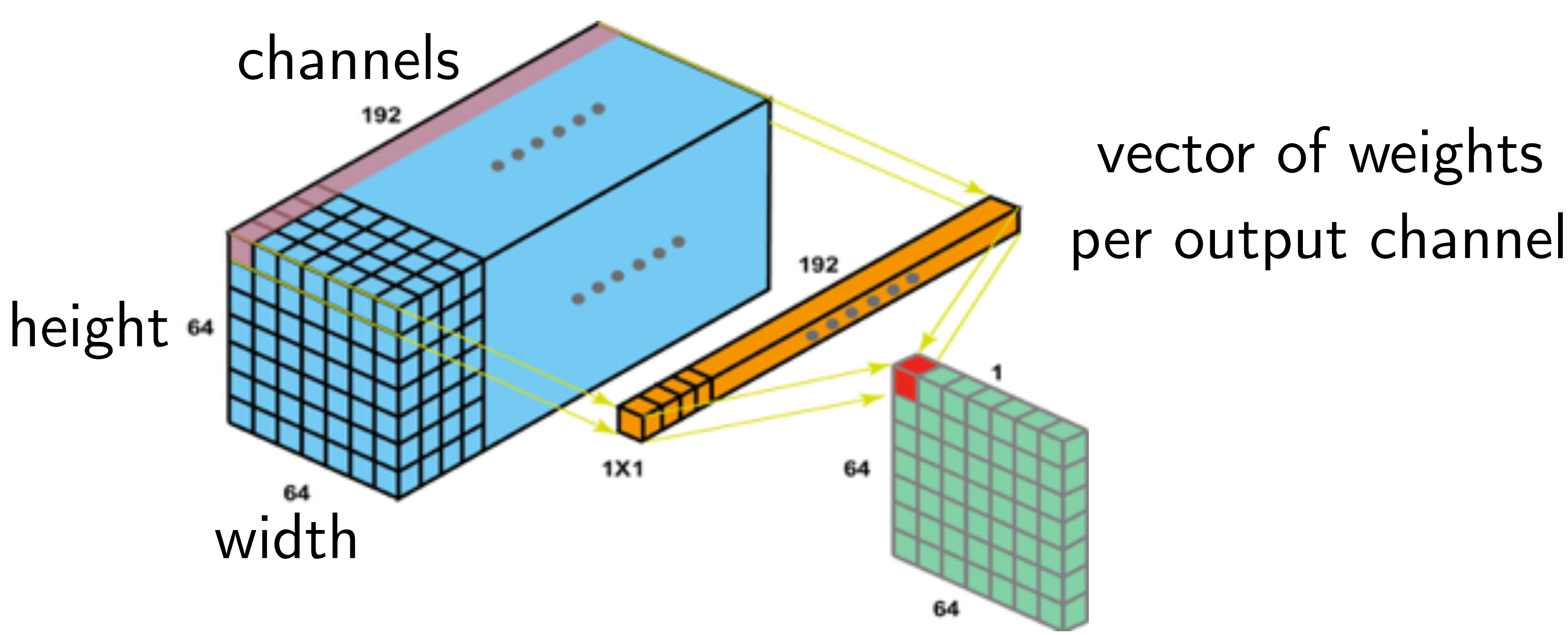
1x1 Convolution

- Kernel size 1×1:

$$y_{o,i,j} = \sum_{c} \sum_{\Delta i=0} \sum_{\Delta j=0} w_{o,c,\Delta i,\Delta j}\; x_{c,i+\Delta i,j+\Delta j}$$

$$= \sum_{c} w_{o,c,0,0}\; x_{c,i,j}$$

- For all i,j a linear transformation on channels with a matrix $w_{o,c,0,0}$



Example 3×3, $256\to256$, is too expensive, simplify:

Input: WxHx256

$\mathrm{conv}(1\times1, 256 \to 64)$

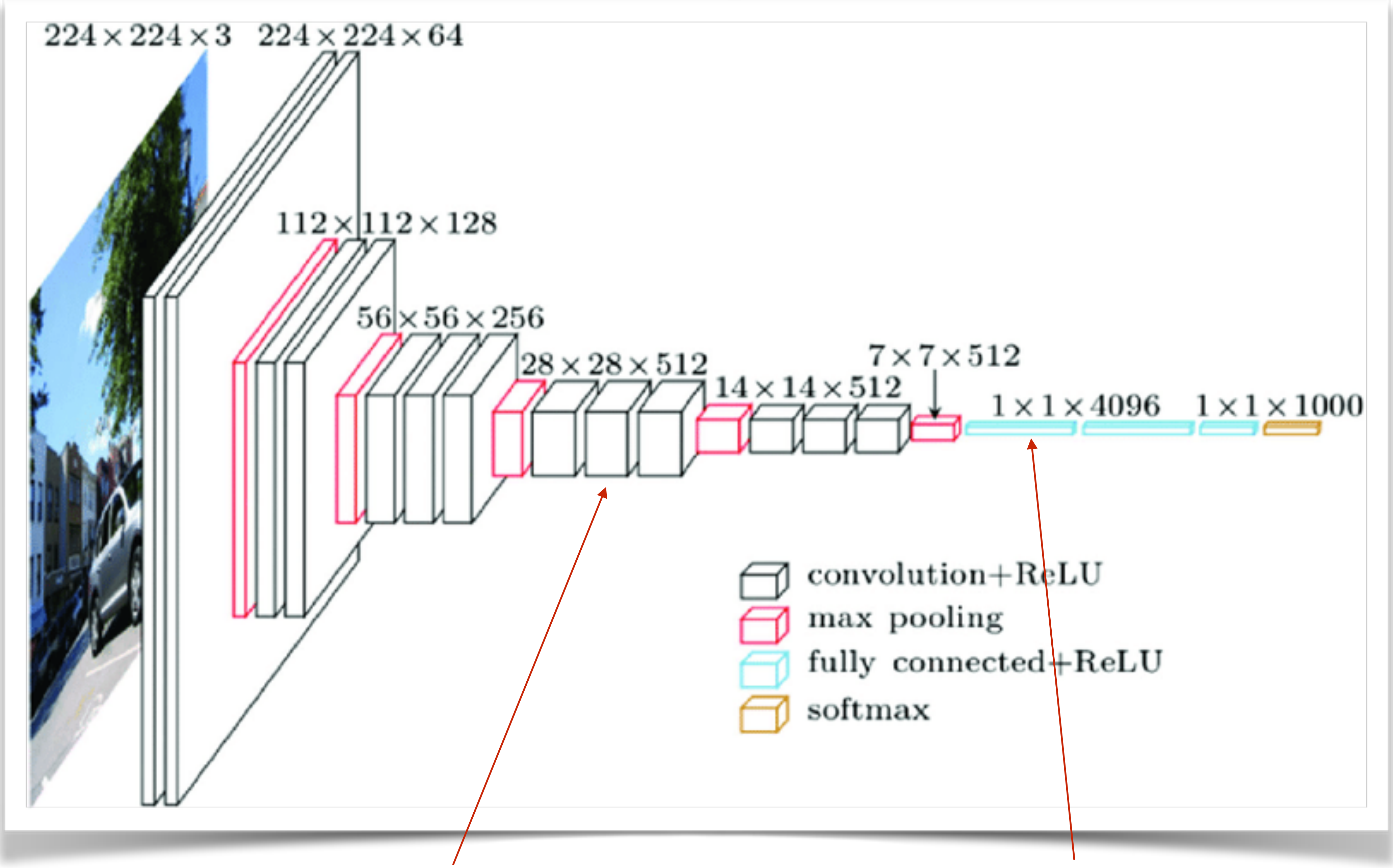
$\mathrm{conv}(3\times3, 64 \to 64)$

$\mathrm{conv}(1\times1, 64 \to 256)$

Output: W'xH'x256

- Useful to perform operations along channels dimension:
  - Increase /decrease number of channels
  - Normalization operations
  - In combination with purely spatial convolution = separable transform

Classification CNN



Could use efficient module here

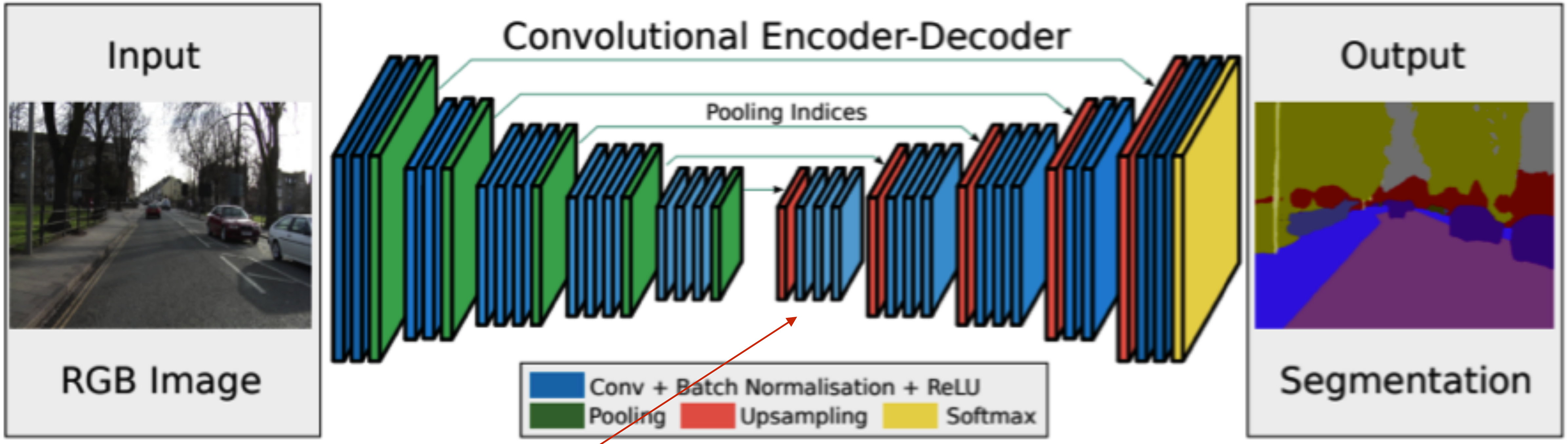
1x1 convolution for input of size 1x1 is equivalent to fully connected

- Second last layer has 4096*4096 =16M parameters!

More Convolutions

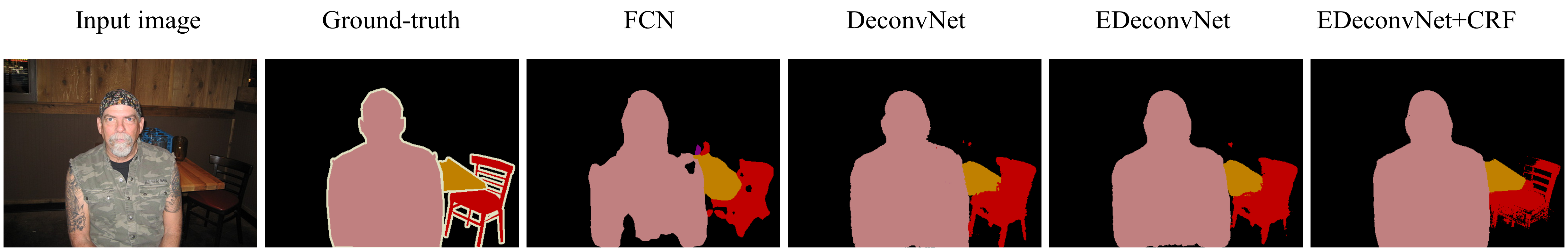
Semantic Segmentation

(Architecture picked to illustrate the task)



There appears "Unpooling"

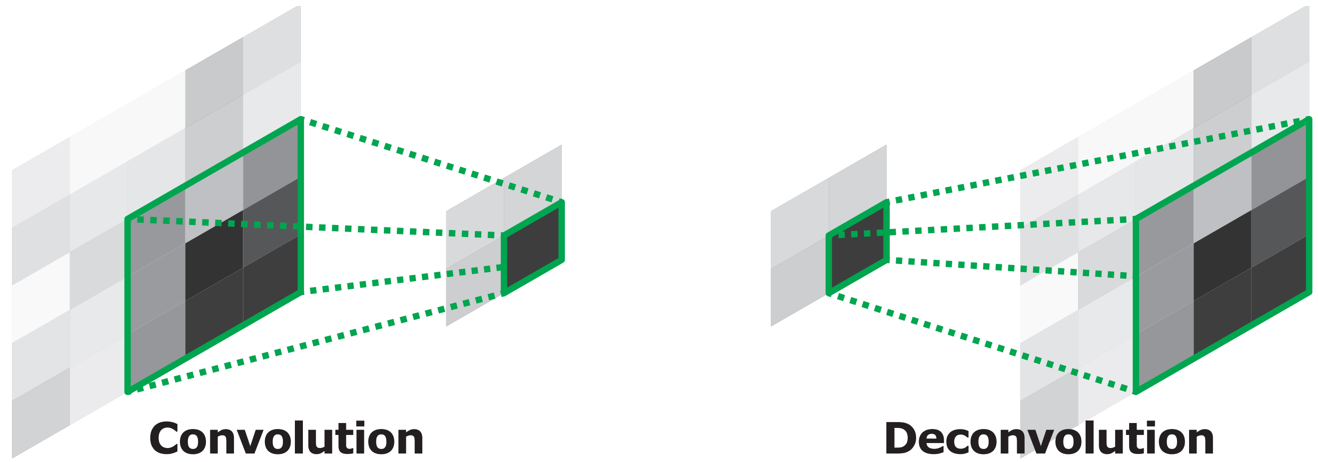
We will look at up-sampling with "transposed" convolution ("deconvolution")



[Noh et al. (2015) Learning Deconvolution Network for Semantic Segmentation]

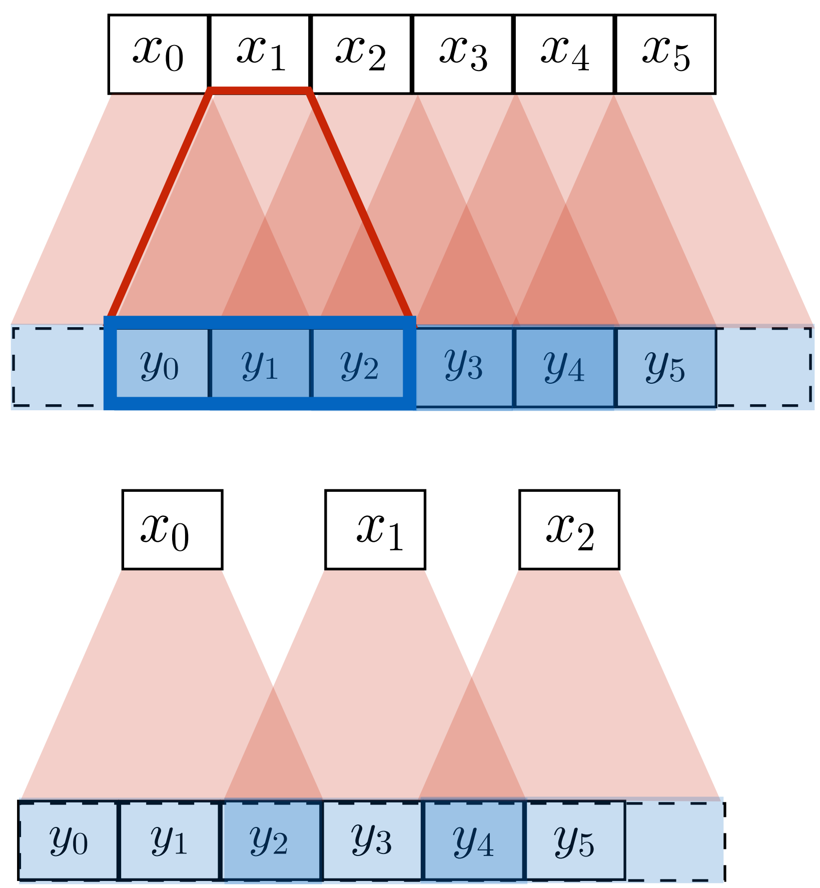
Transposed Convolution

- Deconvolution = Transposed convolution = backprop of convolution



Stride 1

Stride 2

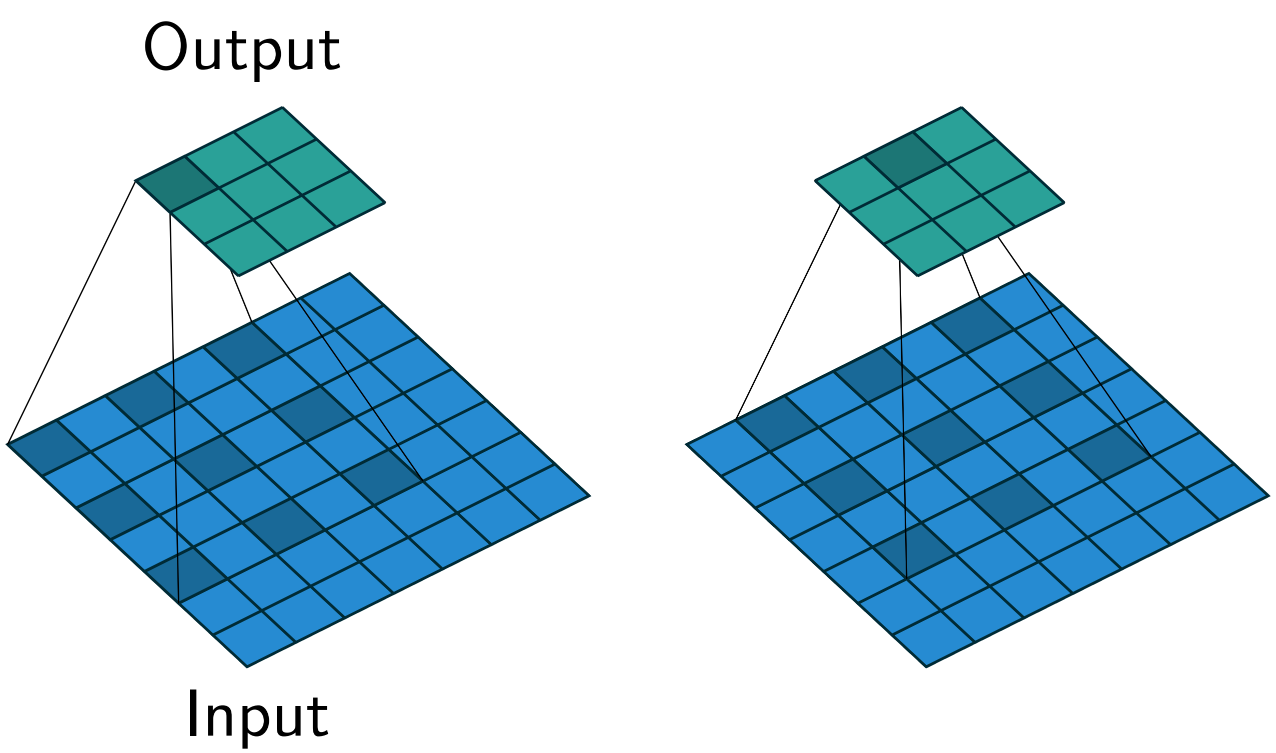


All variants and detail: [Dumoulin, Visin (2018): A guide to convolution arithmetic for deep learning]

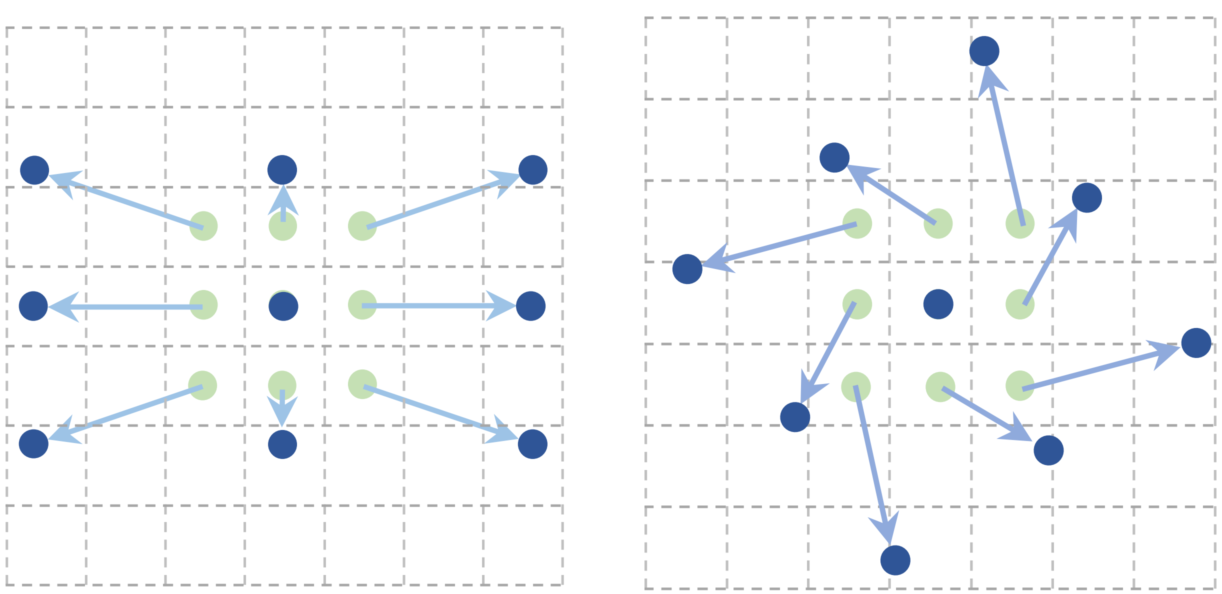
Sparse Convolutions

- ✦ Want to increase receptive field size
  - without decreasing spatial resolution and having too many layers
  - Can increase kernel size, but it was also costly
  - Can use a sparse mask for the kernel

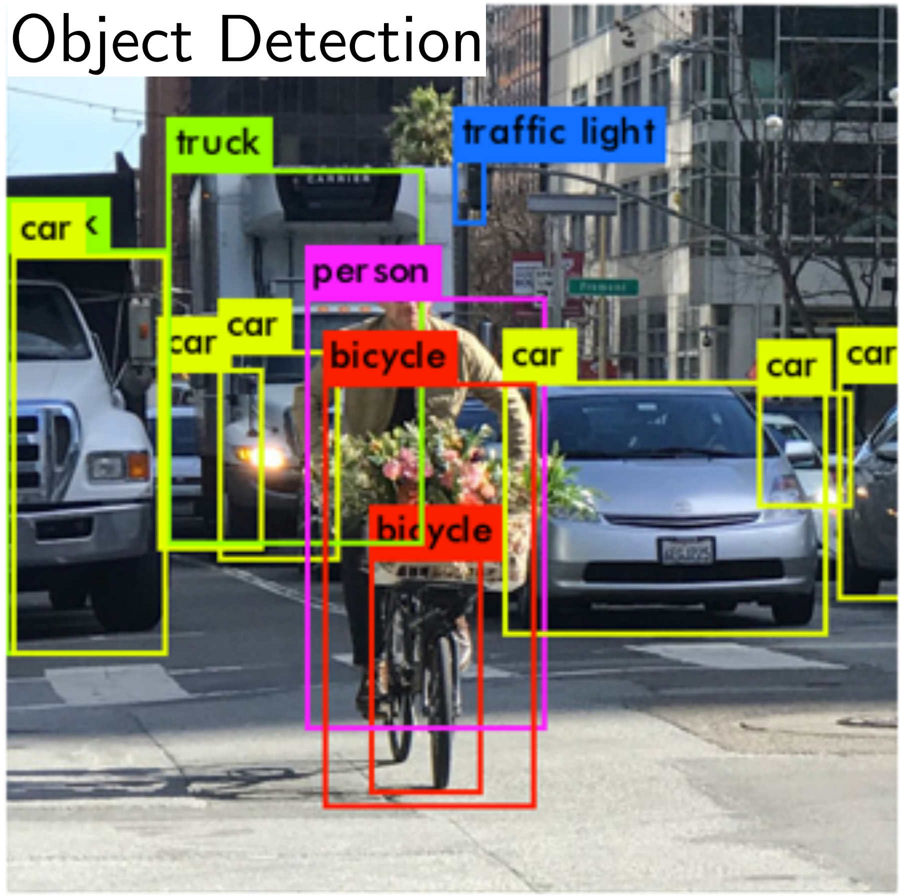
Dilated convolutions

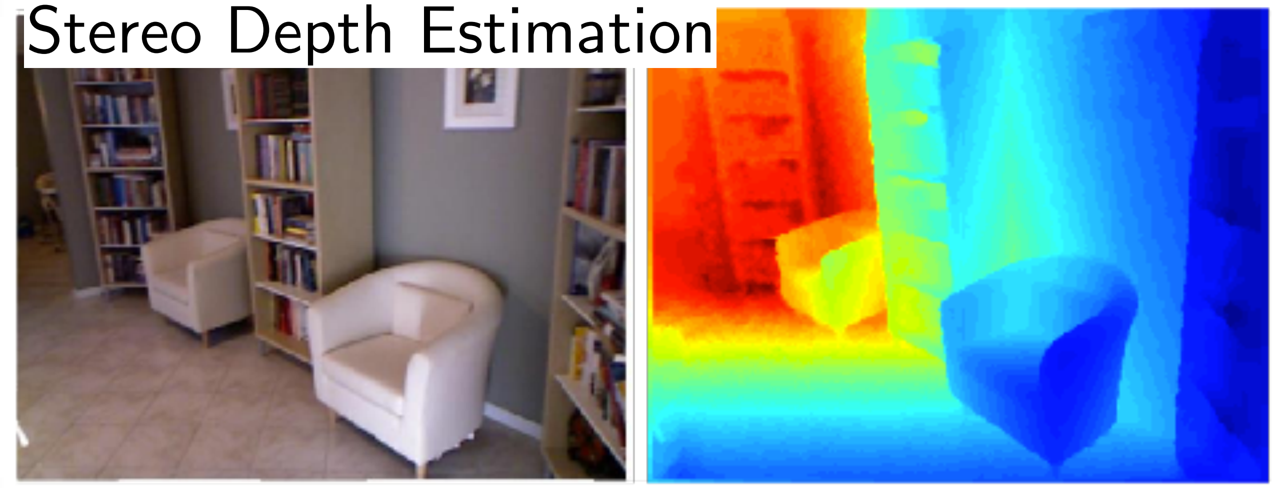


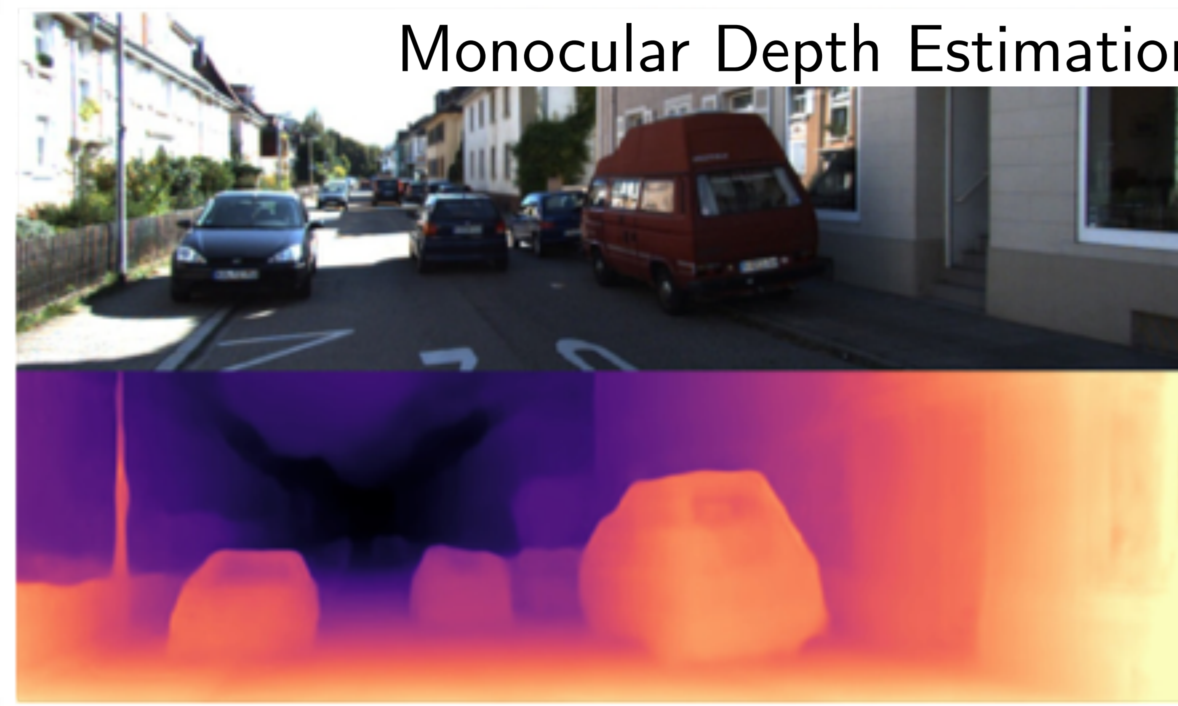
Can even learn sparse locations — **deformable** convolutions

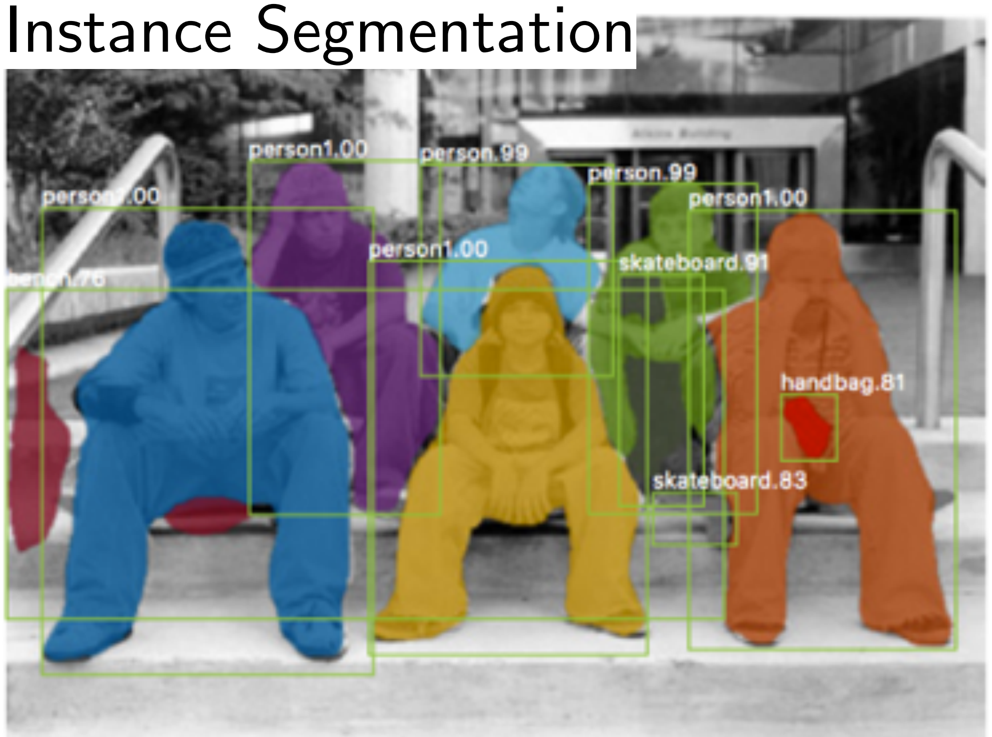


Many More Examples and Smart Architectures





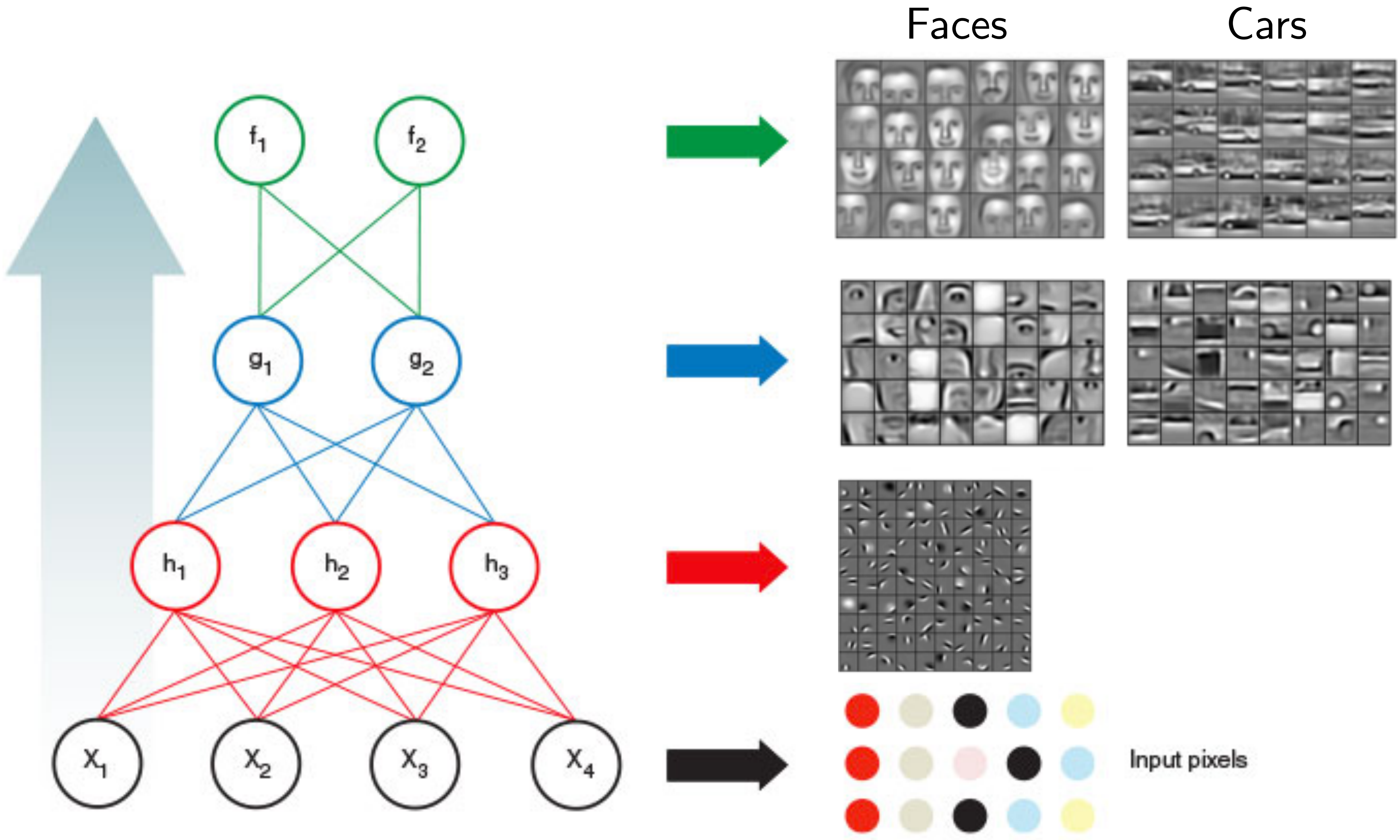




Hierarchy of Parts, Visual Cortex

Hierarch of Parts Phenomenon

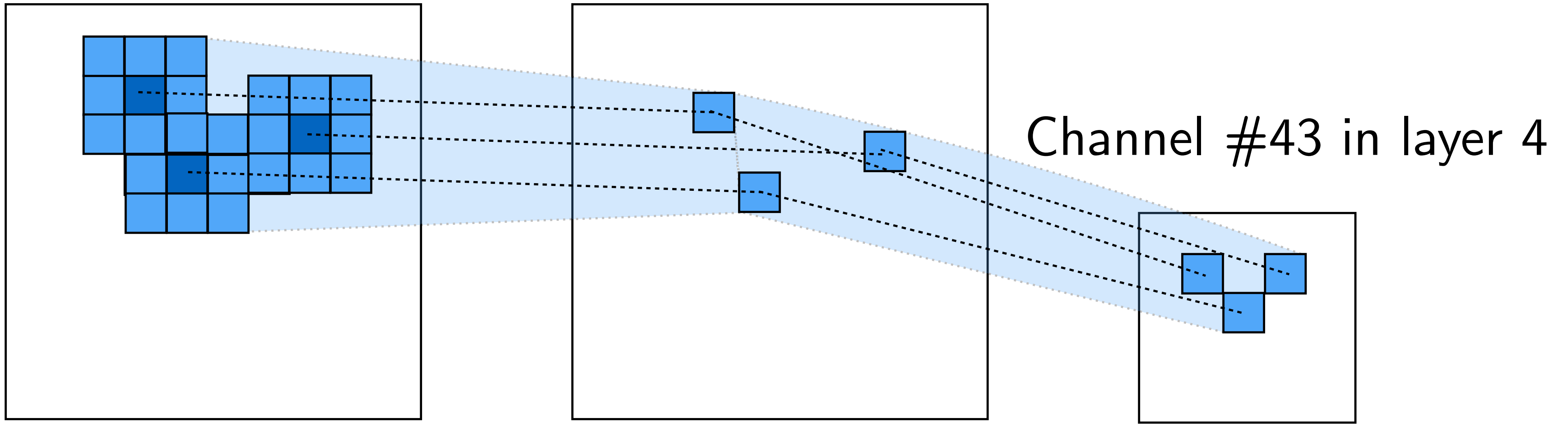
- In networks trained for different complex problems
  - some intermediate layers activations correspond object parts



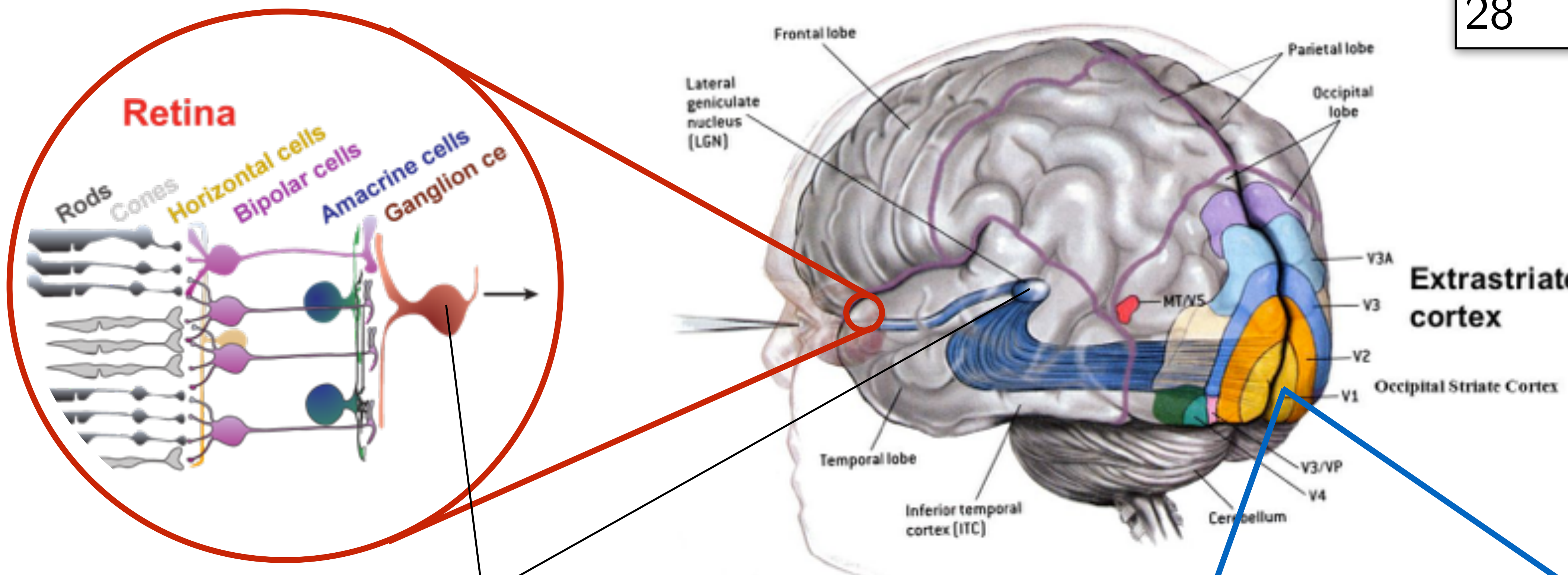
Hierarch of Parts Phenomenon

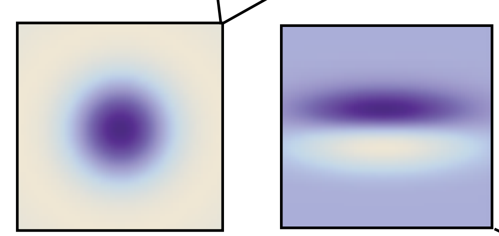
✦ In networks trained for different complex problems

- some intermediate layers activations correspond object parts



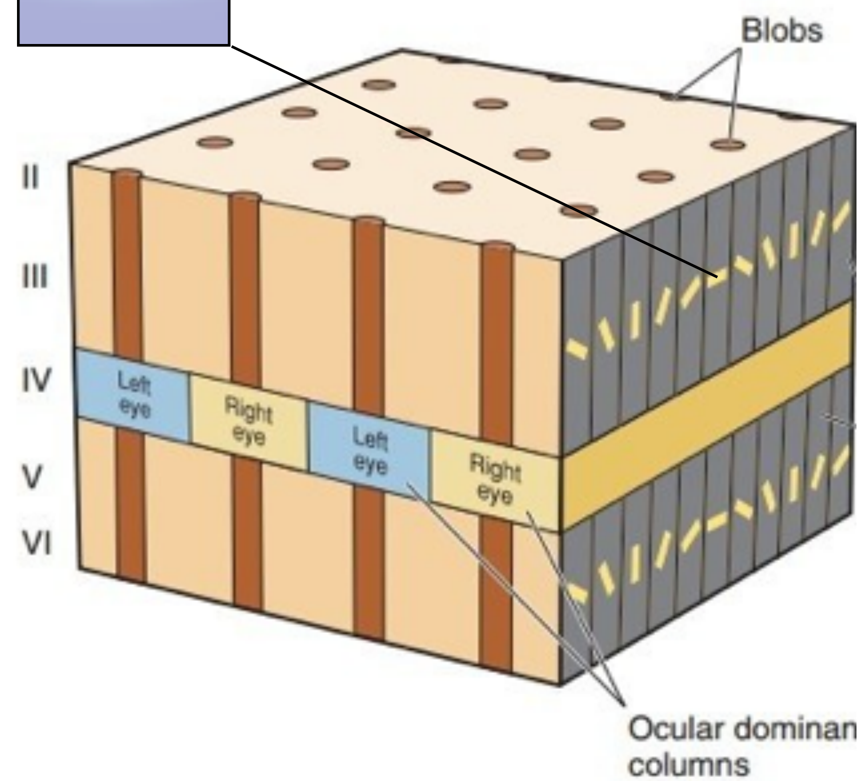
Parallels with Visual Cortex

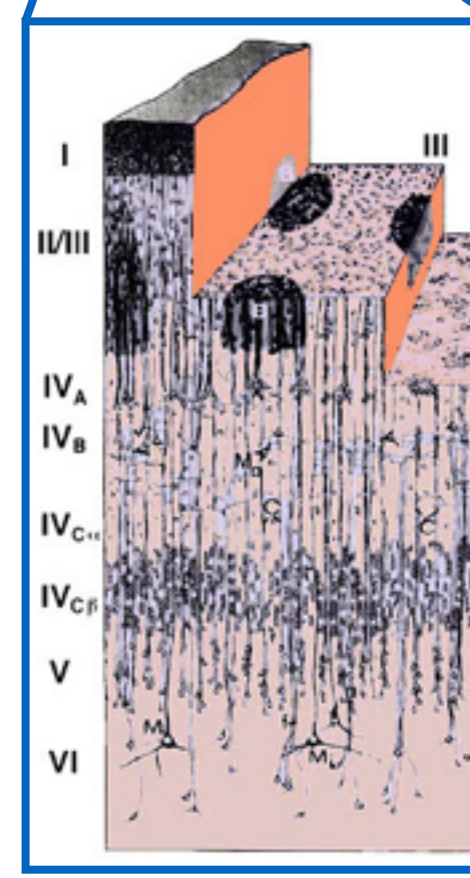




- ✦ LGN:
  no orientation preference
  space-time separable

- ✦ V1 packing in 2D problem:
  - location in the view (retinotopy)
  - orientation
  - ocular dominance
  - motion
- ✦ feedback connections





50000 neurons / mm^3