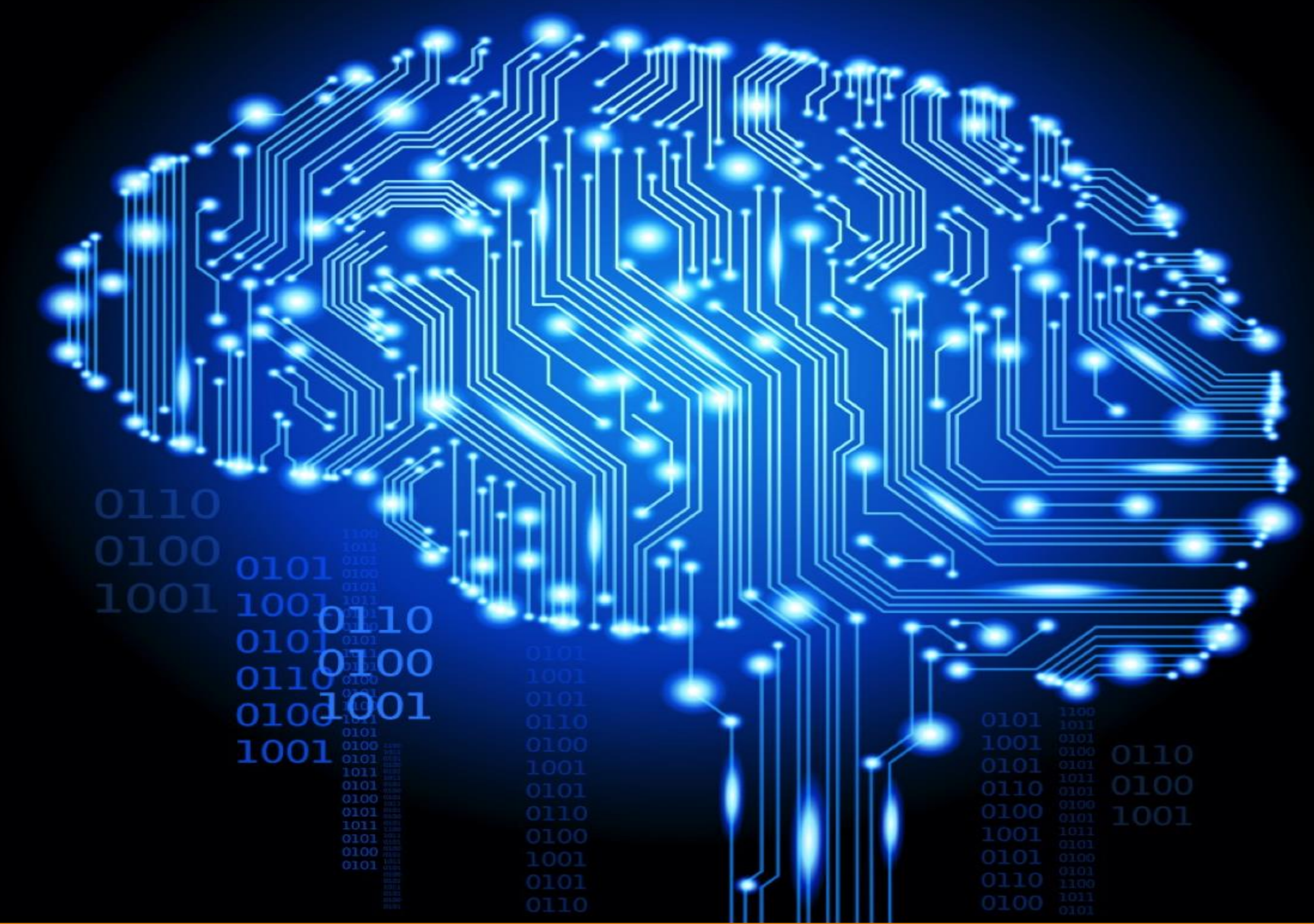


Lecture 4: Convolutional Neural Networks for Computer Vision

Deep Learning @ UvA

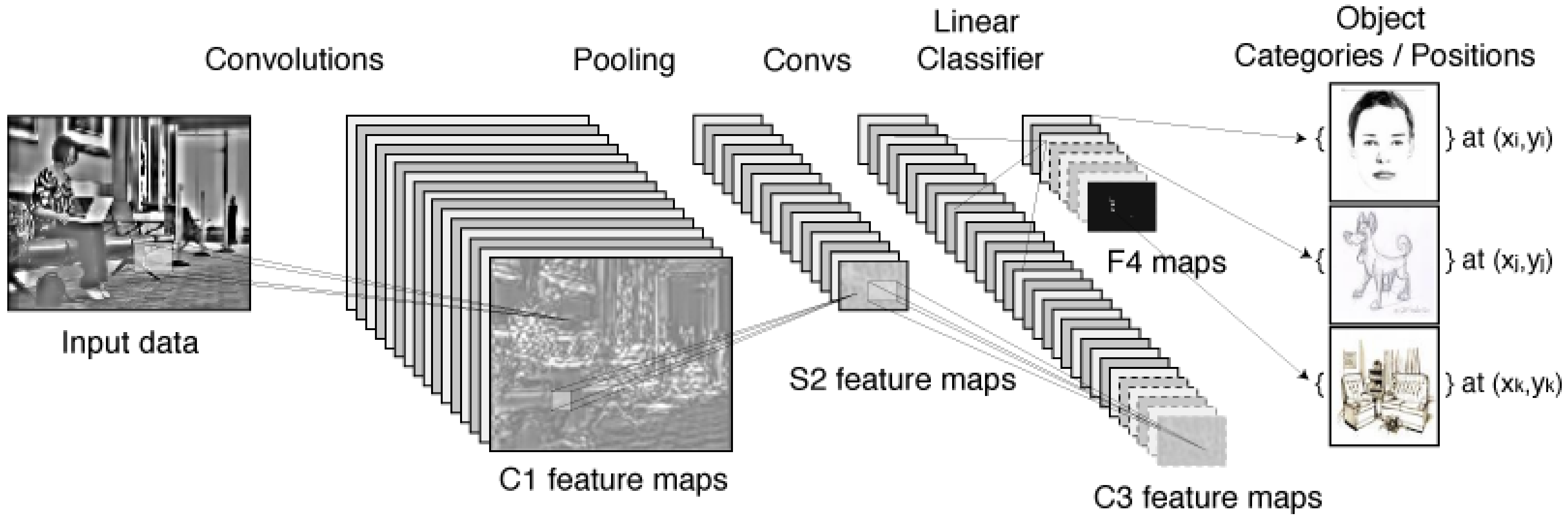
Previous lecture

- How to define your neural network model and optimize it in practice
- Data preprocessing and normalization
- Optimization methods
- Regularizations
- Architectures and architectural hyper-parameters
- Learning rate
- Weight initializations
- Good practices

Lecture overview

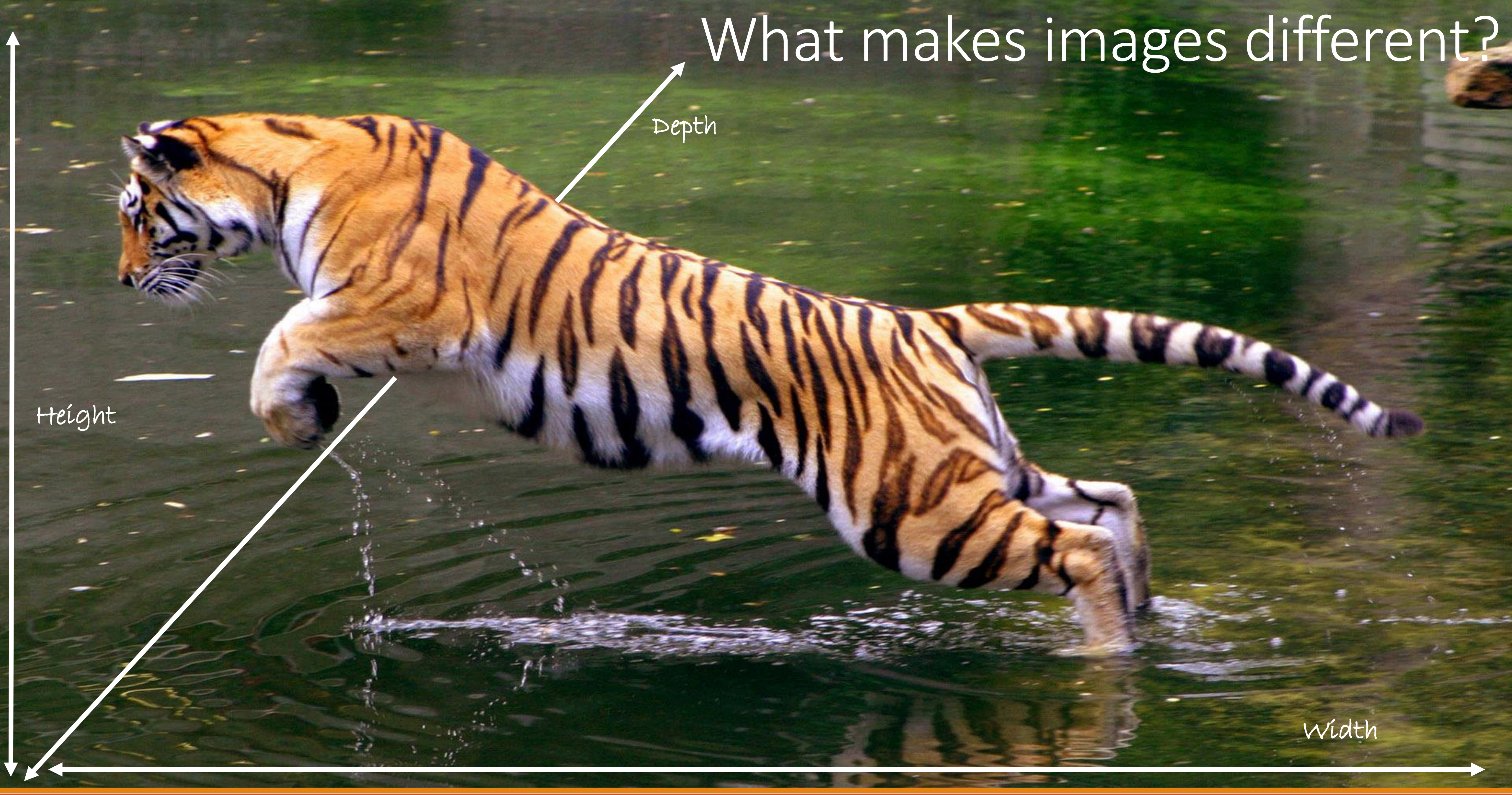
- What are the Convolutional Neural Networks?
- Why are they so important for Computer Vision?
- How do they differ from standard Neural Networks?
- How can we train a Convolutional Neural Network?

Convolutional Neural Networks



What makes images different?

What makes images different?



What makes images different?

1920x1080x3 = 6,220,800 input variables

What makes images different?

What makes images different?

What makes images different?

Image has shifted a bit to the up and the left!

What makes images different?

- An image has spatial structure
- Huge dimensionality
  - A 256x256 RGB image amounts to ~200K input variables
  - 1-layered NN with 1,000 neurons → 200 million parameters
- Images are stationary signals → they share features
  - After variances images are still meaningful
  - Small visual changes (often invisible to naked eye) → big changes to input vector
  - Still, semantics remain
  - Basic natural image statistics are the same

Input dimensions are correlated

Traditional task: Predict my salary!

Shift 1 dimension

Level of education	Age	Years of experience	Previous job	Nationality
"Higher"	28	6	Researcher	Spain

Level of education	Age	Years of experience	Previous job	Nationality
Spain	"Higher"	28	6	Researcher

Vision task: Predict the picture!

First 5x5 values

```
array([[51, 49, 51, 56, 55],
       [53, 53, 57, 61, 62],
       [67, 68, 71, 74, 75],
       [76, 77, 79, 82, 80],
       [71, 73, 76, 75, 75]], dtype=uint8)
```

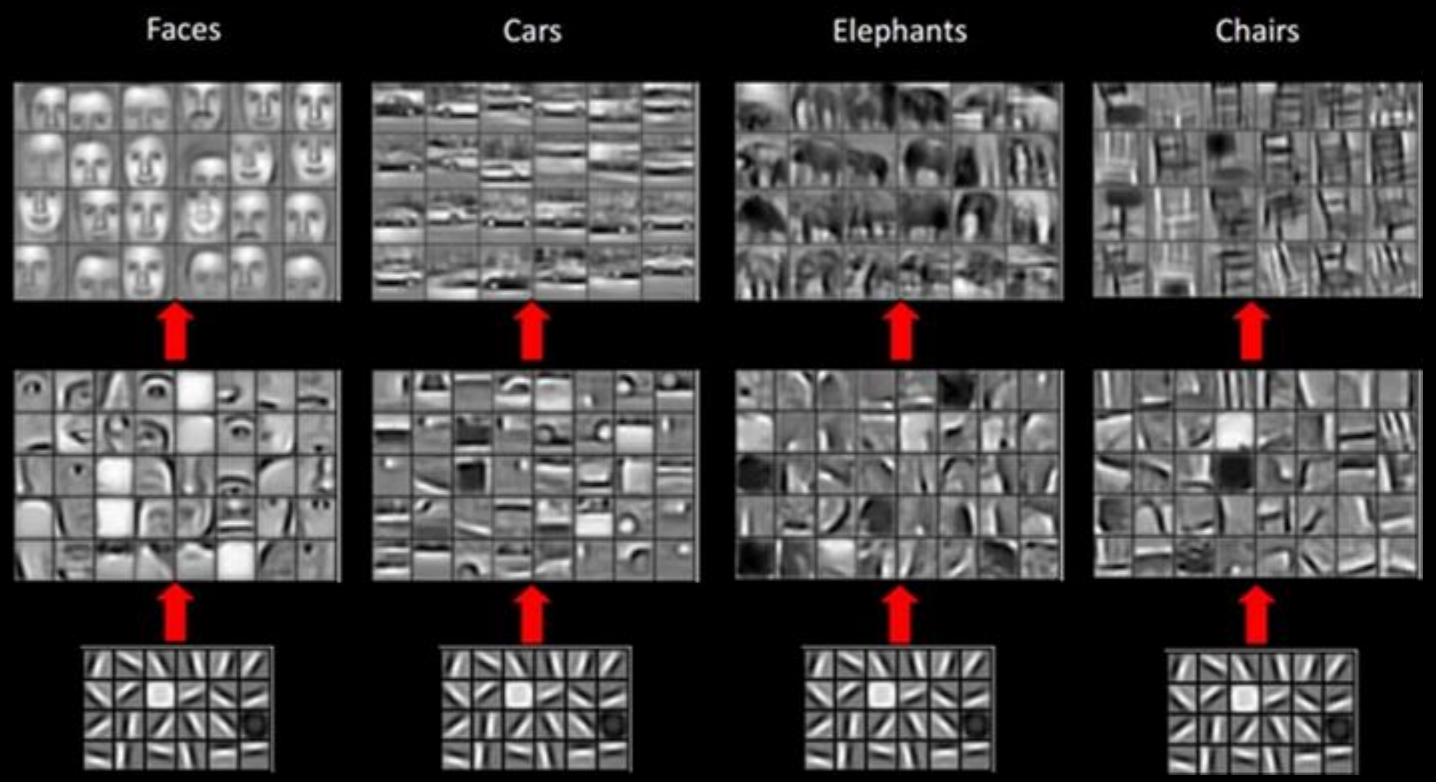
First 5x5 values

```
array([[58, 57, 57, 59, 59],
       [58, 57, 57, 58, 59],
       [59, 58, 58, 58, 58],
       [61, 61, 60, 60, 59],
       [64, 63, 62, 61, 60]], dtype=uint8)
```

Convolutional Neural Networks

- Question: Spatial structure?
  - Answer: Convolutional filters
- Question: Huge input dimensionalities?
  - Answer: Parameters are shared between filters
- Question: Local variances?
  - Answer: Pooling

Preserving spatial structure

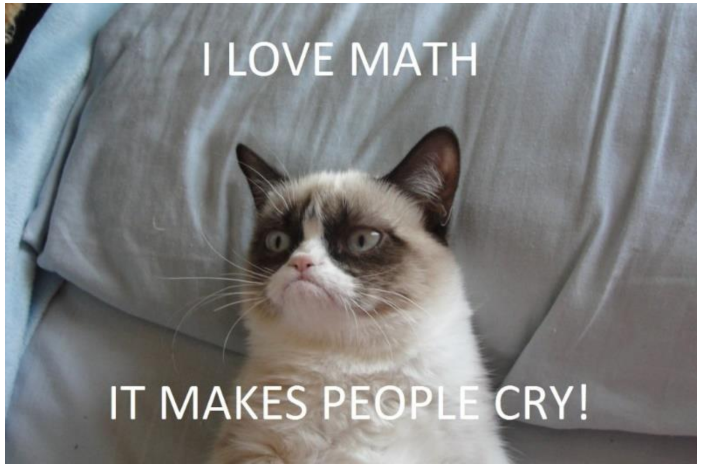


Why spatial?

- Images are 2-D
  - k-D if you also count the extra channels
  - RGB, hyperspectral, etc.

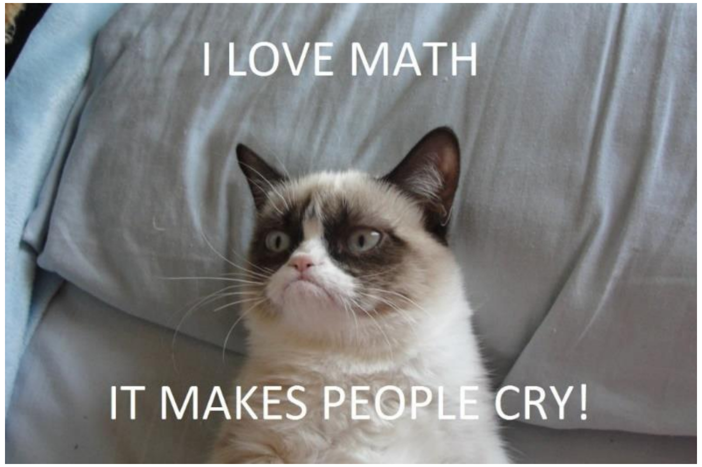
Why spatial?

- Images are 2-D
  - k-D if you also count the extra channels
  - RGB, hyperspectral, etc.



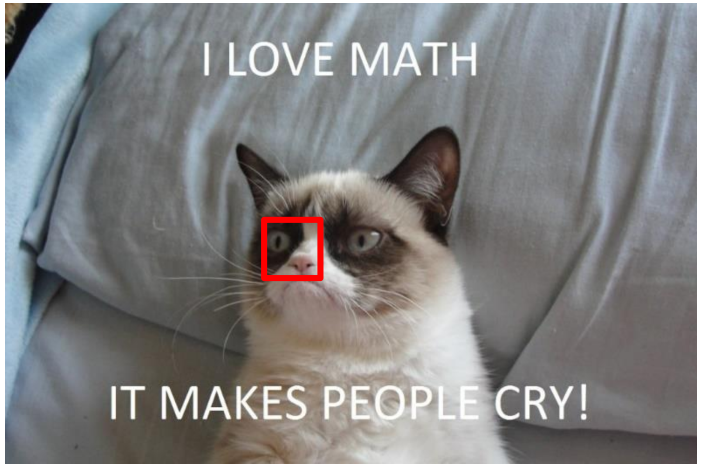
Why spatial?

- Images are 2-D
  - k-D if you also count the extra channels
  - RGB, hyperspectral, etc.

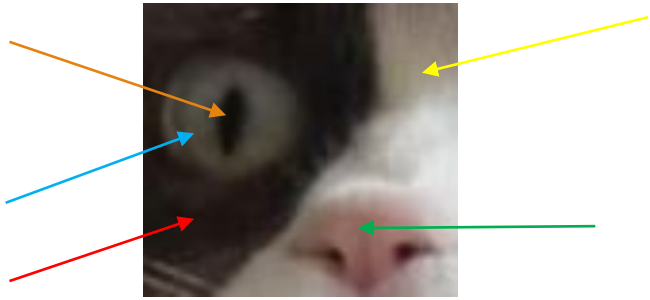


Why spatial?

- Images are 2-D
  - k-D if you also count the extra channels
  - RGB, hyperspectral, etc.

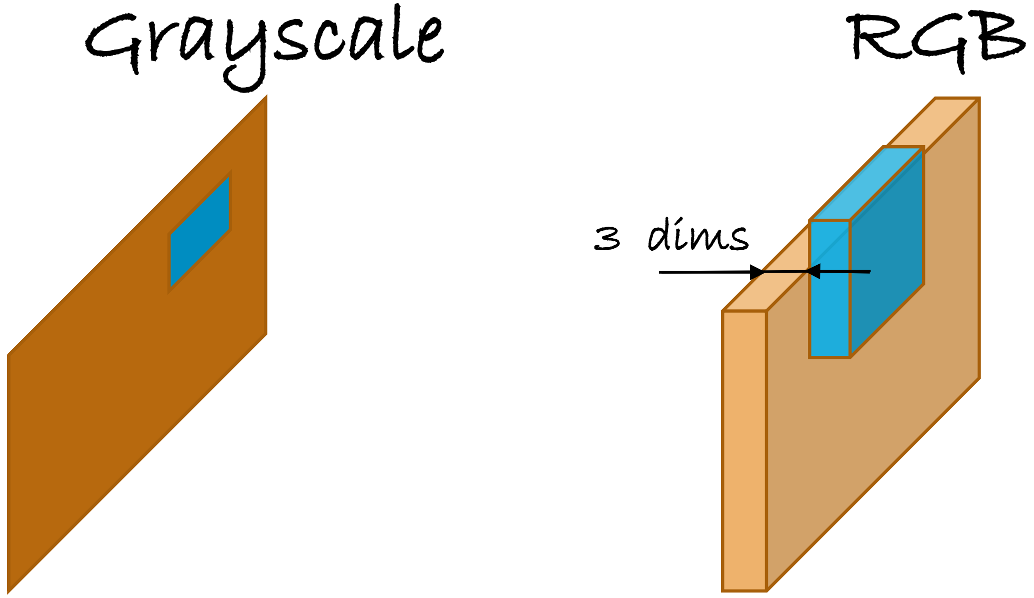


- What does a 2-D input really mean?
  - Neighboring variables are locally correlated



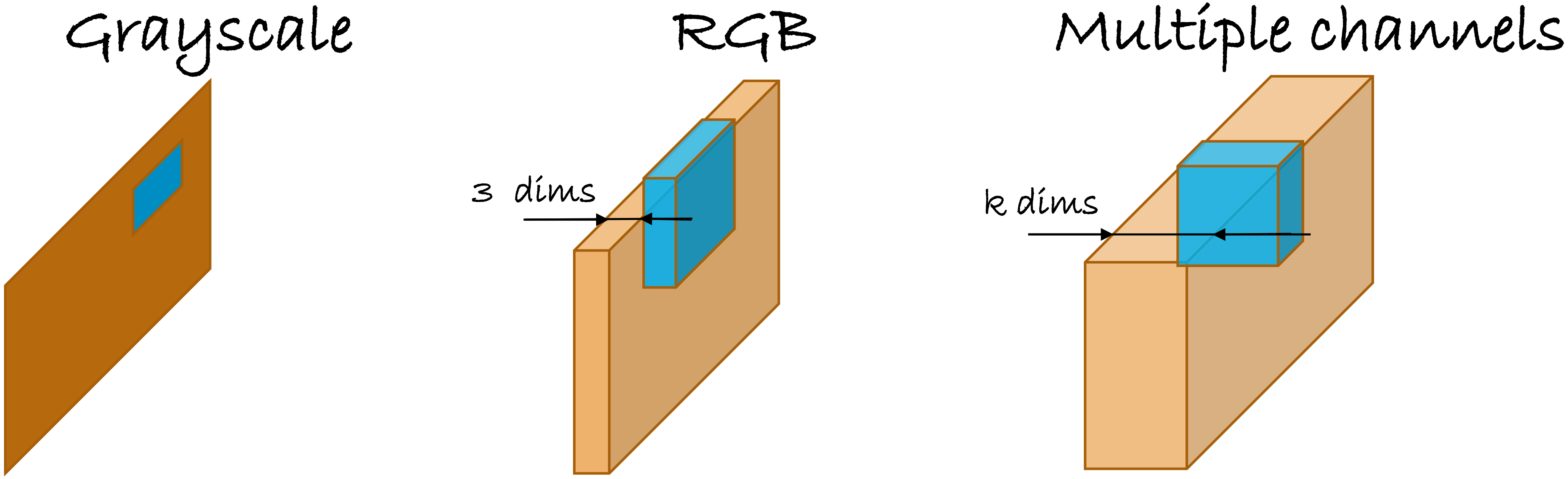
Parameters in k-D == Filters

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can “exploit” the spatial nature of images
- Similarly, if images are k-D, parameters should also be k-D



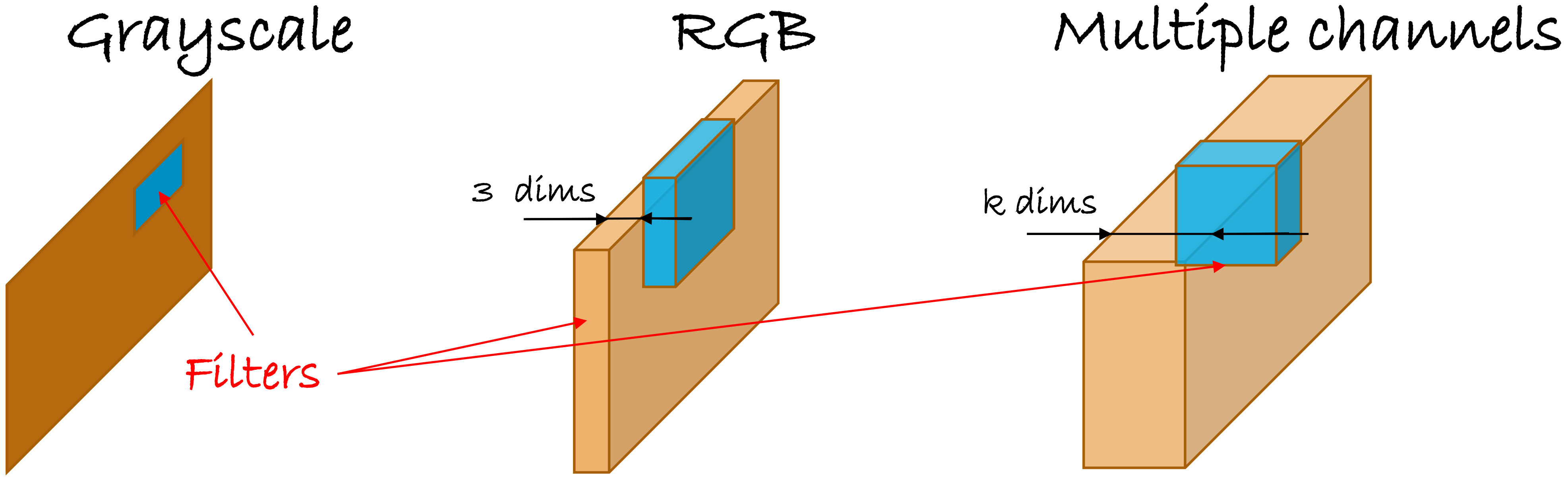
Parameters in k-D == Filters

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can “exploit” the spatial nature of images
- Similarly, if images are k-D, parameters should also be k-D

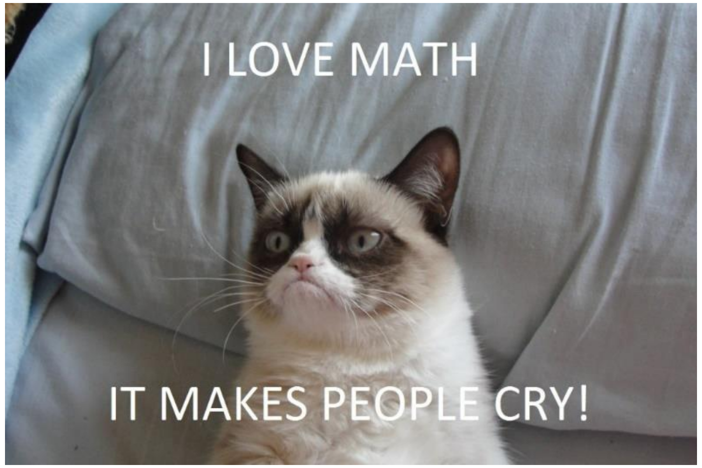


Parameters in k-D == Filters

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can "exploit" the spatial nature of images
- Similarly, if images are k-D, parameters should also be k-D

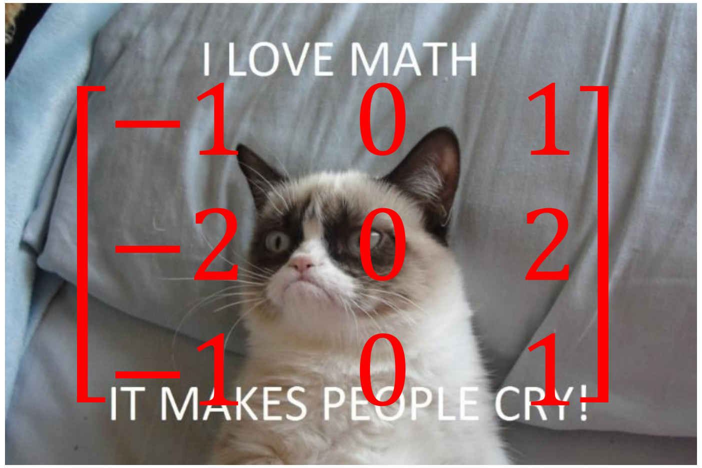


What would a k-D filter look like?



What would a k-D filter look like?

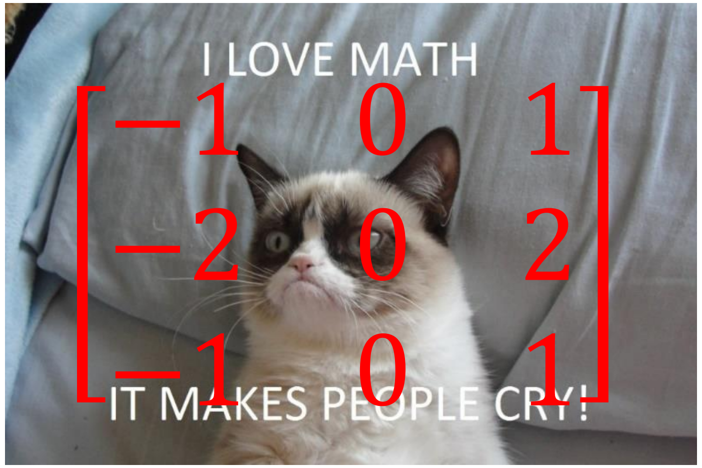
e.g. Sobel 2-D filter



$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

What would a k-D filter look like?

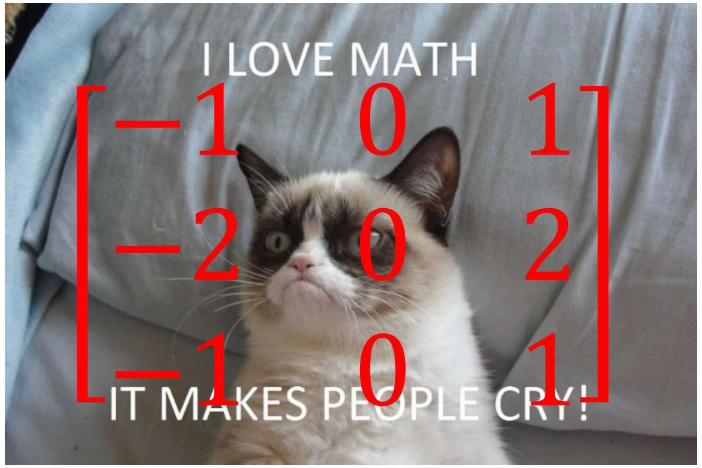
e.g. Sobel 2-D filter



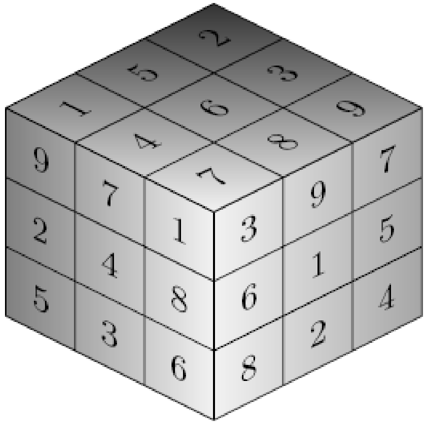
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

What would a k-D filter look like?

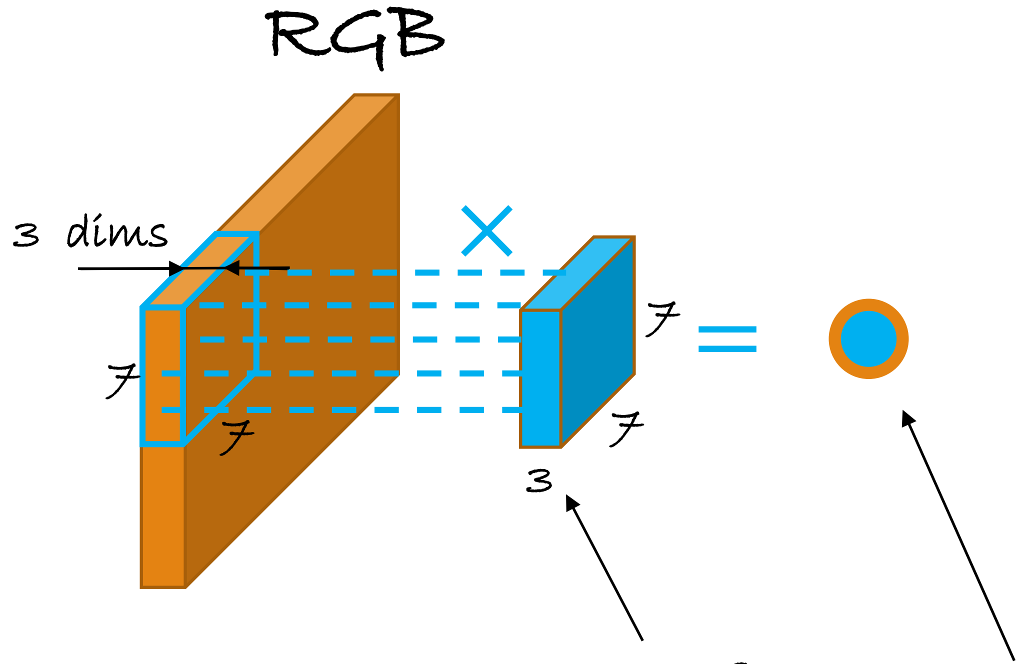
e.g. Sobel 2-D filter



$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

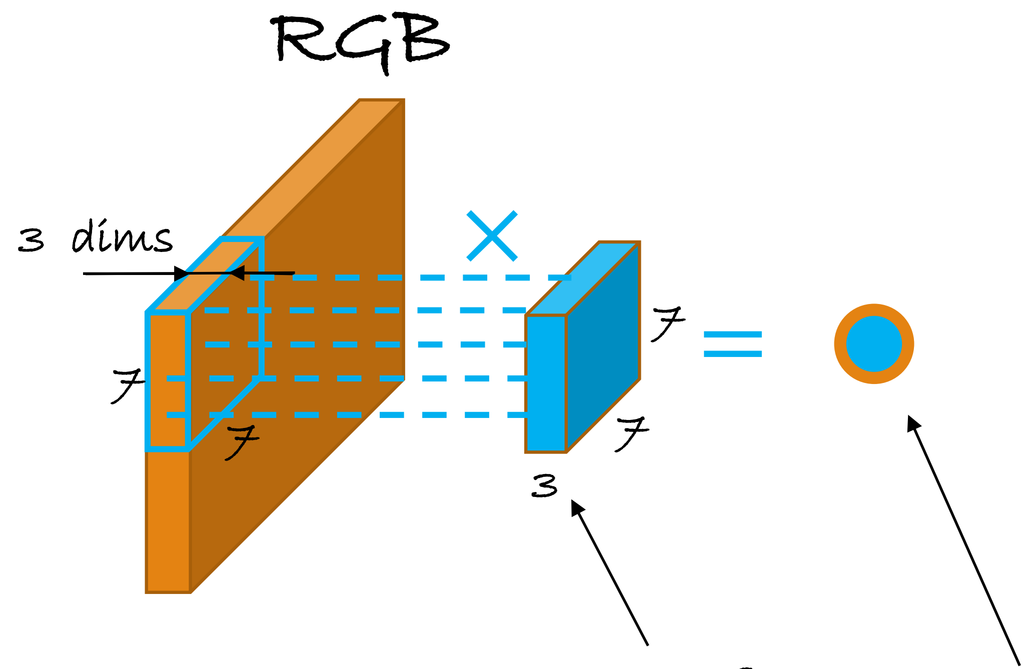


Think in space



How many weights for this neuron?

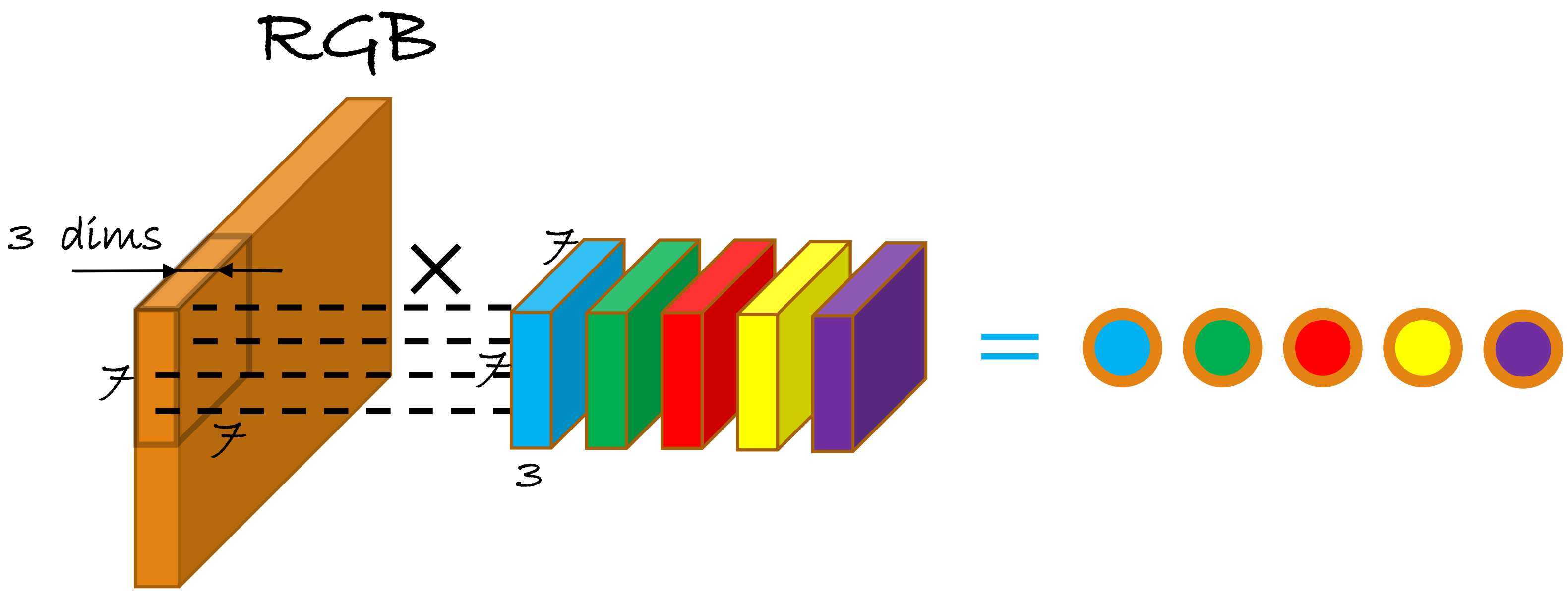
Think in space



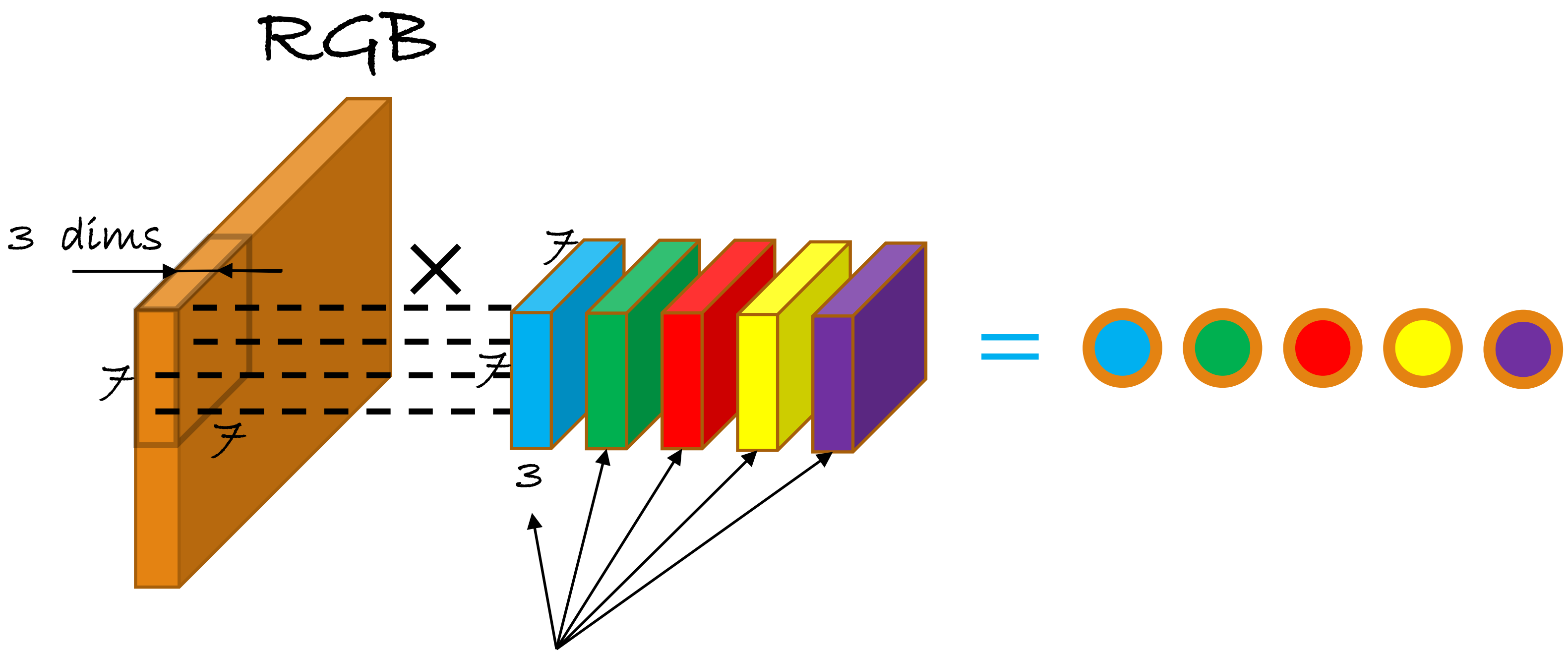
How many weights for this neuron?

$$7 \cdot 7 \cdot 3 = 147$$

Think in space

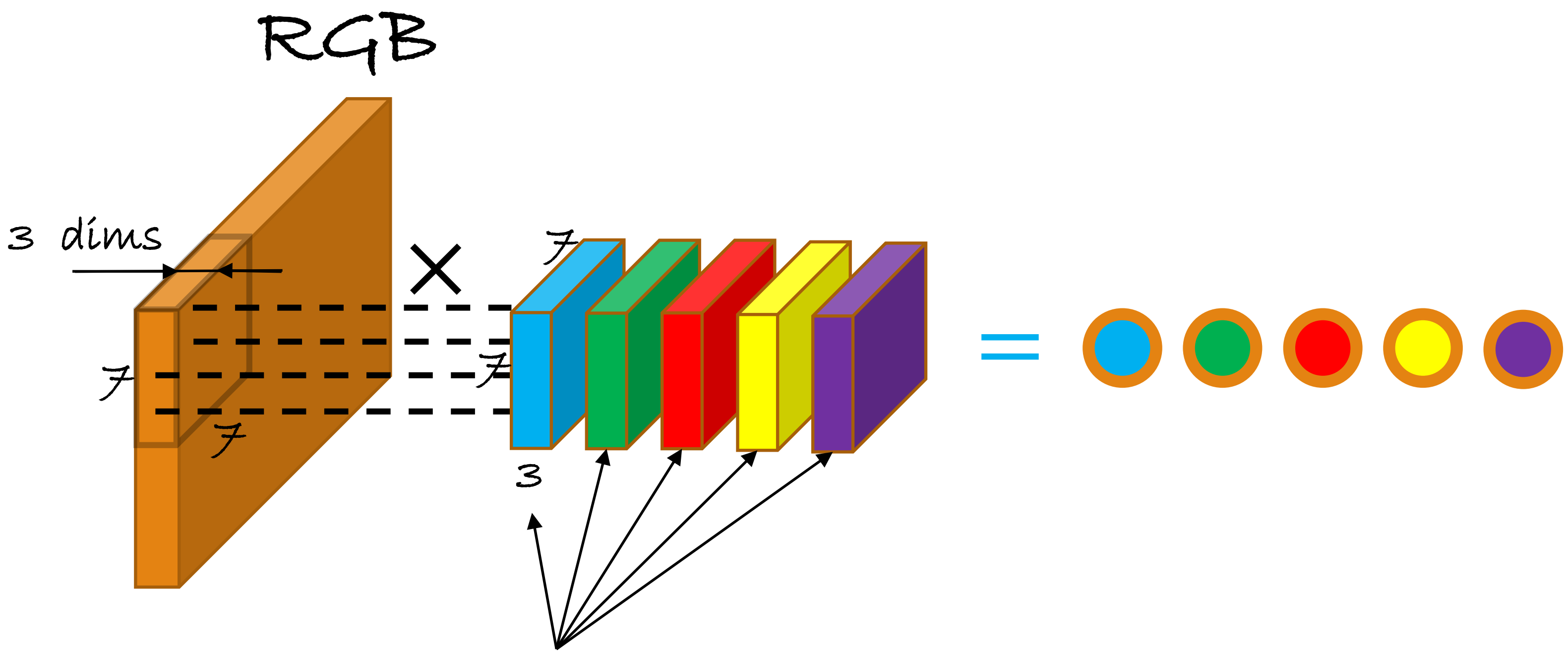


Think in space



How many weights for these 5 neurons?

Think in space

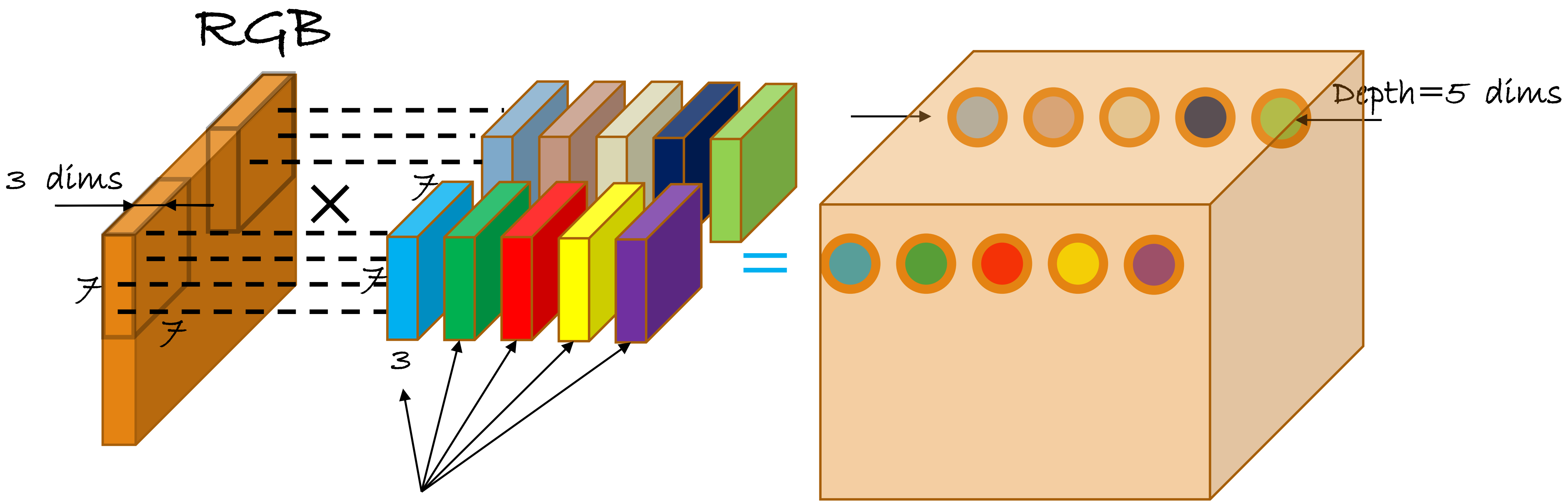


How many weights for these 5 neurons?

$$5 \cdot 7 \cdot 7 \cdot 3 = 735$$

Think in space

The activations of a hidden layer form a volume of neurons, not a 1-d "chain"

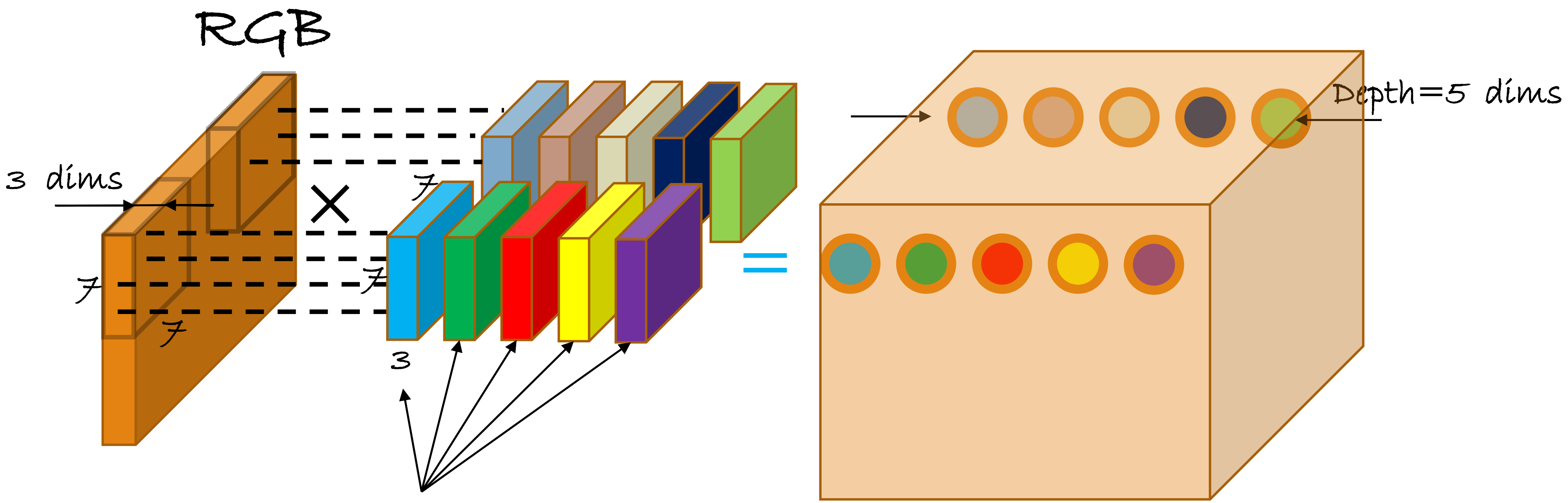


How many weights for these 5 neurons?

$$5 \cdot 7 \cdot 7 \cdot 3 = 735$$

Think in space

The activations of a hidden layer form a volume of neurons, not a 1-d "chain"
This volume has a depth 5, as we have 5 filters

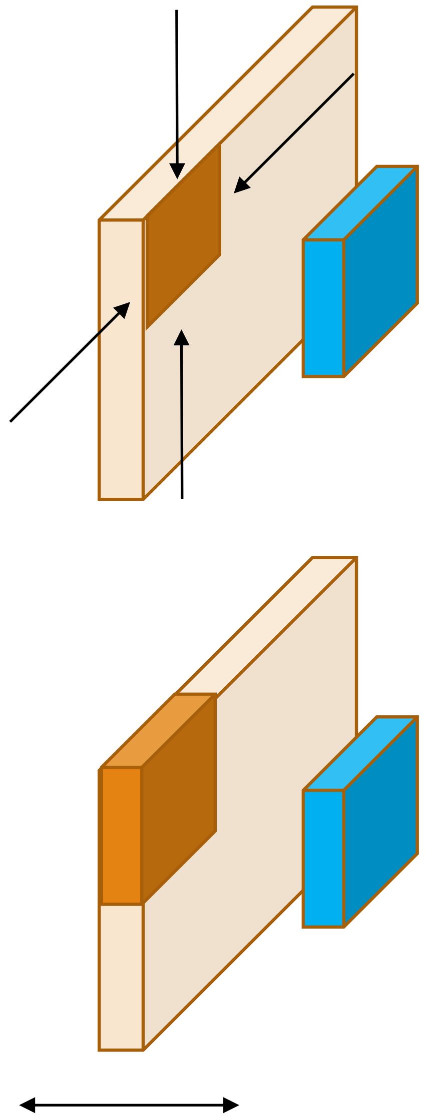


How many weights for these 5 neurons?

$$5 \cdot 7 \cdot 7 \cdot 3 = 735$$

Local connectivity

- The weight connections are surface-wise local!
  - Local connectivity
- The weights connections are depth-wise global

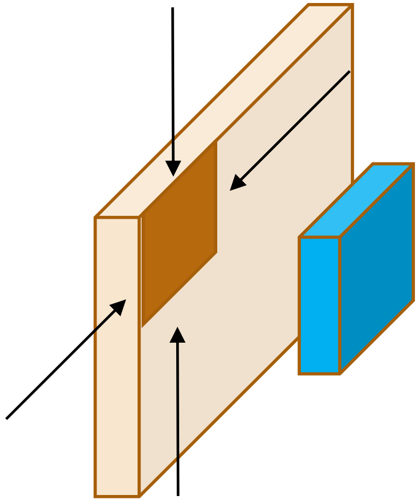


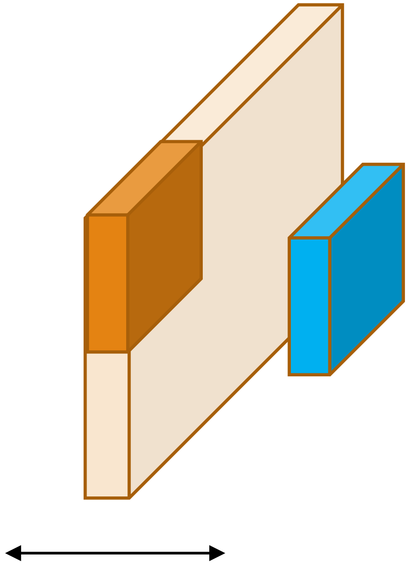
Local connectivity

- The weight connections are surface-wise local!
  - Local connectivity

- The weights connections are depth-wise global

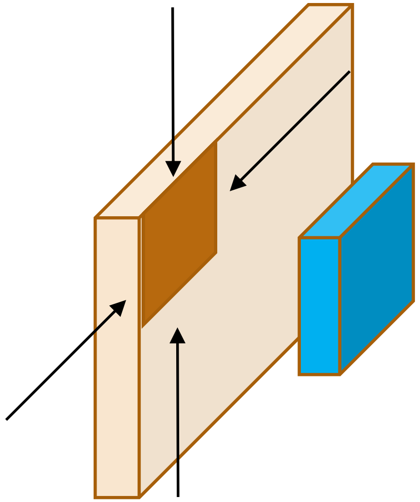
- For standard neurons no local connectivity



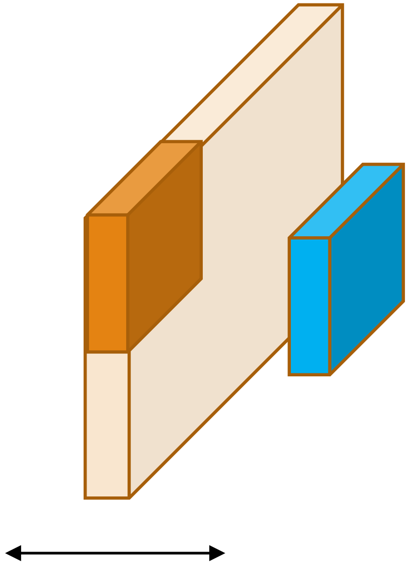


Local connectivity

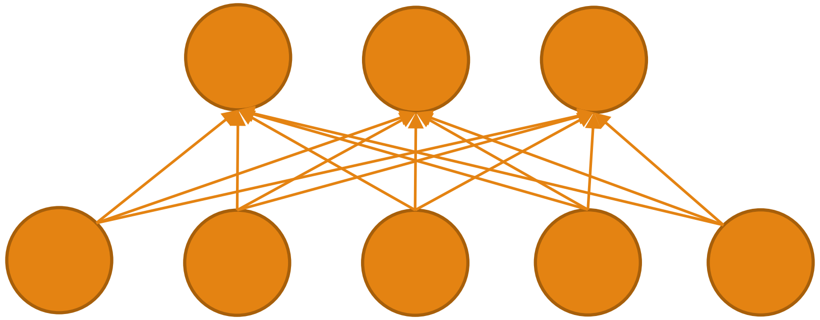
- The weight connections are surface-wise local!
  - Local connectivity



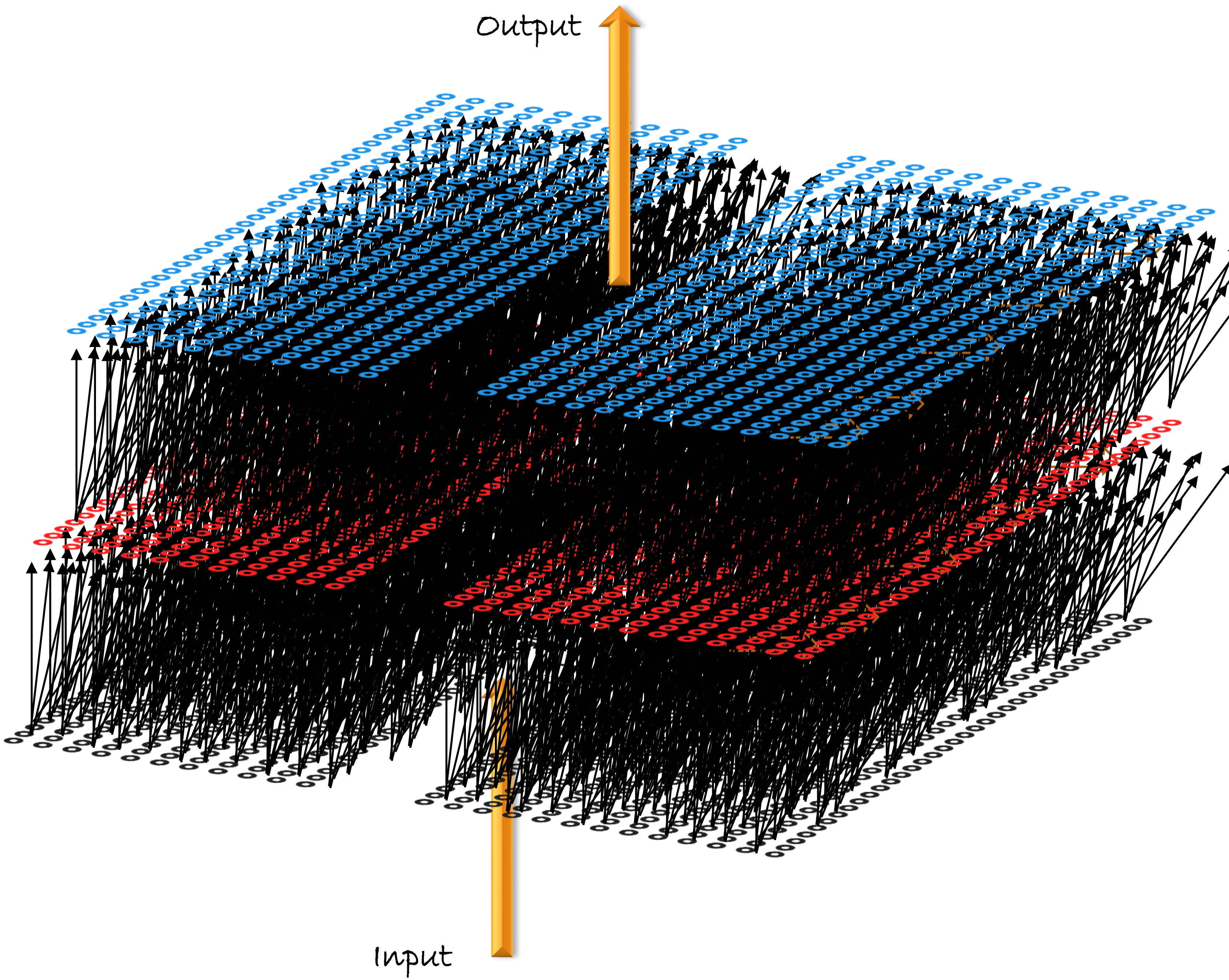
- The weights connections are depth-wise global



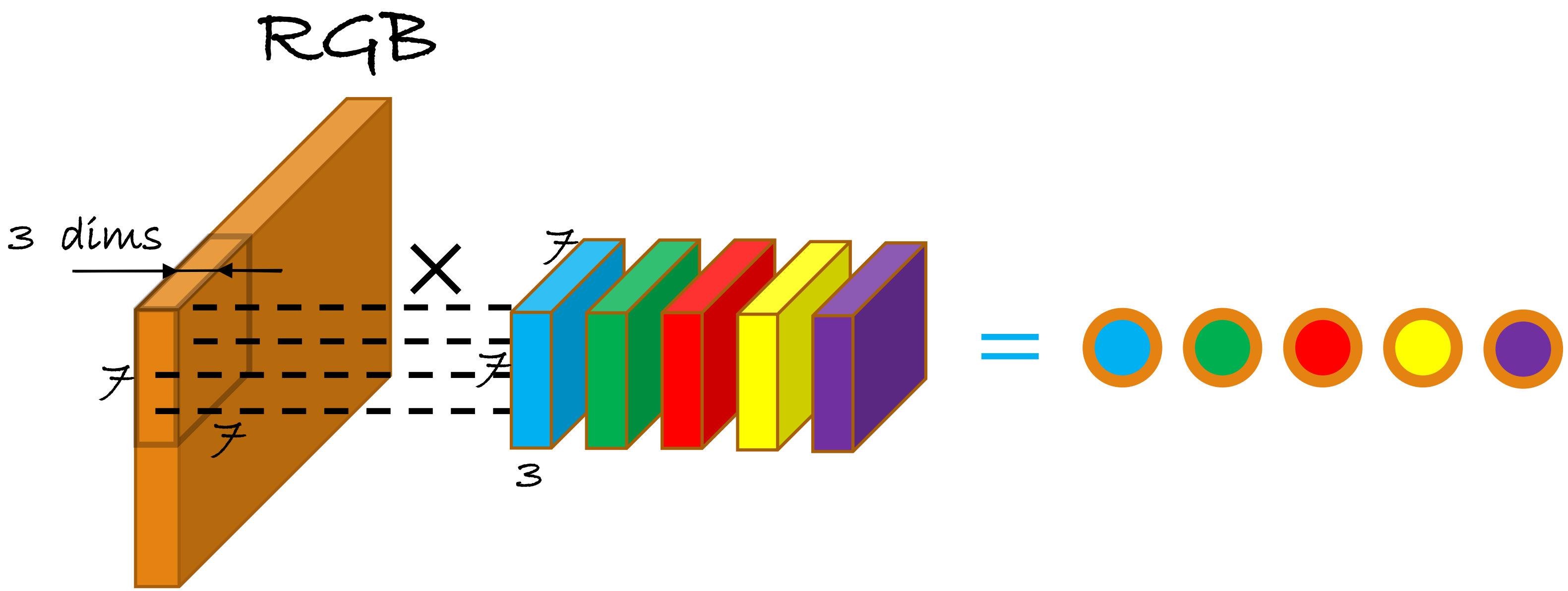
- For standard neurons no local connectivity
  - Everything is connected to everything



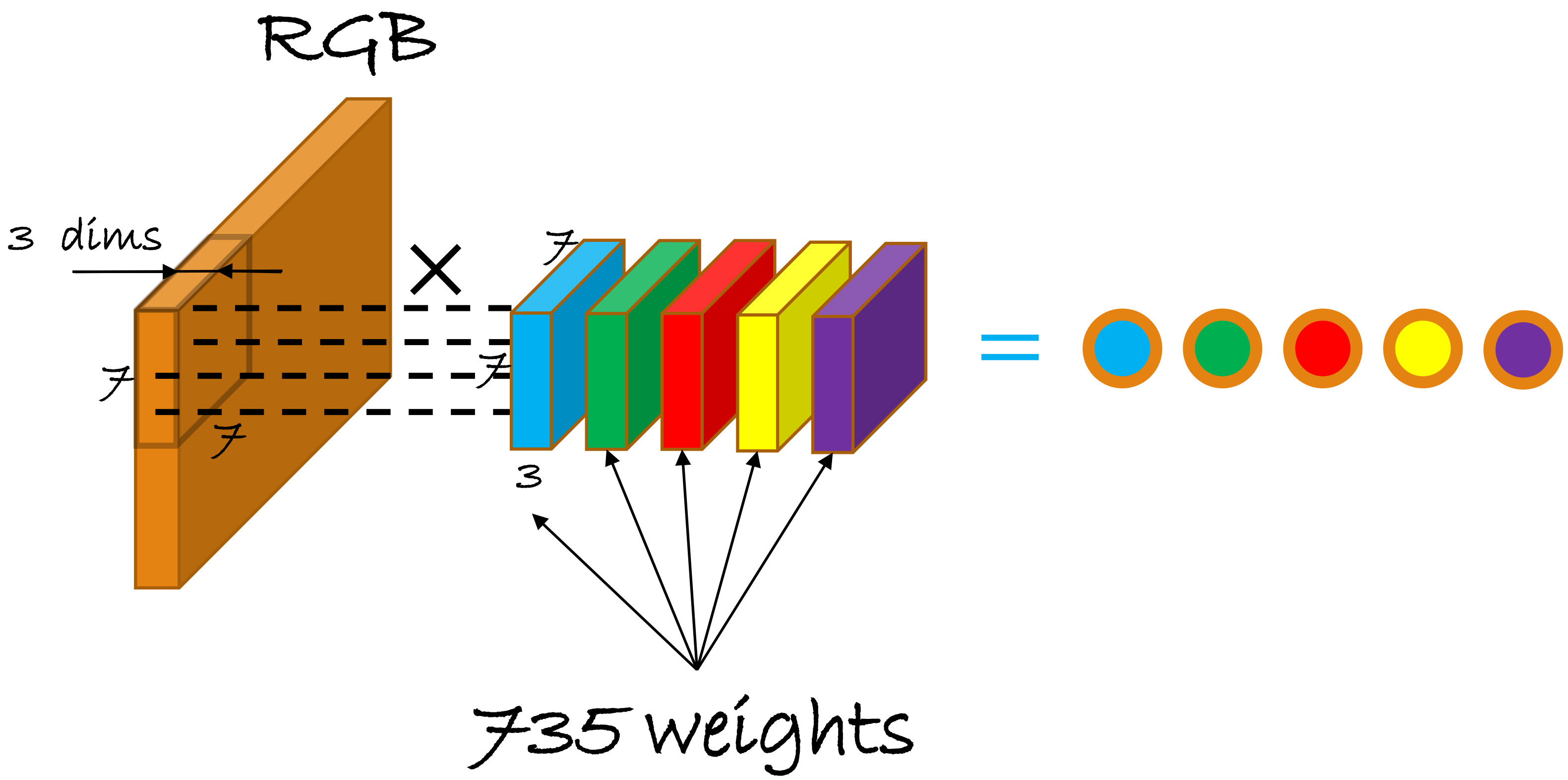
Filters *vs* Convolutional k-d filters



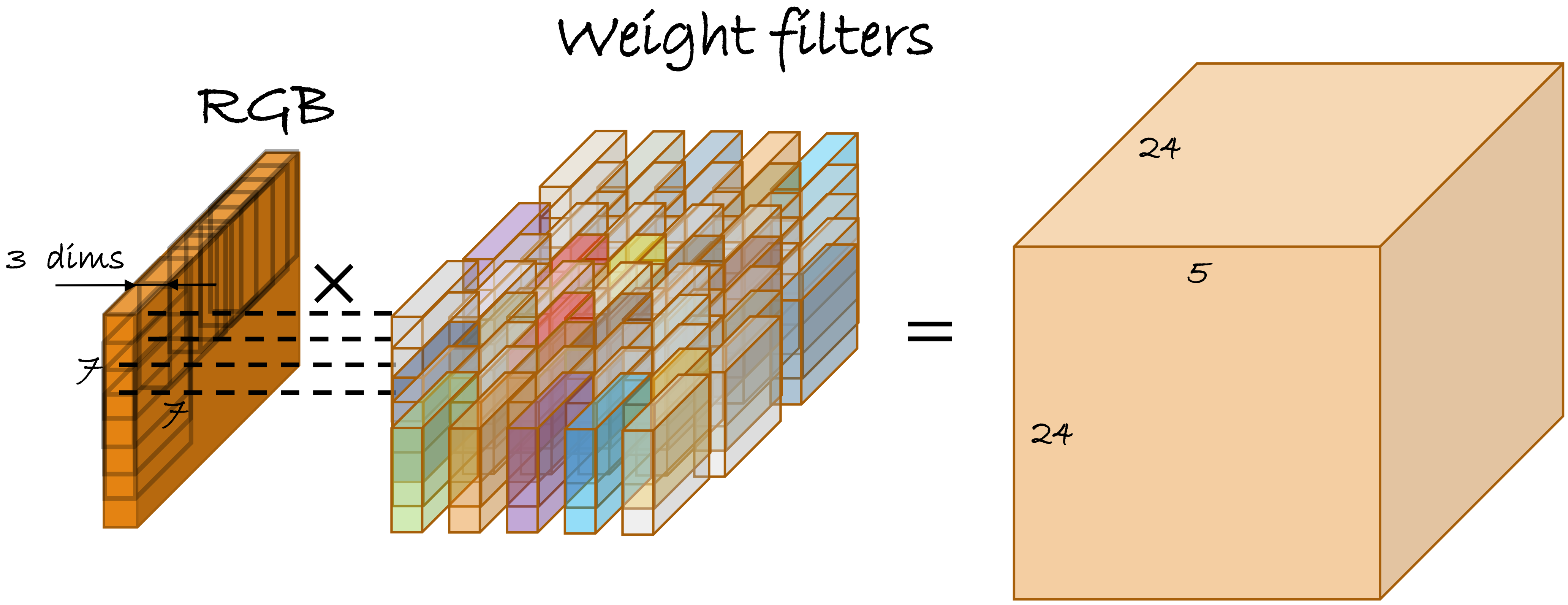
Again, think in space



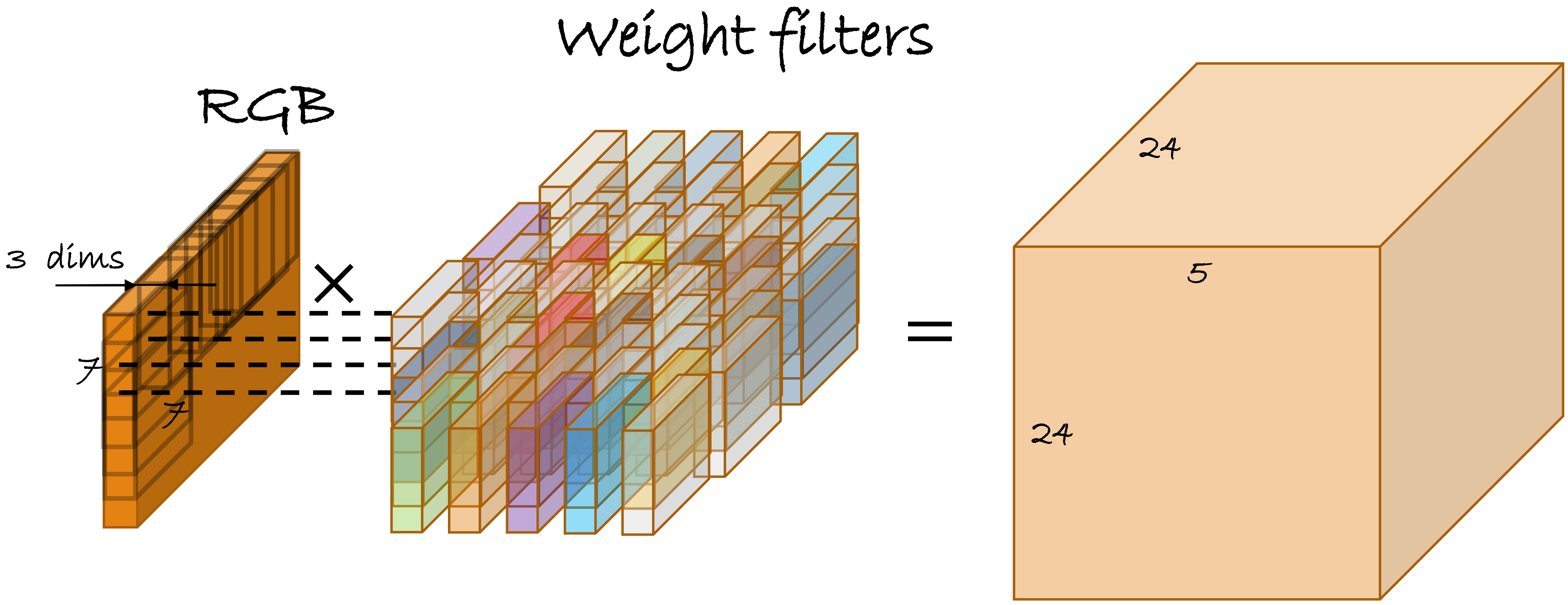
Again, think in space



What about cover the full image with filters?



What about cover the full image with filters?

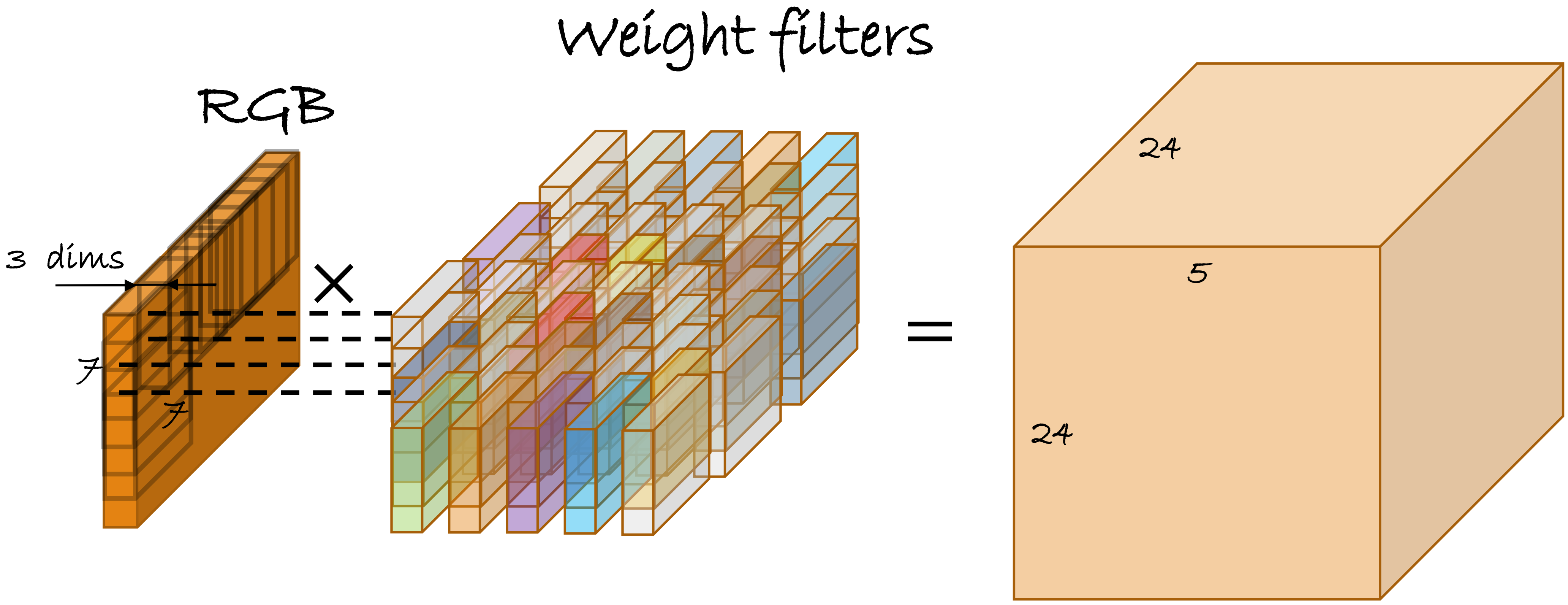


Assume the image is 30x30x3.

1 filter every pixel (stride =1)

How many parameters in total?

What about cover the full image with filters?



Assume the image is 30x30x3.
1 filter every pixel (stride =1)
How many parameters in total?

24 filters along the x axis
24 filters along the y axis
Depth of 5
$\times$ $7 * 7 * 3$ parameters per filter

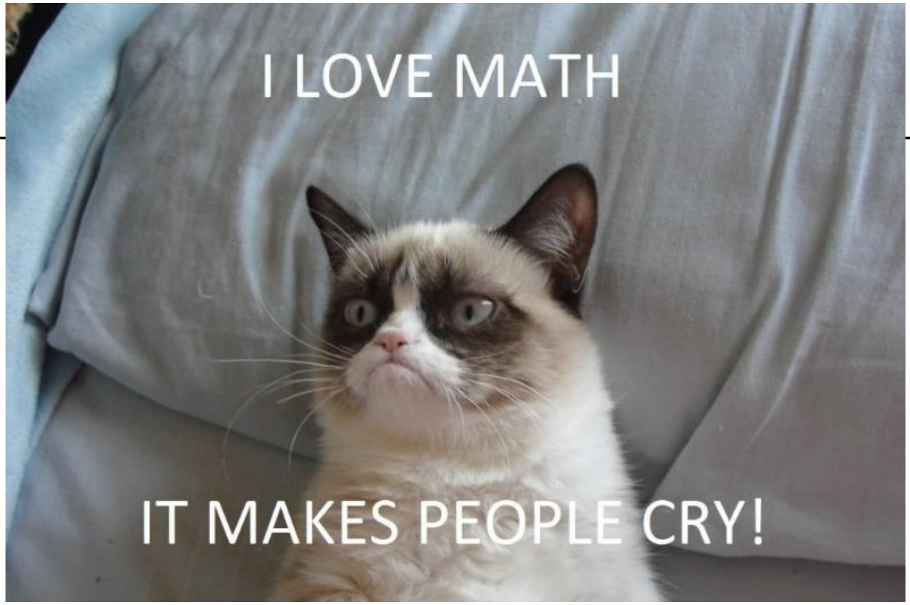
$423K$ parameters in total

Problem!

- Clearly, too many parameters
- With a only 30×30 pixels image and a single hidden layer of depth 5 we would need 85K parameters
  - With a 256×256 image we would need $46 \cdot 10^6$ parameters
- *Problem 1: Fitting a model with that many parameters is not easy*
- *Problem 2: Finding the data for such a model is not easy*
- *Problem 3: Are all these weights necessary?*

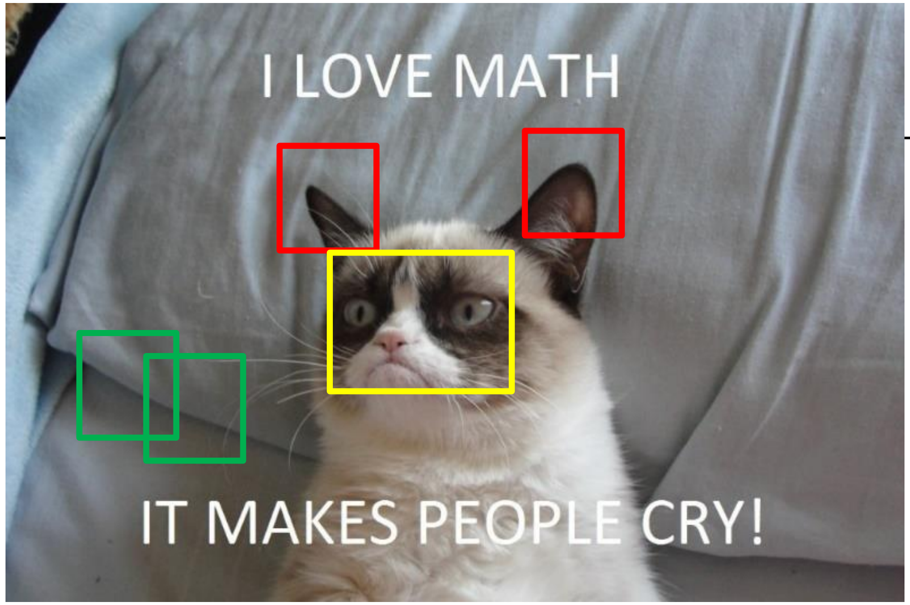
Hypothesis

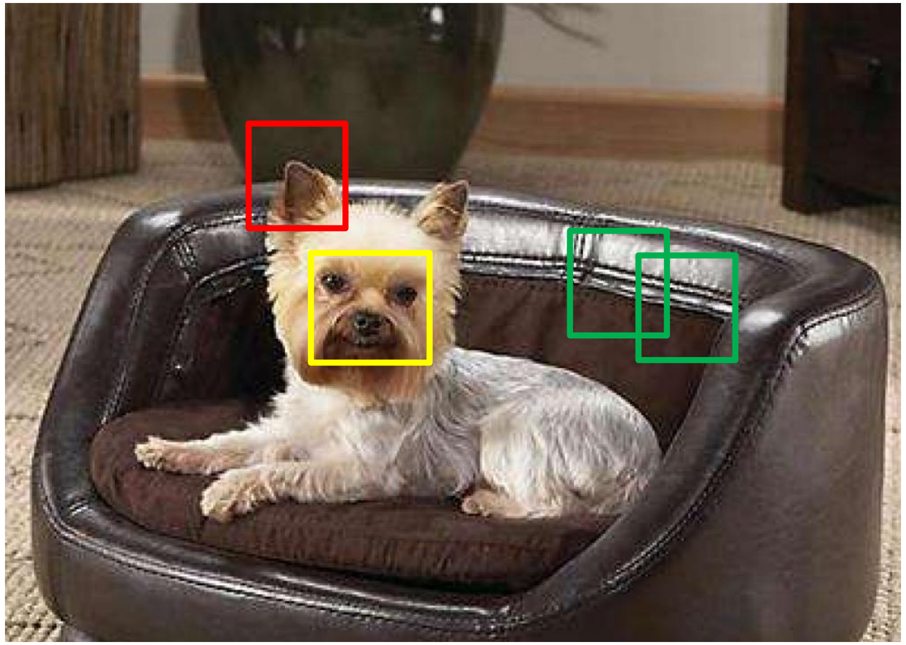
- Imagine
  - With the right amount of data …
  - … and assuming we would connect all input neurons of layer l with all output neurons of layer $l+1$, …
  - … if we would visualize the filters (remember they are 2d) …
  - … we would see very similar plots no matter their location



Hypothesis

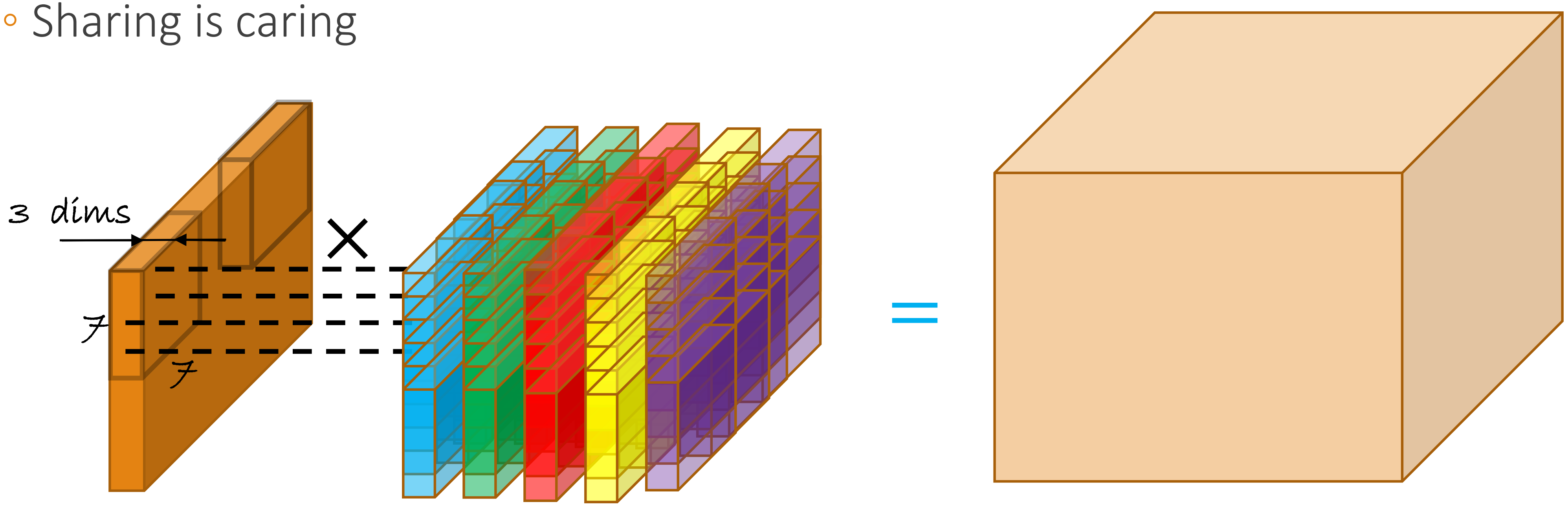
- Imagine
  - With the right amount of data …
  - … and assuming we would connect all input neurons of layer l with all output neurons of layer $l + 1$, …
  - … if we would visualize the filters (remember they are 2d) …
  - … we would see very similar plots no matter their location
- Why?
  - Natural images are stationary
  - Visual features are common for different parts of one or multiple image





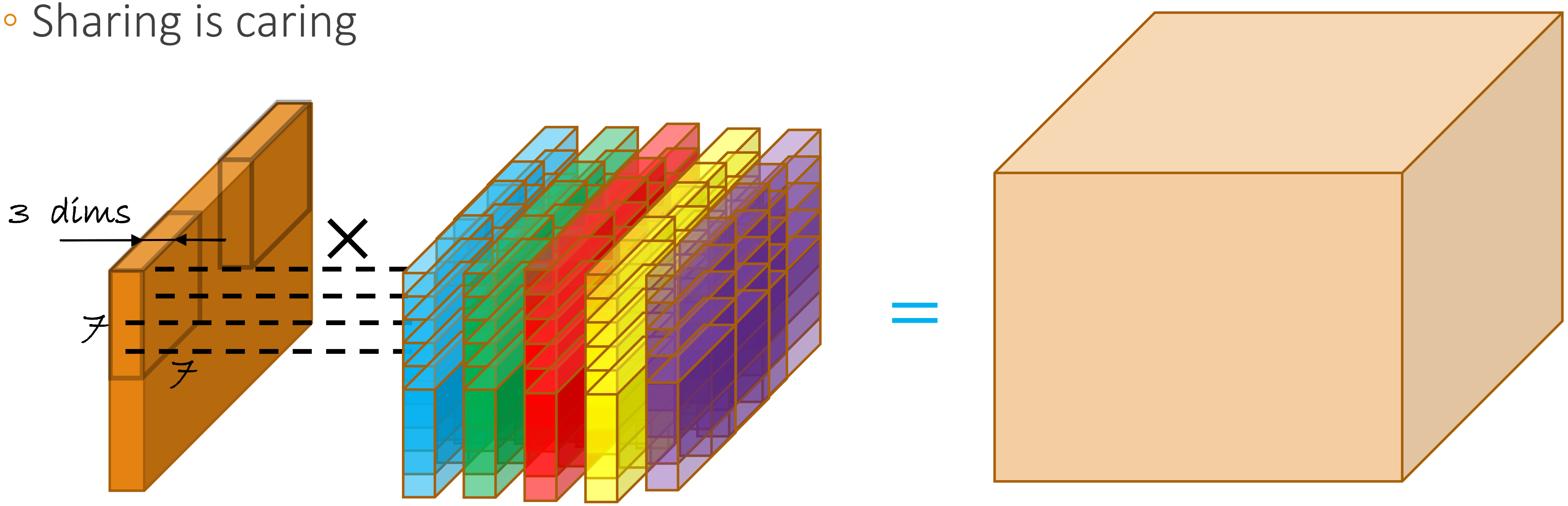
Solution? Share!

- So, if we are anyways going to compute the same filters, why not share?
  - Sharing is caring



Solution? Share!

- So, if we are anyways going to compute the same filters, why not share?
  - Sharing is caring



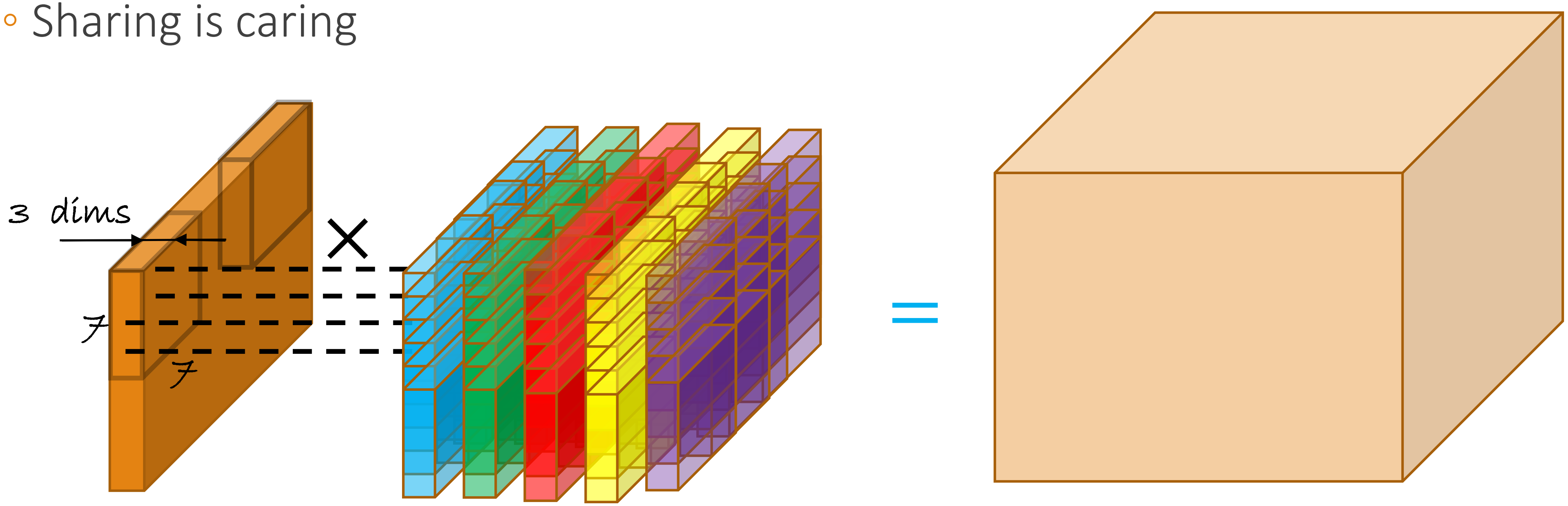
Assume the image is 30x30x3.

1 column of filters common across the image.

How many parameters in total?

Solution? Share!

- So, if we are anyways going to compute the same filters, why not share?
  - Sharing is caring



Assume the image is 30x30x3.
1 column of filters common across the image.
How many parameters in total?

Depth of 5
$\times$ $7 * 7 * 3$ parameters per filter

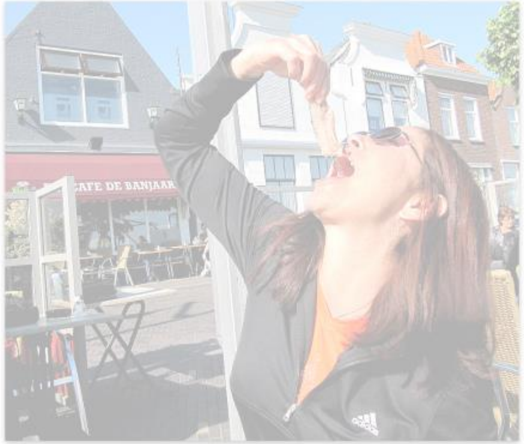
735 parameters in total

Shared 2-D filters → Convolutions

Original image

Shared 2-D filters → Convolutions

Original image



Shared 2-D filters → Convolutions

Original image

Shared 2-D filters → Convolutions

Original image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Convolutional filter 1

0	0	1
0	1	1
1	1	1

Shared 2-D filters → Convolutions

Original image

Convolutional filter 1

0	0	1
0	1	1
1	1	1

Convolving the image

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

Result

4		

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \overbrace{I(x-i,y-j) \cdot h(i,j)}^{\text{Inner product}}$$

Shared 2-D filters → Convolutions

Original image

Convolutional filter 1

0	0	1
0	1	1
1	1	1

Convolving the image

1	1 x1	1 x0	0 x1	0
0	1 x0	1 x1	1 x0	0
0	0 x1	1 x0	1 x1	1
0	0	1	1	0
0	1	1	0	0

Result

4	3	

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x \overbrace{- i, y - j) \cdot h(i,j)}^{\text{Inner product}}$$

Shared 2-D filters → Convolutions

Original image

Convolutional filter 1

0	0	1
0	1	1
1	1	1

Convolving the image

1	1	1 x1	0 x0	0 x1
0	1	1 x0	1 x1	0 x0
0	0	1 x1	1 x0	1 x1
0	0	1	1	0
0	1	1	0	0

Result

4	3	4

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \overbrace{I(x-i, y-j) \cdot h(i,j)}^{\text{Inner product}}$$

Shared 2-D filters → Convolutions

Original image

Convolutional filter 1

0	0	1
0	1	1
1	1	1

Convolving the image

1	1	1	0	0
0 x1	1 x0	1 x1	1	0
0 x0	0 x1	1 x0	1	1
0 x1	0 x0	1 x1	1	0
0	1	1	0	0

Result

4	3	4
2		

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \overbrace{I(x-i,y-j) \cdot h(i,j)}^{\text{Inner product}}$$

Shared 2-D filters → Convolutions

Original image

Convolutional filter 1

0	0	1
0	1	1
1	1	1

Convolving the image

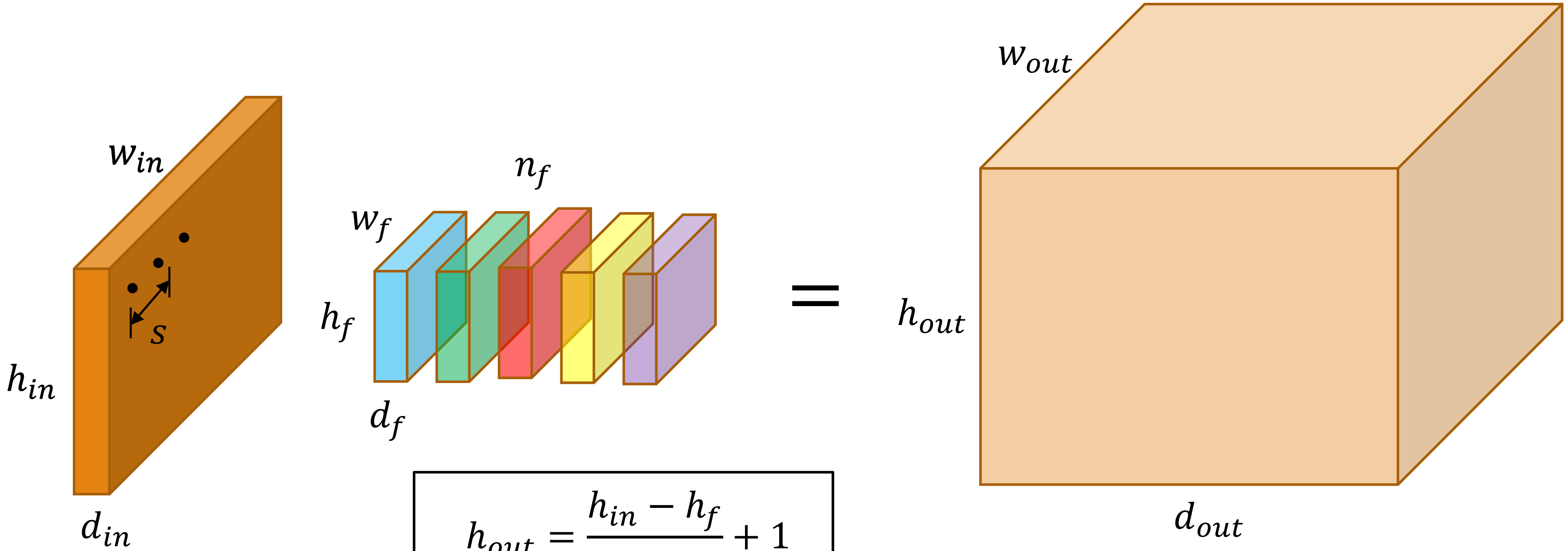
1	1	1	0	0
0	1	1	1	0
0	0	1 x1	1 x0	1 x1
0	0	1 x0	1 x1	0 x0
0	1	1 x1	0 x0	0 x1

Result

4	3	4
2	4	3
2	3	4

$$I(x,y) * h = \sum_{i=-a}^{a} \sum_{j=-b}^{b} I(x \overbrace{- i, y - j) \cdot h(i,j)}^{\text{Inner product}}$$

Output dimensions?



$$h_{out} = \frac{h_{in} - h_f}{s} + 1$$

$$w_{out} = \frac{w_{in} - w_f}{s} + 1$$

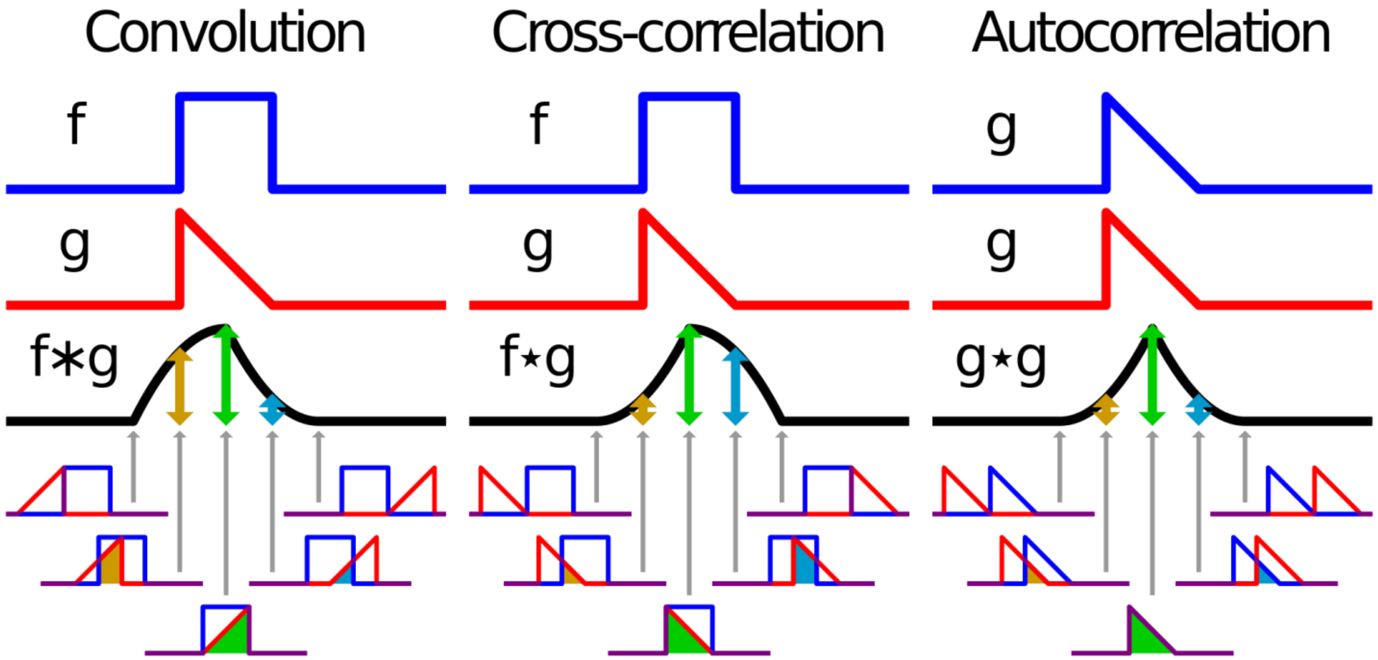
$$d_{out} = n_f$$

$$d_f = d_{in}$$

Why call them convolutions?

Definition *The convolution of two functions f and g is denoted by $*$ as the integral of the product of the two functions after one is reversed and shifted*

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\, d\tau$$

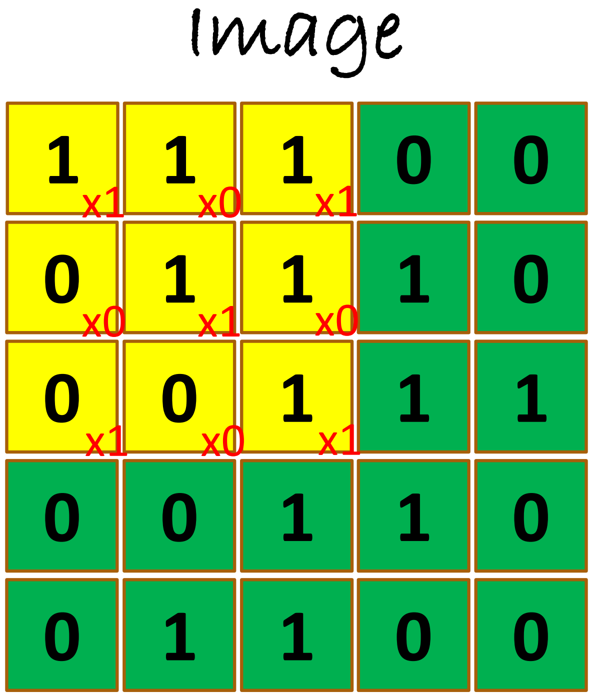


Problem, again :S

- Our images get smaller and smaller
- Not too deep architectures
- Details are lost

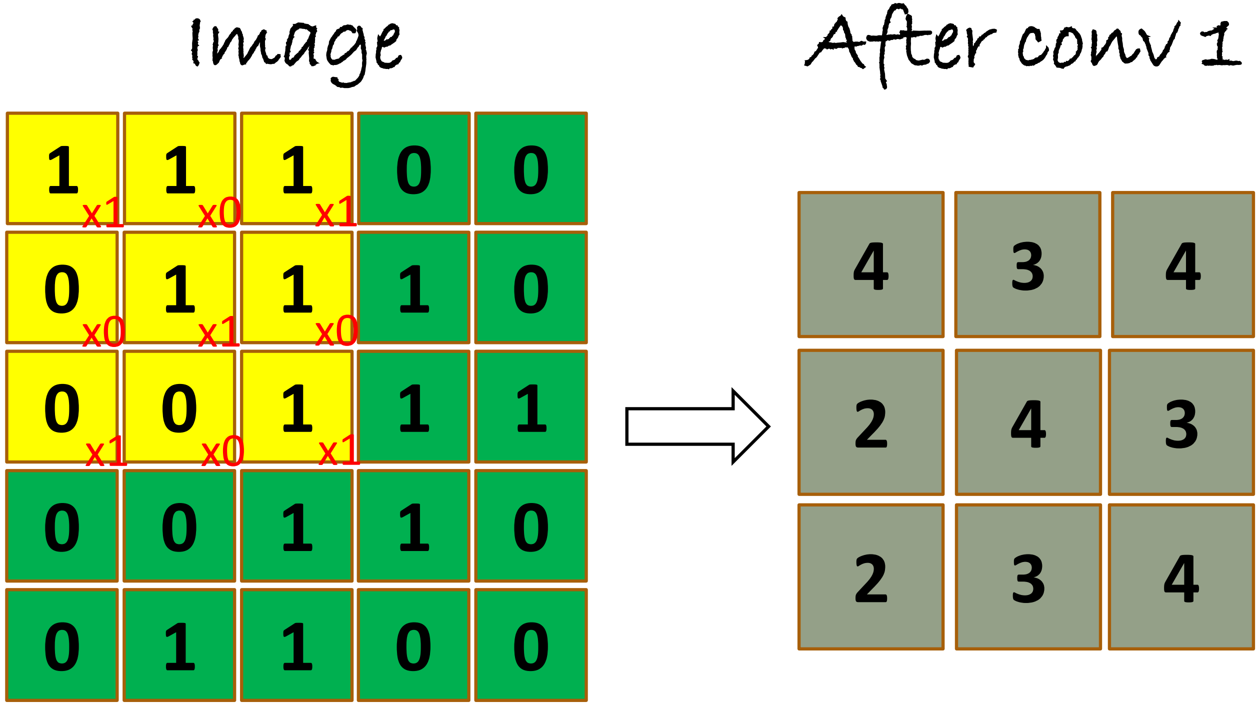
Problem, again :S

- Our images get smaller and smaller
- Not too deep architectures
- Details are lost



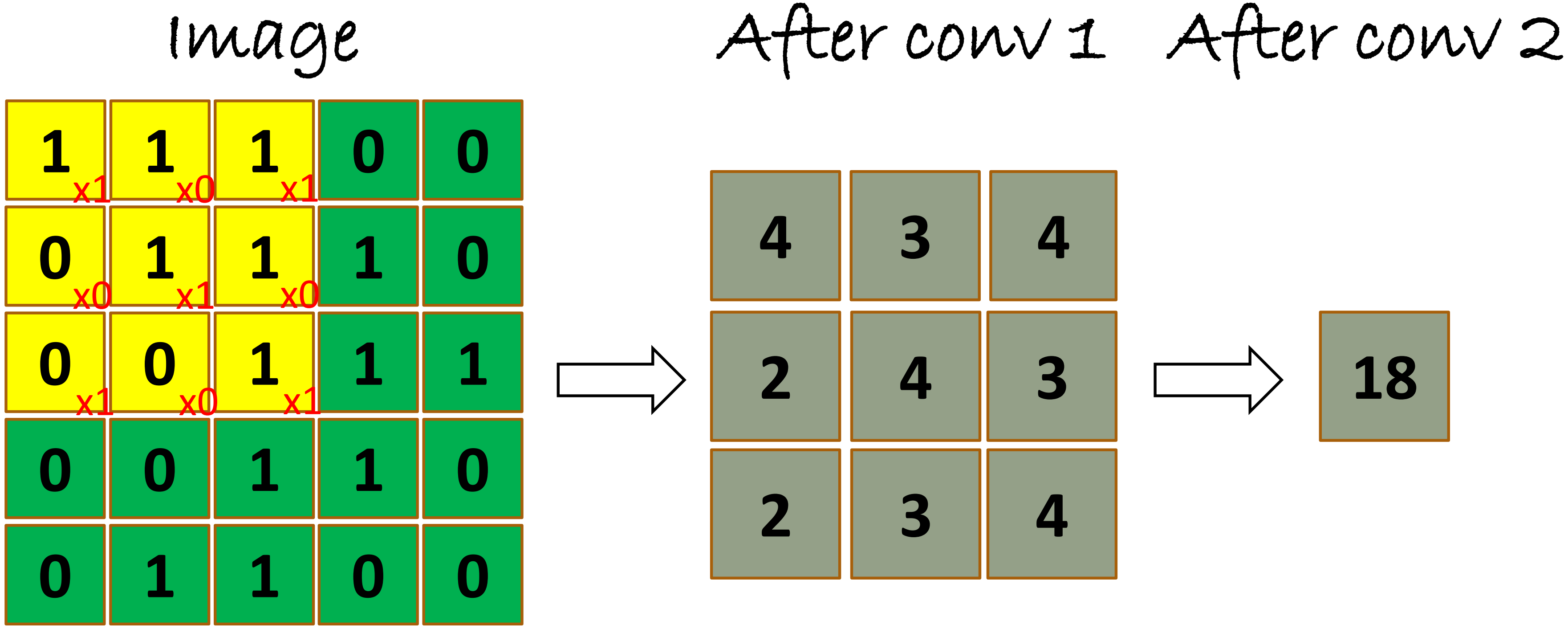
Problem, again :S

- Our images get smaller and smaller
- Not too deep architectures
- Details are lost



Problem, again :S

- Our images get smaller and smaller
- Not too deep architectures
- Details are lost



Solution? Zero-padding!

- For $s = 1$, surround the image with $(h_f - 1)/2$ and $(w_f - 1)/2$ layers of 0

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

1	1	2	0	0
0	1	1	1	0
0	0	1	2	1
1	0	2	1	0
0	1	1	3	0

Solution? Zero-padding!

- For $s = 1$, surround the image with $(h_f - 1)/2$ and $(w_f - 1)/2$ layers of 0

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

1	1	2	0	0
0	1	1	1	0
0	0	1	2	1
1	0	2	1	0
0	1	1	3	0

Solution? Zero-padding!

- For $s = 1$, surround the image with $(h_f - 1)/2$ and $(w_f - 1)/2$ layers of 0

$$
\begin{matrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}
\quad * \quad
\begin{matrix}
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1
\end{matrix}
\qquad
\begin{matrix}
1 & 1 & 2 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 2 & 1 \\
1 & 0 & 2 & 1 & 0 \\
0 & 1 & 1 & 3 & 0
\end{matrix}
$$

Solution? Zero-padding!

- For $\mathrm{s} = 1$, surround the image with $(h_f - 1)/2$ and $(w_f - 1)/2$ layers of 0

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 2 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 2 & 1 \\
1 & 0 & 2 & 1 & 0 \\
0 & 1 & 1 & 3 & 0
\end{bmatrix}
$$

Convolutional module (New module!!!)

- Activation function

$$a_{rc} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} x_{r-i,c-j} \cdot \theta_{ij}$$

- Gradient w.r.t. the parameters

$$\frac{\partial a_{rc}}{\partial \theta_{ij}} = \sum_{r=0}^{N-2a} \sum_{c=0}^{N-2b} x_{r-i,c-j}$$

- Module and variants already implemented in Torch

Convolutional module in Torch

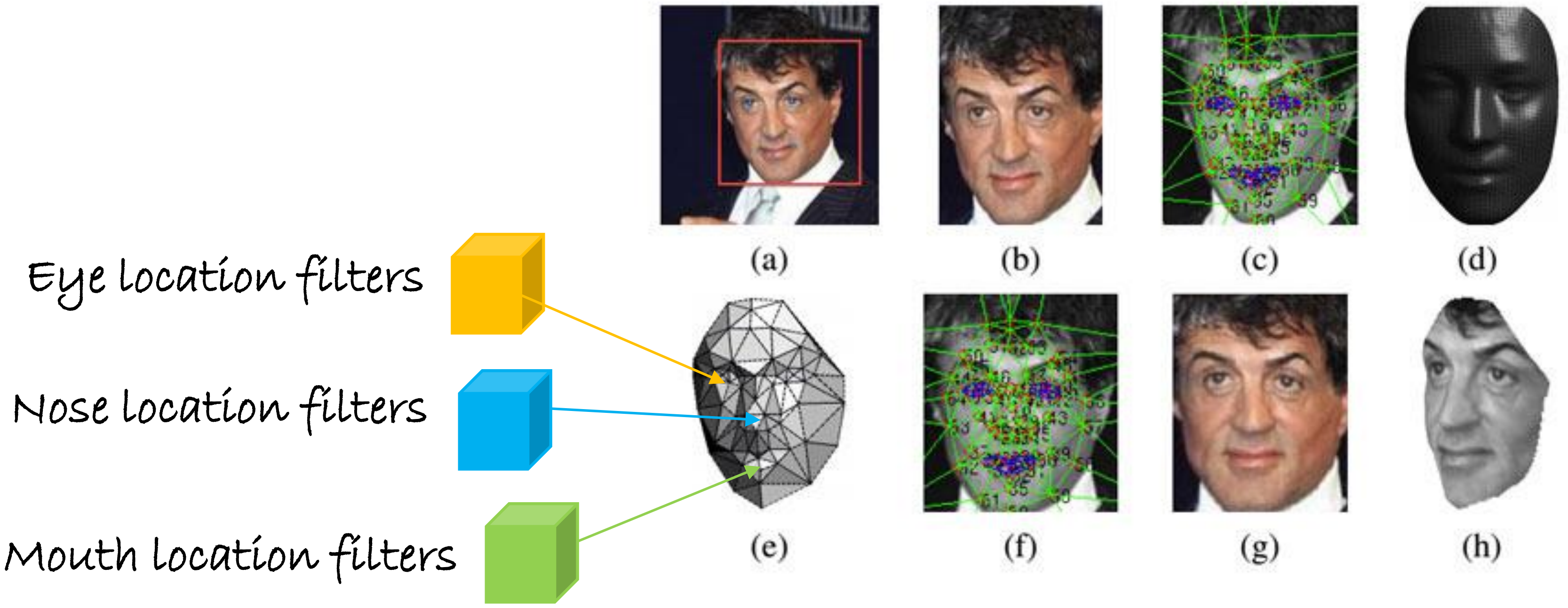
```
require ‘nn’
nn.SpatialConvolution(d_in, d_out, w_f, h_f, s_w, s_h)
```

Good practice

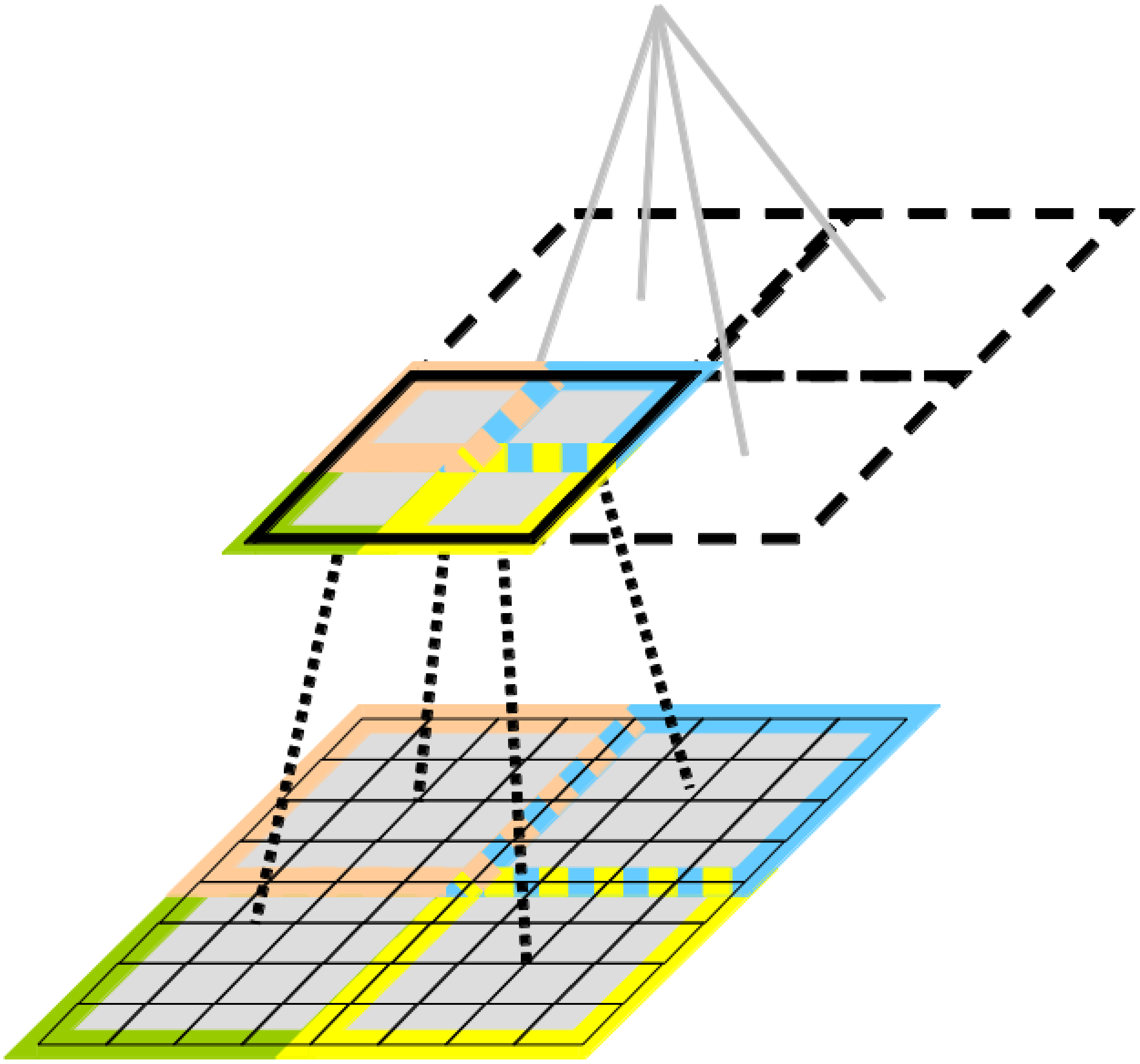
- Resize the image to have a size in the power of 2
- Use stride $s = 1$
- A filter of $\left(h_f, w_f\right) = [3 \times 3]$ works quite alright with deep architectures
- Add 1 layer of zero padding
- In general avoid combinations of hyper-parameters that do not click
  - E.g. $s = 1$
  - $[h_f \times w_f] = [3 \times 3]$ and
  - image size $[h_{in} \times w_{in}] = [6 \times 6]$
  - $[h_{out} \times w_{out}] = [2.5 \times 2.5]$
  - Programmatically worse, and worse accuracy because borders are ignored

P.S. Sometimes convolutional filters are not preferred

- When images are registered and each pixel has a particular significance
  - E.g. after face alignment specific pixels hold specific types of inputs, like eyes, nose, etc.
- In these cases maybe better every spatial filter to have different parameters
  - Network learns particular weights for particular image locations [Taigman2014]

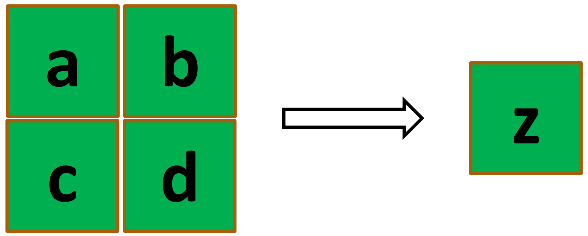


Pooling



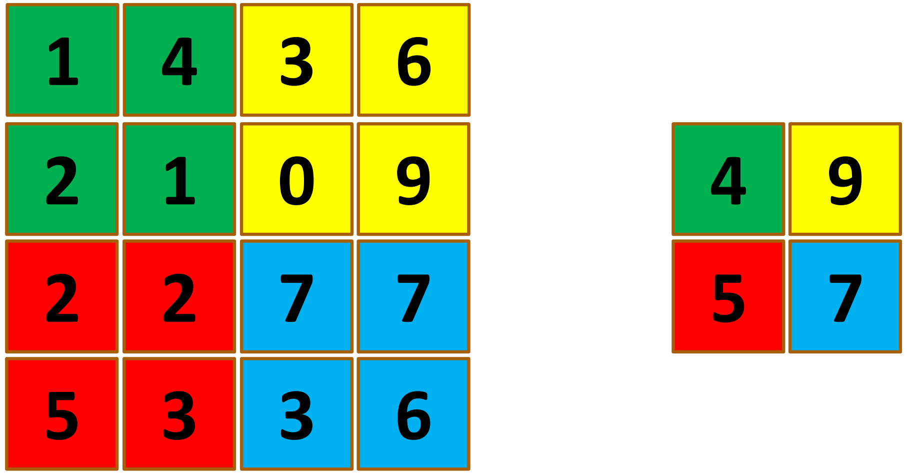
Pooling

- A function that aggregates multiple inputs into a single value
- Reduces the size of the layer output
  - Reduces the input for the next layer
  - Faster computations
  - Keeps the most important information for the next layer
- Max pooling
- Average pooling



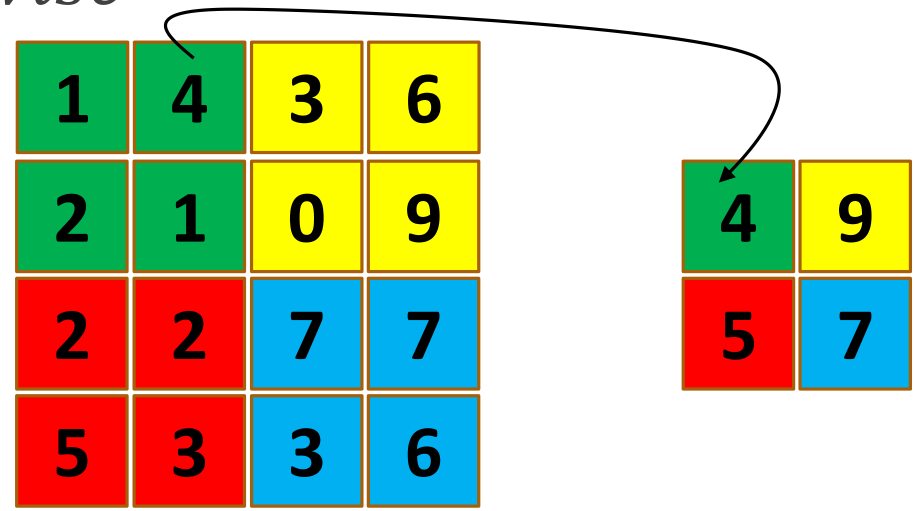
Max pooling (New module!)

- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $\mathrm{i}_{\max}, \mathrm{j}_{\max} = \underset{i,j \in \Omega(r,c)}{\arg\max}\, x_{ij} \rightarrow a_{rc} = x[\mathrm{i}_{\max}, \mathrm{j}_{\max}]$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & \mathit{if}\ i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & \mathit{otherwise} \end{cases}$
- The preferred choice of pooling



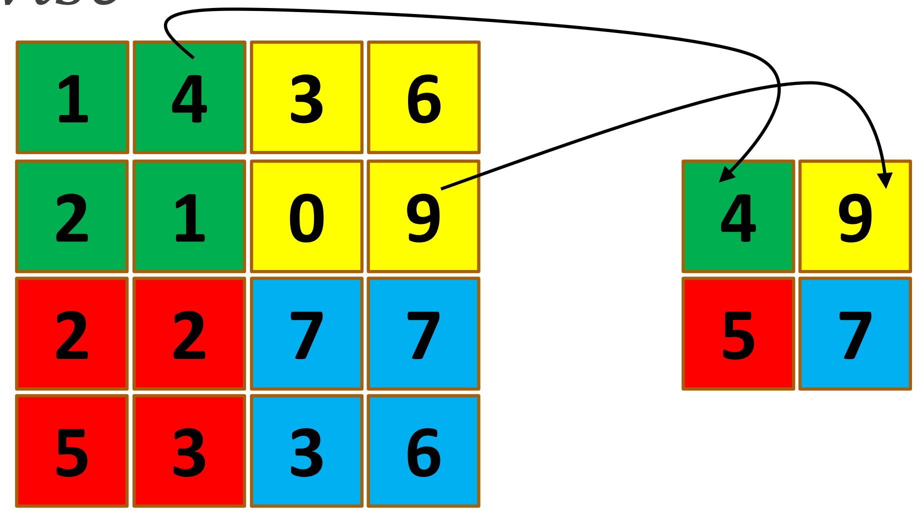
Max pooling (New module!)

- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $\mathrm{i}_{\max}, \mathrm{j}_{\max} = \underset{i,j \in \Omega(r,c)}{\arg\max}\, x_{ij} \rightarrow a_{rc} = x[\mathrm{i}_{\max}, \mathrm{j}_{\max}]$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & if\ i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & otherwise \end{cases}$
- The preferred choice of pooling



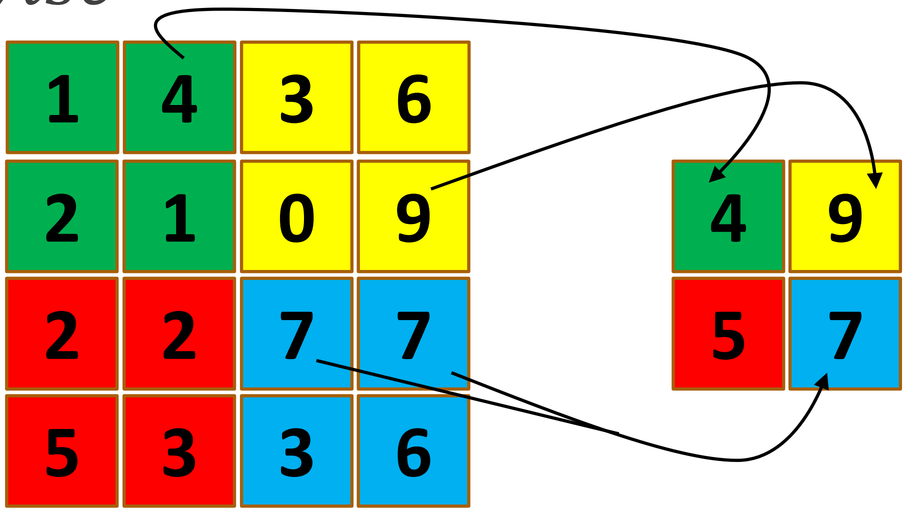
Max pooling (New module!)

- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $\mathrm{i}_{\max}, \mathrm{j}_{\max} = \underset{i,j \in \Omega(r,c)}{\arg\max}\, x_{ij} \rightarrow a_{rc} = x[\mathrm{i}_{\max}, \mathrm{j}_{\max}]$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & if\ i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & otherwise \end{cases}$
- The preferred choice of pooling



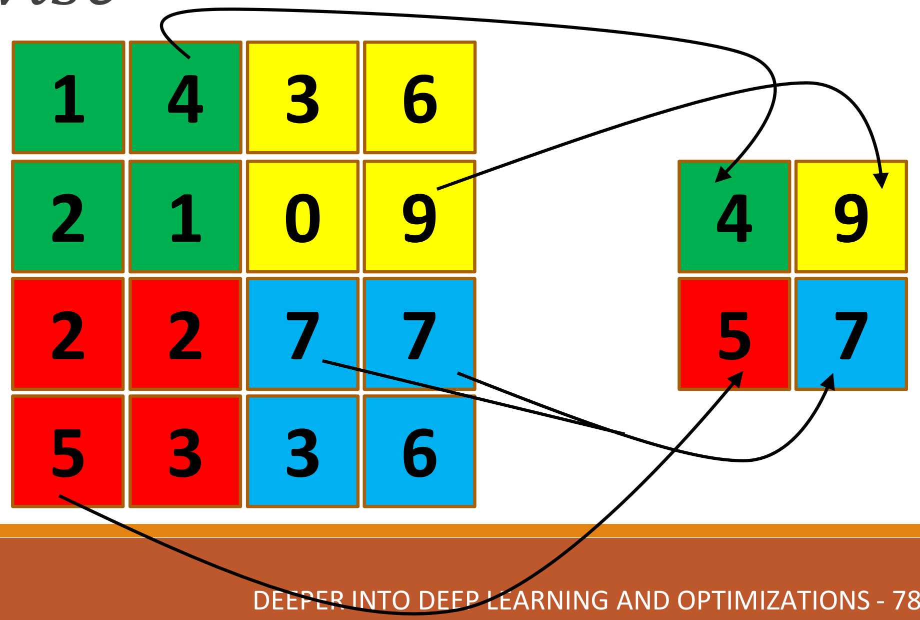
Max pooling (New module!)

- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $\mathrm{i_{max}}, \mathrm{j_{max}} = \arg\max_{i,j \in \Omega(r,c)} x_{ij} \rightarrow a_{rc} = x[\mathrm{i_{max}}, \mathrm{j_{max}}]$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & if\ i = \mathrm{i_{max}}, \mathrm{j} = \mathrm{j_{max}} \\ 0, & otherwise \end{cases}$
- The preferred choice of pooling



Max pooling (New module!)

- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $\mathrm{i}_{\max}, \mathrm{j}_{\max} = \arg\max_{i,j \in \Omega(r,c)} x_{ij} \rightarrow a_{rc} = x[\mathrm{i}_{\max}, \mathrm{j}_{\max}]$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \begin{cases} 1, & if\ i = \mathrm{i}_{\max}, \mathrm{j} = \mathrm{j}_{\max} \\ 0, & otherwise \end{cases}$
- The preferred choice of pooling



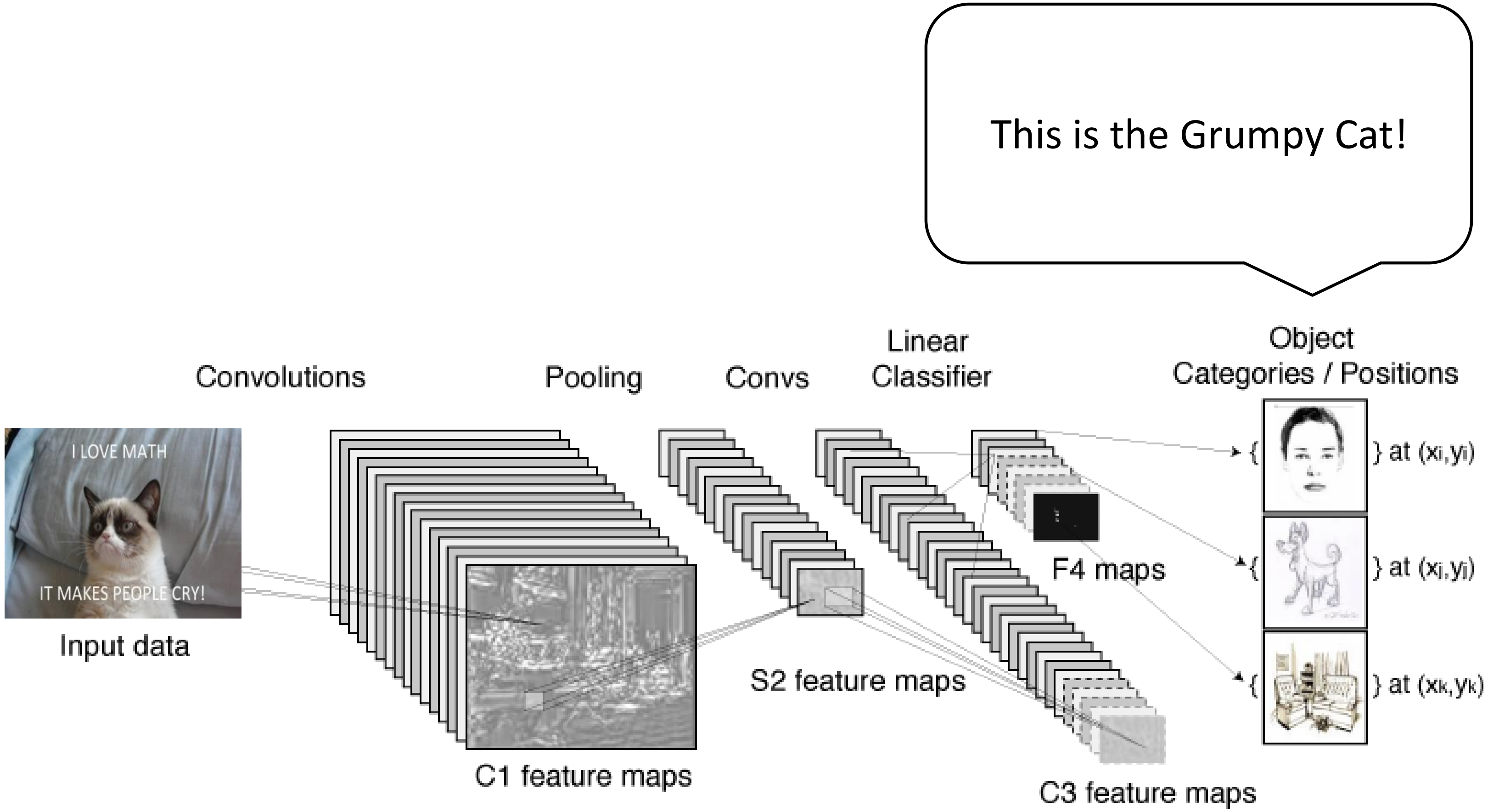
Average pooling (New module!)

- Run a sliding window of size $[h_f, w_f]$
- At each location keep the maximum value
- Activation function: $a_{rc} = \frac{1}{r \cdot c} \sum_{i,j \in \Omega(r,c)} x_{ij}$
- Gradient w.r.t. input $\frac{\partial a_{rc}}{\partial x_{ij}} = \frac{1}{r \cdot c}$

1	4	1	6
2	3	0	9
1	2	7	1
4	1	0	2

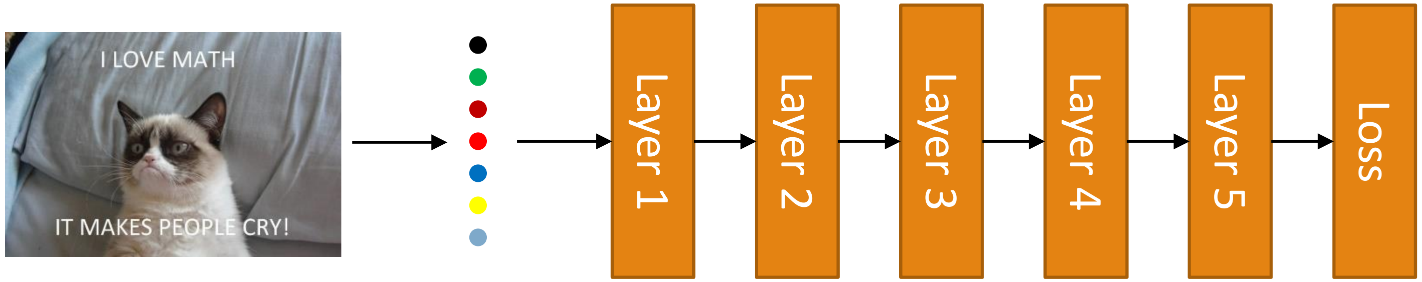
5	8
4	5

Convnets for Object Recognition

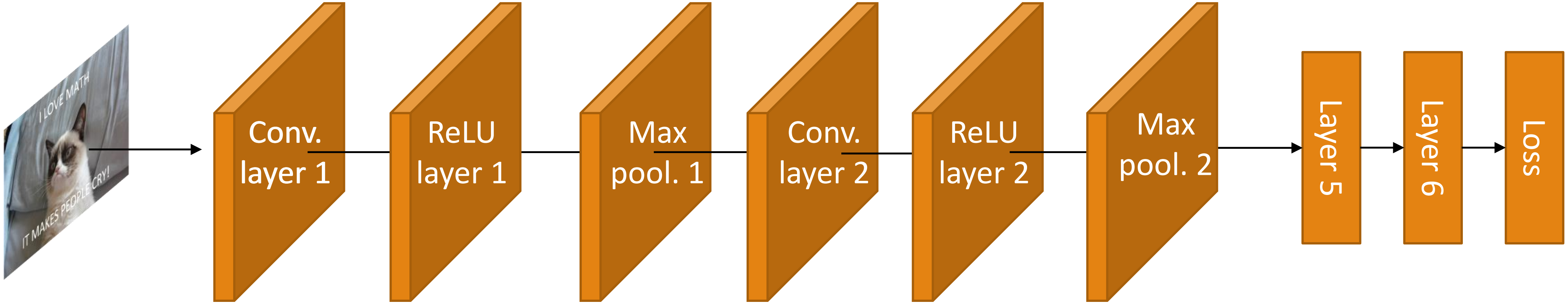


Standard Neural Network vs Convnets

Neural Network



Convolutional Neural Network



Convets in practice

- Several convolutional layers
  - 5 or more
- After the convolutional layers non-linearities are added
  - The most popular one is the ReLU
- After the ReLU usually some pooling
  - Most often max pooling
- After 5 rounds of cascading, vectorize last convolutional layer and connect it to a fully connected layer
- Then proceed as in a usual neural network

CNN Case Study I: Alexnet

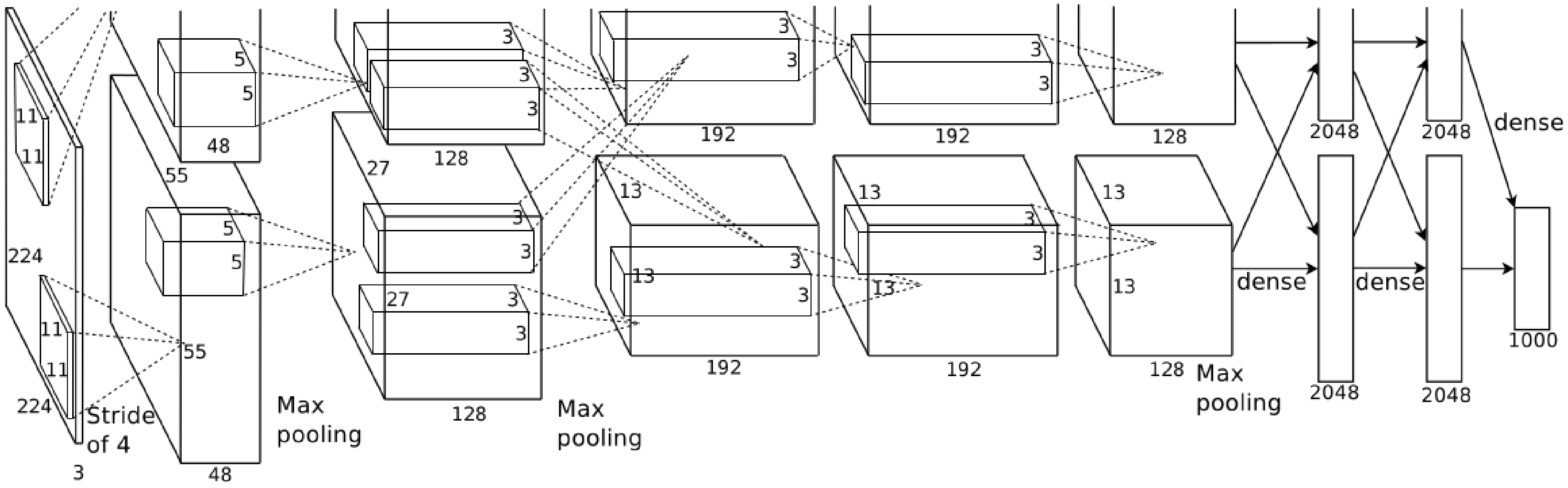
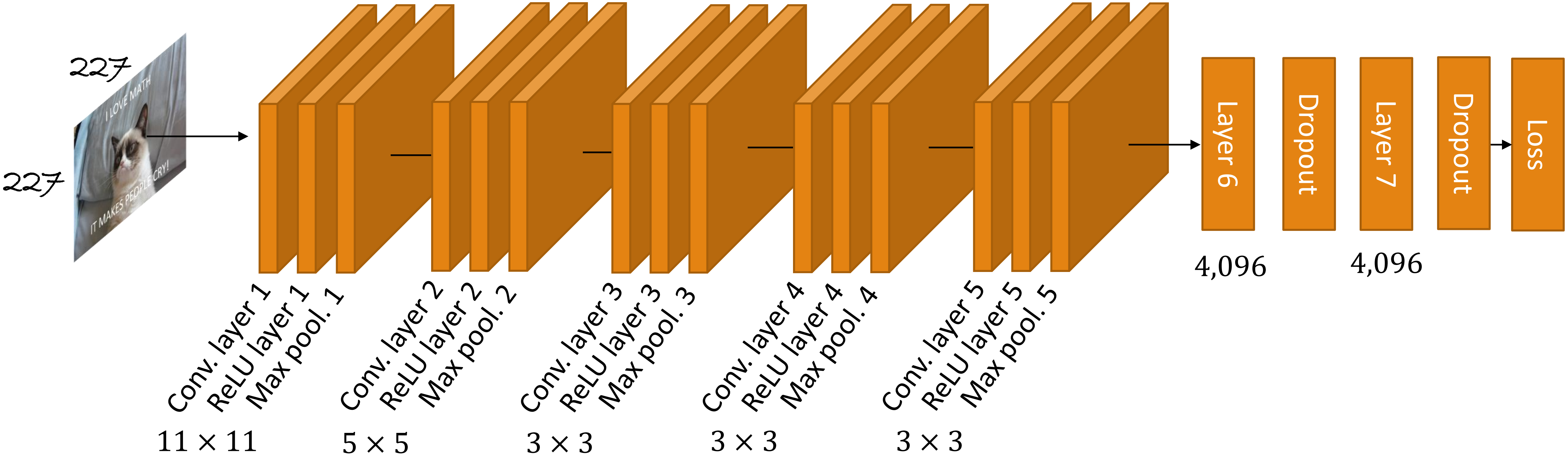


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Alexnet prototxt (Caffe configuration file)

https://github.com/BVLC/caffe/blob/master/models/bvlc_alexnet/train_val.prototxt

Architectural details



CNN Case Study II: VGGnet

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGGnet prototxt

https://gist.github.com/ksimonyan/211839e770f7b538e2d8#file-vgg_ilsvrc_16_layers_deploy-prototxt

More cases

- Two-stream network
  - Moving images (videos)
- Network in Network
- Deep Fried Network
- Resnet
  - Winner of ILSVRC 2016

Summary

- What are the Convolutional Neural Networks?
- Why are they so important for Computer Vision?
- How do they differ from standard Neural Networks?
- How can we train a Convolutional Neural Network?

Next lecture

- What do convolutions look like?
- Build on the visual intuition behind Convnets
- Deep Learning Feature maps
- Transfer Learning