USI/SUPSI Istituto Dalle Molle di studi sull'intelligenza artificiale IDSIA

Convolutional Neural Networks

Machine Learning 2019

Cesare Alippi, Jürgen Schmidhuber, Michael Wand, Paulo Rauber

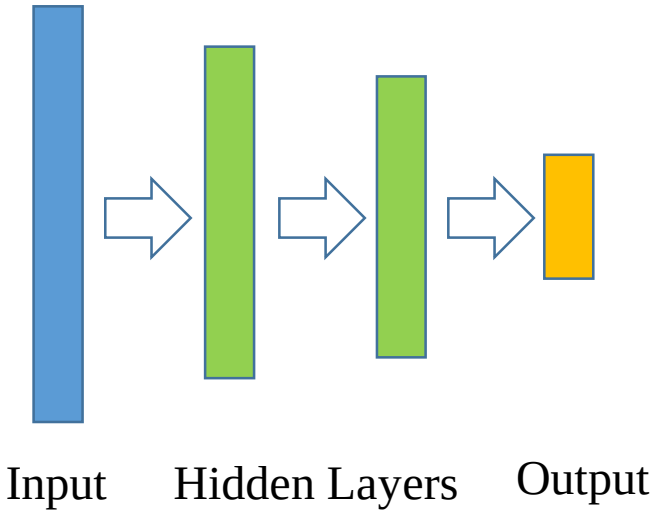
TAs: Róbert Csordás, Krsto Proroković, Xingdong Zuo, Francesco Faccio, Louis Kirsch

Recap: Fully connected neural networks

- So far we have covered *feedforward* neural networks
  - (i.e., data flows from input to output with no way back, no *recurrence*)
- … where *neurons* are organized in *layers* which are *fully connected*
  - (i.e., all neurons of layer *k* are connected to all neurons of layer *k+1)*
  - Training objective is to learn the weight of the connections

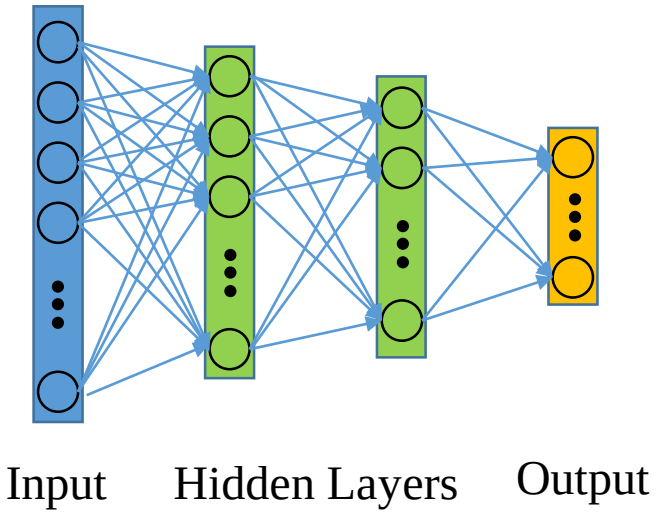
Recap: Fully connected neural networks

- So far we have covered *feedforward* neural networks
  - (i.e. data flows from input to output with no way back, no *recurrence*)
- … where *neurons* are organized in *layers* which are *fully connected*
  - (i.e. all neurons of layer *k* are connected to all neurons of layer *k+1)*
  - Training objective is to learn the weight of the connections



Recap: Fully connected neural networks

- So far we have covered *feedforward* neural networks
  - (i.e. data flows from input to output with no way back, no *recurrence*)
- … where *neurons* are organized in *layers* which are *fully connected*
  - (i.e. all neurons of layer *k* are connected to all neurons of layer *k+1)*
  - Training objective is to learn the weight of the connections



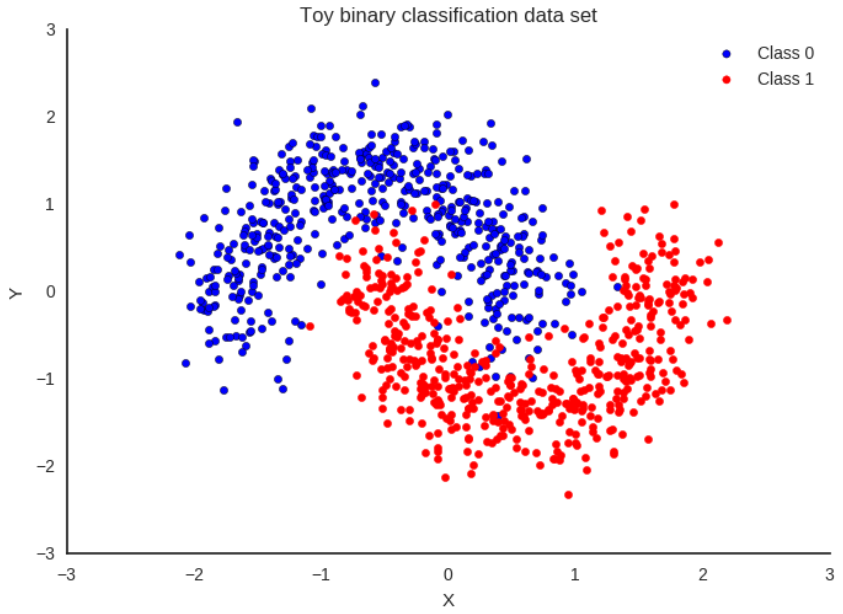
Recap: Fully connected neural networks

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

- We have covered how such a network is trained by *backpropagation*
  - … using a training data set
- Testing should be performed on a different data set
  - You want the network to *generalize:* it should be able to deal with unseen data!
- Also remember the alternative name *Multi-Layer Perceptron (MLP).*

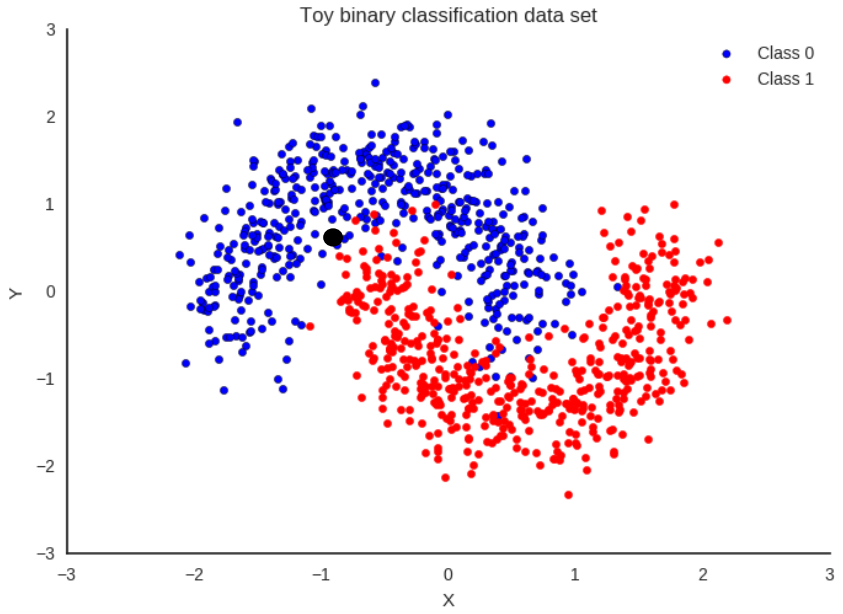
Recap: Fully connected neural networks

- An example: Typical toy data set
  - Absolutely no way to model classes 0 and 1 with a linear classifier
  - Neural networks perform well on such classification tasks



Recap: Fully connected neural networks

- An example: Typical toy data set
  - Absolutely no way to model classes 0 and 1 with a linear classifier
  - Neural networks perform well on such classification tasks
- Assume you need to classify a point as belonging to class 0 or 1.
  How many input neurons do we have?
  And how many outputs?



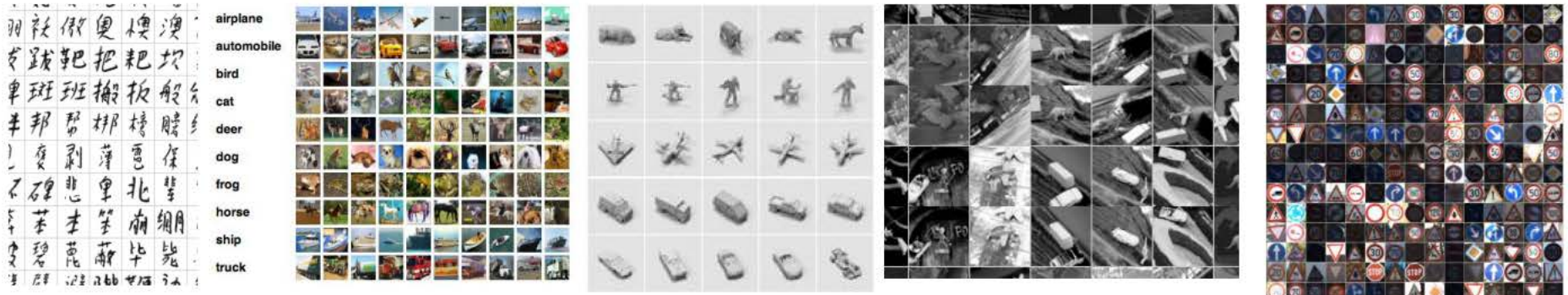
A classical example: MNIST

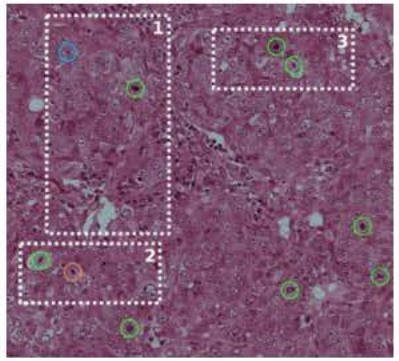
- Task: recognize handwritten digits
- Standard training and test sets available, good benchmark task!
- Record-breaking GPU implementation of MLP classifier on MNIST [1]

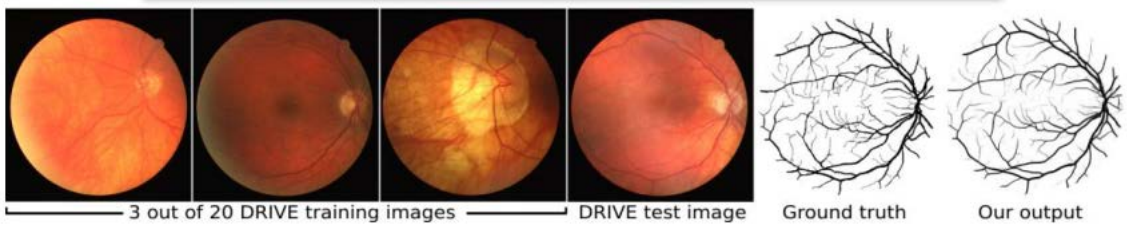
[1] Ciresan et al., *Deep, Big, Simple Neural Nets for Handwritten Digit Recognition*. Neural Computation 2010

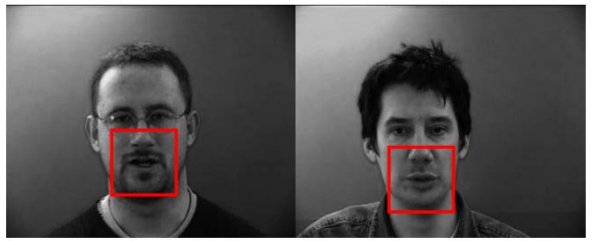
Image Classification – a HUGE field

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA









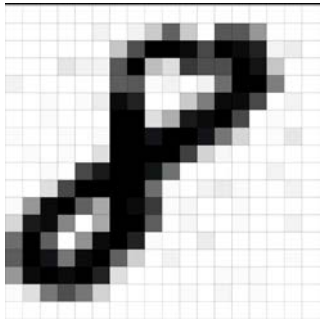
IDSIA Image Classification Networks won Six International Competitions

- **First place** at Assessment of Mitosis Detection Algorithms, MICCAI 2013 Grand Challenge, Nagoya, Japan
- **Best result** at Offline Chinese Character Recognition, ICDAR 2013, Washington D.C., USA
- **First place** at Segmentation of neuronal structures in EM stacks - ISBI 2012, Barcelona, Spain
- **First place** at Mitosis Detection in Breast Cancer Histological Images, ICPR 2012, Tsukuba, Japan
- **First place** at Offline Chinese Character Recognition, ICDAR 2011, Beijing, China
- **First place** at The German Traffic Sign Recognition Benchmark, IJCNN 2011, San Jose, USA

(credits to Dan Ciresan, Alessandro Giusti, and colleagues, see http://people.idsia.ch/~juergen/vision.html)

Image Classification

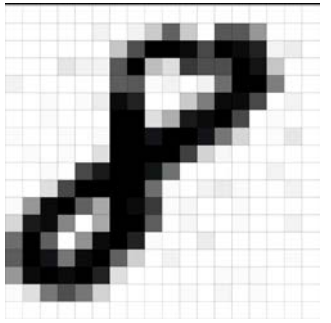
- Is the MLP architecture optimal for image recognition?



"8"

Image Classification

- Is the MLP architecture optimal for image recognition?
  - Certainly, the MLP can learn/approximate any function from input to output!
    - (within reasonable contraints)
- But…
  - Two-dimensional input -> large input dimensionality -> many weights!
  - Disregards spatial information!
  - Does not allow to share "knowledge" across different parts of the image!

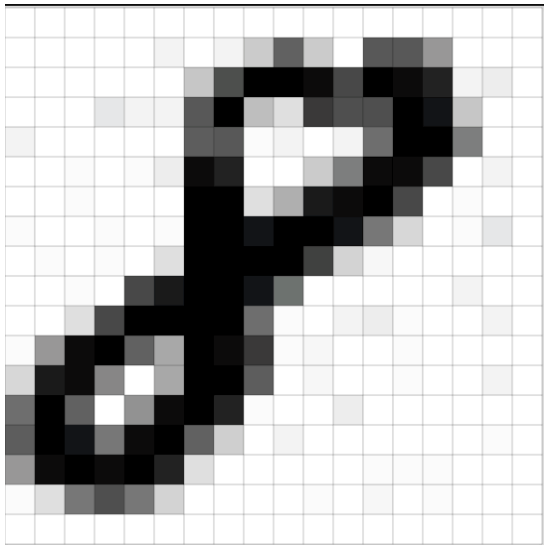


"8"

Image classification

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

- The MLP disregards any spatial information in the input!



(not shown completely)

Image classification

- Furthermore, we can imagine that it is helpful to learn common knowledge from *different* parts of the input image
  - Edge detection, common shapes, materials, …

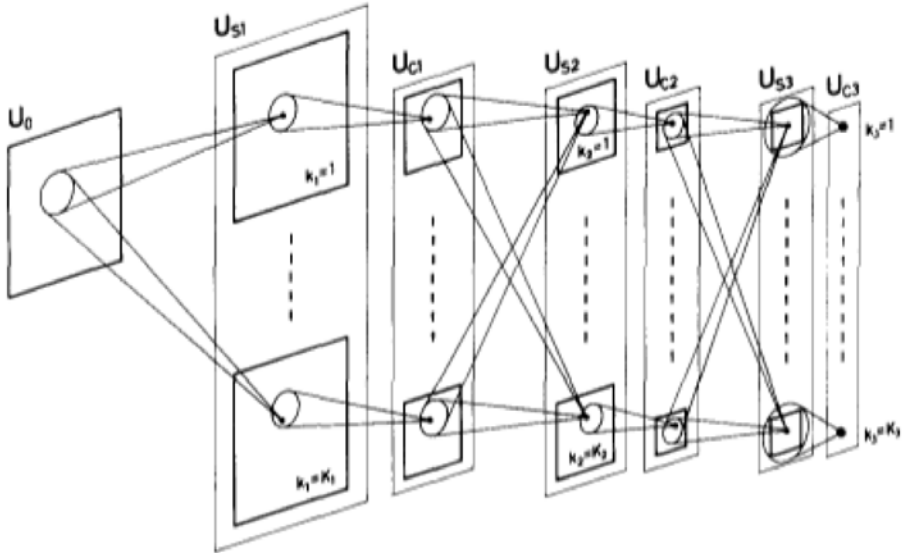
Image classification

- Furthermore, we can imagine that it is helpful to learn common knowledge from *different* parts of the input image
  - Edge detection, common shapes, materials, …
  - Buzzword: ***Parameter sharing***
- Along this line of thought, we could imagine that focusing on a *part* of the image may be a good first step
  - Not look at everything at once
  - Step by step, extend your field of vision
  - Understand image "hierarchically"

Neocognitron (Fukushima ‘79)

USI/SUPSI
Istituto Dalle Molle di studi sull’intelligenza artificiale
IDSIA

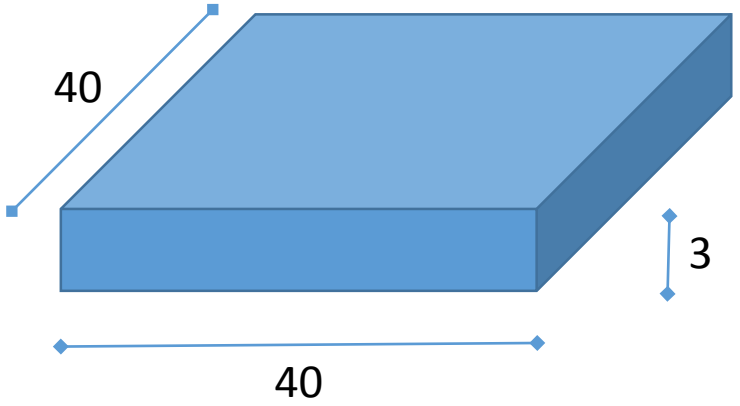
- Introduced Convolutional Neural Networks



- Bioinspired, hierarchical model
- Multiple types of cells:
  - S-cells extract local features
  - C-cells deal with deformations
- Weight sharing
  - Filters are shifted across the input map
  - Trained with local Winner Take All unsupervised rules

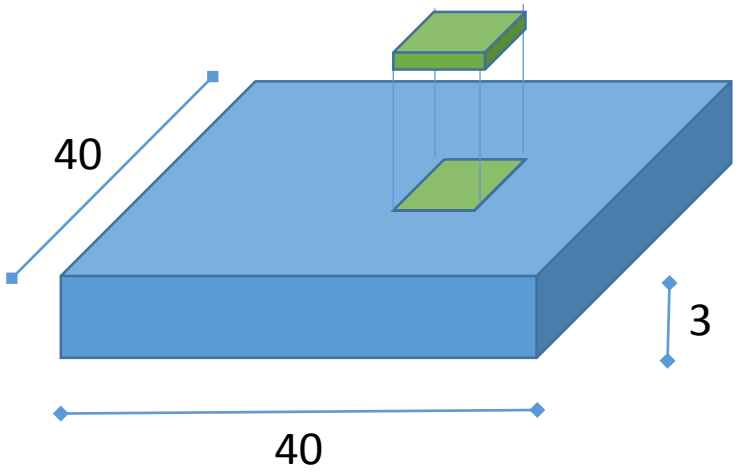
Convolutional Layer

- Let's take a sample image – say, 40x40 pixels and 3 color channels



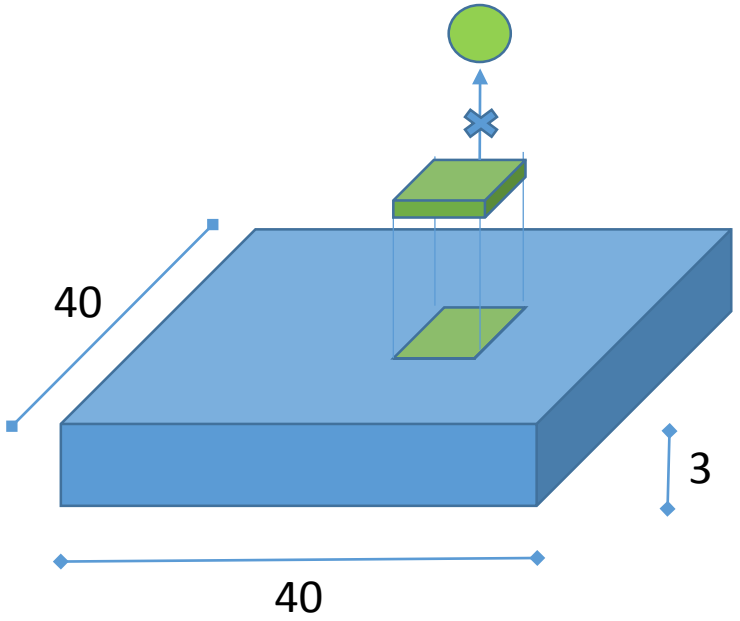
Convolutional Layer

- Let's take a sample image – say, 40x40 pixels and 3 color channels
- Shift a *convolutional filter* over the image, say, 5x5 pixel



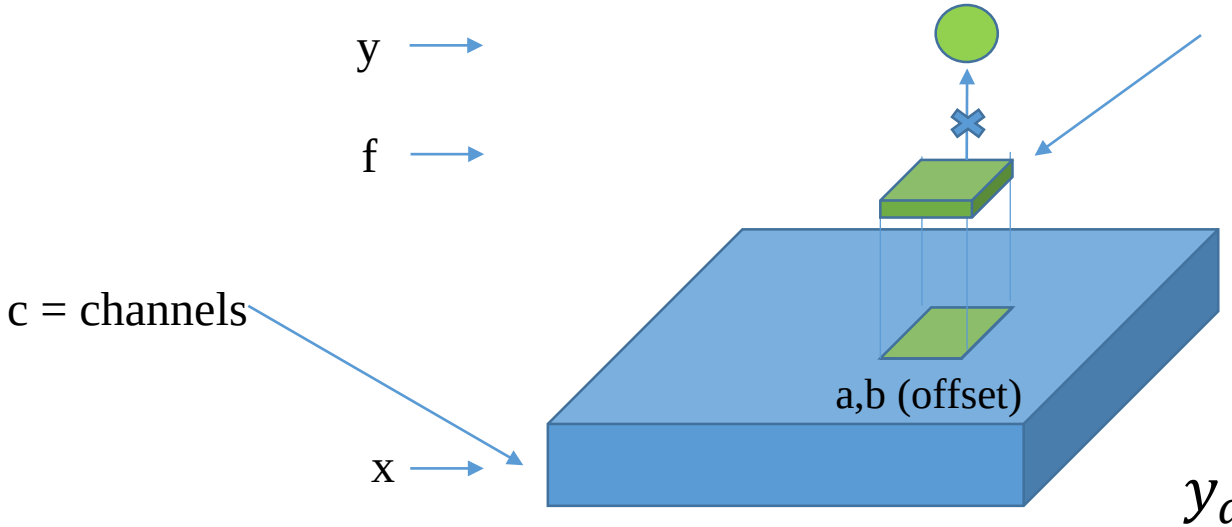
Convolutional Layer

- Let's take a sample image – say, 40x40 pixels and 3 color channels
- Shift a *convolutional filter* over the image, say, 5x5 pixels
- Multiply pixel values by filter values and sum, yielding a single result



Convolutional Layer

- Let's take a sample image – say, 40x40 pixels and 3 color channels
- Shift a *convolutional filter* over the image, say, 5x5 pixels
- Multiply pixel values by filter values and sum (and add bias), get single result



$$y_{a,b} = \sum_{i=0}^{s}\sum_{j=0}^{s}\sum_{c=0}^{C} f_{i,j,c}x_{i+a,j+b,c} + b$$

Convolutional Layer

Why does this operation make sense?

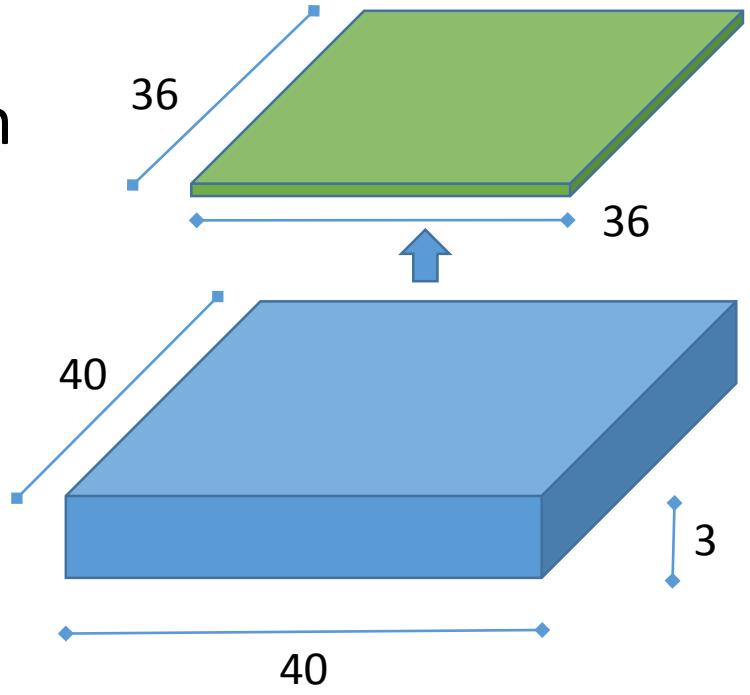
- Think about how you measure similarity in a vector space
  - Compute the *dot product* between two vectors:

$$x = (x_1, x_2, \ldots, x_N);\ y = (y_1, y_2, \ldots, y_N);\ x \cdot y = \sum_i x_i y_i$$

  - If the dot product is zero, *x* and *y* are orthogonal, if not, it is a measure of "affinity" between *x* and *y*
  - The convolution operator is nothing but a dot product: It measures how well the filter fits the respective part of an image!
  - Example: learned filters from an image classification task,
    first convolutional layer: Can you imagine how some of these detect edges?

Convolutional Layer

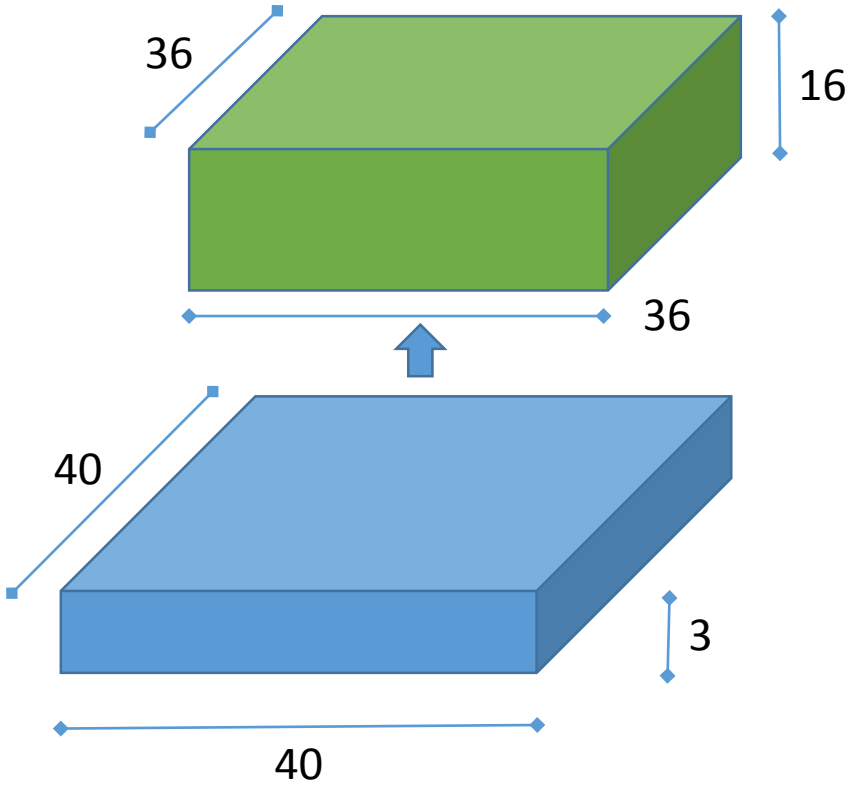
- Performing the convolution operation at all possible offsets yields a *feature map* which describes the presence of the *feature* represented by the filter across the image
- (Example: edges…)
- Slightly smaller than the image due to the filter size
- Spatial structure preserved!



You can also pad the input to retain the size of the input data.

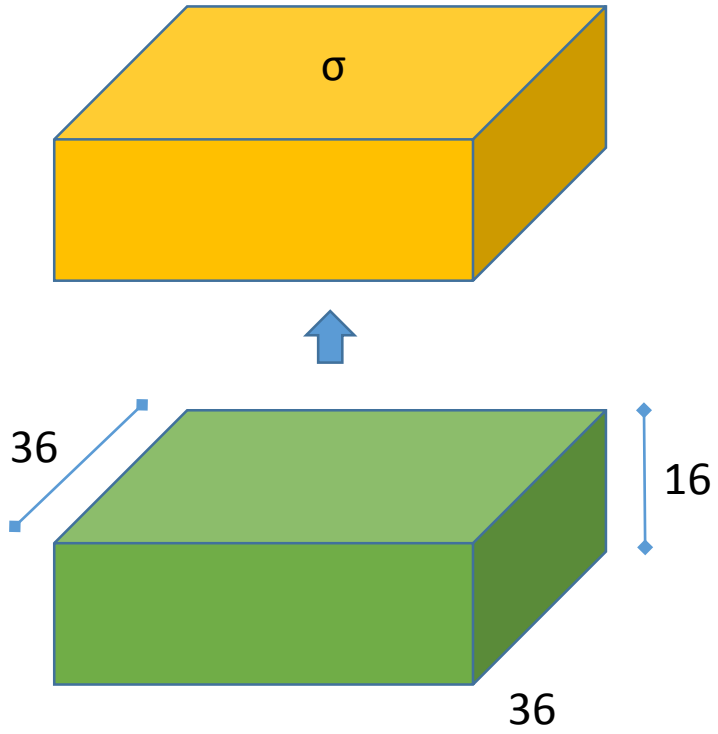
Convolutional Layer

- Of course, we need several features, thus we get several feature maps



Convolutional Layer

- After the feature map, we will frequently apply a nonlinearity as usual
- We can save parameters by applying the filter only every *n*-th pixel (*stride,* not shown)



Convolutional Layer

- The operation of a convolutional layer, summarized:
  - Filter with a certain size (often called *kernel size)* is shifted over the image, with a specific step size *(stride)*
  - At each offset, filter coefficients are multiplied by pixel values across *all* channels, result is summed to create single output
  - Yields *feature maps* (one per filter) which preserve spatial structure

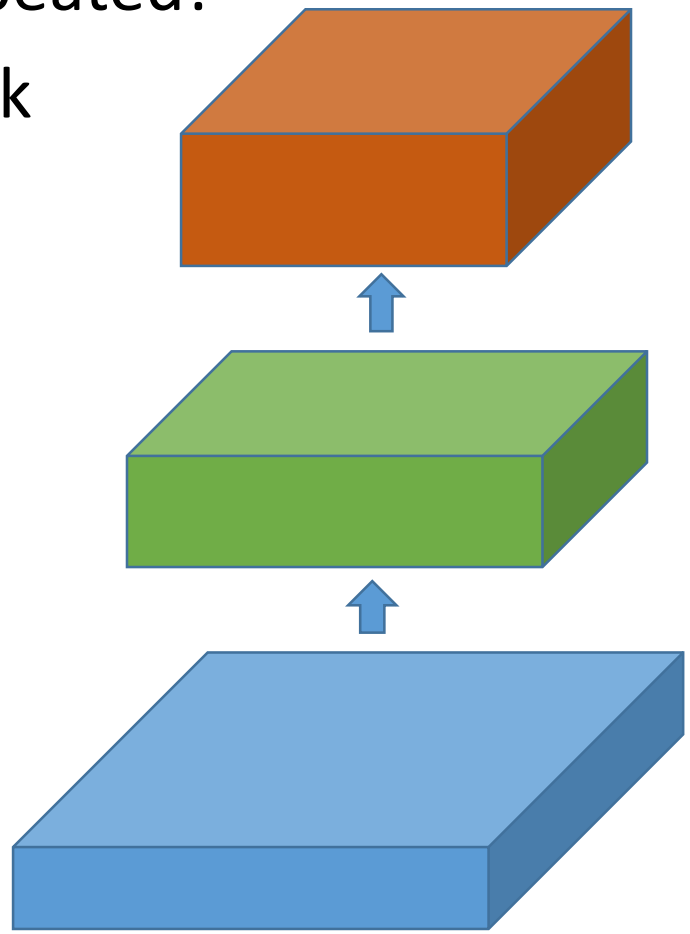
$$y_{a,b} = \sum_{i=0}^{s} \sum_{j=0}^{s} \sum_{c=0}^{C} f_{i,j,c} x_{i+a,j+b,c} + b$$

(*x* = image, *f* = one single filter, *b* = bias, *y* = one single feature map)

  - Finally, compute $z = \sigma(y)$ component-wise
- As with the MLP, the filter parameters must be empirically determined

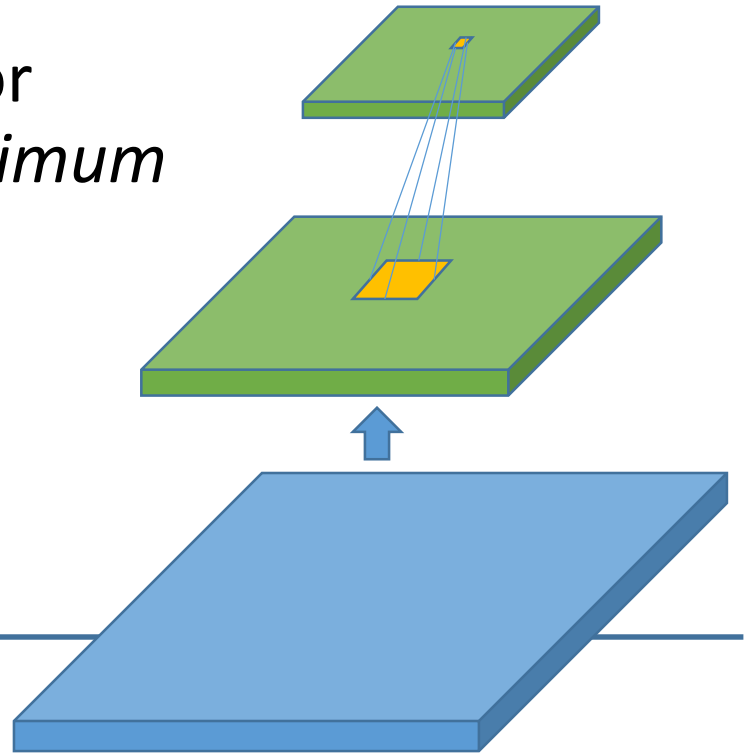
Convolutional Layers Iterated

- The application of a convolutional layer can be repeated!
- But there is an issue: We do not have a way to look at the image as a whole!
  - Always small parts, as determined by the filter size



Pooling layers

- How to consider consecutively larger "receptive fields"?
- *Pooling layers* join several adjacent feature values, thus shrinking the size of the feature map
- Most common version: *max pooling*
- As with the convolutions, the max pooling operator is shifted across each feature map, taking the *maximum* over several adjacent values



Pooling layers

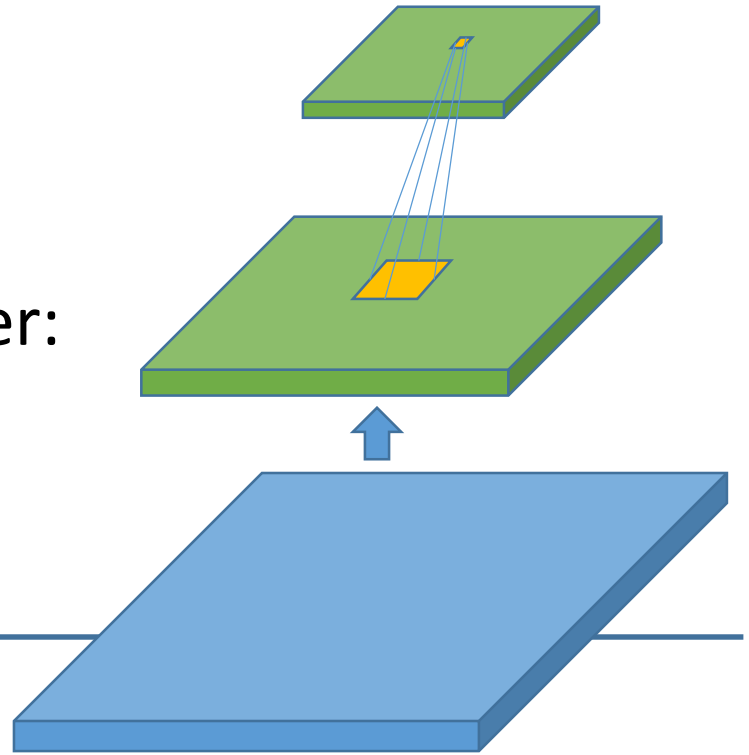
USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

Example with a pooling kernel size of *k* :

- Input: Feature map *y* of size *N*N*
- Output: Pooled feature map *z* of size (*N/k,N/k*)

$$z_{a,b} = \max_{i,j=0\ldots k-1} y_{a\cdot k+i,b\cdot k+j}$$

- this can also be varied by adding a stride parameter:
  - Small stride makes pooling operations overlap

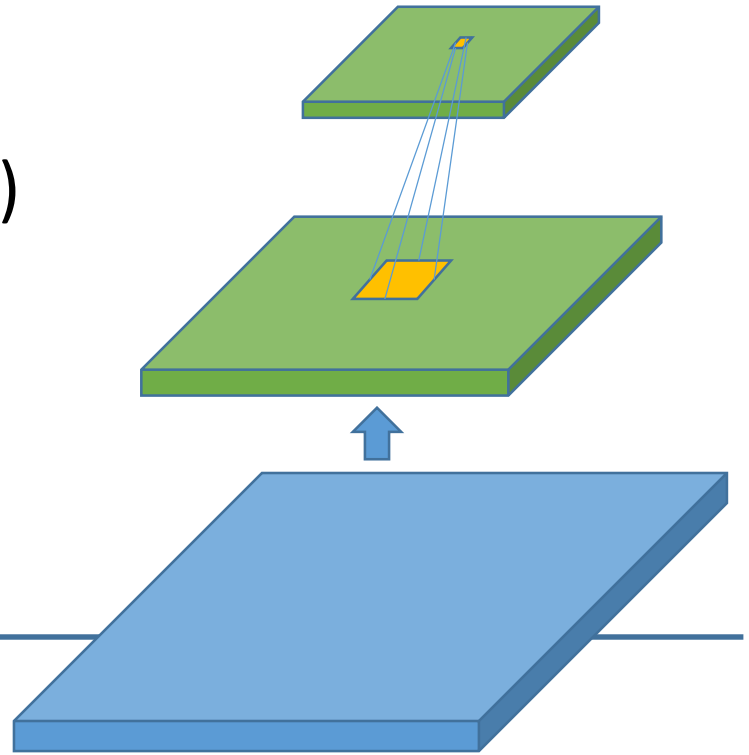


Pooling layers

Max pooling allows hierarchies of only convolutional layers. Furthermore, it

- improves generalization
- induces robustness to small position shifts
- decreases dimensionality of the representation!

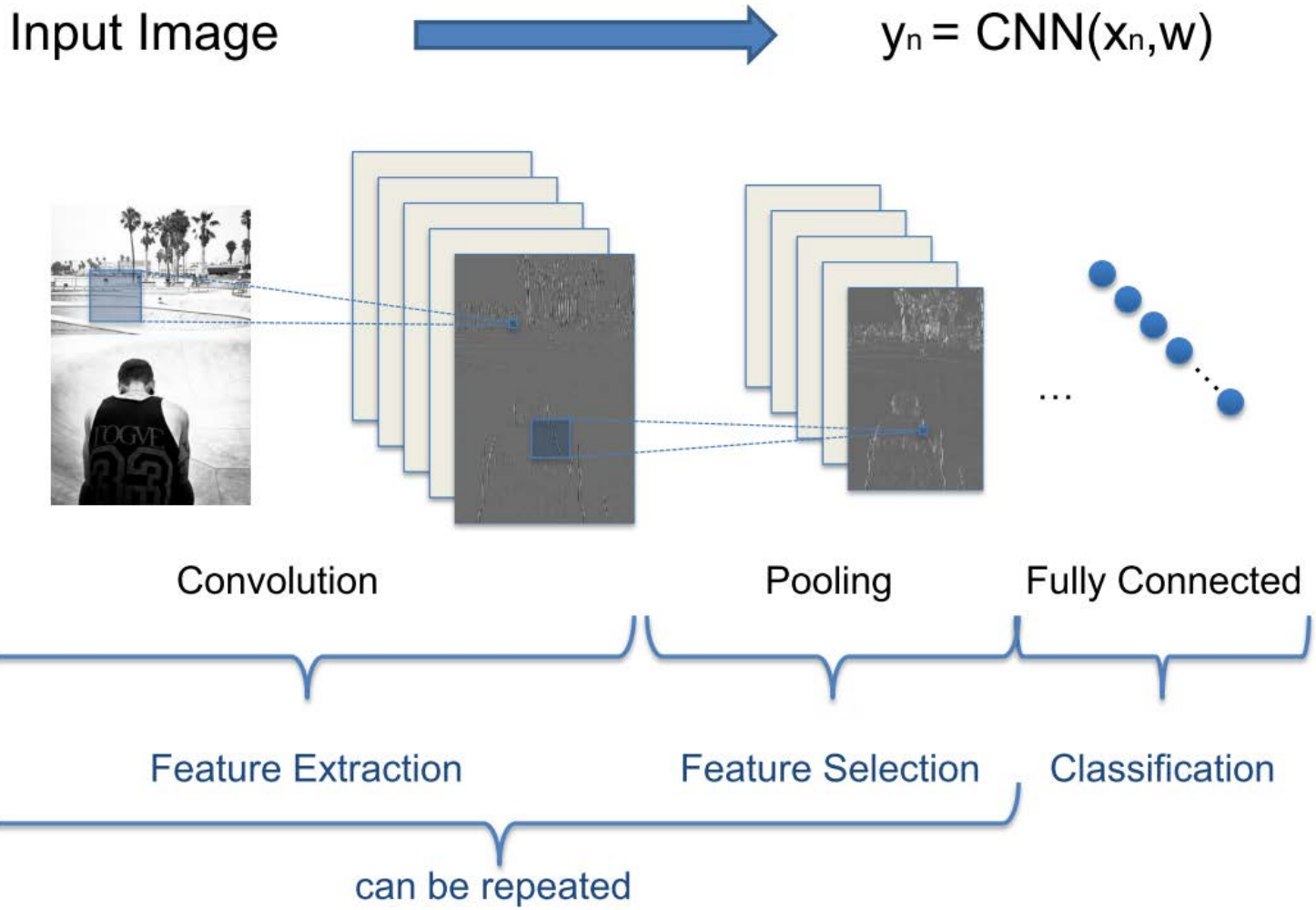
There are also variants around (e.g. average pooling)



Convolutional Network Architecture

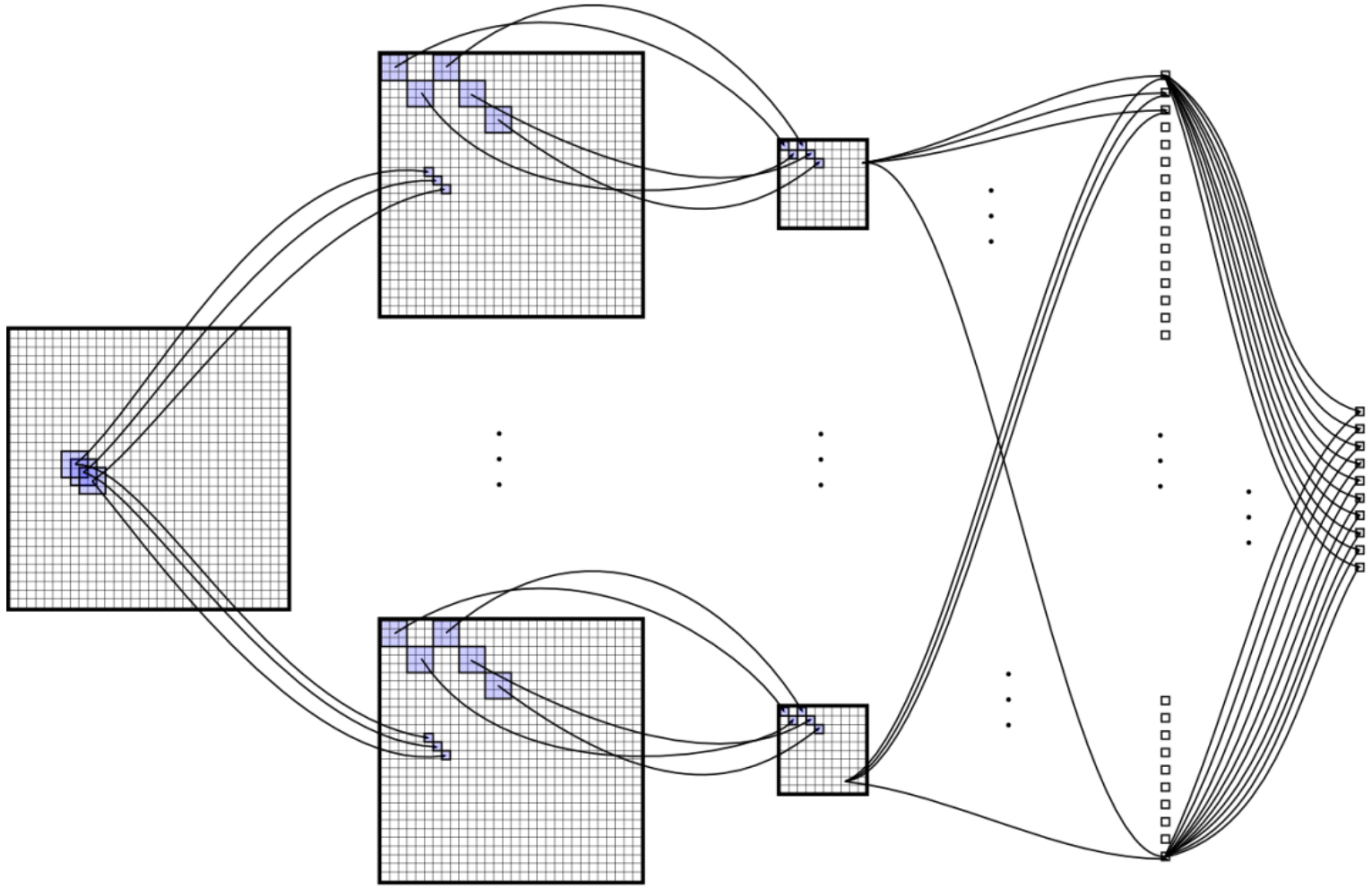
- Very common architecture: alternating convolutional / max-pooling layers
  - Convolutions e.g. of kernel size 3x3 … 7x7, stride 1, followed by nonlinearity
  - Max-pooling with kernel size 2x2 or 3x3, stride == kernel
- Hierarchically compute representations (feature maps) which cover larger and larger areas
  - Size of representations decreases exponentially, e.g. 256x256 -> 128x128 ->64x64 …
- Finally, add a few fully connected layers, and the final layer (e.g. softmax for classification)
- *End-to-end* architecture without handcrafted features

Convolutional Network Architecture

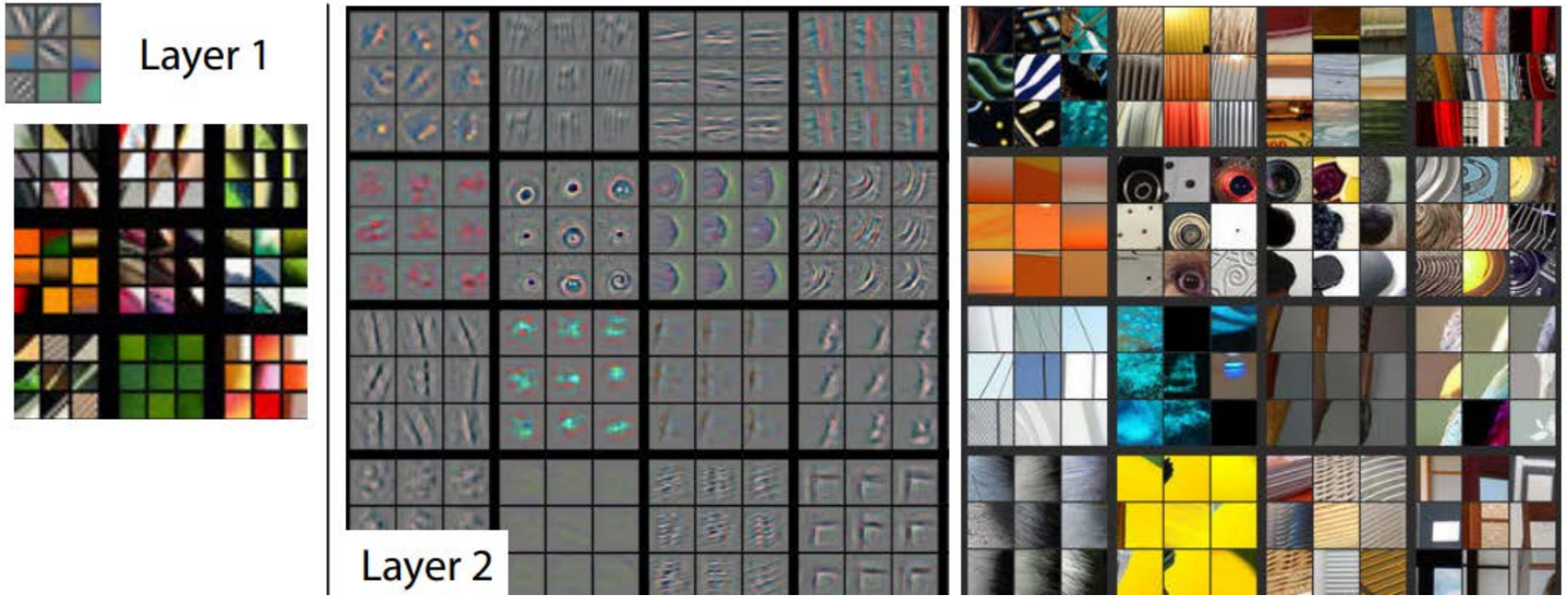


Convolutional Network Architecture

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

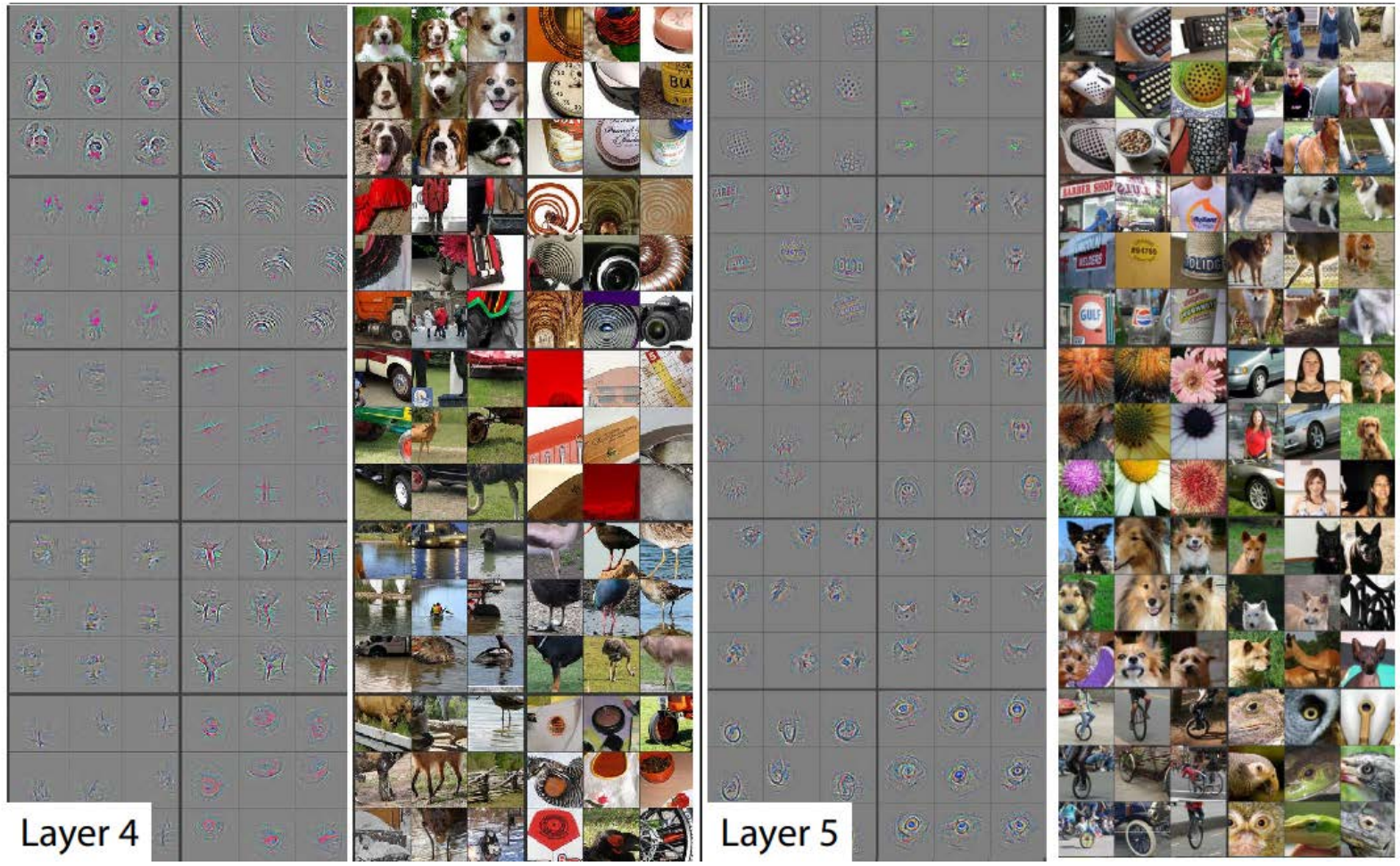


Visualizing learned filters



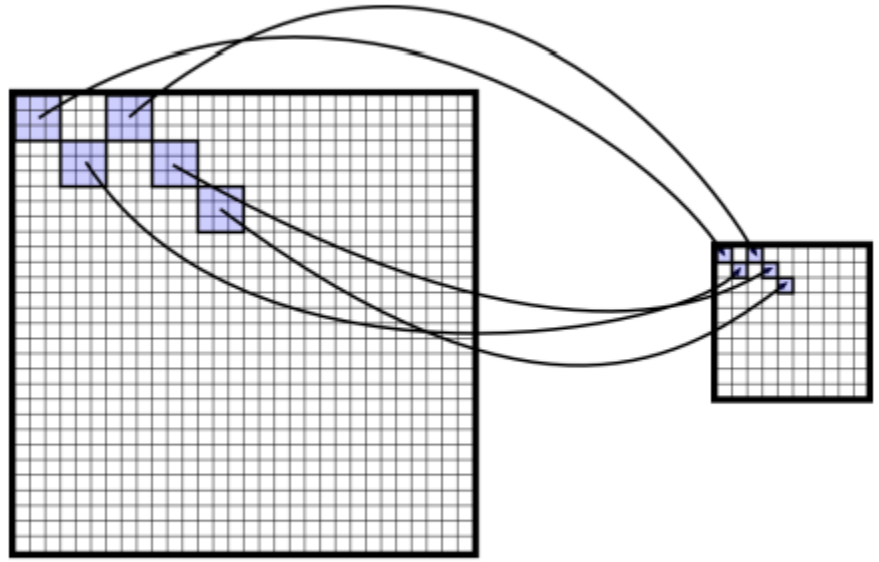
Learned filters on an image classification task (Imagenet), and highest activations.
From Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, 2013

Visualizing learned filters



Training of Convolutional Networks

- Training is done with backpropagation, layer for layer
- So, we need to find out how to do backprop for convolutional and max-pooling layers
- Max-pooling is easy:
  - In the forward step, remember which source neuron had maximum activation
  - the error which arrives at the max-pooling layer is propagated to that one neuron
  - The neurons which were not maximal receive gradient zero since changing them slightly does not change the output



Training of Convolutional Networks

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

- What about the convolutional layer itself?
- Recap fully connected layer:

$$y_j = \sum_i w_{ij} x_i \qquad \text{and} \qquad z_j = f(y_j)$$

- We need the derivative of z_j w.r.t. w_{ij}

$$\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial z_j}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}} \qquad \text{with} \qquad \frac{\partial z_j}{\partial y_j} = f'(y_j) \qquad \text{and} \qquad \frac{\partial y_j}{\partial w_{ij}} = x_i$$

Training of Convolutional Networks

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

- For the convolutional layer, the situation is almost the same!
- With offset *a,b* and map index *c,* for a single output $z_{a,b,c}$

$$z_{a,b,c} = f(y_{a,b,c}) \quad \text{and} \quad y_{a,b,c} = \sum_{i,j,k} w_{i,j,k} x_{i+a,j+b,k}$$

  where *k* is the map index in the source layer.
- Consequently, we have $\frac{\partial y_{a,b,c}}{\partial w_{i,j,k}} = x_{i+a,j+b,k}$

Training of Convolutional Networks

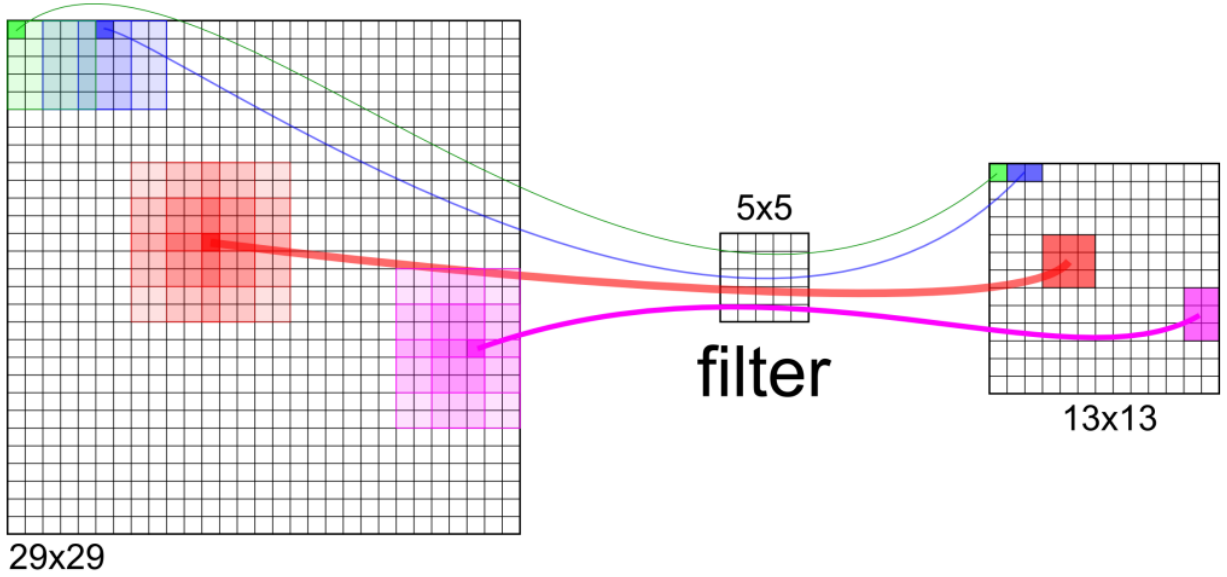
- For the convolutional layer, the situation is almost the same!
- With offset *a,b* and map index *c,* for a single output $z_{a,b,c}$

$$z_{a,b,c} = f(y_{a,b,c}) \quad \text{and} \quad y_{a,b,c} = \sum_{i,j,k} w_{i,j,k} x_{i+a,j+b,k}$$

  where *k* is the map index in the source layer.
- Consequently, we have $\frac{\partial y_{a,b,c}}{\partial w_{i,j,k}} = x_{i+a,j+b,k}$
- The only difference: In the fully connected case, weight updates for w_{ij} come only from output neuron *j*, whereas for the convolutional layer, weight updates for any given weight come from the entire feature map.

Training of Convolutional Networks

- Need to sum weight updates from all offsets. (Why???)



- In practice, first collect all weight deltas, then perform update (requires just one write)

Training is Computationally Expensive

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

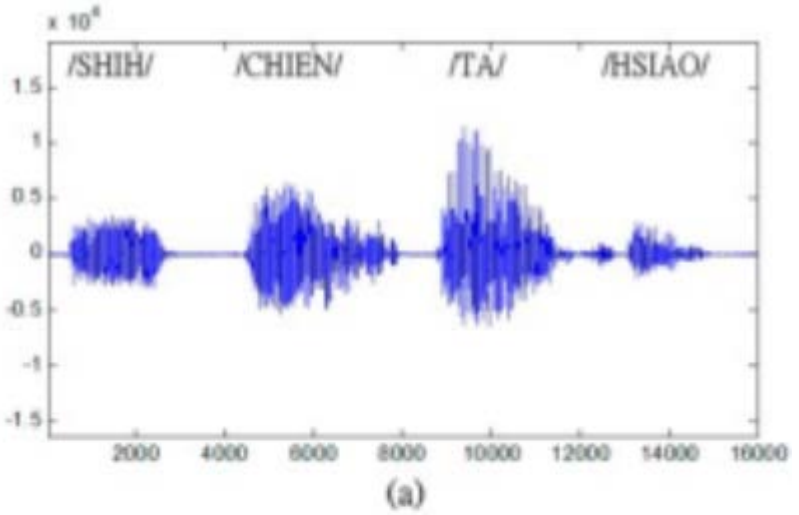
9HL-48x48-100C3-MP2-200C2-MP2-300C2-MP2-400C2-MP2-500N-3755		
Net for Chinese OCR	#weights	#connections
Feature extraction layer:	321'800	26.37 Million
Classification layer:	1'881'255	1.88 Million

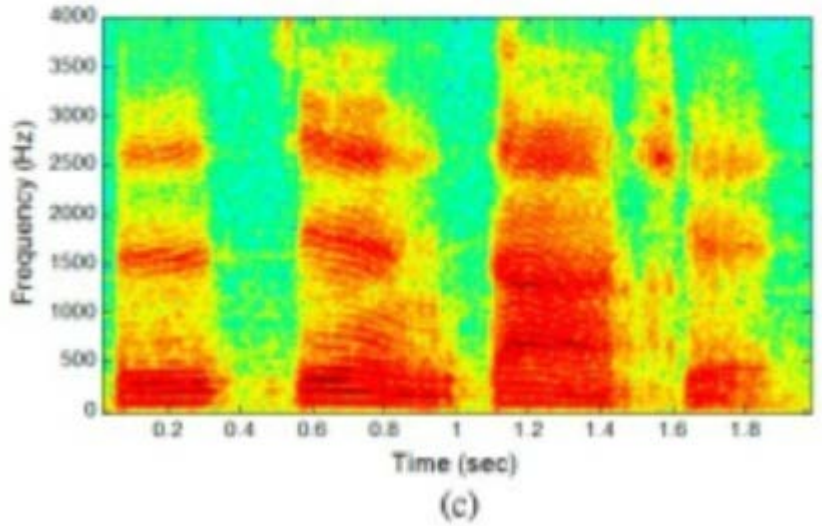
1'121'749 training samples
CPU (i7-920, one thread):
27h to forward propagate one epoch
14 months to train for 30 epochs

… or one week on a GPU (2016)!

Final remarks

- Convolutional networks have revolutionized image processing
- … which uses *two-dimensional* convolutions
  - But they can also have other dimensionalities
  - …and be applied to a variety of other signals
- Example: Speech recognition with ConvNets
  - Compute *spectrogram* of speech signal (image)
  - … or apply one-dimensional convolutions to raw signals!
  - (Compare Alex Waibel's time delay neural networks)
- Michael works on *Lipreading*: process images to recognize speech ☺

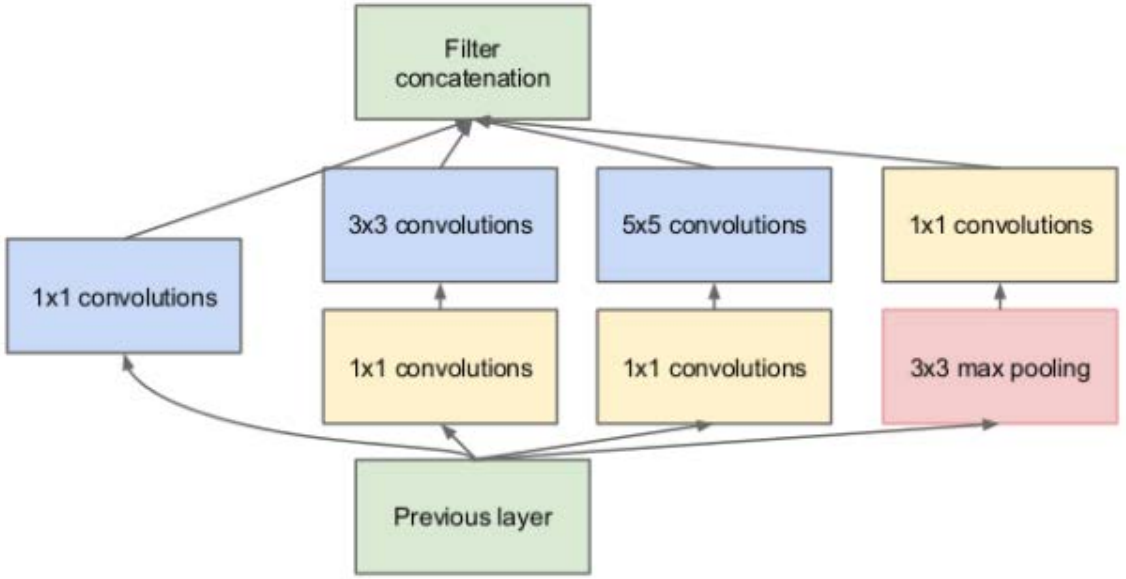




Final remarks

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

- You now know the standard convolutional architecture
  - For examples have a look at the papers of Dan Ciresan (IDSIA), or google AlexNet
- The convolutional architecture can be varied in a variety of ways
- Example: *Inception modules* which let the model decide on the size of the convolution kernel
  (Szegedy et al., *Going Deeper with Convolutions*, 2014)



Conclusion / Summary

USI/SUPSI
Istituto Dalle Molle di studi sull'intelligenza artificiale
IDSIA

Today you should have learned

- what is the idea of convolutional layers / networks
- why they are useful in image processing (but not only there)
- how forward and backward propagation change in the convolutional case (not so much!)
- how you (in principle) construct a full network for an image classification task