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

Convolutional Neural Networks

Machine Learning

Michael Wand

TA: Vincent Herrmann (vincent.herrmann@idsia.ch)

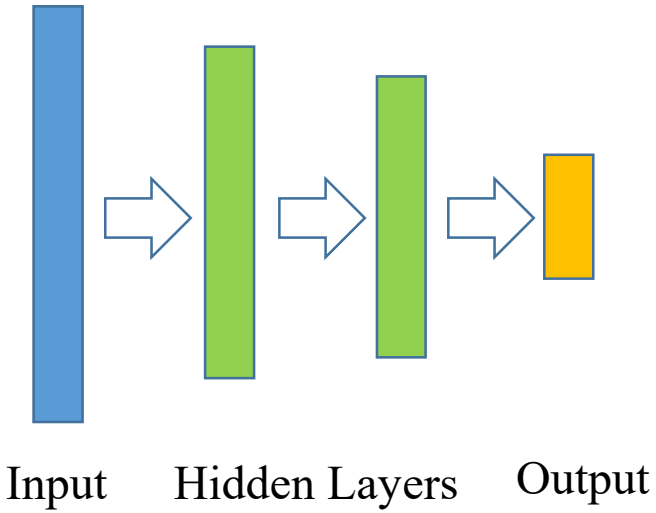
Fall semester 2023

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 *l* are connected to all neurons of layer *l+1*)
  - Training objective is to learn the weight of the connections.

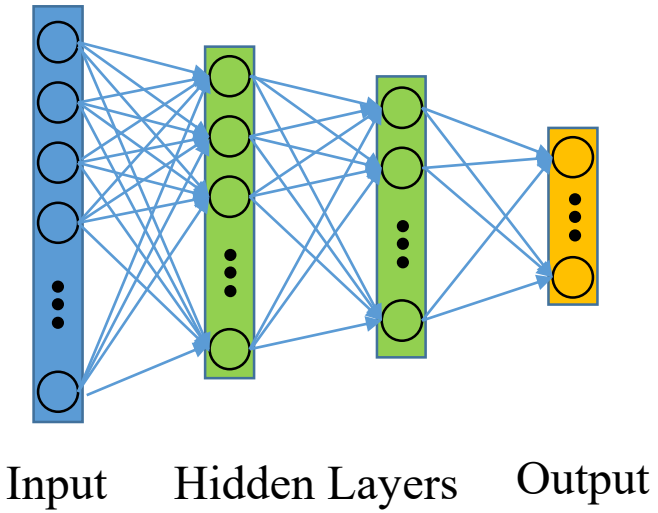
Recap: Fully connected neural networks

- 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 *l* are connected to all neurons of layer *l+1*)
  - Training objective is to learn the weight of the connections.



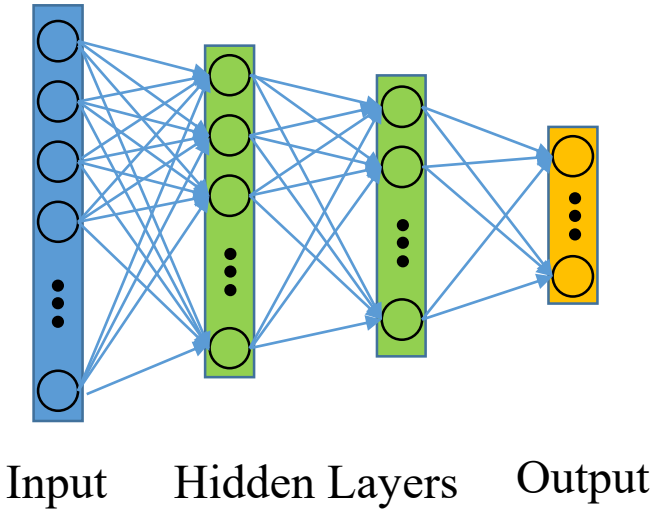
Recap: Fully connected neural networks

- 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 *l* are connected to all neurons of layer *l+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 *l* are connected to all neurons of layer *l+1*)
  - Training objective is to learn the weight of the connections.



Also called
Multi-Layer Perceptron (MLP)

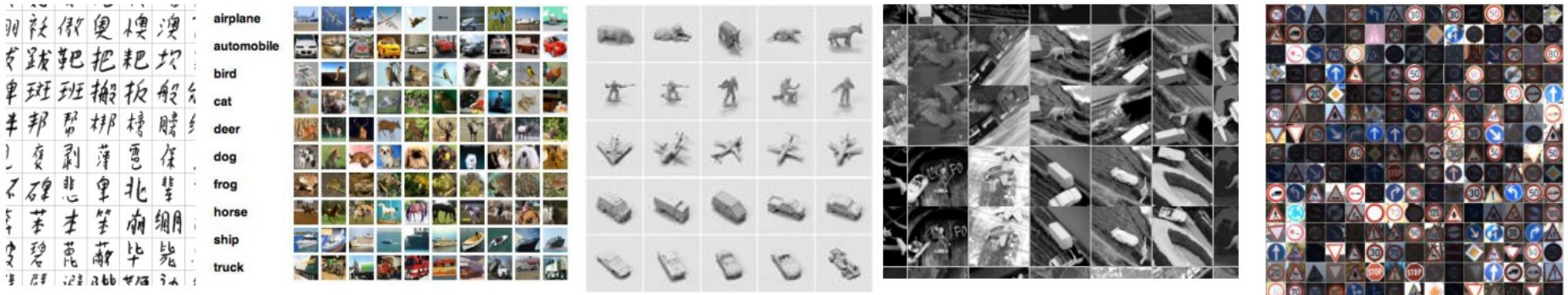
A classical example: MNIST

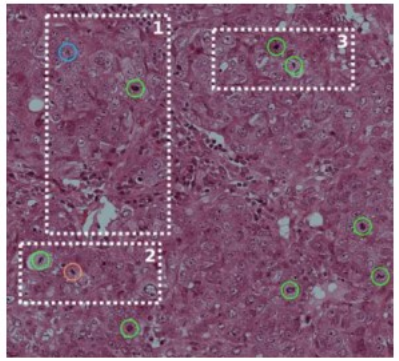
- Let us look at image processing tasks!
- Task: recognize handwritten digits
- Standard training and test sets available, good benchmark task
- Record-breaking GPU implementation of MLP classifier on MNIST [1]

[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





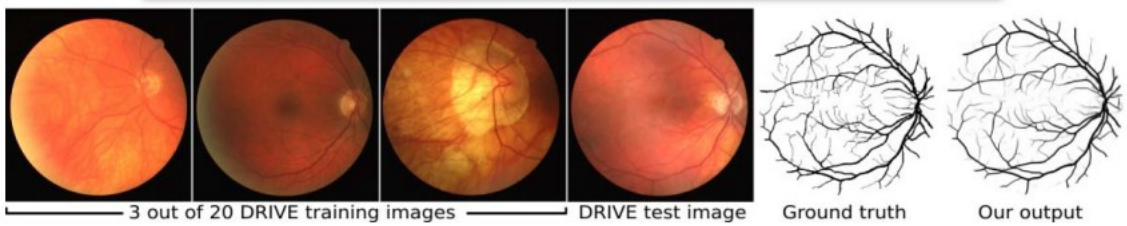
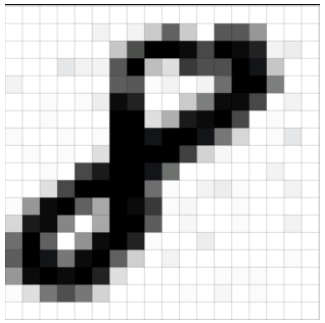


Image Classification

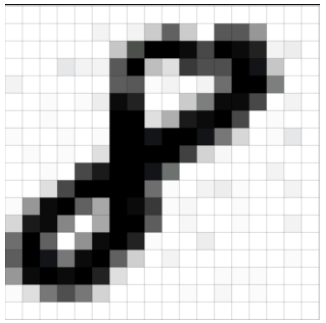
- Is the MLP architecture optimal for image recognition?



"8"

Image Classification

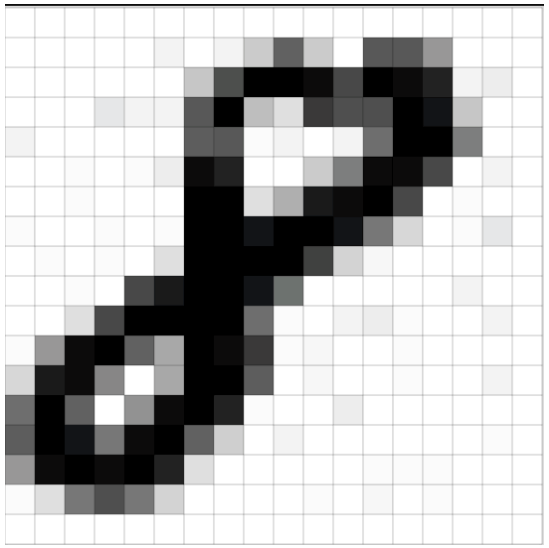
- Is the MLP architecture optimal for image recognition?
  - The MLP can learn/approximate any function from input to output!
    - (within reasonable constraints, which we have not covered)
- But…
  - Two-dimensional input -> large input dimensionality -> many weights!
  - Disregards spatial information!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



"8"

Image classification

- This is what the MLP sees…



(not shown completely)

Image classification

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

- Furthermore, intuitively it is helpful to learn *common* knowledge from *different* parts of the input image.
  - Edge detection, common shapes, materials, shift invariance …

Image classification

- Furthermore, intuitively it is helpful to learn *common* knowledge from *different* parts of the input image.
  - Edge detection, common shapes, materials, shift invariance …
  - Buzzword: ***Parameter Sharing***

Image classification

- Furthermore, intuitively it is helpful to learn *common* knowledge from *different* parts of the input image.
  - Edge detection, common shapes, materials, shift invariance …
  - Buzzword: ***Parameter Sharing***
- Similarly, we may wish to apply a common computation to different parts of the image.

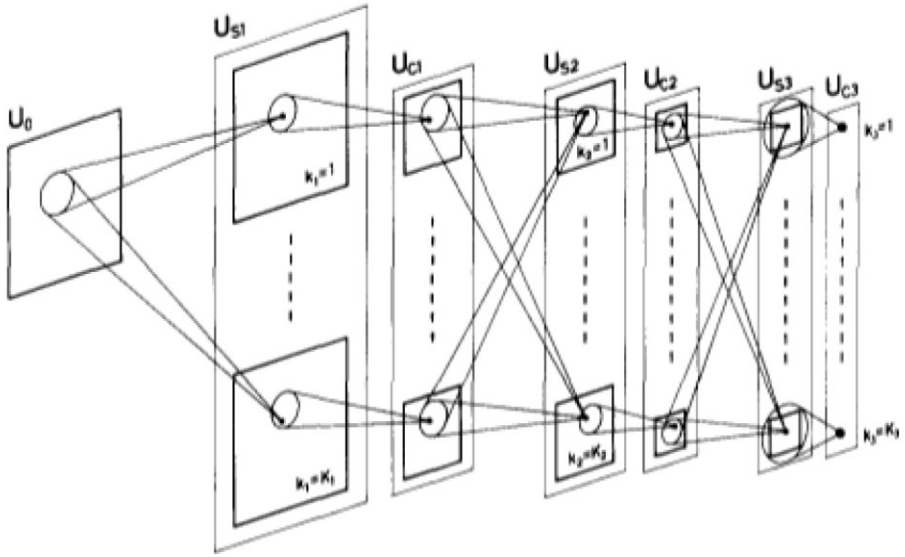
Image classification

- Furthermore, intuitively it is helpful to learn *common* knowledge from *different* parts of the input image
  - Edge detection, common shapes, materials, shift invariance …
  - Buzzword: ***Parameter Sharing***
- Similarly, we may wish to apply a common computation to different parts of the image.
- We derive a method to focus at *parts* of the image, while *sharing* information across the image
  - Not look at everything at once, but learn from everywhere
  - 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

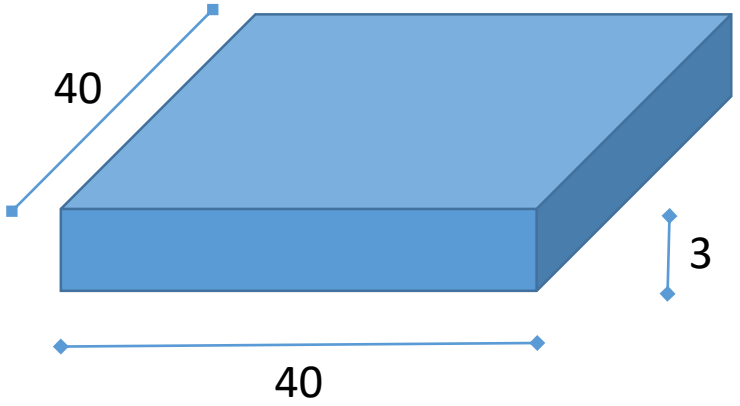
- Introduced Convolutional Neural Networks



- 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

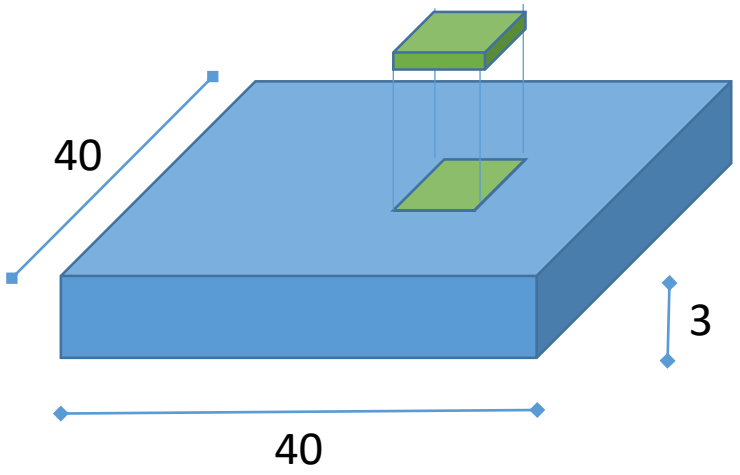
Convolutional Layer

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



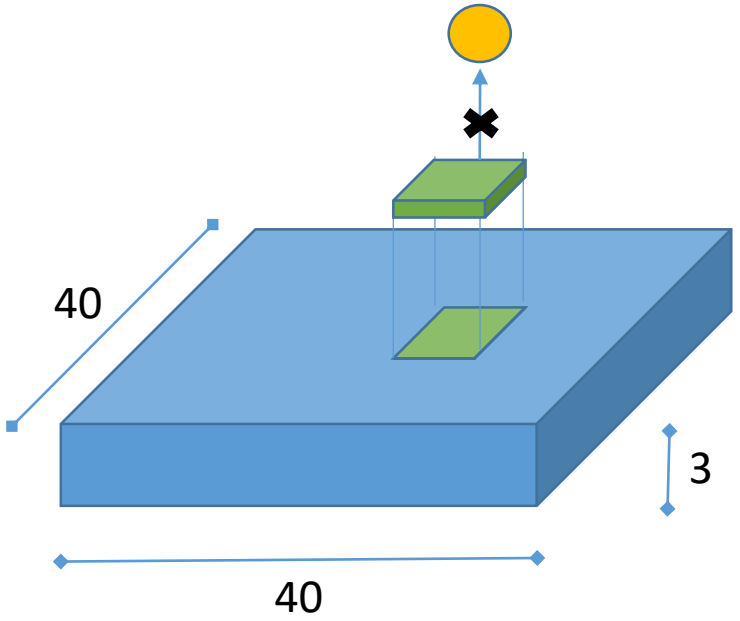
Convolutional Layer

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



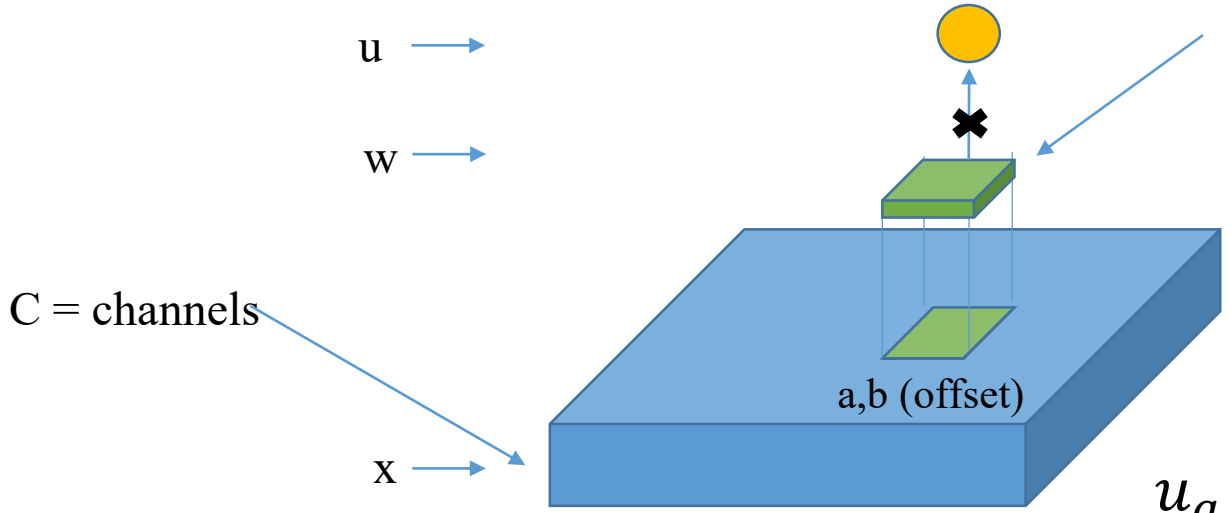
Convolutional Layer

- Let's take a sample image – say, 40x40 pixels and 3 color channels
- Slide a convolutional filter over the image, say, 5x5 pixels
- At each offset: Multiply pixel values by filter values and sum, add bias, get scalar result



Convolutional Layer

- Let's take a sample image – say, 40x40 pixels and 3 color channels
- Slide a convolutional filter over the image, say, 5x5 pixels
- At each offset: Multiply pixel values by filter values and sum, add bias, get scalar result



$$u_{a,b} = \sum_{i=0}^{s} \sum_{j=0}^{s} \sum_{c=0}^{C} w_{i,j,c} x_{i+a,j+b,c} + w_0$$

Convolutional Layer

How to interpret this operation?

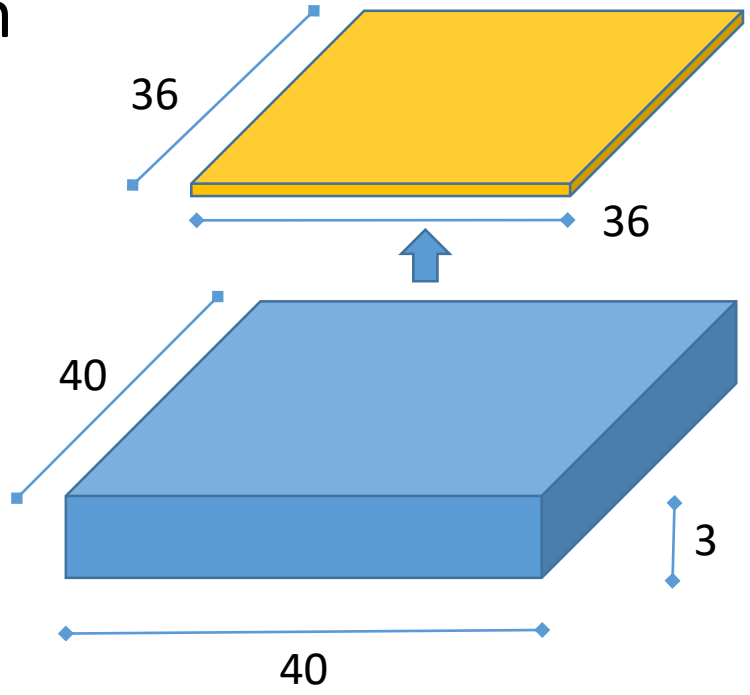
- Think about how you measure similarity in a vector space.
  - Compute the *dot product* between two vectors:

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

  - The dot product is a measure of "alignment" between *x* and *y* (in particular, it is zero if *x* and *y* are orthogonal).
  - 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

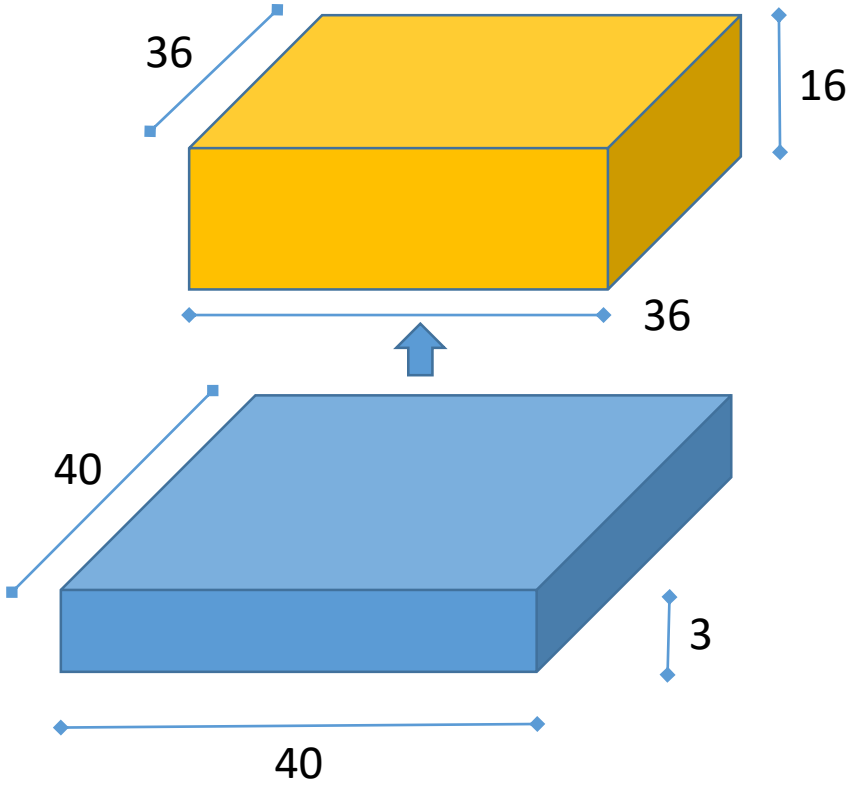
- Performing the convolution operation at all possible offsets yields a feature map which describes the presence of the feature represented by the filter across the image.
- Slightly smaller than the image due to the filter size
  - example: 5x5 filter
- Spatial structure preserved!



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

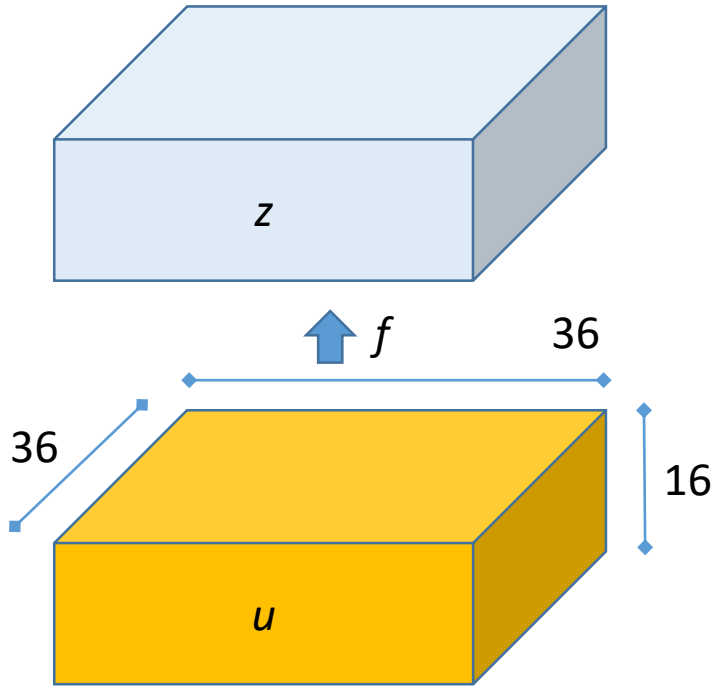
Convolutional Layer

- Of course, we need several features (i.e. several filters): say 16 for example.
- Thus we get several feature maps.



Convolutional Layer

- After the feature map, we frequently apply a nonlinearity (*f*), usually component-wise.
- We can reduce the representation size by applying the filter only every *n*-th pixel (*stride,* not shown)



Convolutional Layer Summarized

- Multiple filters with a certain size (often called kernel size) are shifted over the image, with a specific step size (stride).
- At each offset, the filter coefficients are multiplied by the pixel values (across all channels), result is summed to create scalar output.
- Yields feature maps (*K* filters – *K* feature maps) which preserve spatial structure

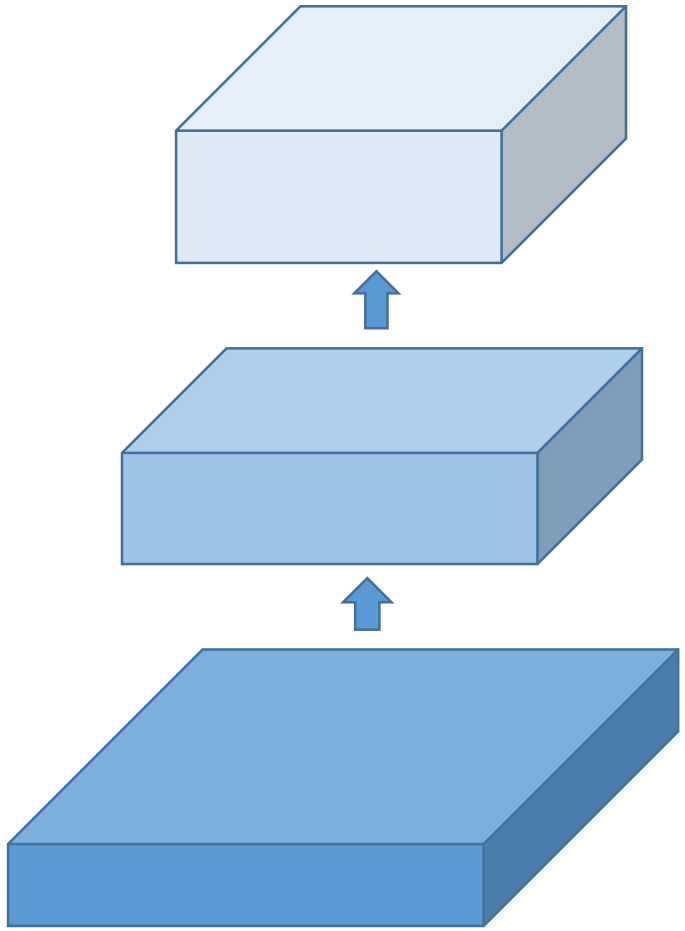
$$u_{a,b,k} = \sum_{i=0}^{s} \sum_{j=0}^{s} \sum_{c=0}^{C} w_{i,j,c}^{(k)} x_{i+a,j+b,c} + w_0^{(k)}$$

(*x* = image, *w* = filter, w_0 = bias, *c* = source channel, *k* = feature map)

- Finally, compute *z = f(u),* usually component-wise
- The filter meta-parameters (size, stride, number of filters) must be empirically determined.

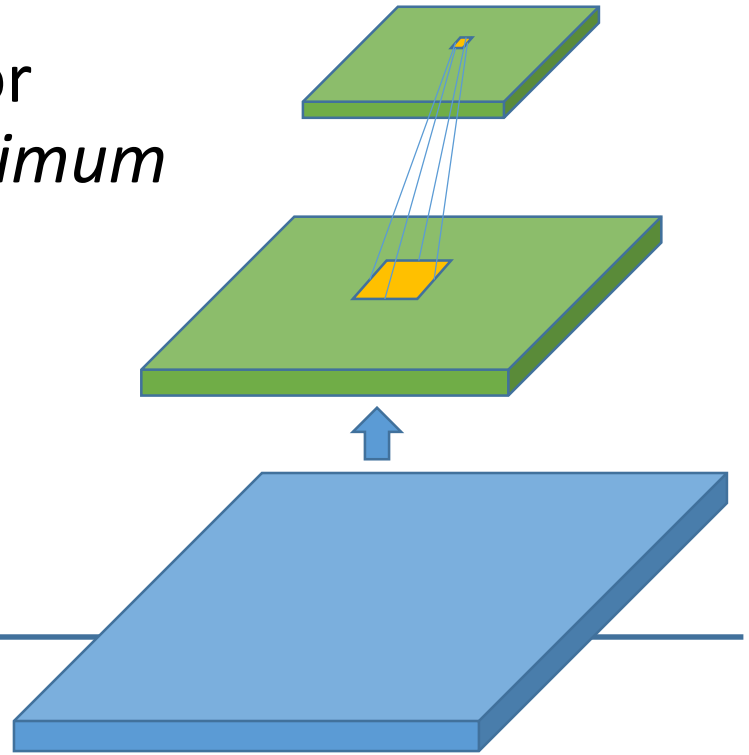
Convolutional Layers Iterated

- The application of a convolutional layer can be repeated!
  - The feature maps of the previous layer take the role of input channels in the subsequent layer.
- But there is an issue: We do not have a way to look at the image as a whole.
  - Each pixel of a feature map "sees" only a small part of the input, determined by the filter size.
  - This is called the *receptive field*.



Pooling layers

- How to consider consecutively larger receptive fields?
- Pooling layers join several adjacent pixels, 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.



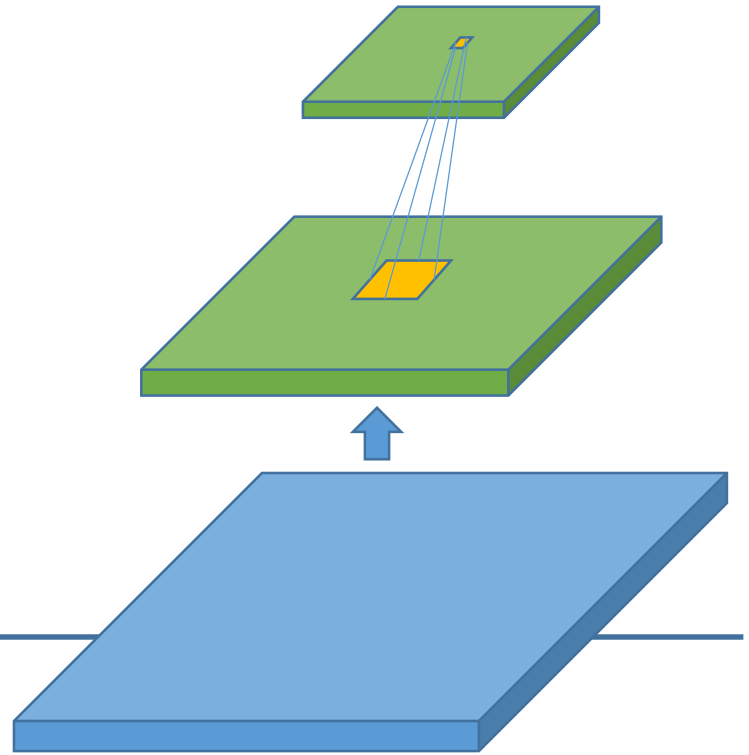
Pooling layers

Example with a pooling kernel size of *k* and a stride of *s*:

- Input: Feature map *u* of size *N*N*
- Output: Pooled feature map $\tilde{u}$ of size (*N/k,N/k*)

$$\tilde{u}_{a,b} = \max_{i,j=0\ldots k-1} u_{a\cdot k+i,b\cdot k+j}$$

- Also this operation can be varied by adding a stride parameter.
  - Small stride makes pooling operations overlap.



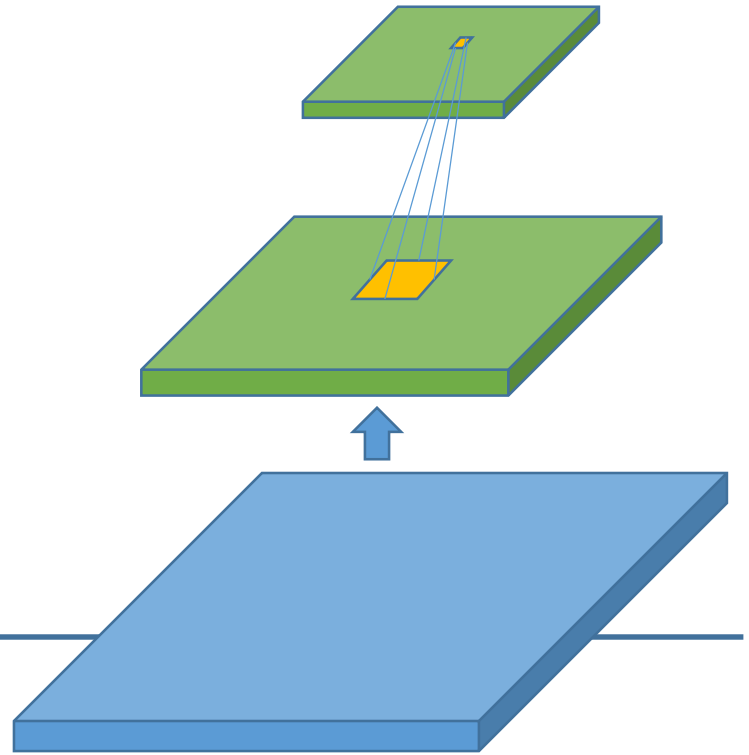
Pooling layers

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

Max pooling allows hierarchies of convolutional layers. Furthermore, it

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

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



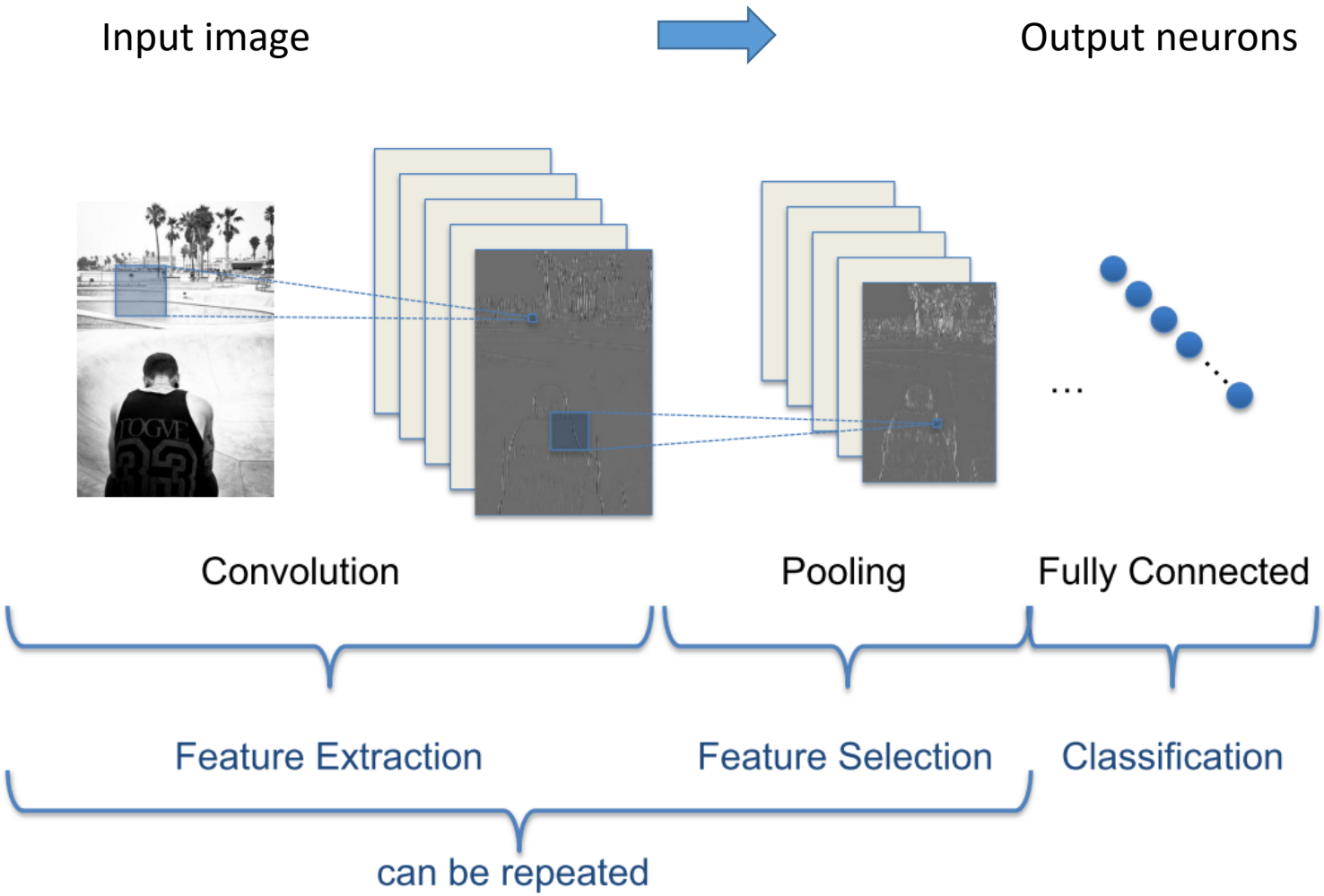
Convolutional Network Architecture

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

- 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 …
  - Depth of computation increases in parallel with decreasing representation size
- Finally, add a few fully connected layers, and the final layer (e.g. softmax for classification)
- *End-to-end* architecture without hand-engineered features
  - Convolutional feature maps take their role instead

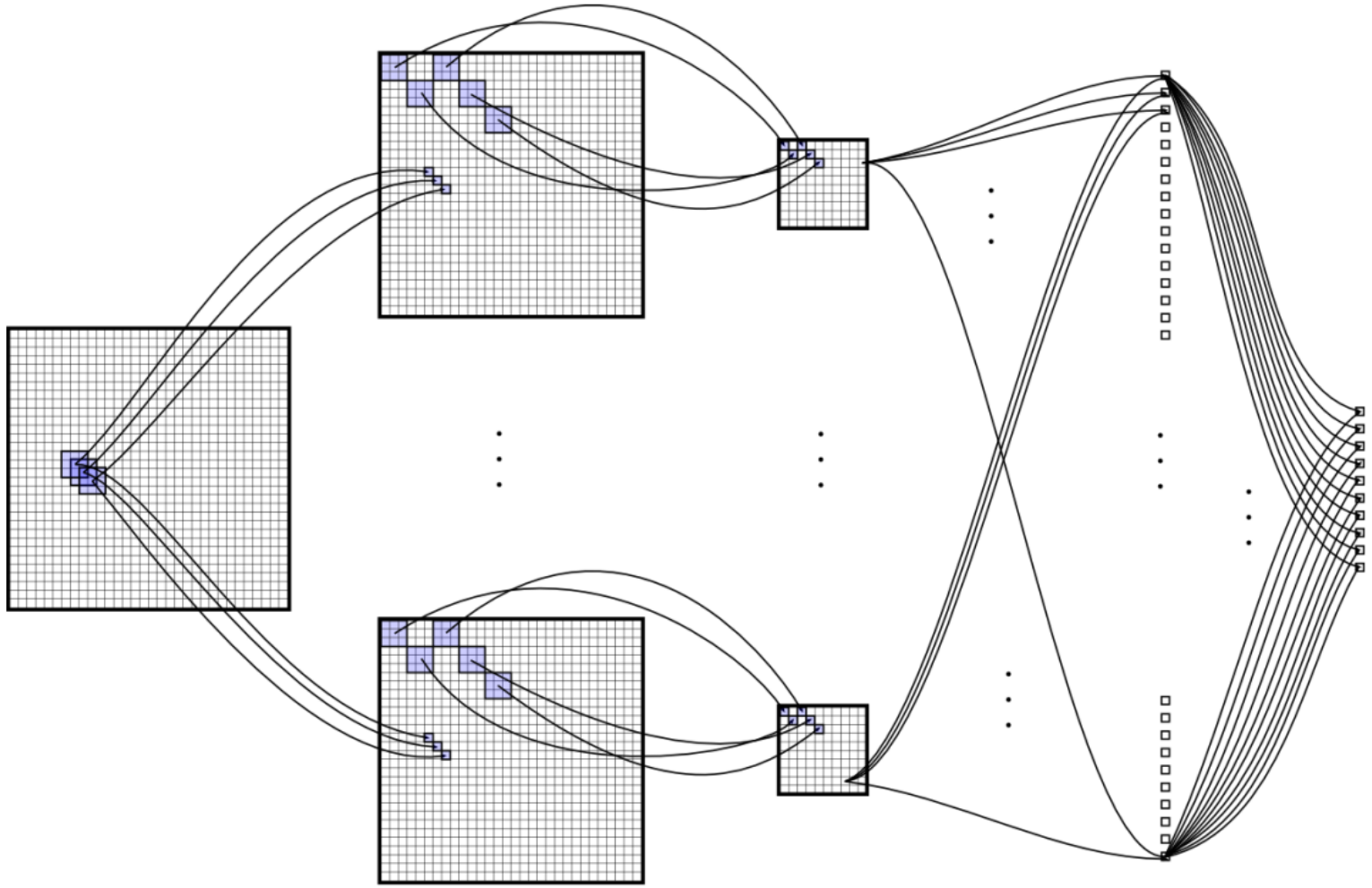
Convolutional Network Architecture

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

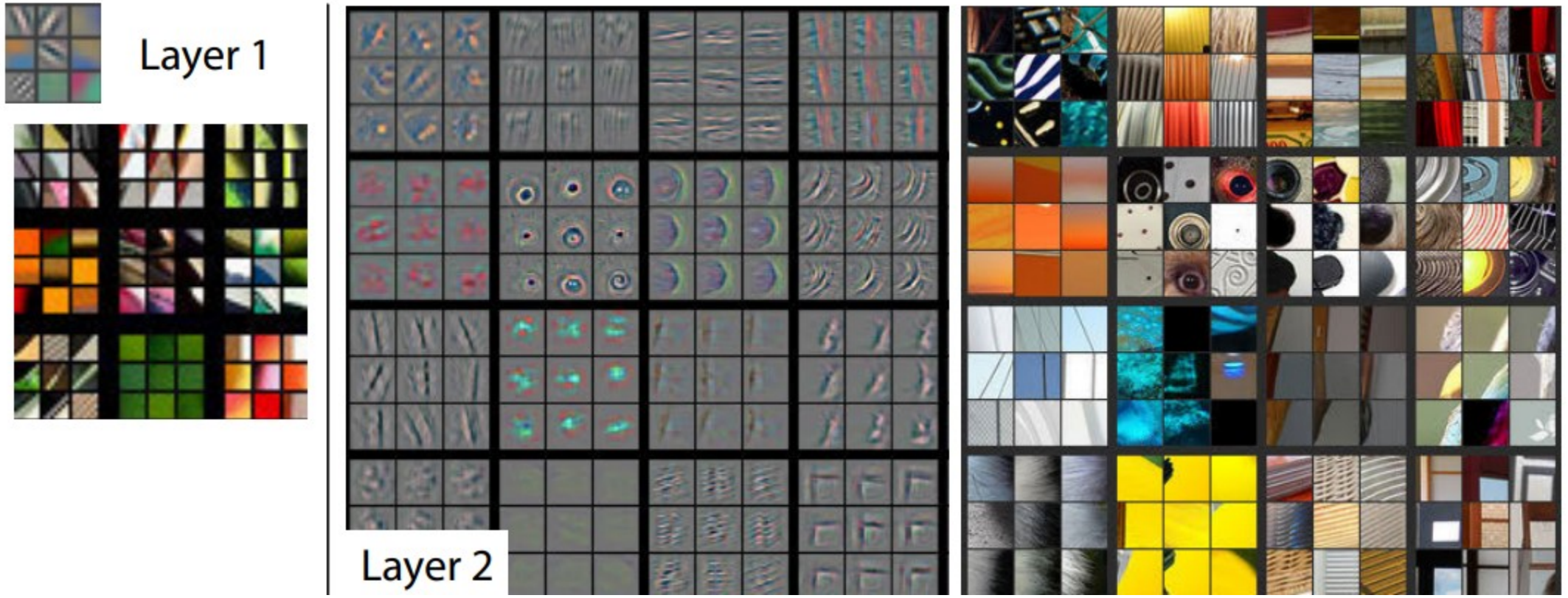


Convolutional Network Architecture

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

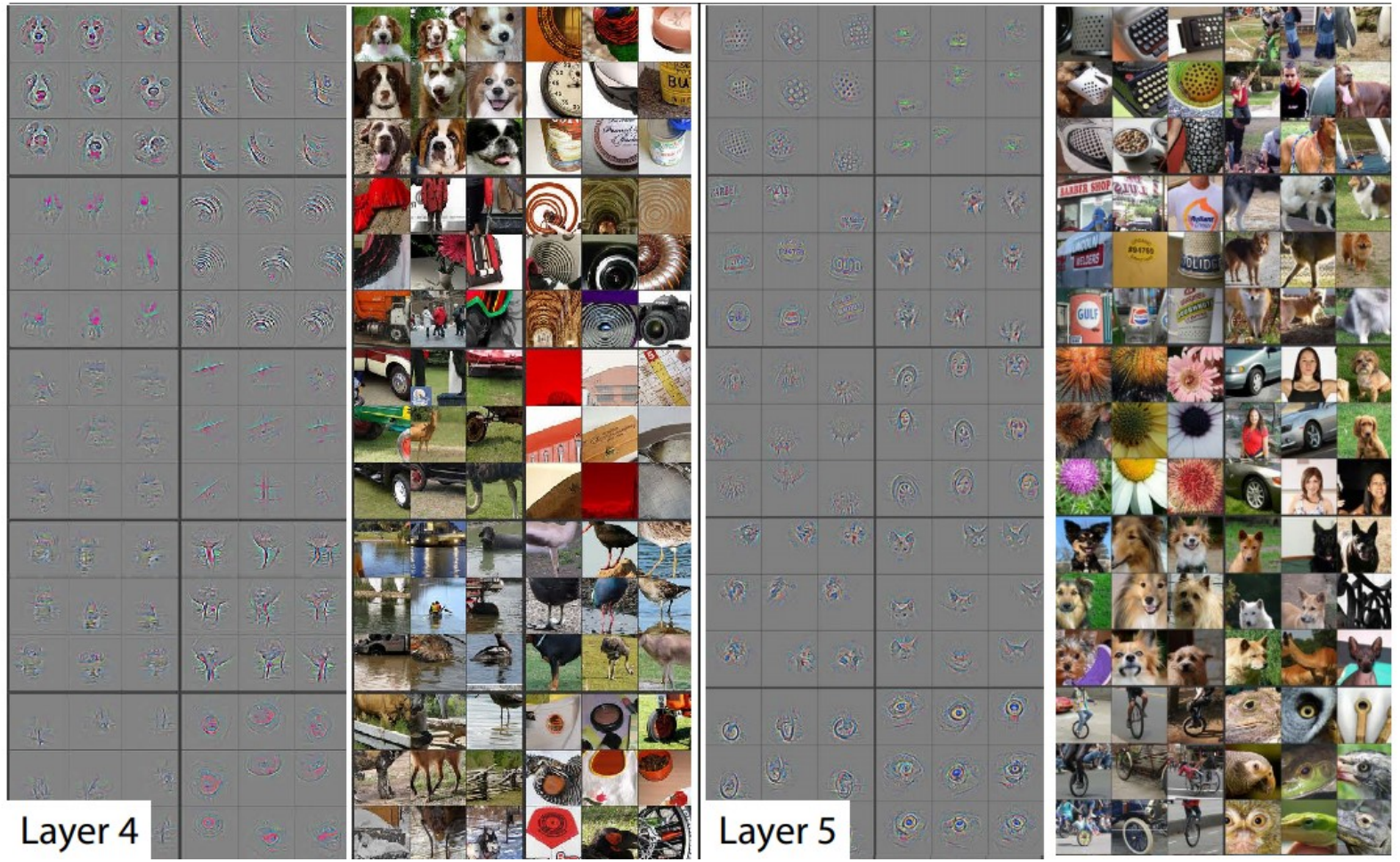


Visualizing learned filters



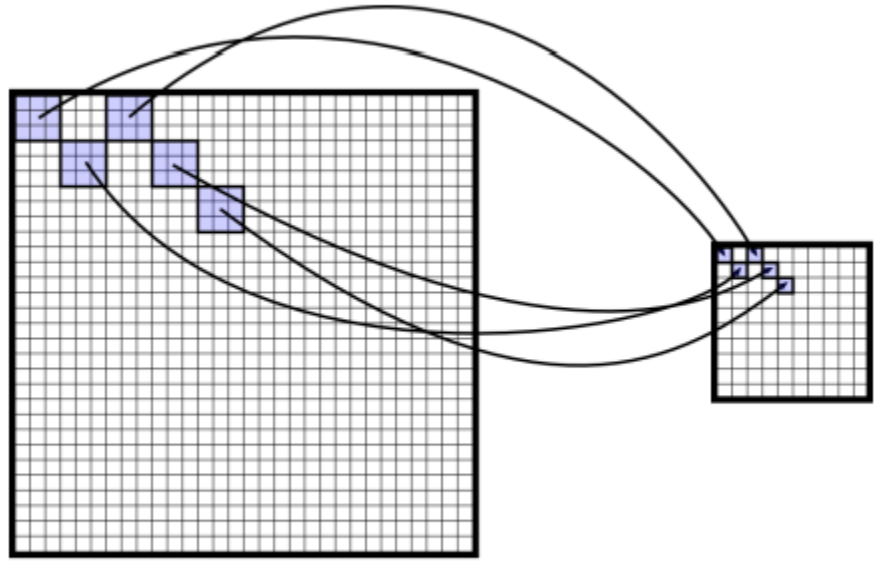
Learned filters on an image classification task (Imagenet), and highest activations.
From Zeiler and Fergus, Visualizing and Understanding Convolutional Networks, 2013

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?
- Recap fully connected layer:

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

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

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

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 channel (feature) *k,* for a single pixel $z_{a,b,k}$

$$z_{a,b,k} = f(u_{a,b,k}) \qquad \text{and} \qquad u_{a,b,k} = \sum_{i,j,c} w_{i,j,c}^{(k)} x_{i+a,j+b,c} + w_0^{(k)}$$

  where *c* is the source channel.
- Consequently, we have $\frac{\partial u_{a,b,k}}{\partial w_{i,j,c}^{(k)}} = x_{i+a,j+b,c}$

Training of Convolutional Networks

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

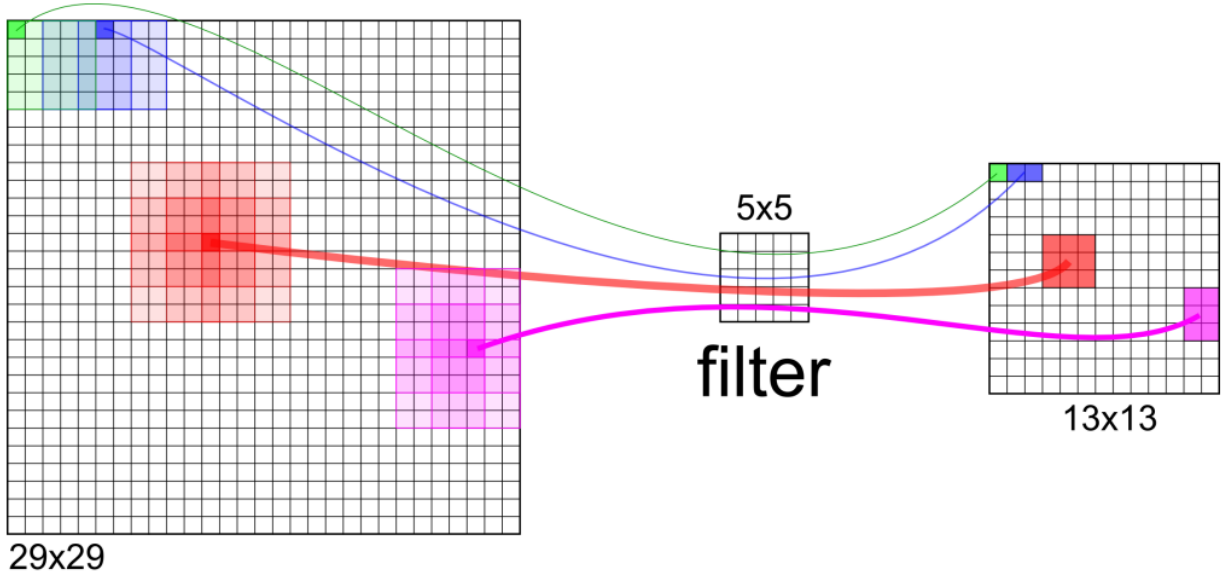
- For the convolutional layer, the situation is almost the same!
- With offset *a,b* and channel (feature) *k,* for a single pixel $z_{a,b,k}$

$$z_{a,b,k} = f(u_{a,b,k}) \quad \text{and} \quad u_{a,b,k} = \sum_{i,j,c} w_{i,j,c}^{(k)} x_{i+a,j+b,c} + w_0^{(k)}$$

  where *c* is the source channel.
- Consequently, we have $\frac{\partial u_{a,b,k}}{\partial w_{i,j,c}^{(k)}} = x_{i+a,j+b,c}$
- 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 $w^{(k)}_{i,j,c}$ come from the entire feature map.

Training of Convolutional Networks

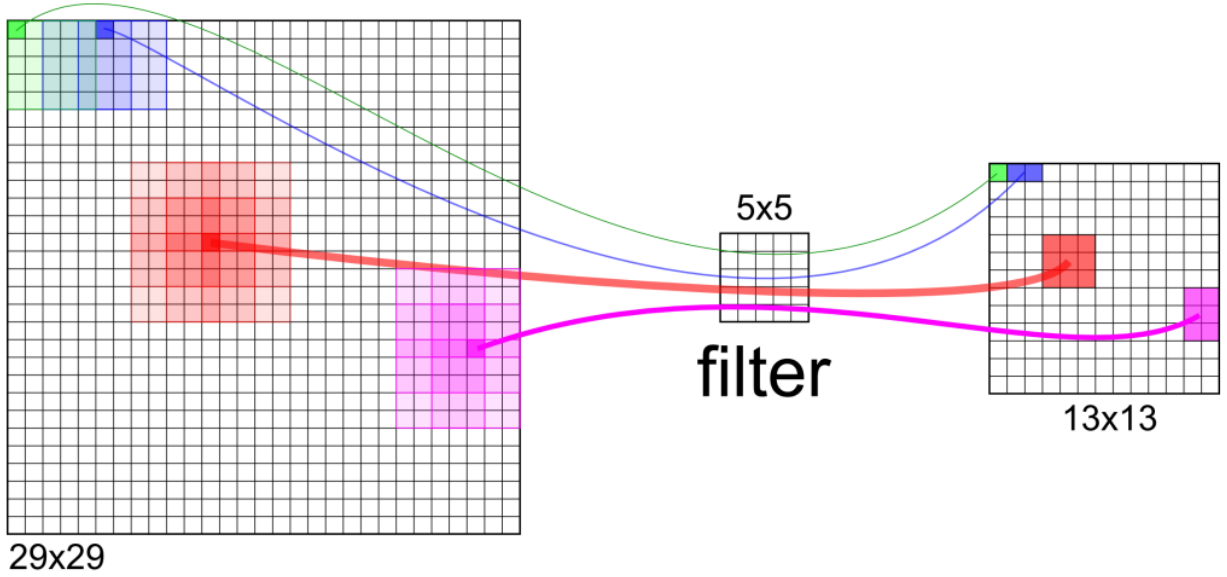
- Need to sum weight updates from all offsets. (Why?)



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

Training of Convolutional Networks

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

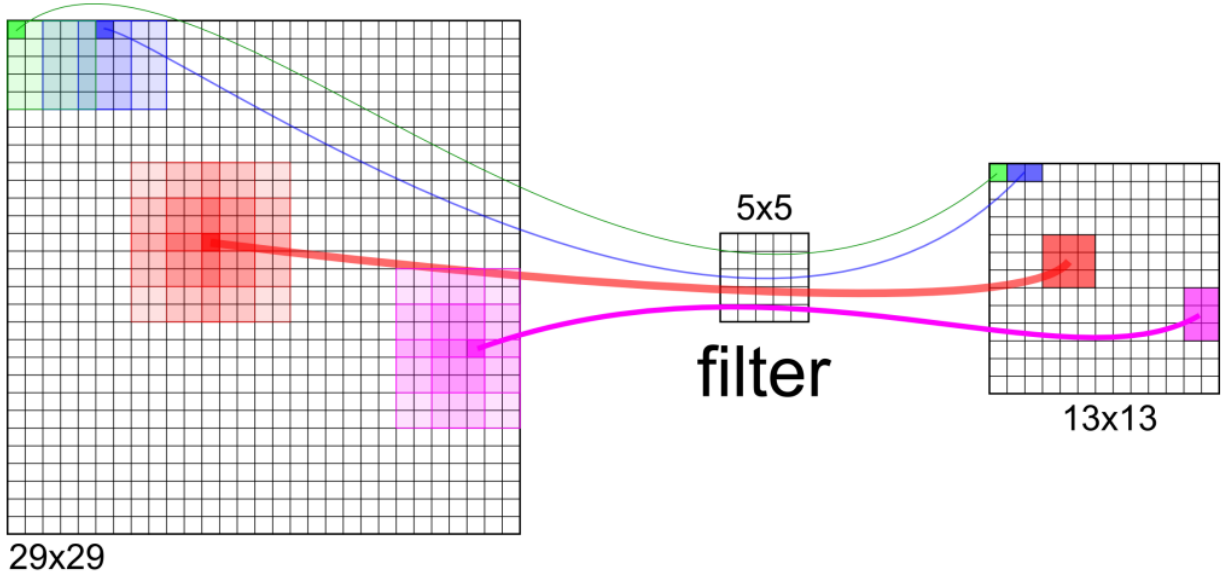


because we need to sum over all the "paths" in which a filter influences the output

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

Training of Convolutional Networks

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



because we need to sum over all the "paths" in which a filter influences the output

$$\frac{\partial \mathbf{Z}_{\cdot,\cdot,k}}{\partial w_{i,j,c}^{(k)}} = \sum_{a,b} \frac{\partial z_{a,b,k}}{\partial w_{i,j,c}^{(k)}}$$

- 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

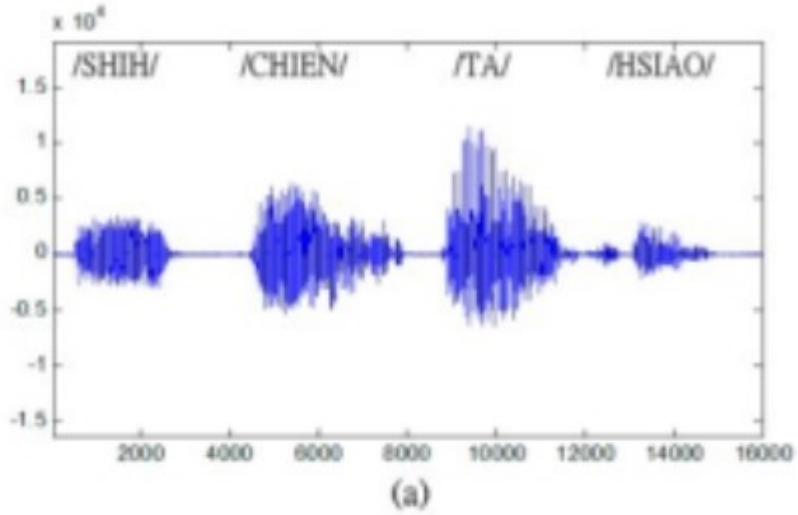
9HL-48x48-100C3-MP2-200C2-MP2-300C2-MP2-400C2-MP2-500N-3755		
Net for Chinese OCR	#weights	#connections
Feature extraction layer:	321'800	26.37 Million
Classification layer:	1'881'255	1.88 Million

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

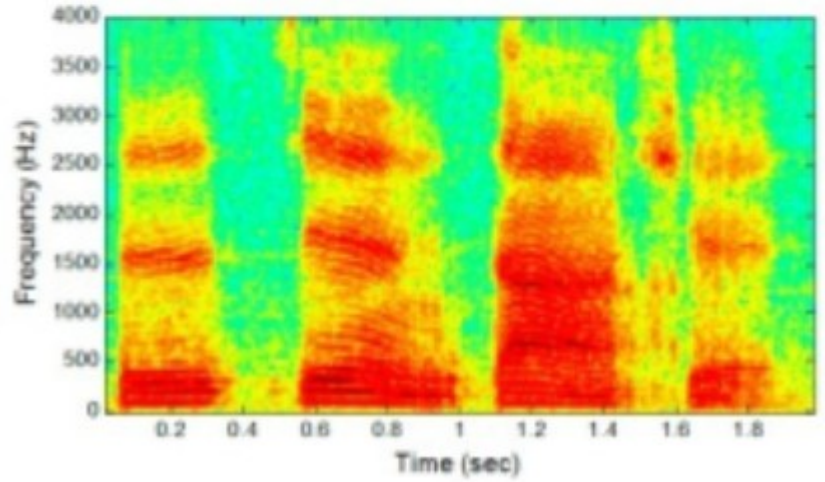
… or one week on a GPU (2016)!

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 ☺



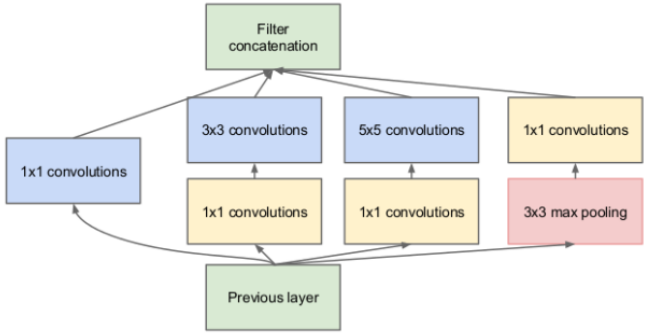
(a)



(c)

Final remarks

- 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
- *Inception modules* let the model decide on the size of the convolution kernel (Szegedy et al., *Going Deeper with Convolutions*, 2014)
- *ResNet* (He et al., *Deep Residual Learning for Image Recognition*, 2015) uses *skip connections* to allow deeper networks, most standard architecture nowadays



Inception block

ResNet34

Conclusion / Summary

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 are implemented in the convolutional case
- how you construct a neural network for an image classification task