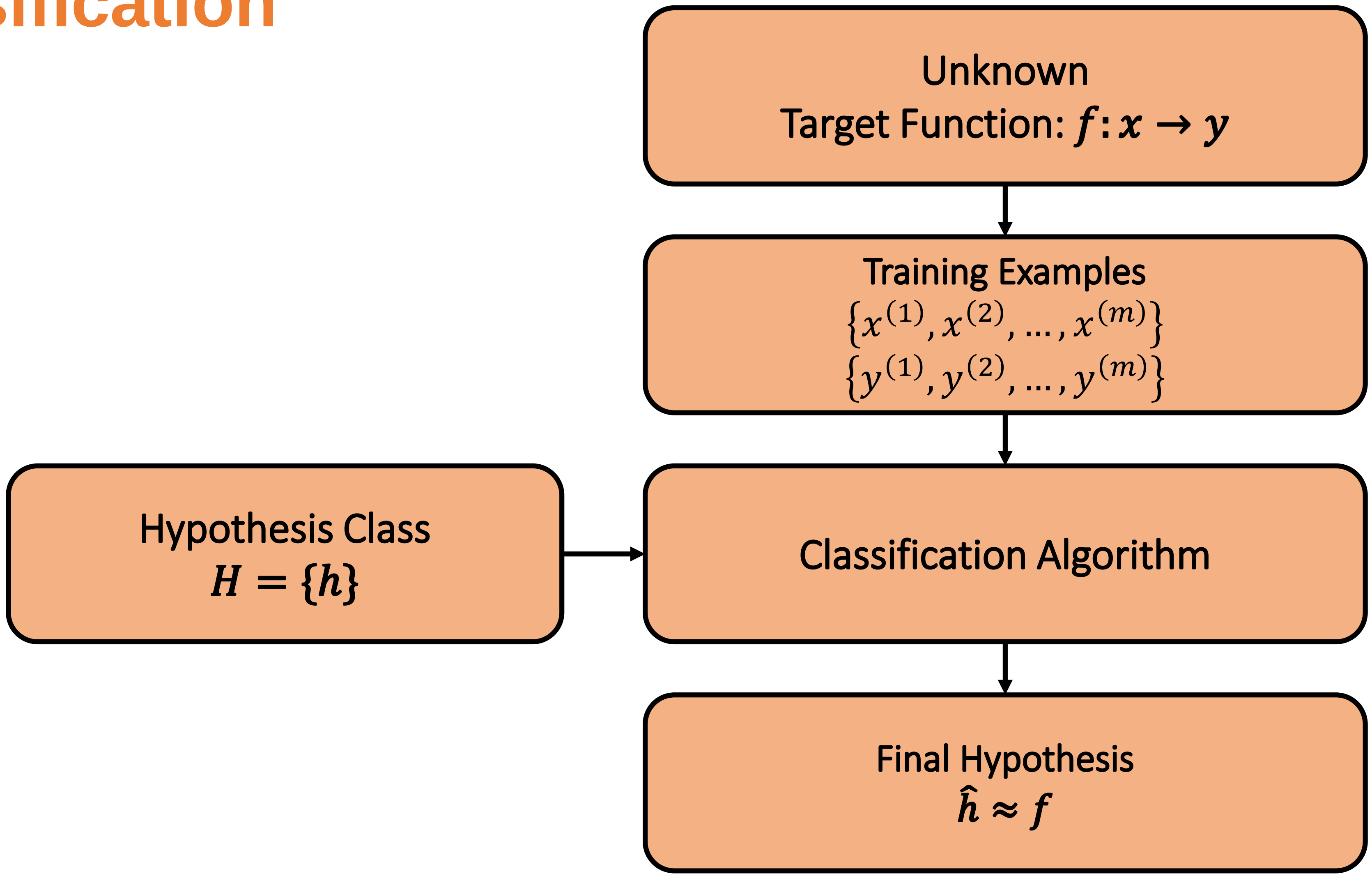
Machine Learning in Healthcare

#C20 Introduction to Convolutional Neural Network

Technion-IIT, Haifa, Israel

Assist. Prof. Joachim Behar
Biomedical Engineering Faculty
Technion-IIT

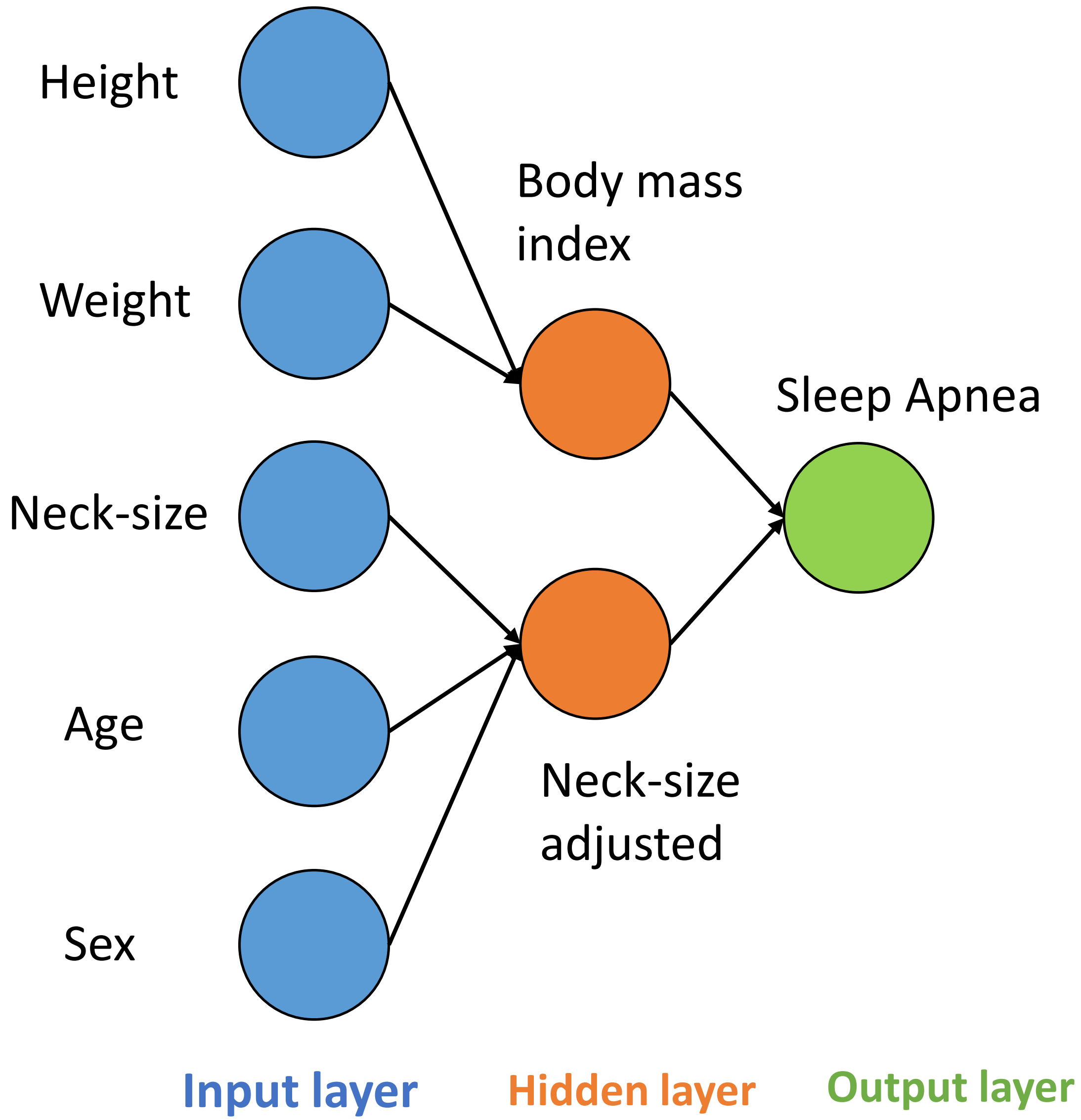
Classification



Intuition

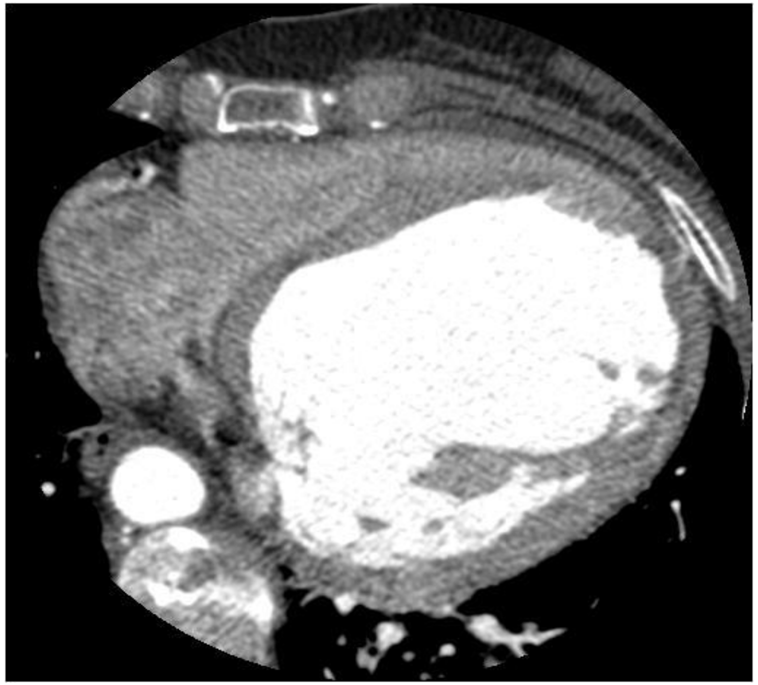
Parameters estimation in NN

- Categorical data with a limited number input features.
- Not many parameters to estimate, here: 10 x 2 +2 = 22 weights parameters.



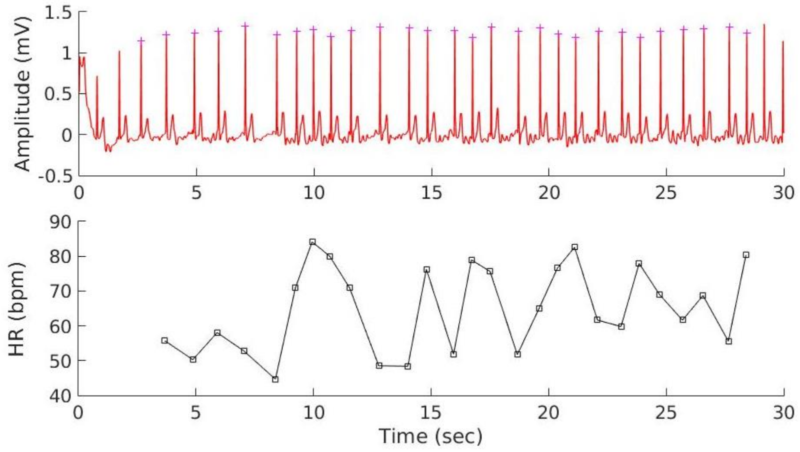
Parameters estimation in NN, images

- Consider an image which is 1000 x 1000 pixels, with three color channels (RGB) and one a NN with 1000 neurons in the hidden layer.
- How many weight parameters do we have to estimate?
- $W^{[1]} \in \mathbb{R}^{1000 \cdot 3M}$ which makes 3 billions parameters to estimate.
- And this is considering only one hidden layer.



Parameters estimation in NN, temporal time series

- Consider an ECG time series sampled at 1kHz and a window size of 30 seconds for classifying the ECG segment as arrhythmia or not.
- How many weight parameters do we have to estimate?
- $W^{[1]} \in \mathbb{R}^{1000 \cdot 30000}$ which makes 30 millions parameters to estimate.
- And this is considering only one hidden layer.



Parameters estimation in NN

- Learning such a high number of parameters is challenging. How can we better deal with this type of data?
- Recall the intuition of Deep learning as a type of representation learning:
  - Learn more and more complex features as we go deeper in the network.
- How, would we get the first level of features without having “3 billions” weights parameters to estimate?
- How could we detect edges in a “cheap” manner?
- Can we instead kind of feed the first hidden layer with edges being detected from filters?

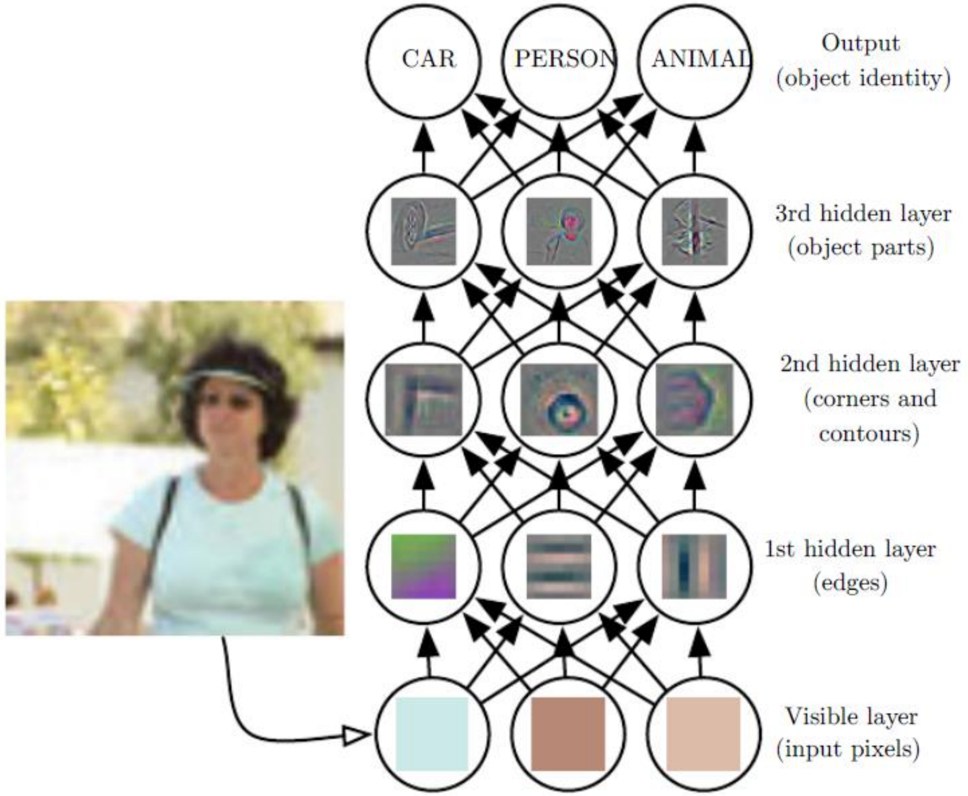
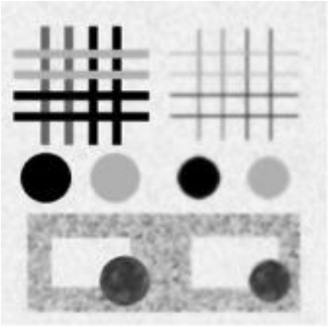
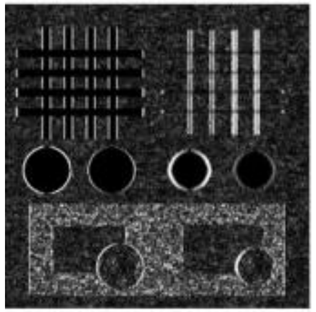


Image from Zeiler and Fergus (2014).

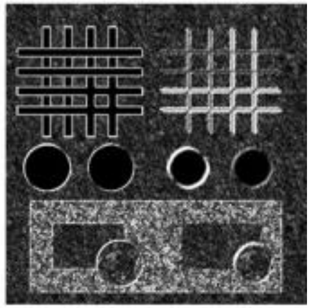
How do we detect edges in images?

- Horizontal derivatives:
  - Gradient:
    - $G_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * X$
  - Sobel:
    - $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * X$
- Vertical derivatives:
  - Gradient:
    - $G_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * X$
  - Sobel:
    - $G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * X$


Original


Sobel X

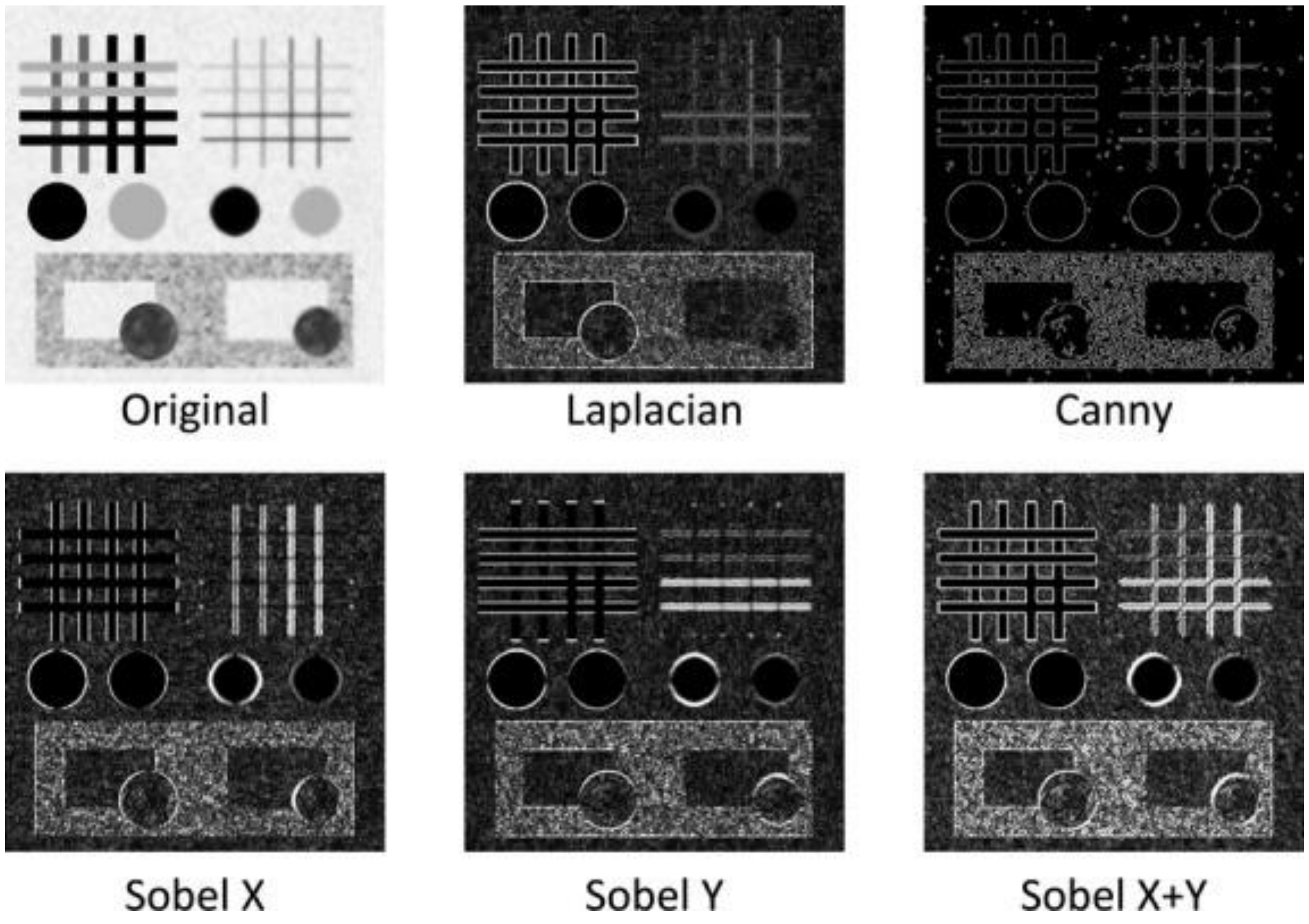

Sobel Y


Sobel X+Y

How do we detect edges in images?

- What about edges that would be at a specific angle (e.g. $40°$)?
- What if images have noise embedded?
  - Canny edge detector:
    - $B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * X$
    - Gaussian smoothing + edge detection.

How do we detect edges in images?



Uchida, Seiichi. "Image processing and recognition for biological images." Development, growth & differentiation 55.4 (2013): 523-549.

How do we detect edges in images?

- So there are different flavors of “edge detection filters”. Instead, of using a specific defined filter, could we learn it from data?

- Derivative along x-axis:
  - Defined filter (e.g. gradient):
    - $G_x = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * X,$
  - Learn from data $\{w_{i,j}\}$:
    - $G_x = \begin{bmatrix} w_{11} & w_{21} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} * X$

Summary

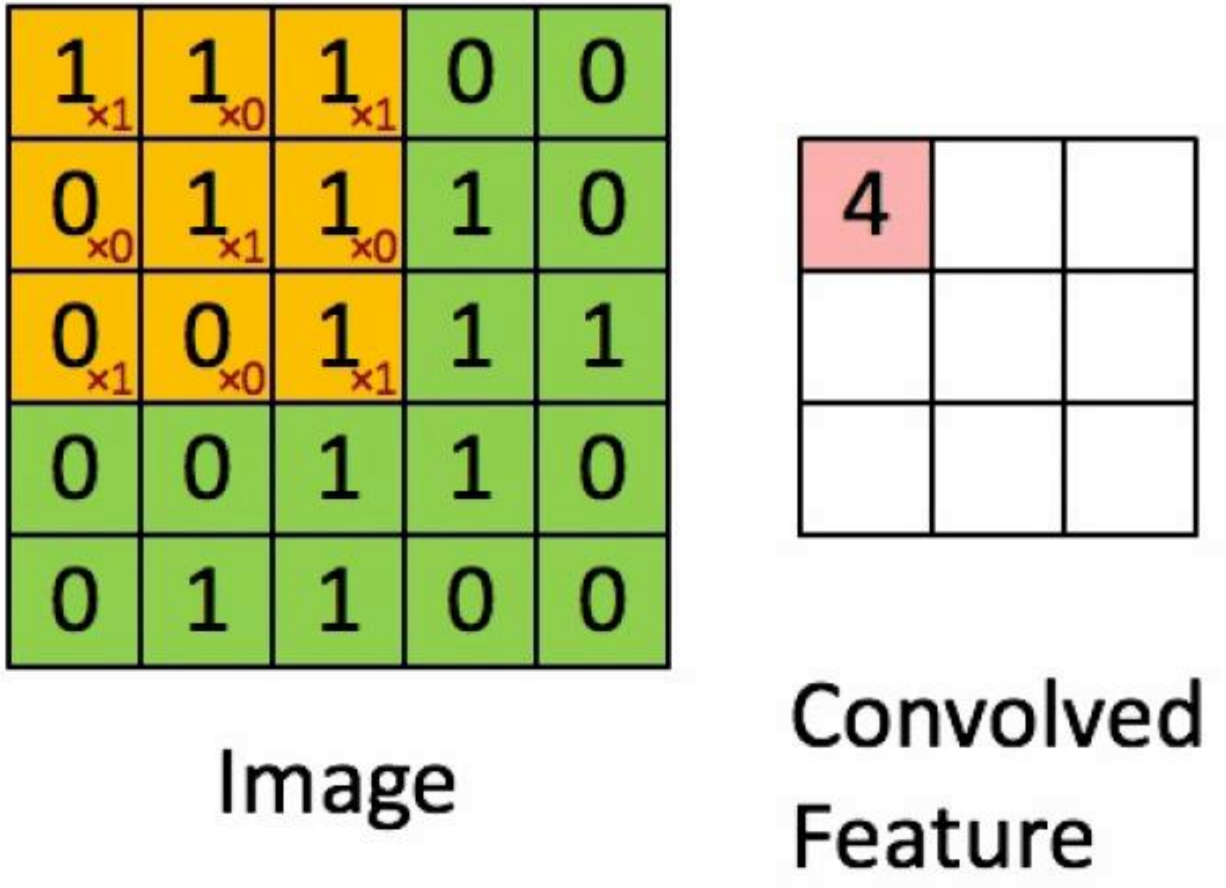
- Too many parameters to estimate when dealing with images or large time series.
- We seek a way to reduce the number of free parameters.
- We elaborated on the feasibility to detect edges using pre-defined filters (Sobel, Canny etc.). We could feed a NN with these “engineered” first level features.
- We pointed to the fact that these filters are the implementation of different ideas/insights but that there would be value in learning from data what filter coefficients to use rather than using a pre-defined template.
  - If we take back our initial example of an image 1000 x 1000 with RGB channels and one hidden layer of 1000 neurons, we had 3 billions parameters.
  - If we now consider 10 filters of size 3 x 3 we wish to learn then we have 28*10 = 280 parameters.
- This provides the insight behind Convolutional Neural Network.

Cross-correlation versus Convolutions

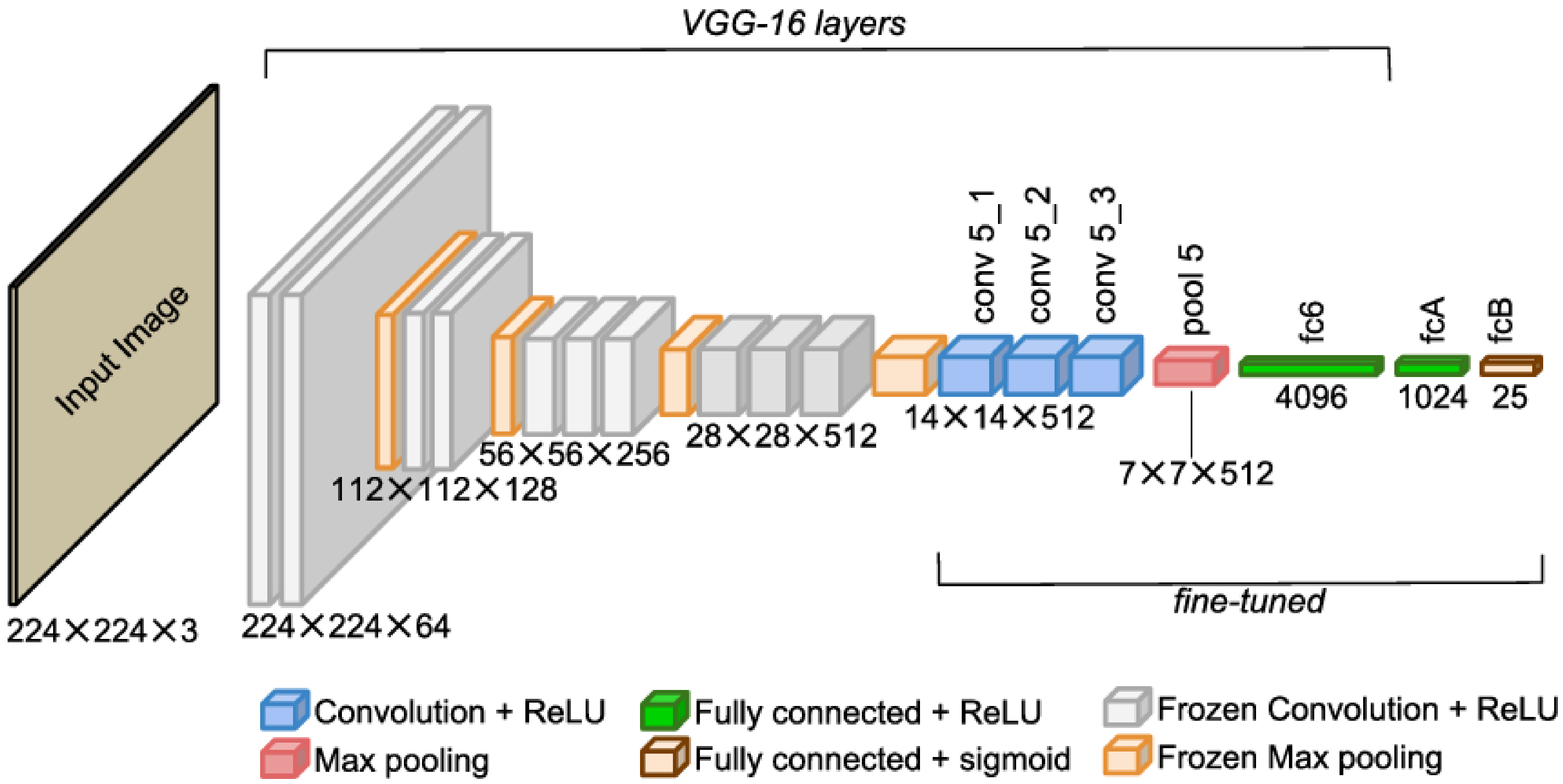
- Cross-correlation
  - $G = h \otimes F$
  - $G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] F[i+u, j+v]$
- Convolution
  - $G = h \otimes F$
  - $G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] F[i-u, j-v]$
- Practically, what we do in CNN are **cross-correlation** operations and not **convolutions** per se. But for historical reasons Convolutional Neural Network is the terminology we use.

CNN

Convolution



Convolution

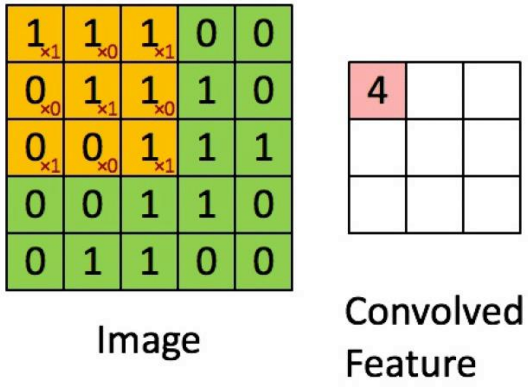


Banik, Soubarna, Mikko Lauri, and Simone Frintrop. "Multi-label Object Attribute Classification using a Convolutional Neural Network." *arXiv preprint arXiv:1811.04309* (2018).

AIMLab.

Padding

- When applying a convolution the image shrinks
  - 5 x 5 ➔ 3 x 3
  - Also we intrinsically use less the information at the edges than the information in the center of the image.
- To address these issues we use padding.



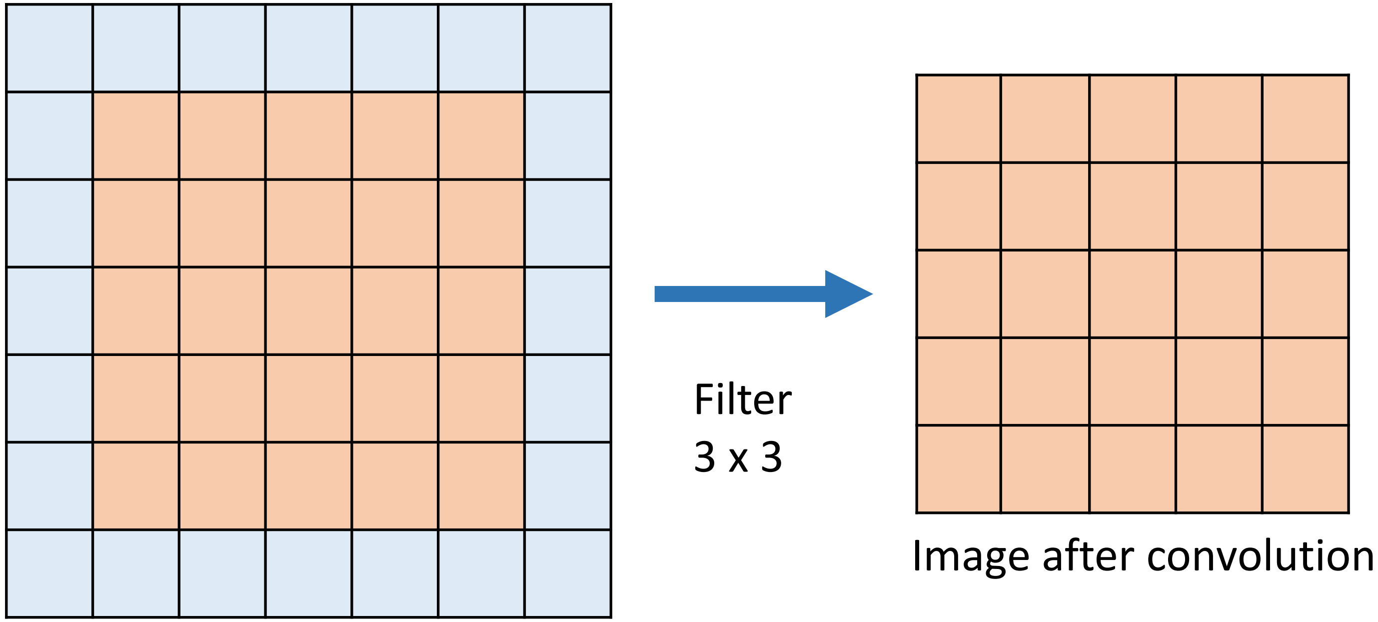


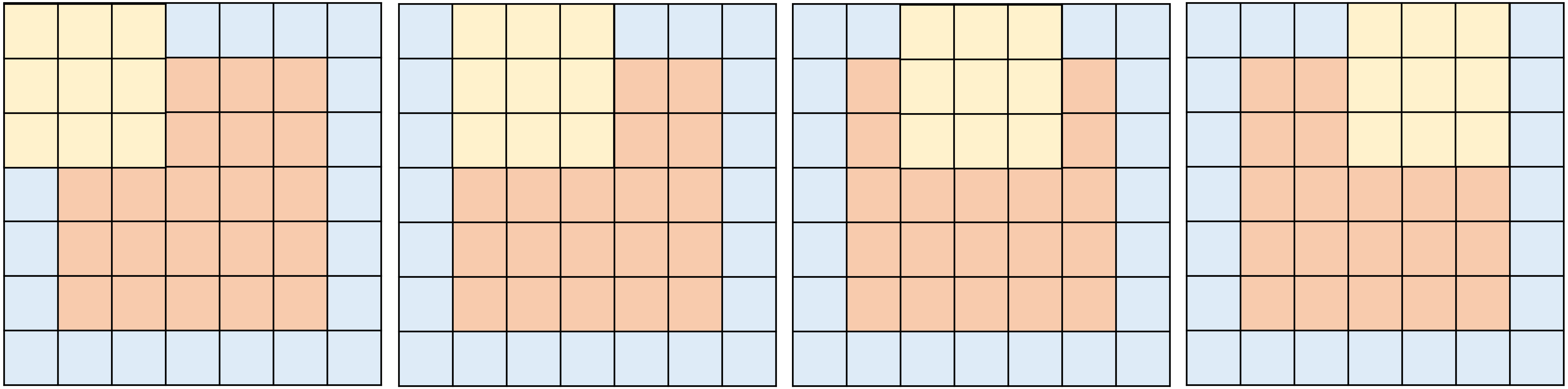
Image after convolution: 5 x 5

Image: 5 x 5
Image + padding: 7 x 7

Striding

- No striding

Input: 7 x 7
Output: 5 x 5

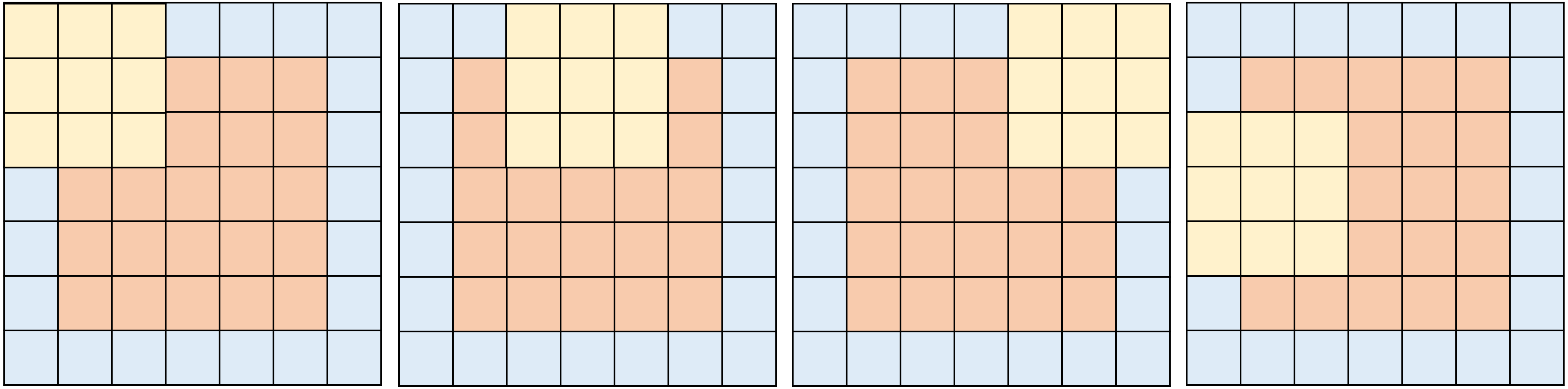


...

Striding

Input: 7 x 7
Output: 3 x 3

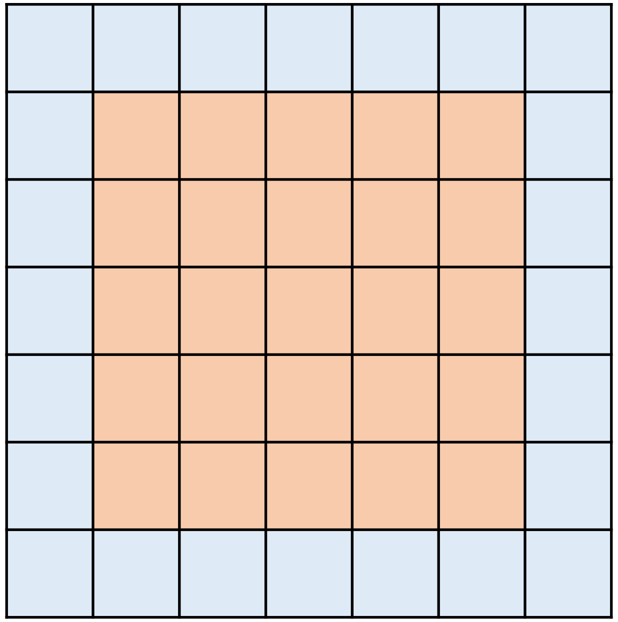
- Striding with stride (s) of 2



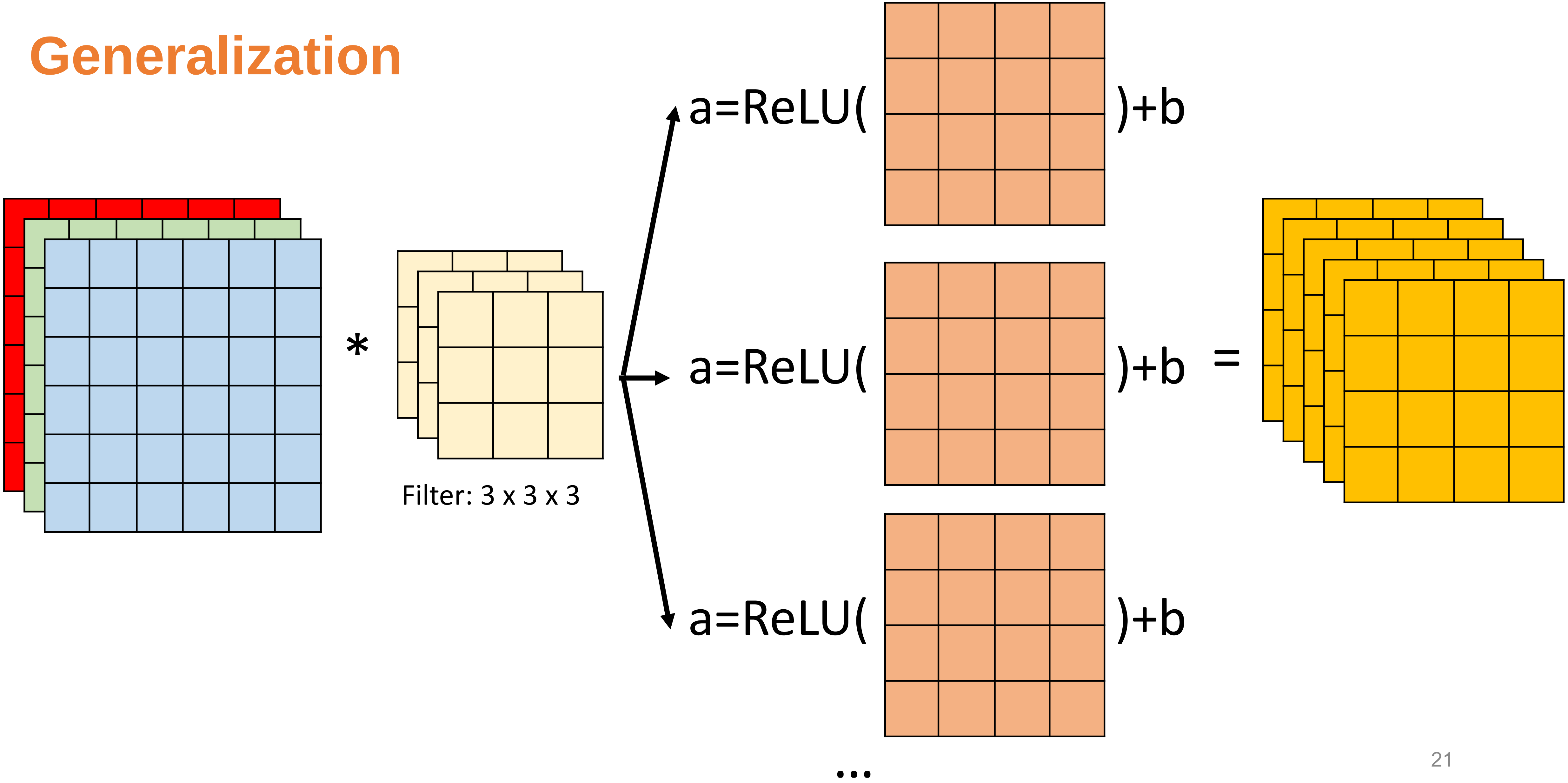
…

Notations

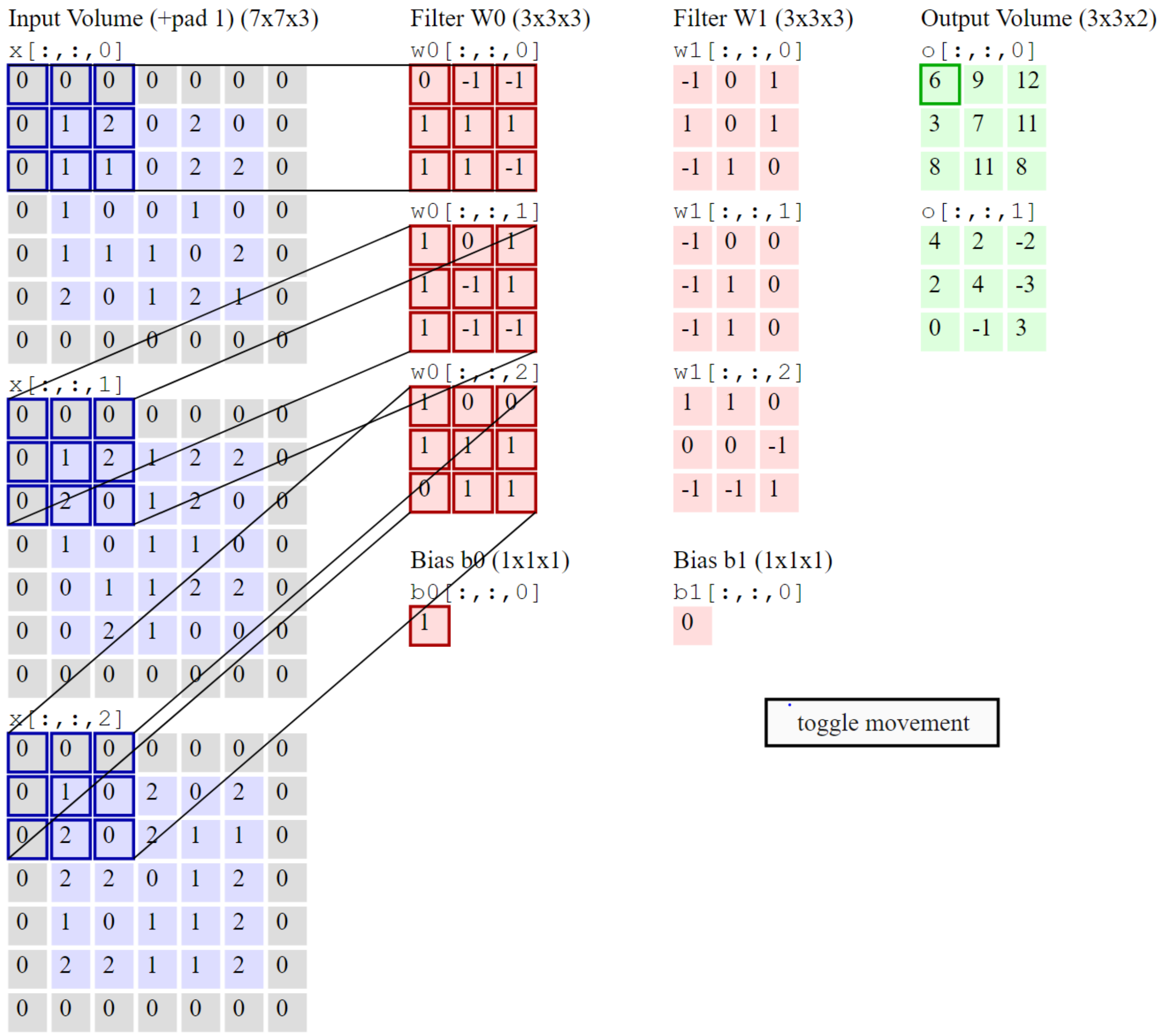
- We write
  - f: filter size.
  - p: padding.
  - s: stride.
  - n: size of the image in pixels.
- Sizing: $(n \cdot n) * (f \cdot f) \to \left(\frac{n+2p-f}{s} + 1\right) \times \left(\frac{n+2p-f}{s} + 1\right)$



Generalization



Generalization



http://cs231n.github.io/convolutional-networks/

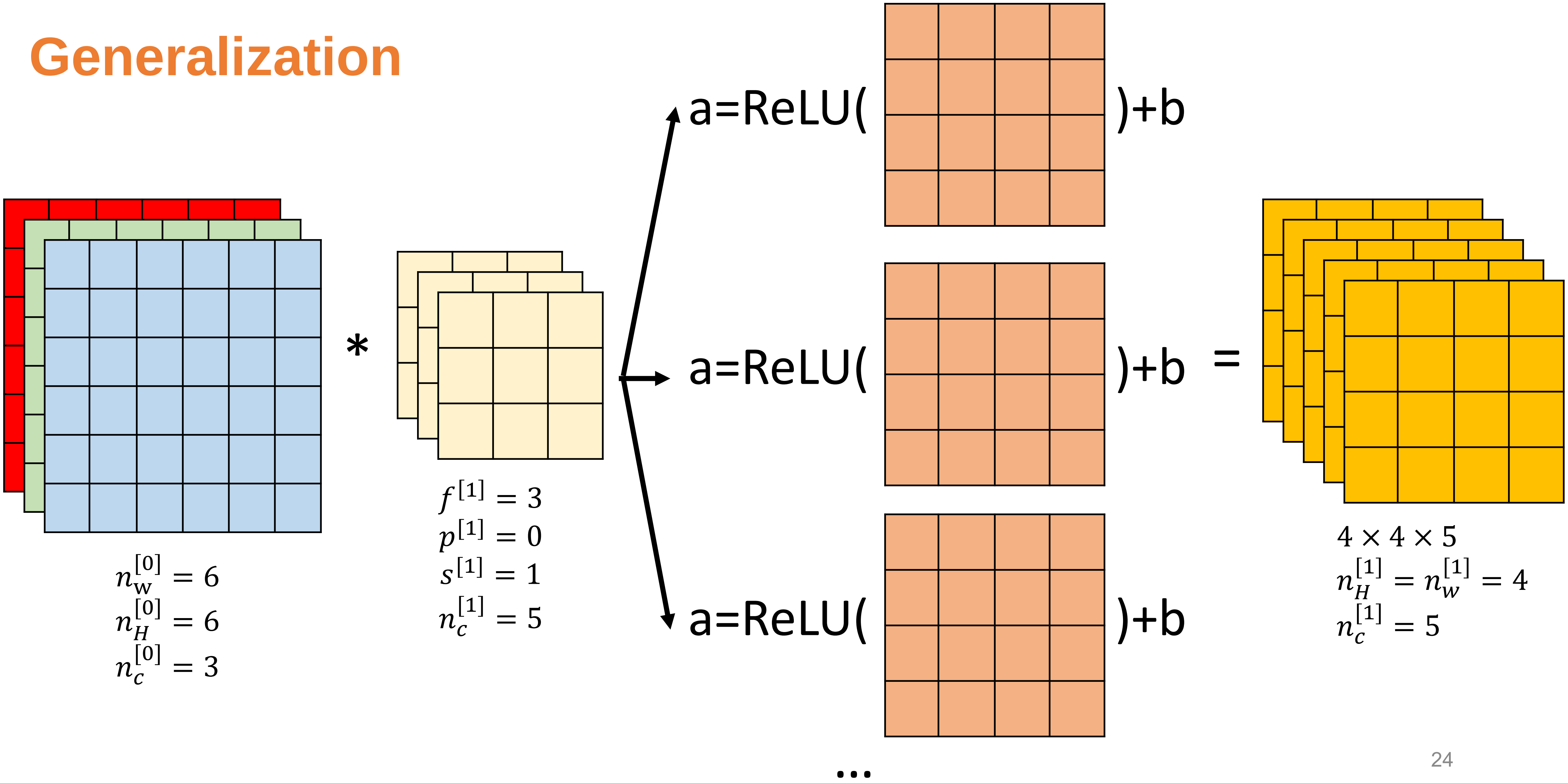
Notations - generalization

- For a layer l and for an image of width n_w and height n_H:

Symbol	
$f^{[l]}$	Filter size.
$p^{[l]}$	Padding.
$s^{[l]}$	Stride.
$n_c^{[l]}$	Number of filters.
$n_w^{[l]}$	Width at layer l.
$n_H^{[l]}$	Height at layer l.

- Sizing:
  - $n_w^{[l]} \cdot n_H^{[l]} = \left(\frac{n_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right) \times \left(\frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right)$

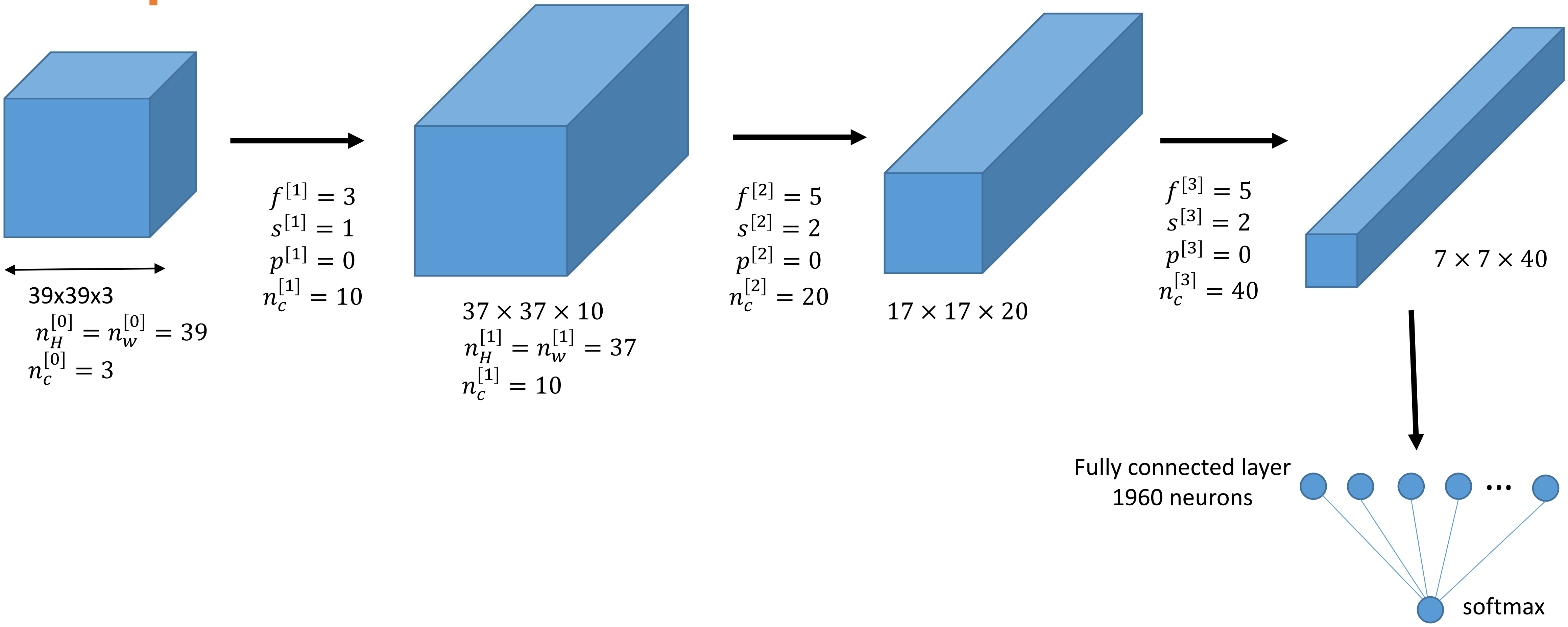
Generalization



Notations - generalization

- Input: $n_H^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$
- Output: $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$
- Number of weights to learn at layer l: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
- Activation at layer l: $n_H^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

Example


39x39x3
$n_H^{[0]} = n_w^{[0]} = 39$
$n_c^{[0]} = 3$
$f^{[1]} = 3$
$s^{[1]} = 1$
$p^{[1]} = 0$
$n_c^{[1]} = 10$
$37 \times 37 \times 10$
$n_H^{[1]} = n_w^{[1]} = 37$
$n_c^{[1]} = 10$
$f^{[2]} = 5$
$s^{[2]} = 2$
$p^{[2]} = 0$
$n_c^{[2]} = 20$
$17 \times 17 \times 20$
$f^{[3]} = 5$
$s^{[3]} = 2$
$p^{[3]} = 0$
$n_c^{[3]} = 40$
$7 \times 7 \times 40$
Fully connected layer
1960 neurons
...
softmax

Take home

- CNN as a way to take advantage of “convolutions” for elaborating features. Rather than hand-crafting the convolution filters we learn their coefficients.
- Practically we use cross-correlation and not convolutions but for historical questions we call CNN this convolutional neural network.
- Padding.
- Striding.
- Notations.

References

[1] Andrew Ng, Coursera, Neural Networks and Deep Learning. Coursera.
[2] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.