CONVOLUTIONAL NEURAL NETWORKS --- BASICS

Convolutional Neural Networks (CNNs, ConvNets)

- Has a root in Neocognitron [Fukushima80]
- LeNet: A success in handwritten character recognition [LeCun+89]
- Basis in findings in neuroscience
  - Simple cells, Complex cells [Hubel-Wiesel59]
  - Local receptive field

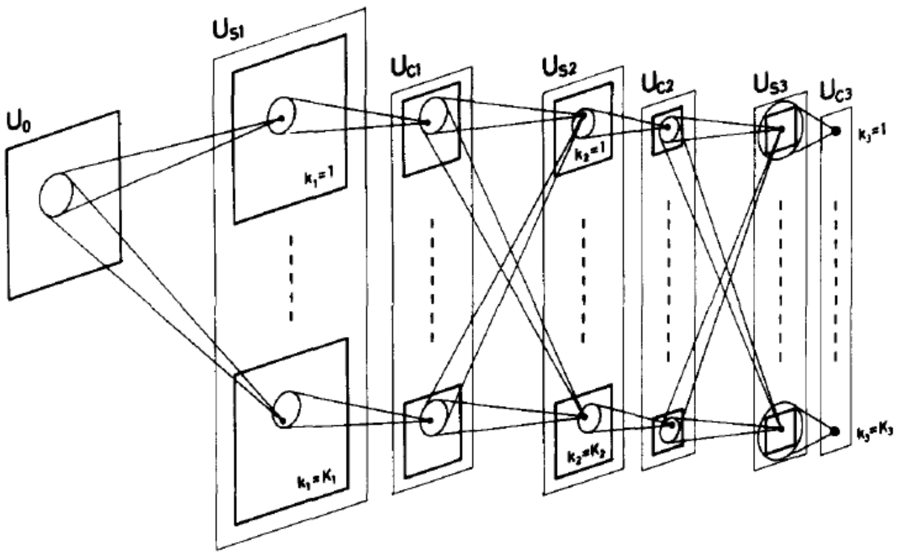


Fig 4 Schematic diagram illustrating the interconnections between layers in the neocognitron

[Fukushima+83]

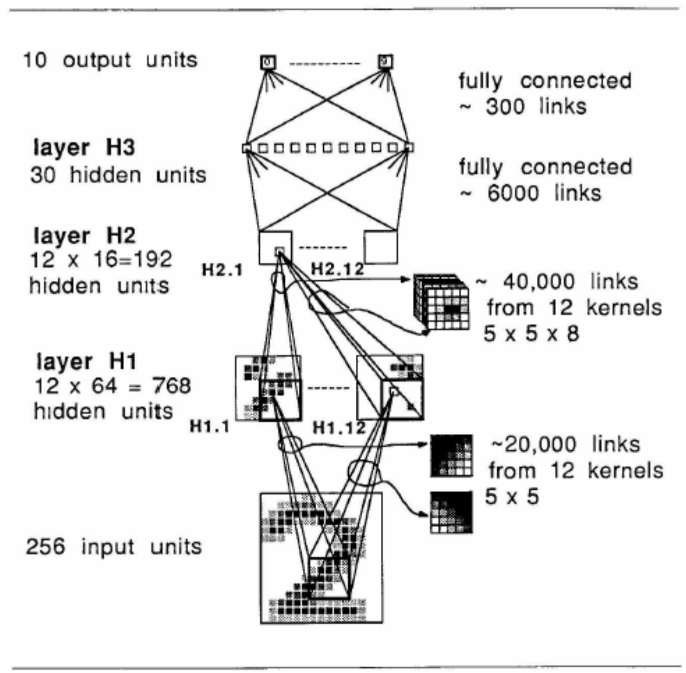
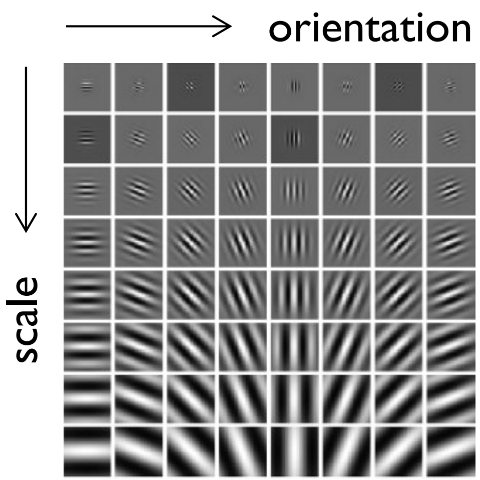


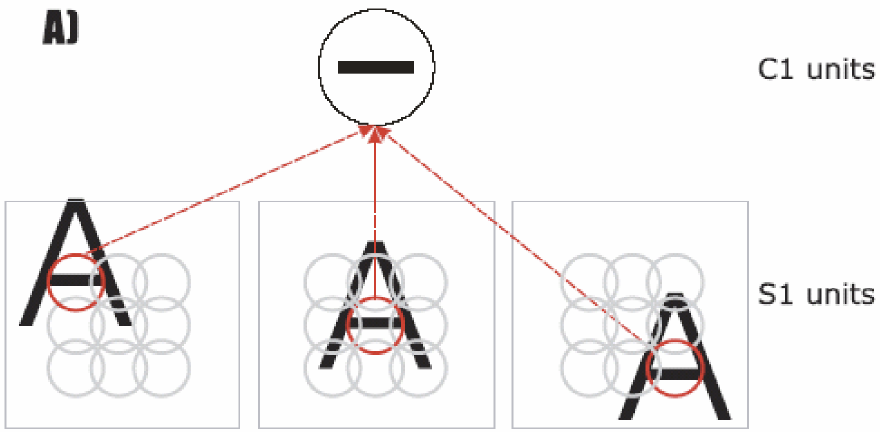
Figure 3 Log mean squared error (MSE) (top) and raw error rate (bottom) versus number of training passes

[LeCun+89]

Simple cells & complex cells [Huber-Wiesel59]

- Two types of cells exist in early visual cortex of animals
  - There exist a large number of cells, each of which is selective to particular position/orientation/scale
- Simple cells: neurons that have selectivity on the orientation and position of a feature (e.g., a bar)
  - In the 2D retinal space
  - Has selectivity = Selectively fires for the feature
- Complex cells: neurons that are less selective to position





Serre et al, Object Recognition with Features Inspired by Visual Cortex, CVPR05

Convolution

$$u_{ij} = \sum_{p=0}^{H-1} \sum_{q=0}^{H-1} x_{i+p,j+q} h_{pq}$$

Input

77	80	82	78	70	82	82	140
83	78	80	83	82	77	94	151
87	82	81	80	74	75	112	152
87	87	85	77	66	99	151	167
84	79	77	78	76	107	162	160
86	72	70	72	81	151	166	151
78	72	73	73	107	166	170	148
76	76	77	84	147	180	168	142

$x_{i+p,j+q}$

$*$

Filter (Kernel)

0.01	0.08	0.01
0.08	0.62	0.08
0.01	0.08	0.01

h_{pq}

$=$

Output

79	80	81	79	79	98
82	81	79	75	81	114
85	83	77	72	99	144
79	77	77	79	112	155
73	71	73	89	142	162
73	73	77	110	160	166

u_{ij}

The red 3x3 square in the input is called *receptive field* of the convolution outputting the red value

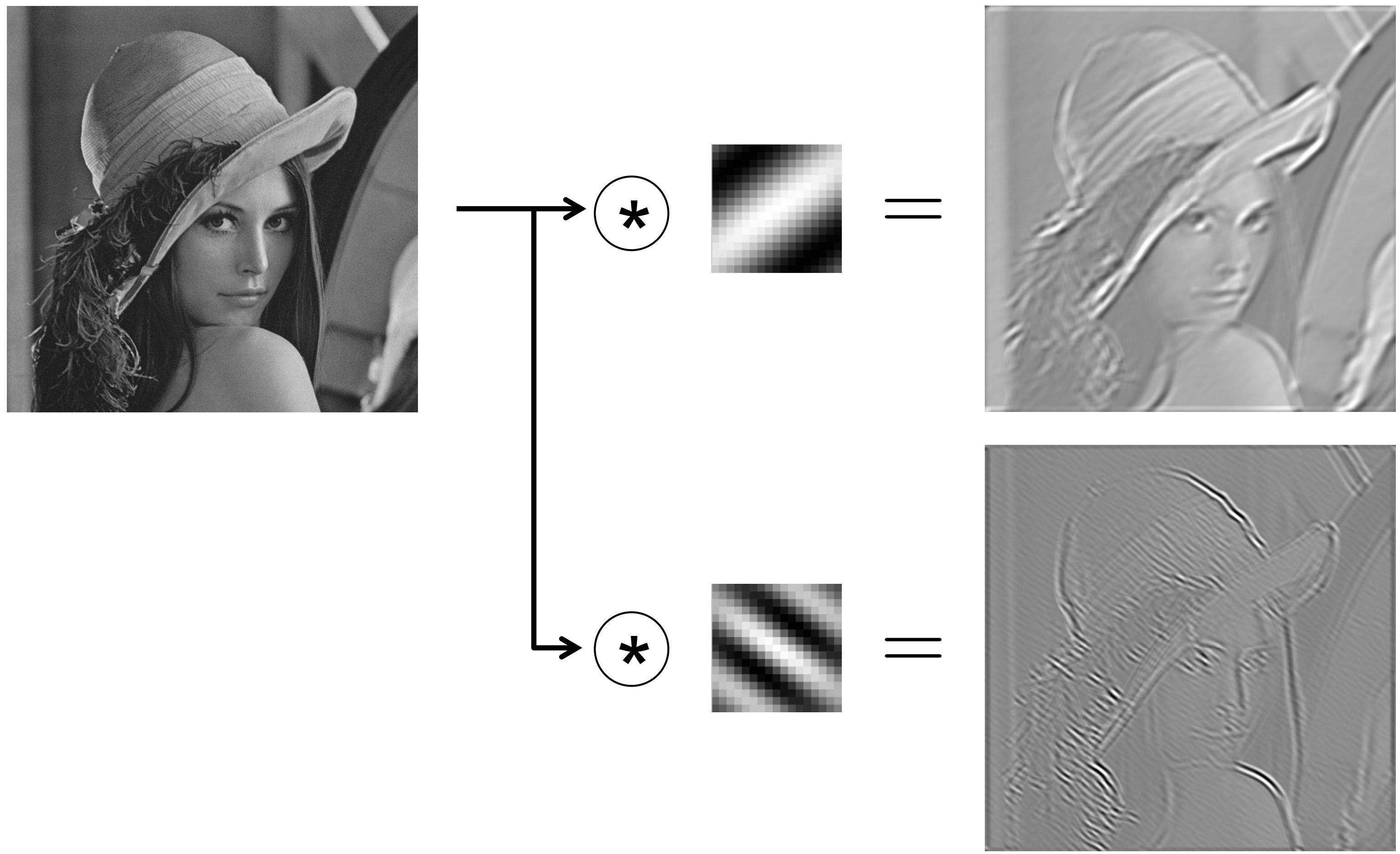
Convolution

$$u_{ij} = \sum_{p=0}^{H-1} \sum_{q=0}^{H-1} x_{i+p,j+q} h_{pq}$$

Input

Filter (Kernel)

Output



$x_{i+p,j+q}$ h_{pq} u_{ij}

Pooling

- Selects a single value representing a local area (usually a square)
  - Pooling has a local receptive field similar to convolution
  - Stride > 1 yields output with lower resolution (downsampling)

Input

62	71	72	69	65	71	79	107
73	79	80	81	79	79	98	128
76	82	81	79	75	81	114	132
77	85	83	77	72	99	144	145
74	79	77	77	79	112	155	142
74	73	71	73	89	142	162	137
69	73	73	77	110	160	166	134
60	67	68	78	124	154	148	116

$\rightarrow$

Output

82	82	114	132
85	85	155	155
85	110	166	166
79	124	166	166

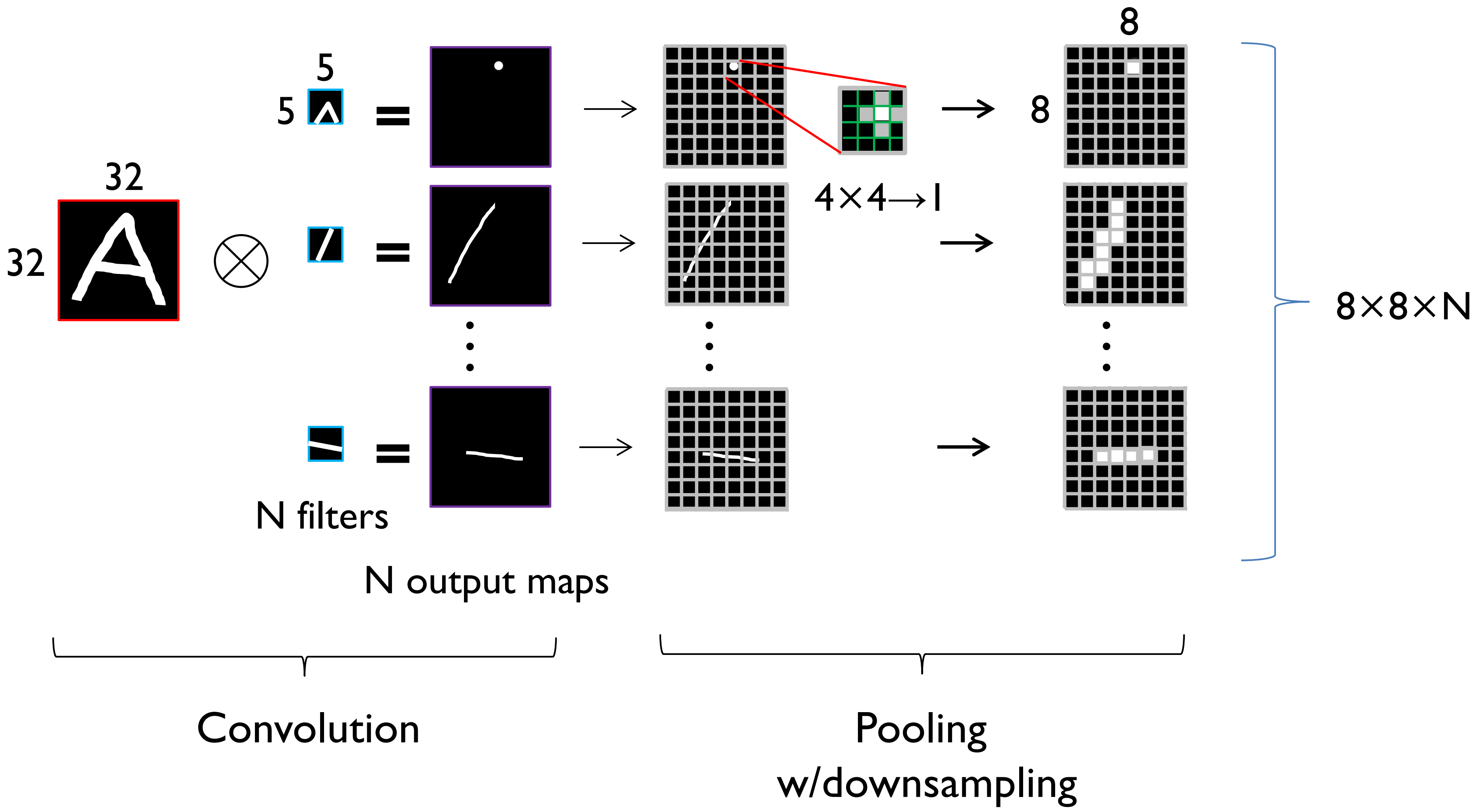
Max pooling

$$u_{ijk} = \max_{(p,q)\in P_{ij}} z_{pqk}$$

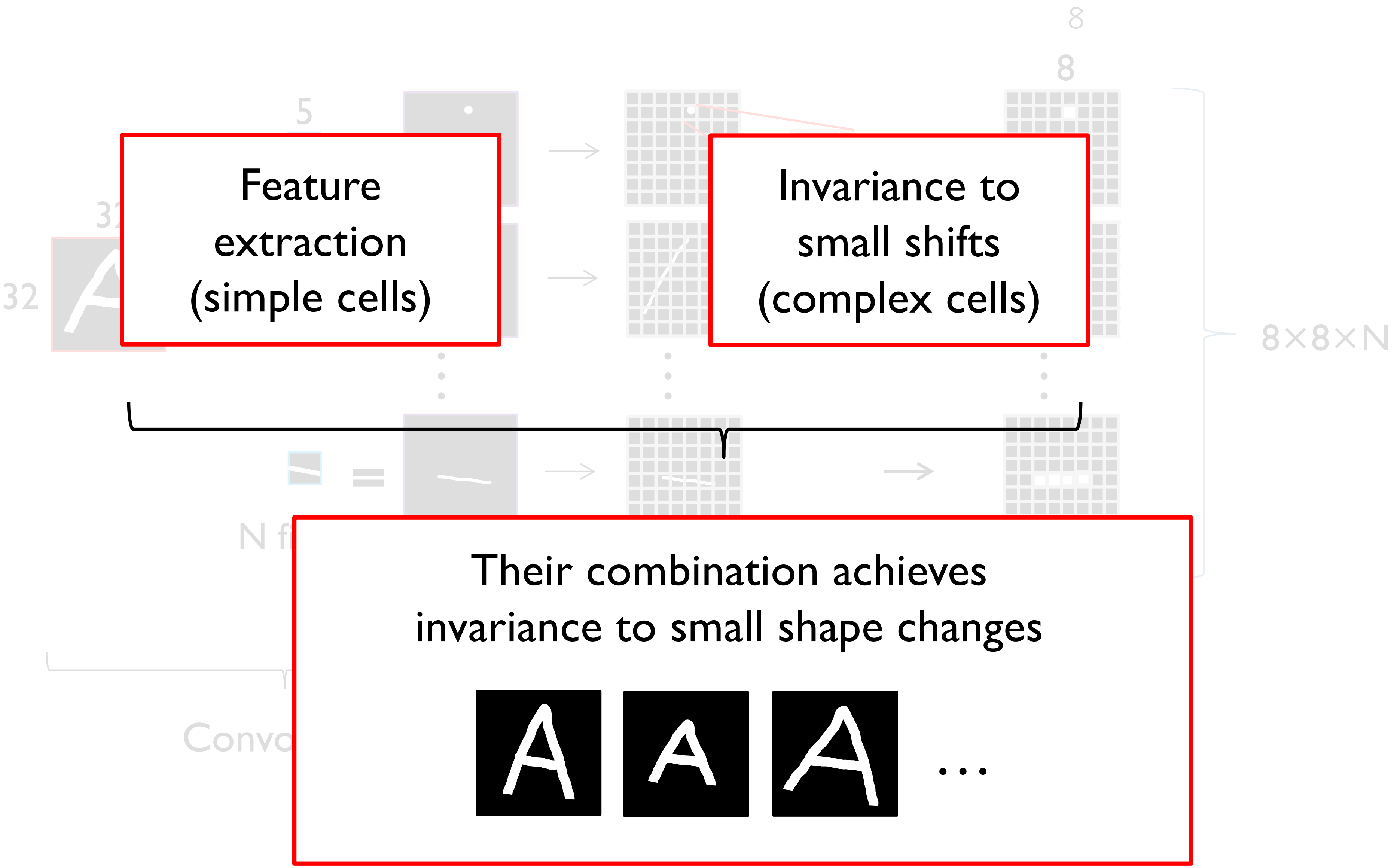
Average pooling

$$u_{ijk} = \frac{1}{H^2} \sum_{(p,q)\in P_{ij}} z_{pqk}$$

Convolution + pooling



Convolution + pooling



Convolution as a layer of NNs

- Convolution operation w/ a filter can be implemented by a network layer w/
  - *Sparse connection*: Each output unit has connections only to input units in its *receptive field*
  - *Shared weights*: Connection weights are element values of the filter → As the same filter is applied to the entire input, the layer weights are shared by the output units

$$u_{ij} = \sum_{p=0}^{H-1} \sum_{q=0}^{H-1} x_{i+p,j+q} h_{pq}$$

Input

77	80	82	78	70	82	82	140
83	78	80	83	82	77	94	151
87	82	81	80	74	75	112	152
87	87	85	77	66	99	151	167
84	79	77	78	76	107	162	160
86	72	70	72	81	151	166	151
78	72	73	73	107	166	170	148
76	76	77	84	147	180	168	142

$x_{i+p,j+q}$

$\otimes$

Filter (Kernel)

0.01	0.08	0.01
0.08	0.62	0.08
0.01	0.08	0.01

h_{pq}

$=$

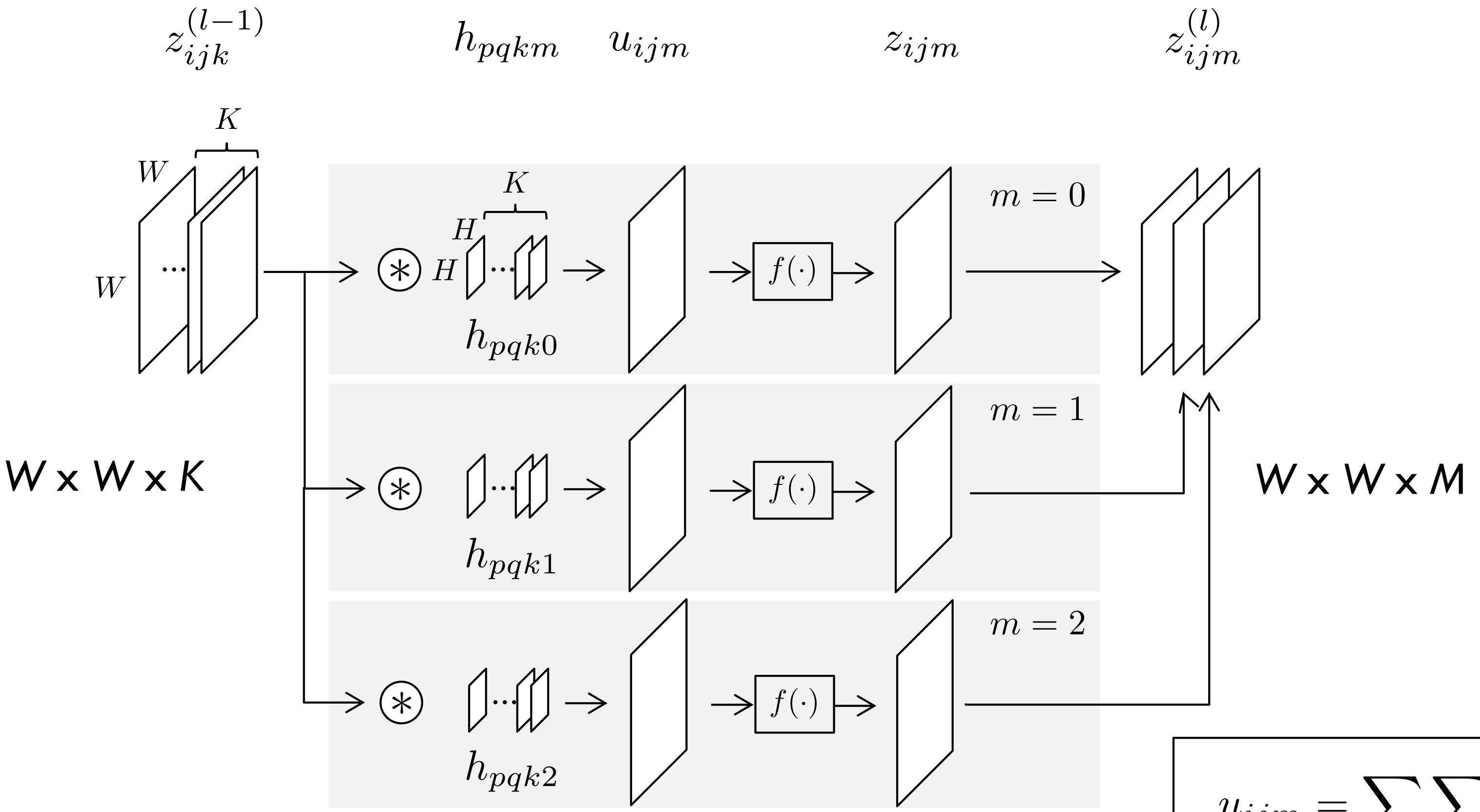
Output

79	80	81	79	79	98
82	81	79	75	81	114
85	83	77	72	99	144
79	77	77	79	112	155
73	71	73	89	142	162
73	73	77	110	160	166

u_{ij}

Conv. layer in a general form

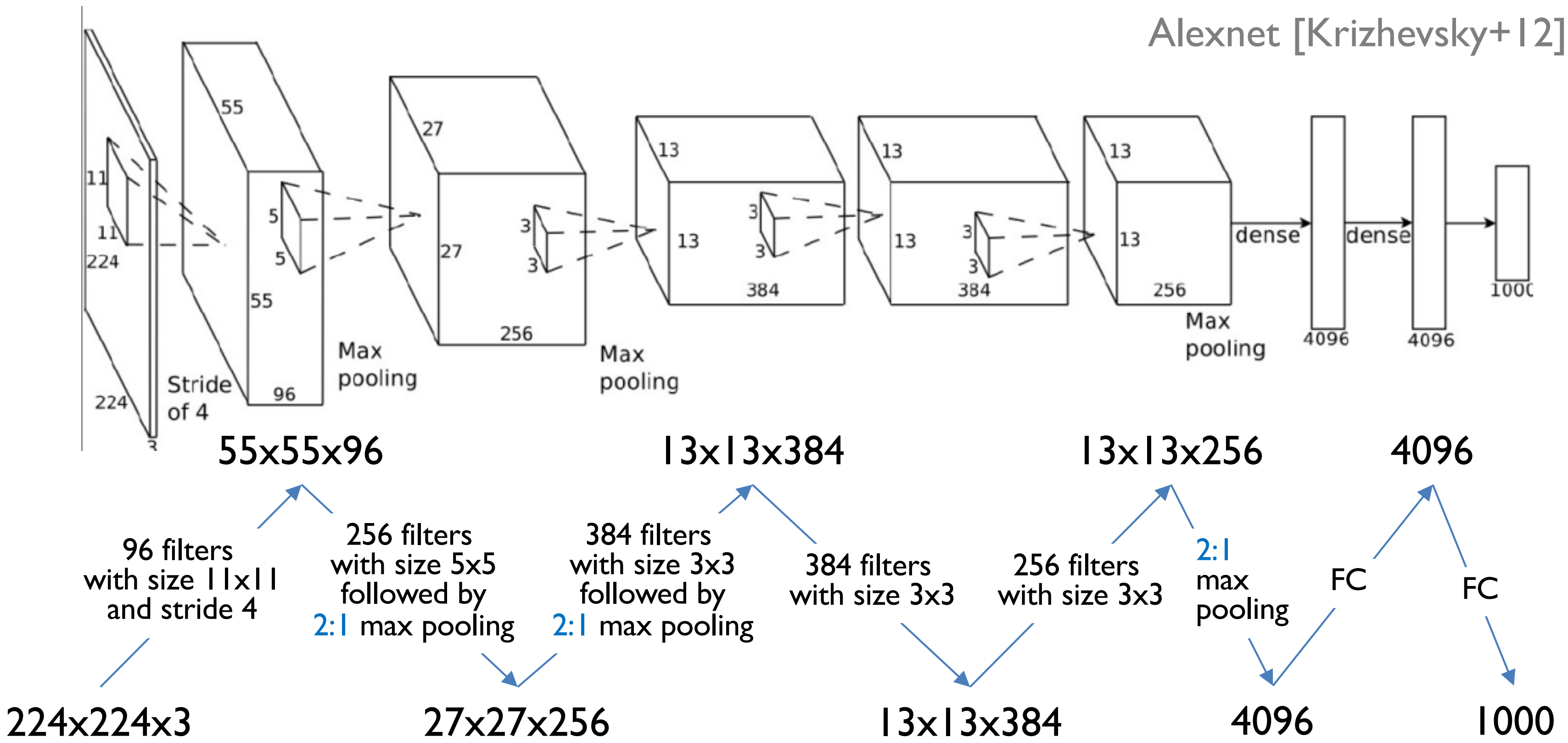
- In a general conv-layer, multiple filters are applied to multi-channel inputs, yielding multi-channel outputs
  - Each filter has the same number of channels as the input: *K*
  - The number of filters specifies the number of output channels: *M*



$$u_{ijm} = \sum_k \sum_p \sum_q z_{i+p,j+q,k} h_{pqkm}$$

Convolutional neural networks

- A CNN is a feedforward network consisting of several alternating convolution layers and pooling layers (or mere downsampling), on top of which additional layers computing an output
  - Each box below indicate the output of a conv. layer



Visual recognition of object category

→ CNN → ‘lion’

⇒ ‘lion’

⇒ ‘table lamp’

⇒ ‘acoustic guitar’

⇒ ‘Blenheim spaniel’

⇒ ‘electric guitar’

⇒ ‘Japanese spaniel’

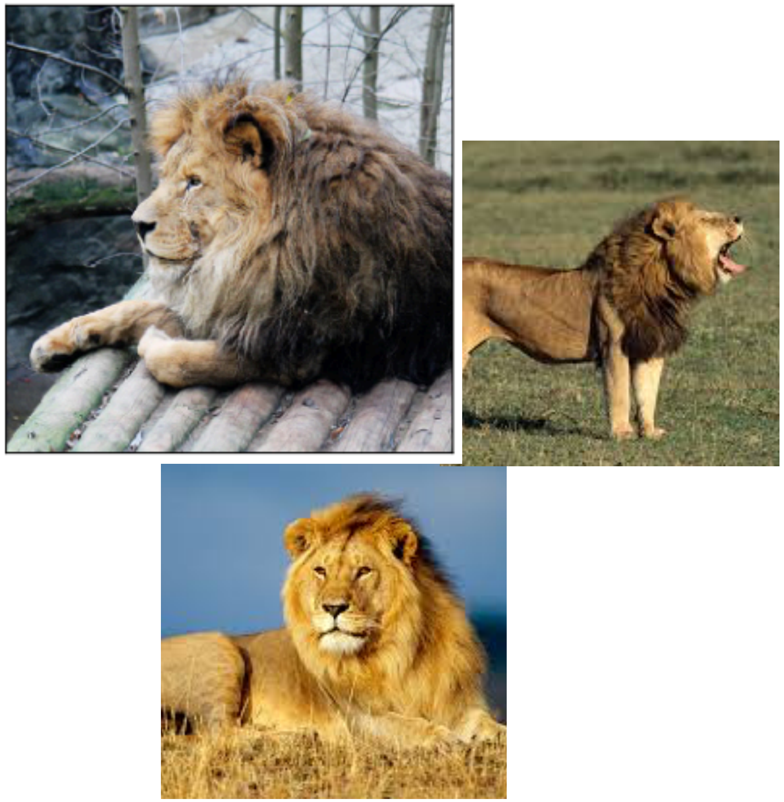
⇒ ‘chambered nautilus’

⇒ ‘crane’

Difficulty with visual object recognition

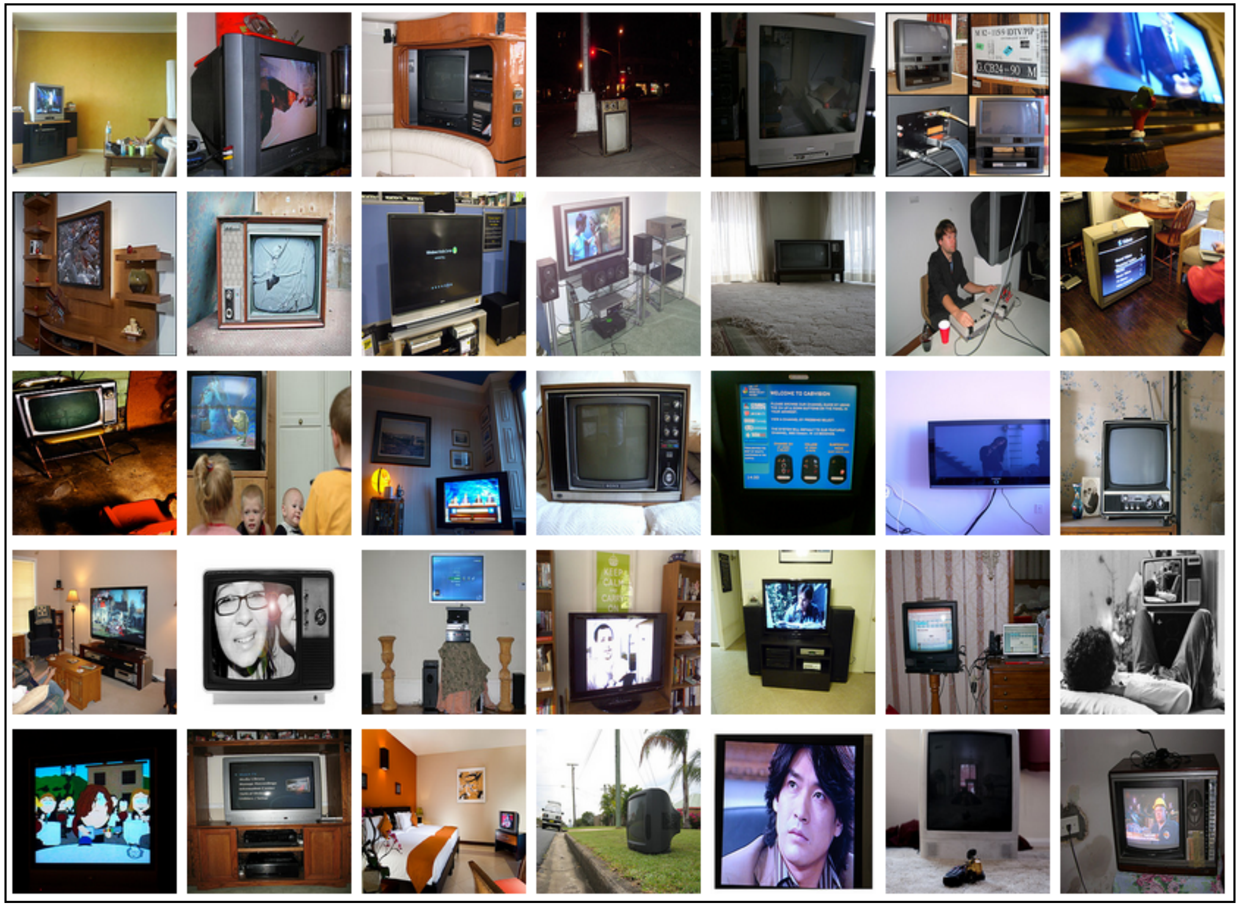
- Previously, researchers tried to solve it in two steps:

Input → Feature extraction → Feature → Classify → Lion / Cat / Dog / ⋮



Difficulty with visual object recognition

- Feature needs to have invariance, which tolerates various types of *variations within the category*

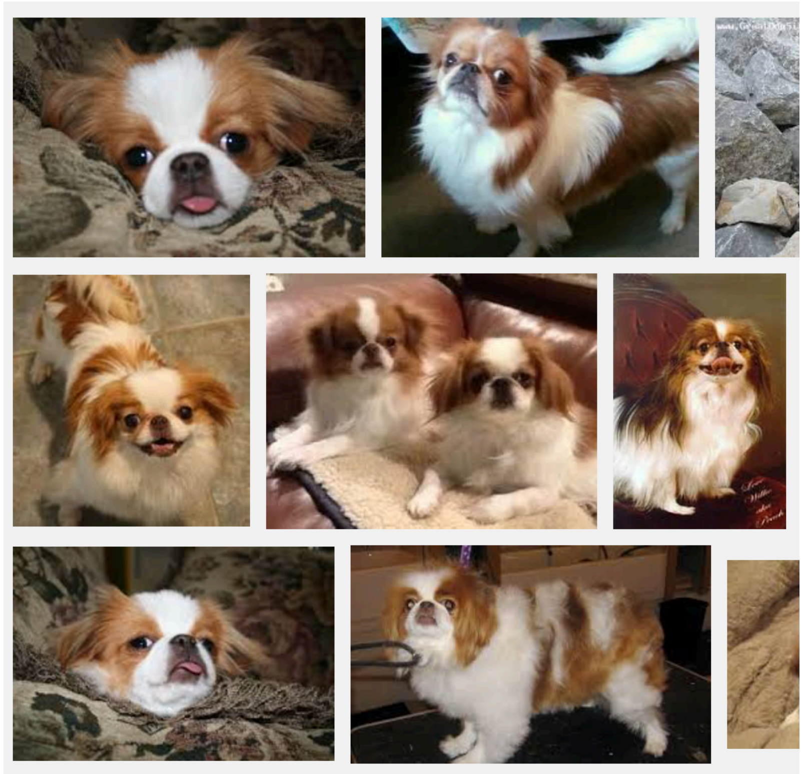


“Television set”

Difficulty with visual object recognition

- Feature needs to have sensitivity, which can distinguish subtle difference between different classes

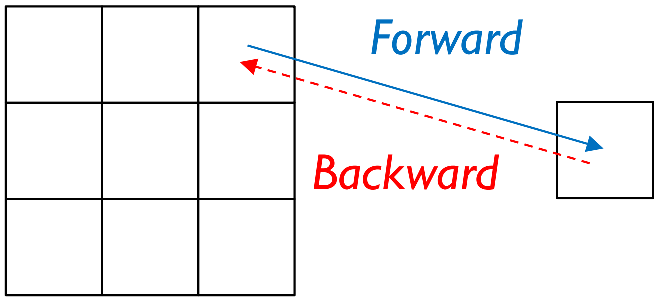
'Blenheim spaniel'



'Japanese spaniel'

Training of CNNs

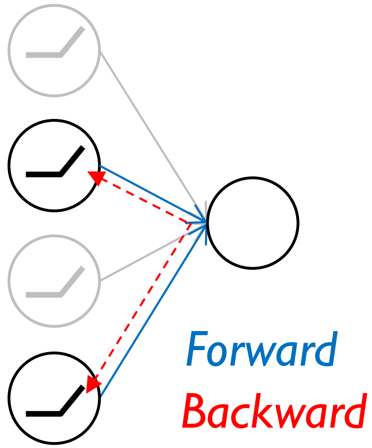
- As they are just feedforward nets, they can be trained similarly to standard FF nets.
  - Weights are randomly initialized based on *fan-ins*
- Note: Backpropagation of deltas
  - In a max pooling layer, they are backpropagated to the unit which was selected in the pooling operation in the forward computation; other units are ignored
  - This is similar to backprop at ReLU; units that outputted zero are ignored



Input units

Output unit

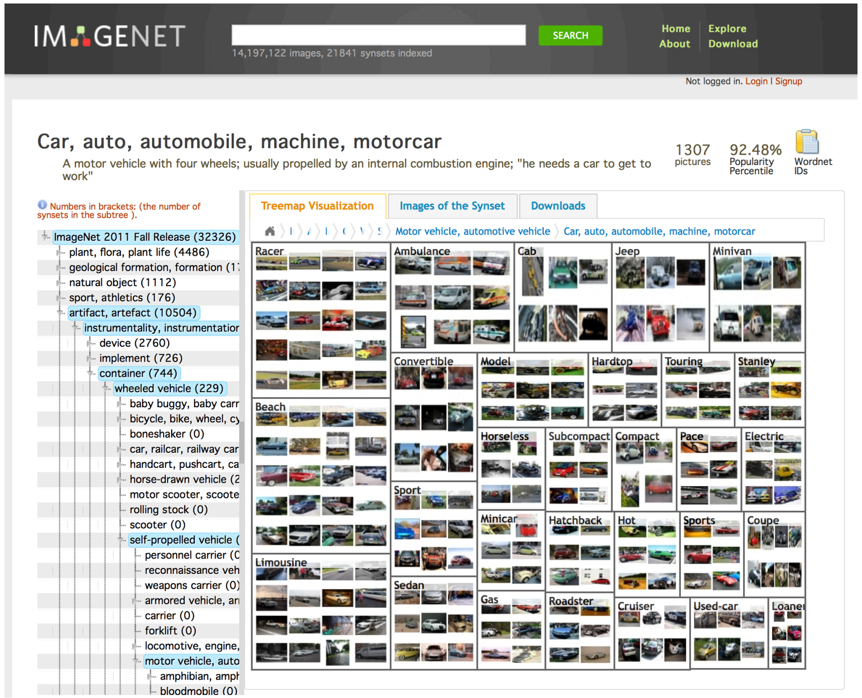
Max pooling layer



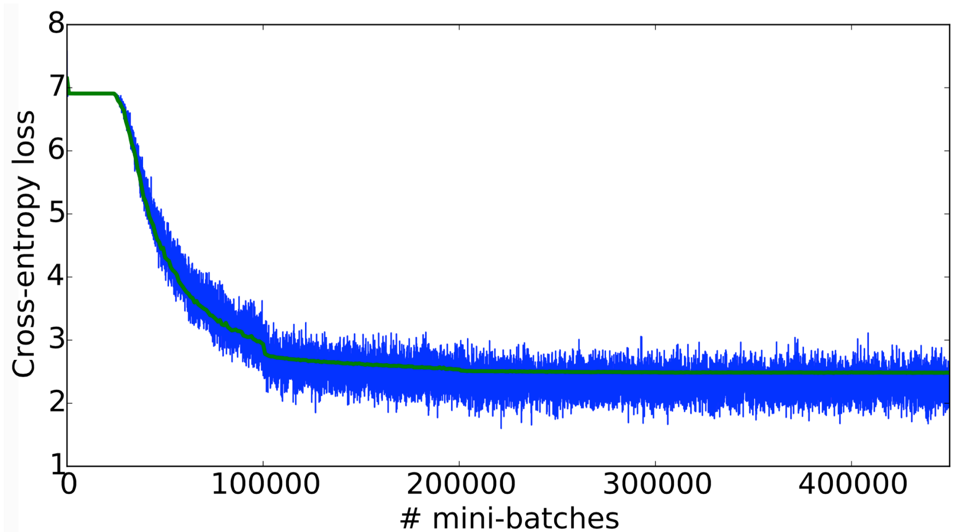
ReLUs

Object recognition --- ImageNet

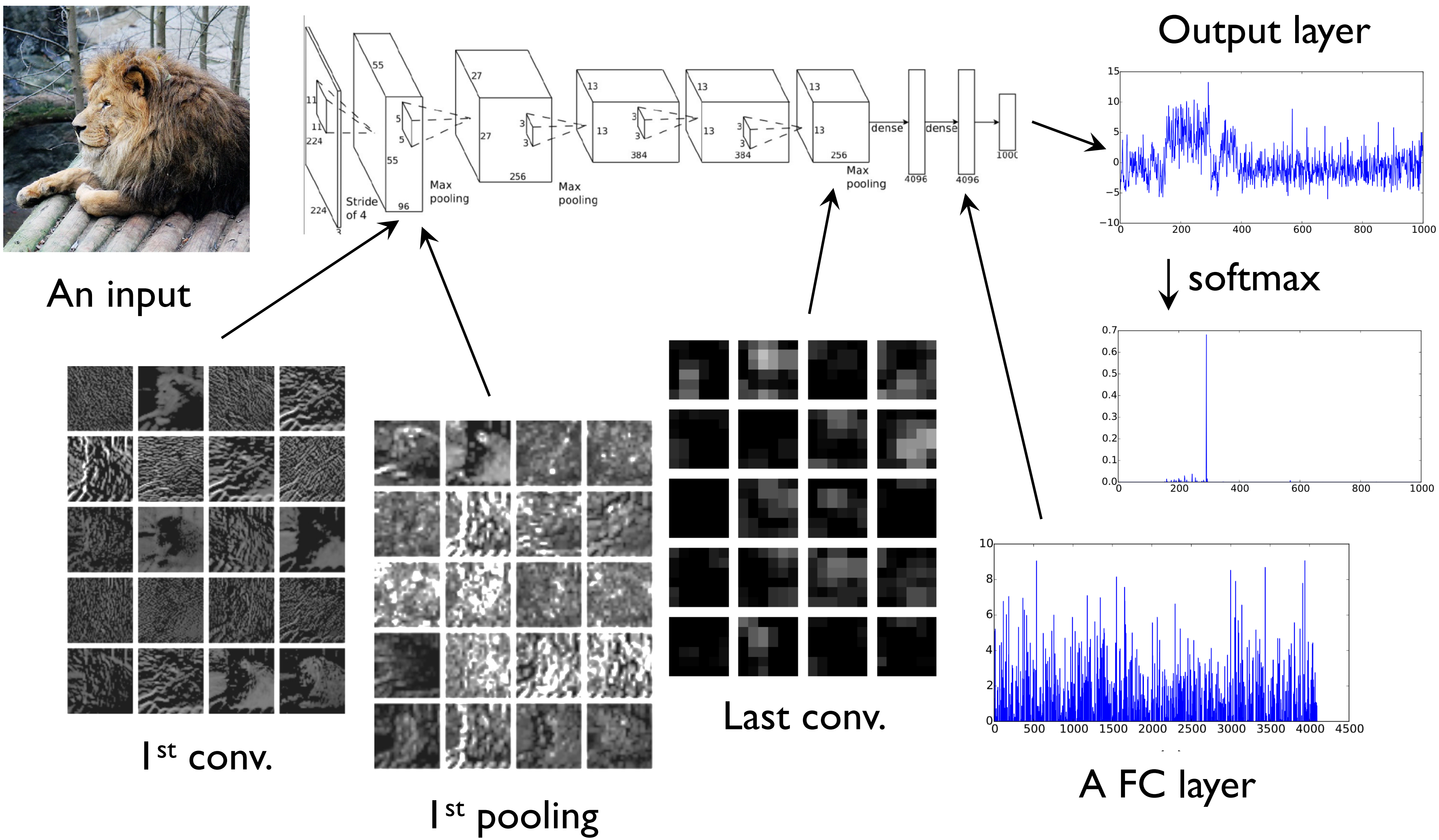
- The **ImageNet** project: database designed for research [Colab notebook]
  - More than 14 million images have been hand-annotated
    - Third-party image URLs; the actual images are not owned by ImageNet
  - Contains more than 20,000 categories
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) from 2011
  - 1,000 object classes ~ one million images
- It was reported CNNs surpass human performance (He+, Delving deep into rectifier, 2015)



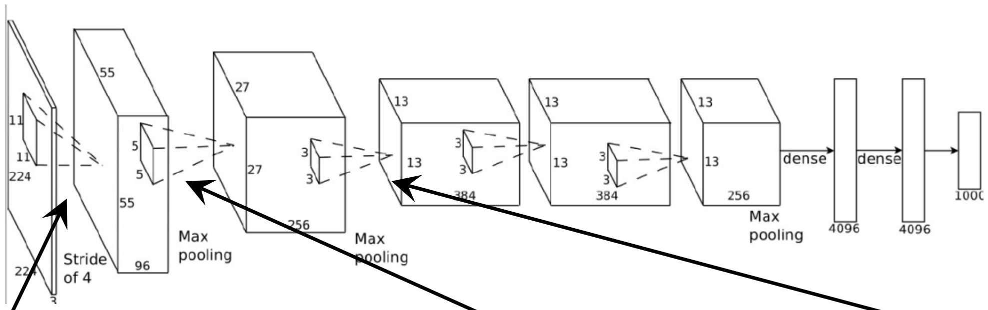
- Training w/ a GPU usually takes days
- Distributed system w/ many GPUs enables training less than an hour

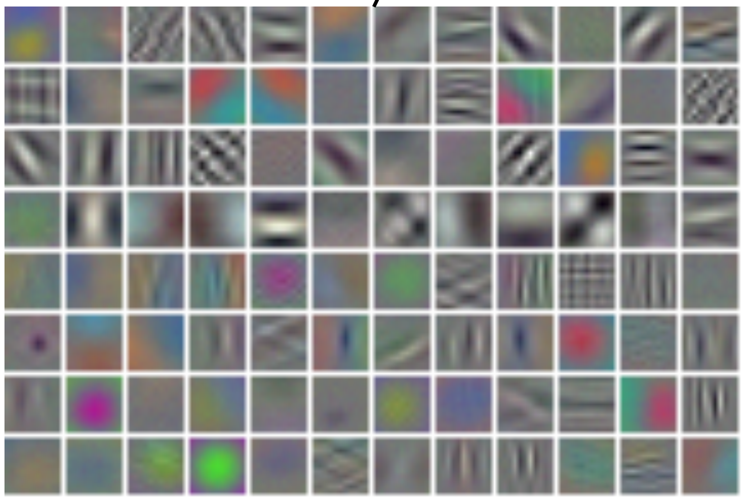


Layer activations for an input

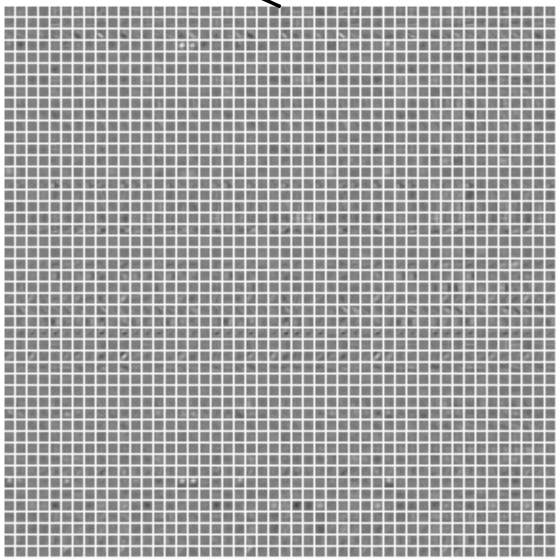


Learned filters

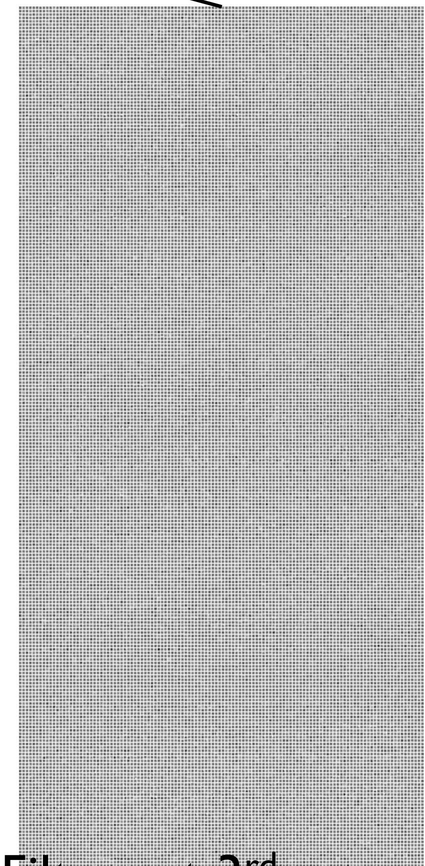




Learned filters at 1st conv.



Filters at 2nd conv.

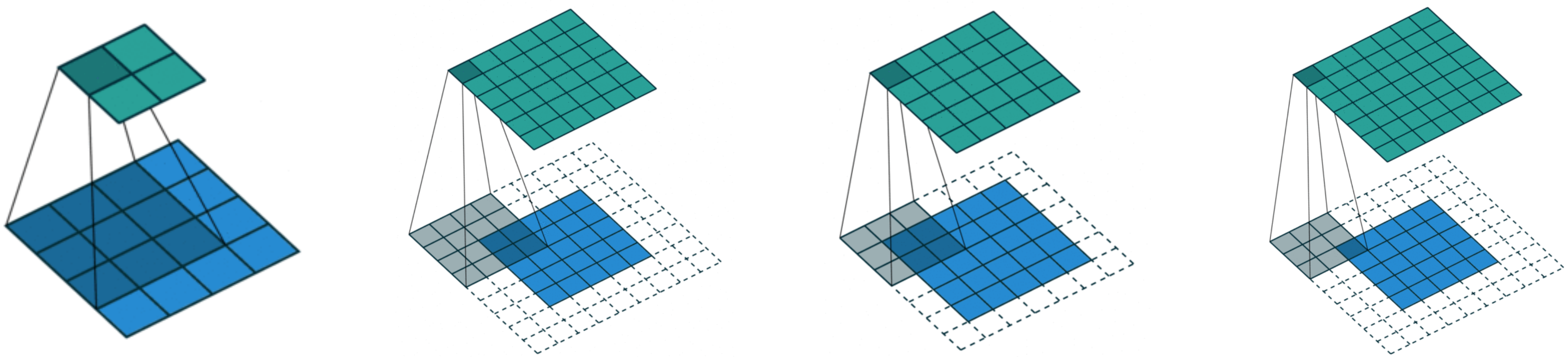


Filters at 3rd conv.

Note: Hard to interpret except for 1st conv filters

Padding

- Zero-padding: We often pad zeros around the input so that the output will have the same size as the input
  - Otherwise, the output will be smaller by the filter size than the input



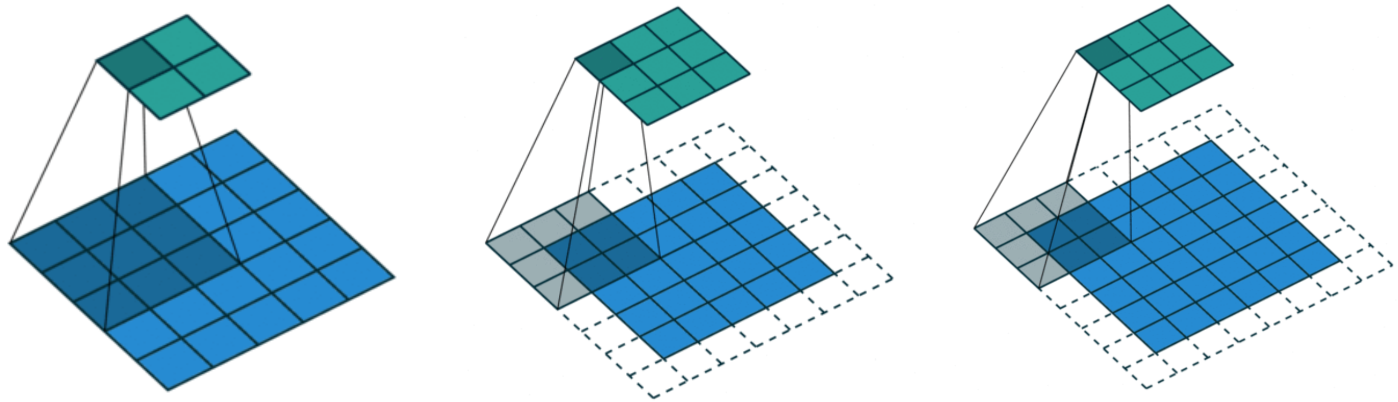
(filter_size, stride, padding) = (3, 1, 0)	(4, 1, 2)	(3, 1, 1)	(3, 1, 2)
(input, output) = (4, 2)	(5, 6)	(5, 5)	(5, 7)

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning (BibTeX)

Stride

[Colab notebook for LeNet]

- We can apply filters *sparsely* (i.e., at every few pixels)



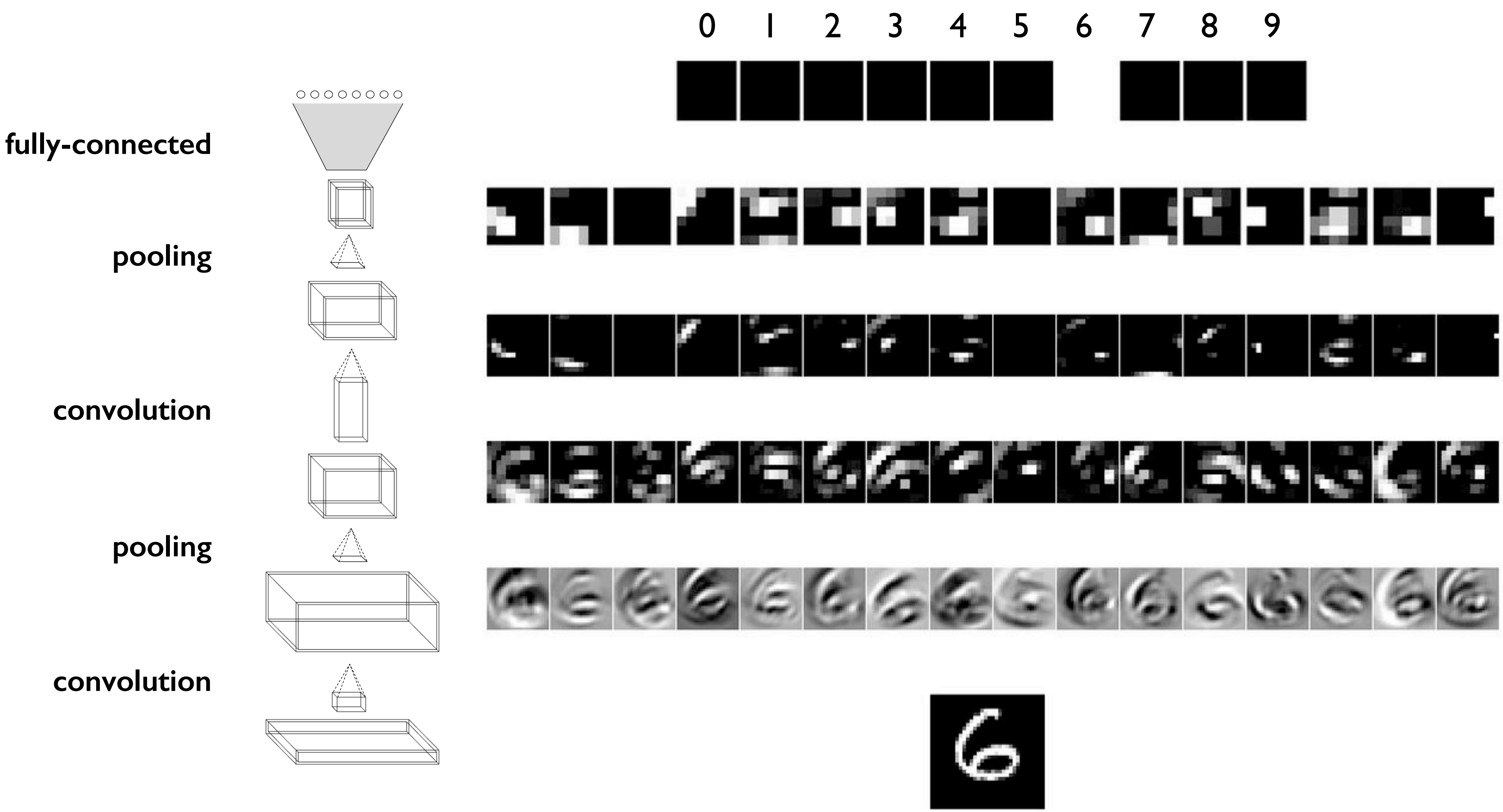
(filter_size, stride, padding) = (3, 2, 0)	(3, 2, 1)	(3, 2, 1)
(input, output) = (5, 2)	(5, 3)	(6, 3)

- Calculation of output size:

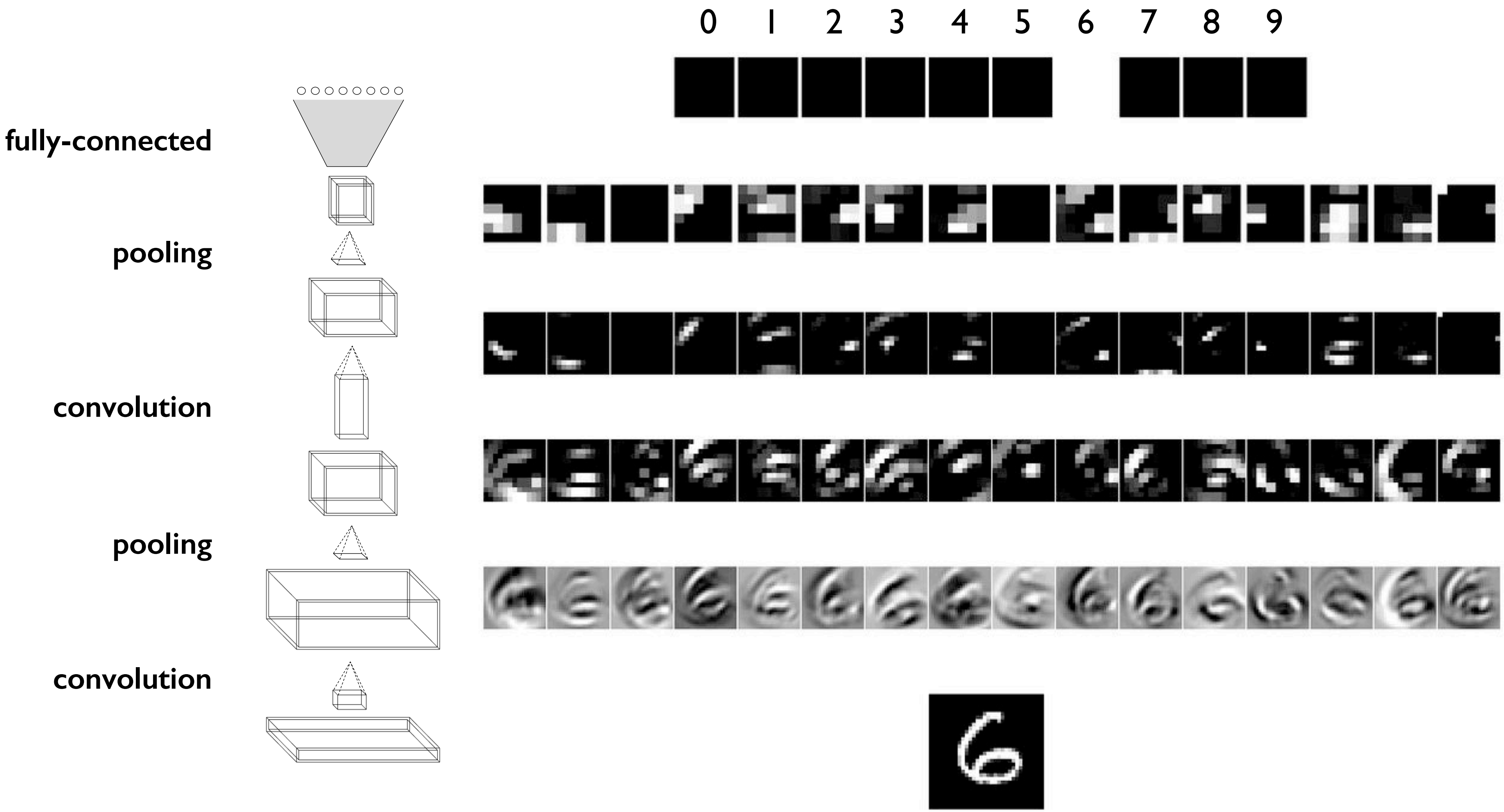
$$(\text{output size}) = \left\lfloor \frac{(\text{input size}) + (\text{padding}) \times 2 - (\text{filter size})}{(\text{stride})} \right\rfloor + 1$$

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning (BibTeX)

Behavior of a CNN at inference time



Behavior of a CNN at inference time



Assignments 1

[Colab notebook for loading fewer data]

- Mission: Analyze how the structure of a network affects its prediction accuracy and how it depends on the size of training data
- Minimum requirements:
  - Create at least 10 networks (models) that have different structures, e.g., number of layers, layer type (conv/fc), number of units, channels, filter size, etc.
  - Train each model on 1,000 and 50,000 samples until convergence, respectively
  - Test each model on 10,000 test samples to get mean prediction accuracy and create a table like the one below
  - Observe your results and explain what you have found
  - Don't forget to report the details of each model, e.g. the output of `print(net),` and training method, e.g., `optim.SGD(net.parameters(), lr=0.001, momentum=0.9)`

Model	**1000 samples**	**5000 samples**
1) 2FC_512	70.00%	92.00%
2) 3FC_128_128		
3) LeNet		
…		
10) *****		