UNIVERSITÄT BERN

# Convolutional Neural Networks

Paolo Favaro

# Contents

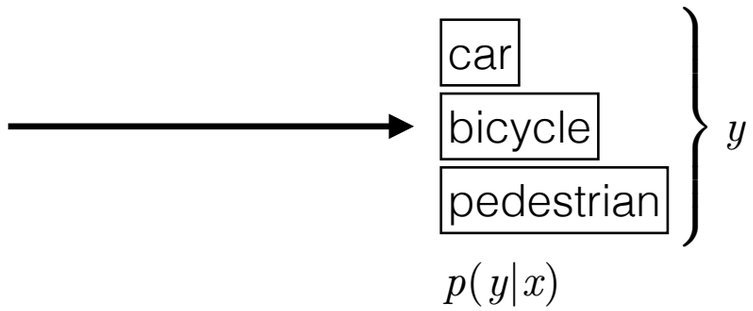
- Motivation and classic pattern recognition, Convolutional Neural Networks
  - Convolutions (standard, unshared, tiled), pooling, structured outputs
- Based on **Chapter 9** of Deep Learning by Goodfellow, Bengio, Courville
- Credits also to Yan LeCun “Learning Invariant Feature Hierarchies” presentation

Note

# Pattern Recognition

- The task is to classify an image as one of several possible objects

$x$

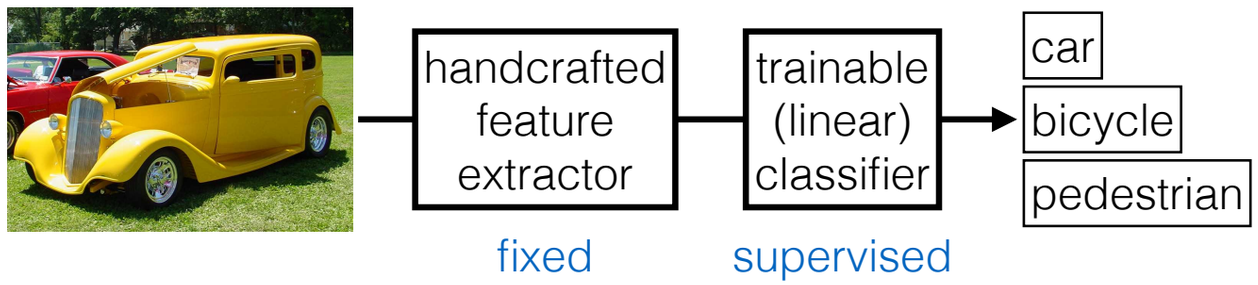



$p(y|x)$

Note

# Pattern Recognition

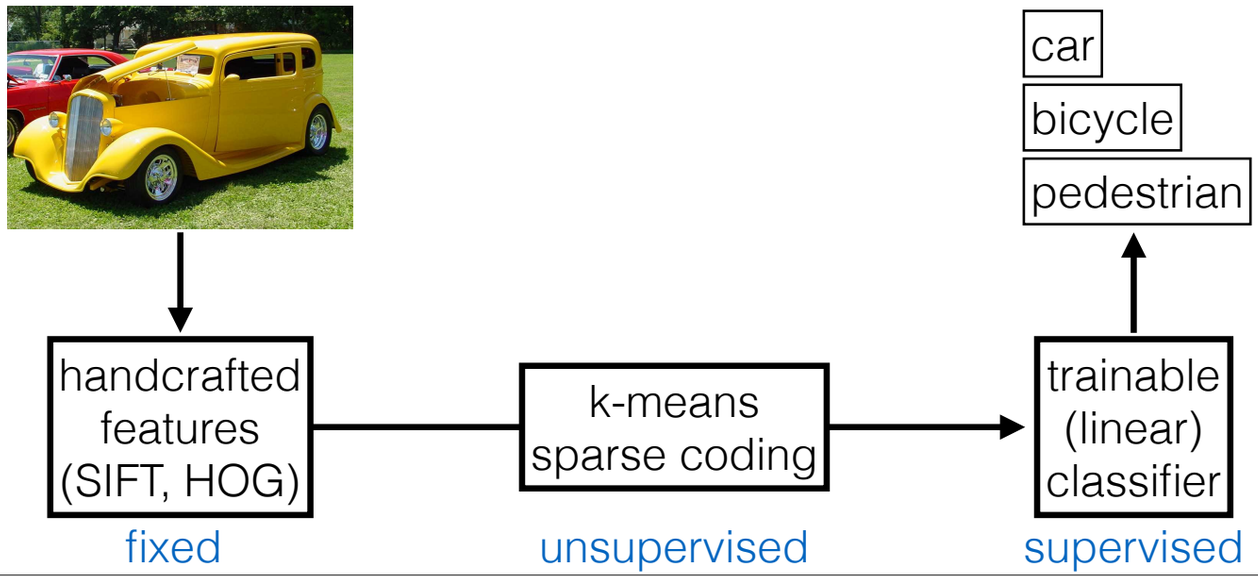
- The classic pipeline




Fixed handcrafted feature extractor

# Pattern Recognition

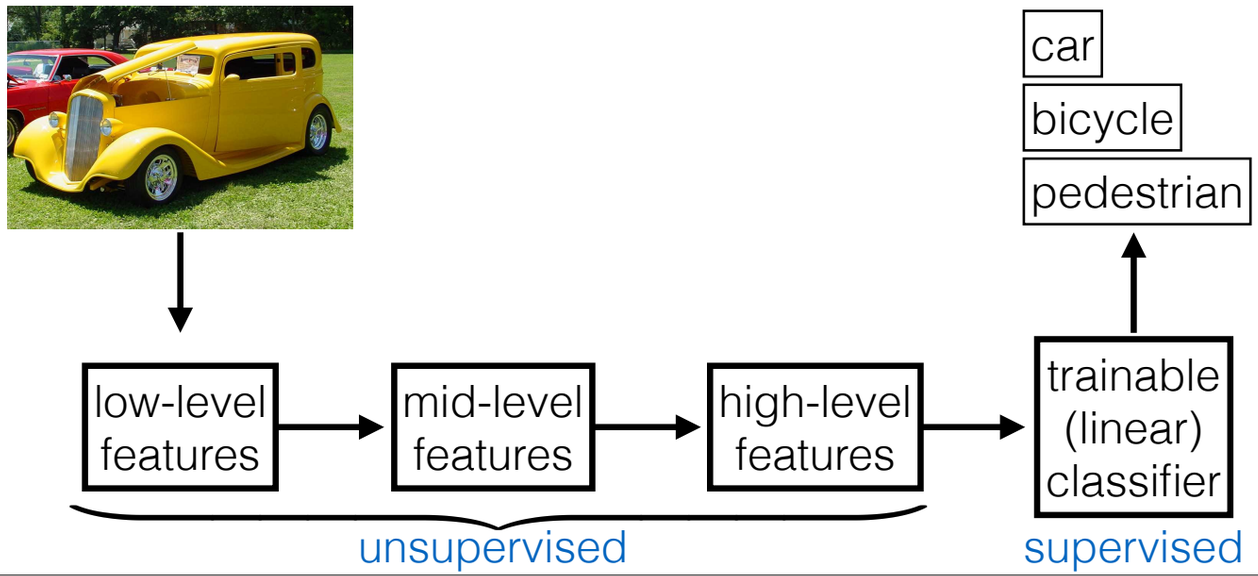
- The modern pipeline (2000s)




Handcrafted features are also called low-level features.
The k-means/sparse coding stage introduces mid-level features.
Unsupervised and mid-level features

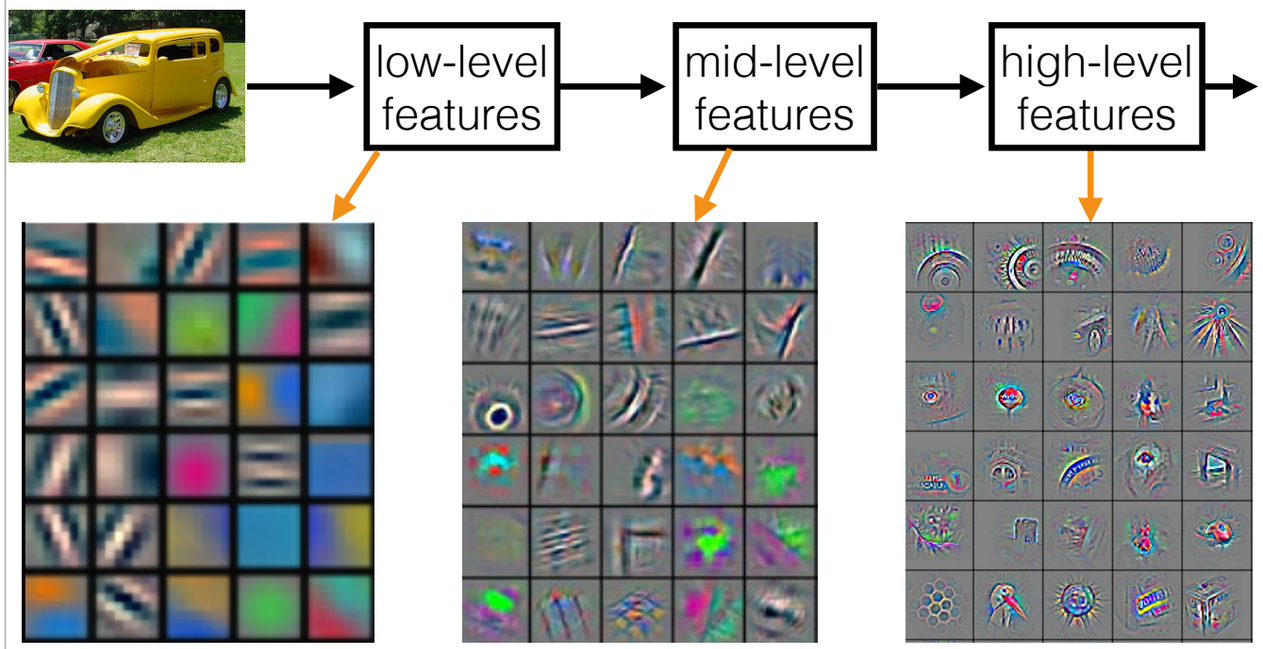
# Pattern Recognition

- Deep learning




Representations are hierarchical and trained

# Pattern Recognition




Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# Trainable Feature Hierarchy

- Hierarchy builds increasing level of abstraction, where each level is trainable
- Hierarchy examples
  - Image: Pixel → edge → texton → motif → part → object
  - Text: Character → word → word group → clause → sentence → story

# Learning Representations

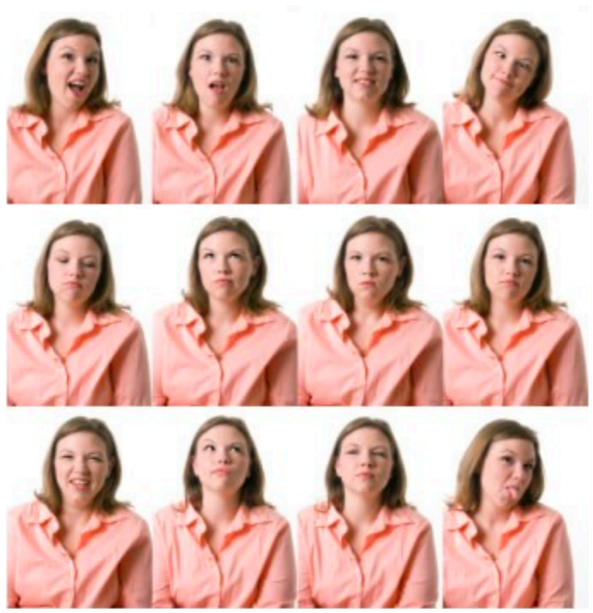
- How do we learn representations of the perceptual world?
- How can a perceptual system build **itself** by looking at the world?
- How much prior structure is necessary?

# Learning Representations

- **ML/AI**: How do we learn features or feature hierarchies? What is the fundamental principle? What is the learning algorithm? What is the architecture?
- **Neuroscience**: How does the cortex learn perception? Does the cortex “run” a single, general learning algorithm? (or a small number of them?)
- **CogSci**: How does the mind learn abstract concepts on top of less abstract ones?
- **Deep Learning** addresses the problem of learning hierarchical representations with a single algorithm (or perhaps with a few algorithms)

# What Are Good Features?

- Discover & disentangle the independent **explanatory factors**
- The manifold hypothesis
  - Natural data lives in a low-dim (non-linear) manifold, because variables in natural data are mutually dependent



The ideal feature extractor should map an image to a list of attributes such as: face/no-face, pose, lighting, expression etc.

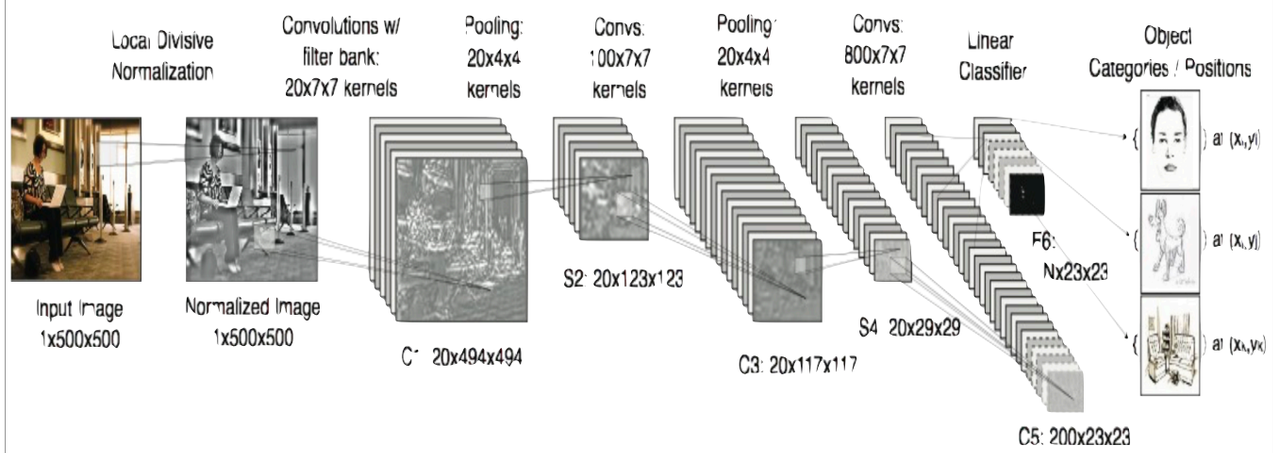
# Feature Learning Scheme

- Embed features in a high dimensional space (data points become separable)
  - Pattern matching
  - Nonlinearity
- Pool regions that have similarities

# Deep Learning Architecture

- Stack multiple stages of
  - **Normalization**: builds invariance to nuisance factors
  - **Filtering**: mapping to high-dim space
  - **NonLinearity**: separation of features
  - **Pooling**: aggregation by similarity

# The Convolutional Network

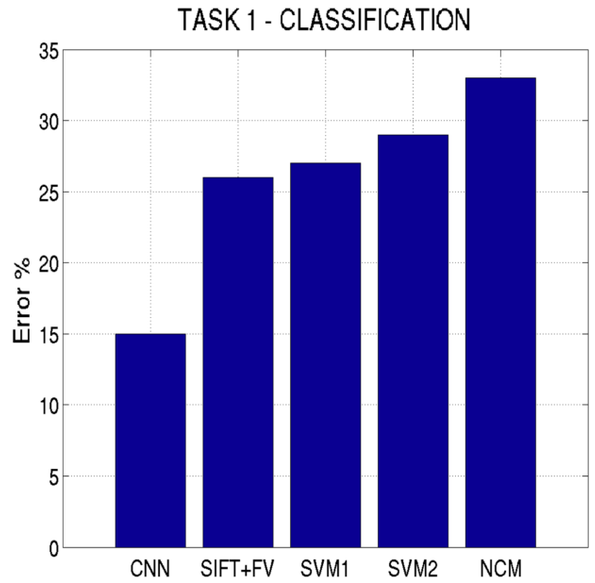


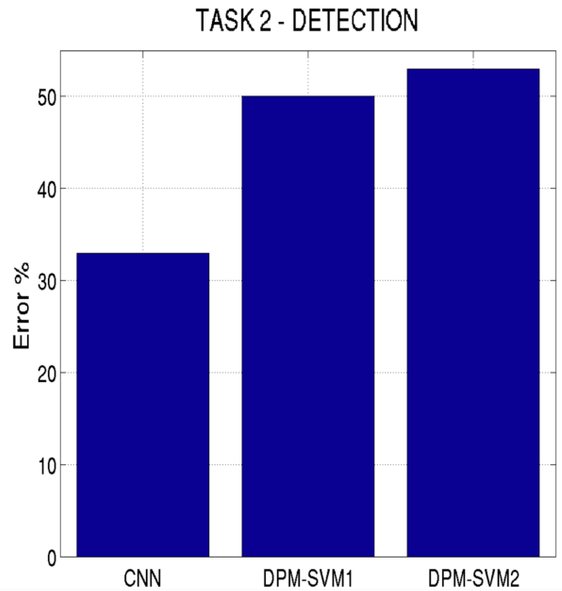
# Convolutional Networks (ConvNets)

- Deployed in many practical applications
  - Image/speech reco, Google's and Baidu's photo taggers
- Have won several competitions
  - ImageNet, Kaggle Facial Expression/Multimodal Learning, German Traffic Signs, etc
- Applicable to array data where nearby values are correlated
  - Images, sound, time-frequency representations, video, volumetric images, RGB-Depth images, etc

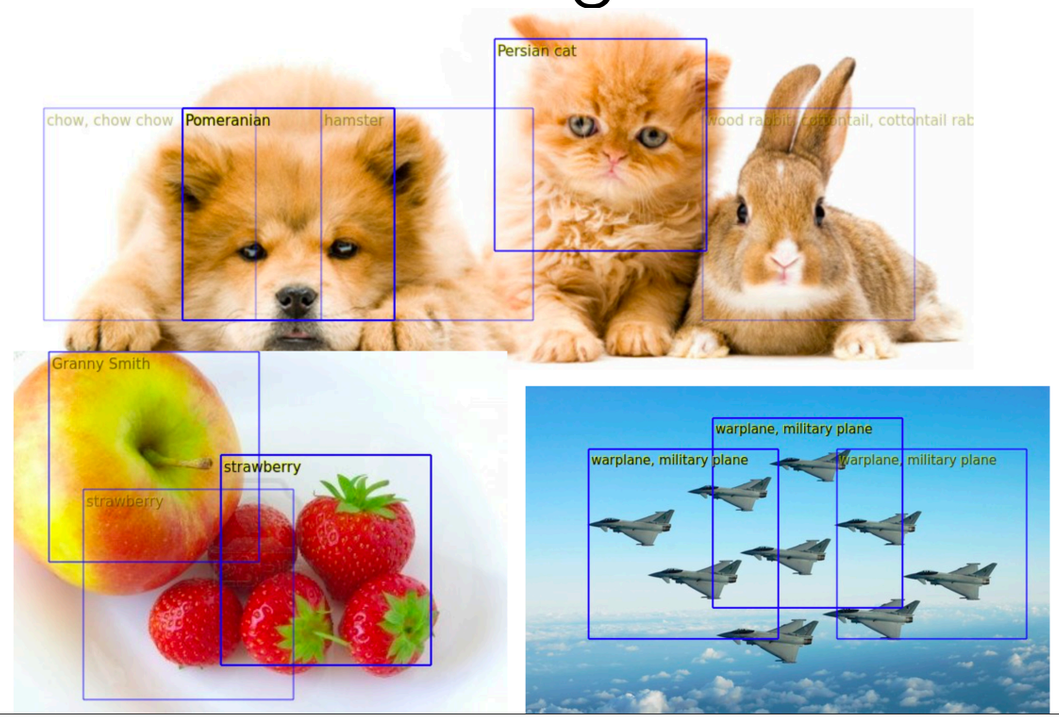
# Object Recognition with ConvNets

- AlexNet (Krizhevsky, Sutskever, Hinton 2012) won the 2012 ImageNet LSVRC (1K categories, 1.3M labeled training samples)

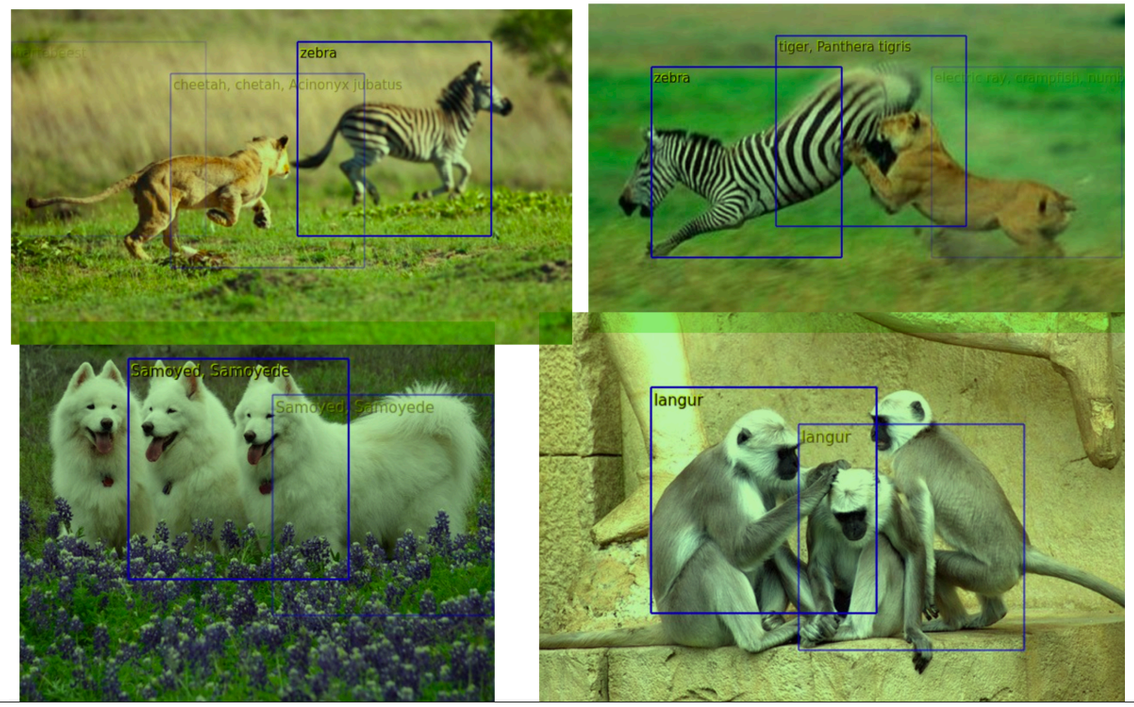


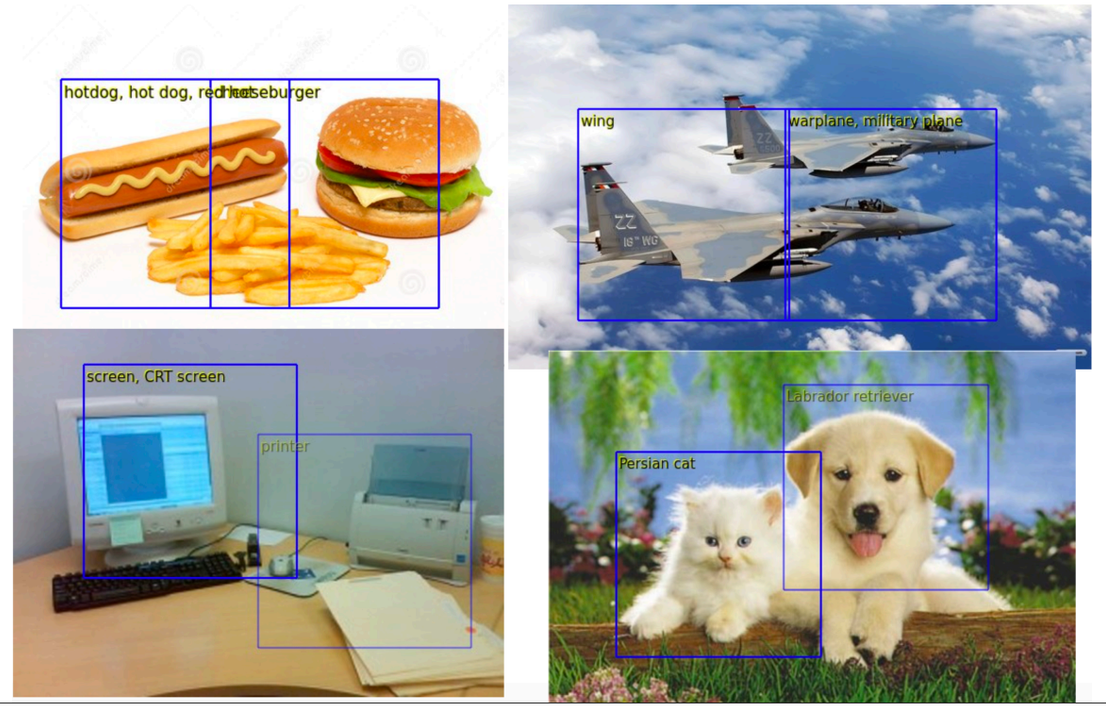
# Detection with a Sliding Window



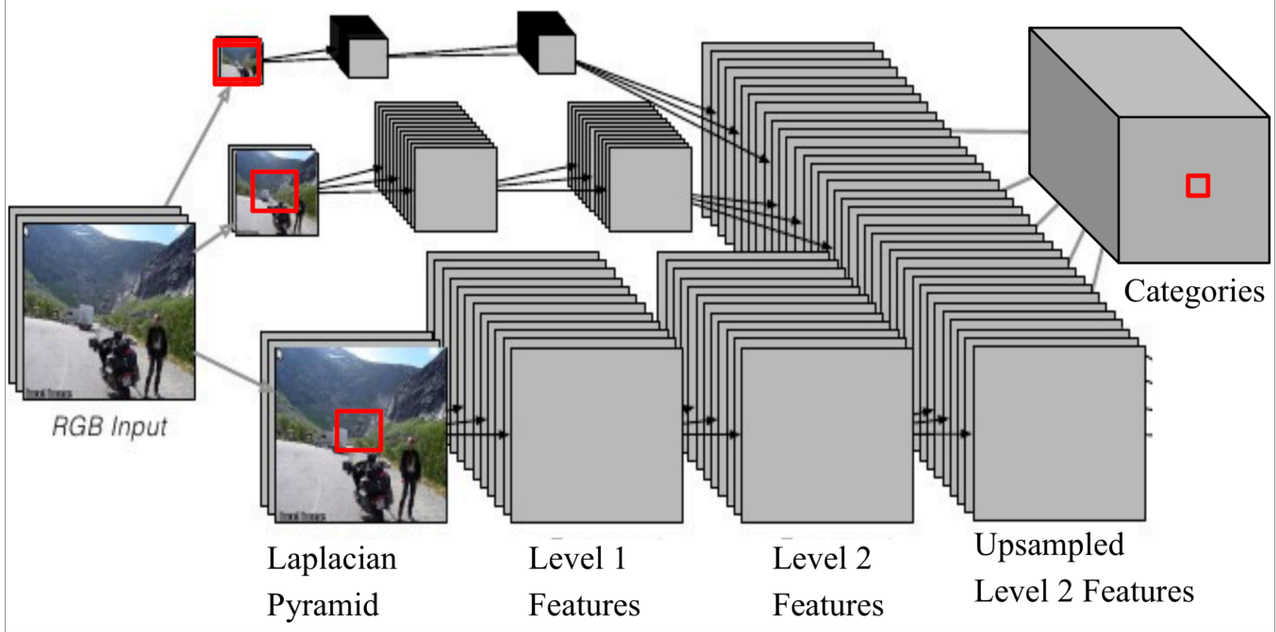
# Detection with a Sliding Window



# Detection with a Sliding Window

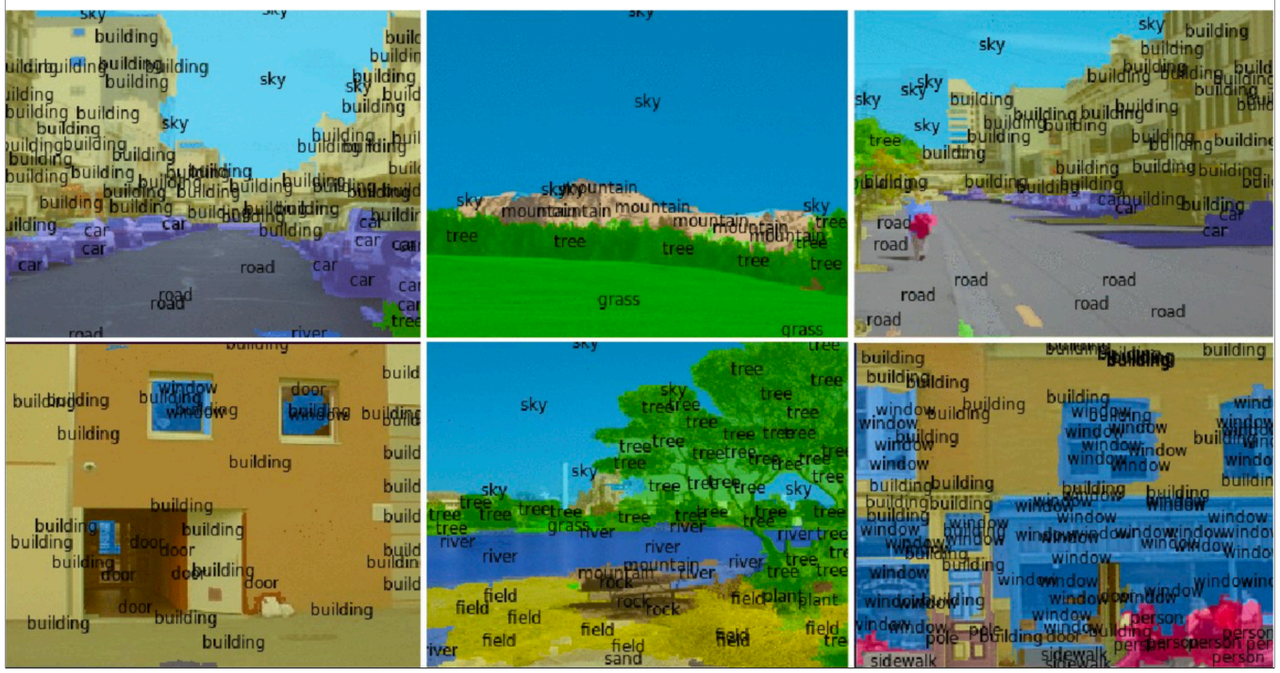


# Semantic Segmentation with ConvNets



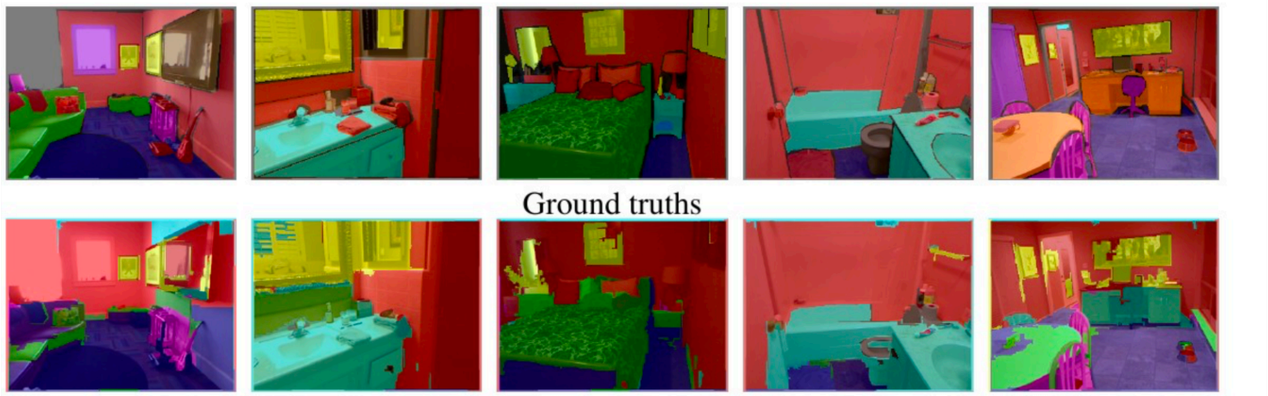

The Pyramid gives more context to each classification

# Semantic Segmentation with ConvNets



# Semantic Segmentation with ConvNets

# Semantic Segmentation with ConvNets




Scene parsing on RGB-depth images

# Commercial Applications with ConvNets

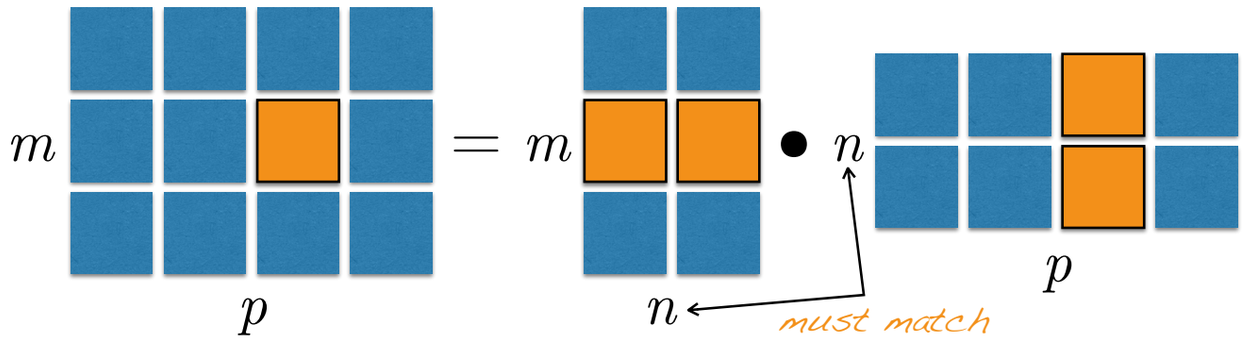
- Form Reading: AT&T 1994
- Check reading: AT&T 1996 (read 10-20% of all US checks in 2000) Handwriting recognition: Microsoft early 2000
- Face and person detection: NEC 2005
- Face and License Plate Detection: Google/StreetView 2009
- Gender and age recognition: NEC 2010 (vending machines)
- OCR in natural images: Google 2013 (StreetView house numbers) Photo tagging: Google 2013
- Image Search by Similarity: Baidu 2013
- Today: Lots of google services, etc

# Convolutional Networks

- A specialized neural network for data arranged on a grid (e.g., audio signals, images)
- Allow neural networks to deal with high-dimensional data
- Key idea is to substitute fully connected layers with a convolution

Note

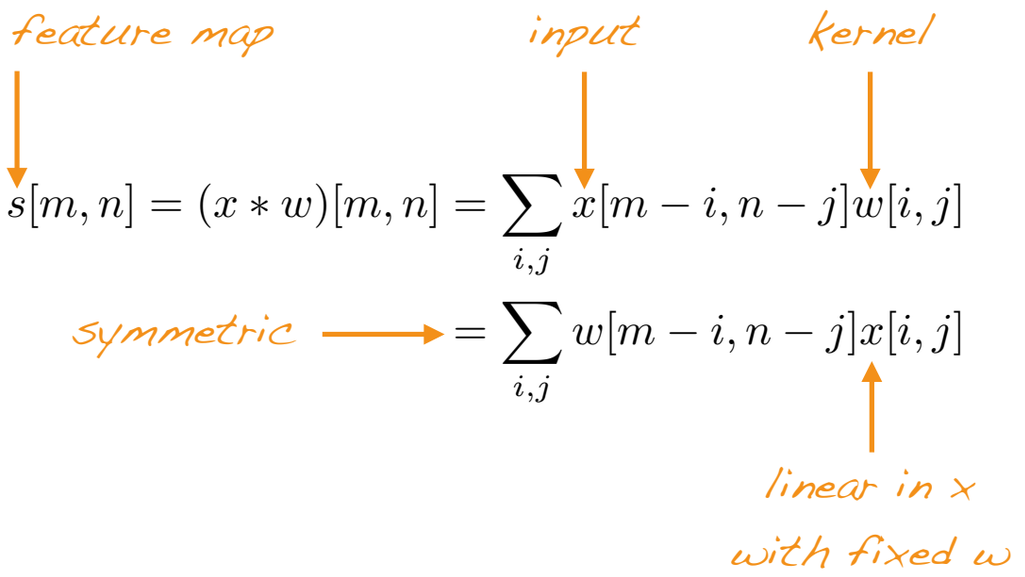
# Fully Connected Layers




**matrix product**

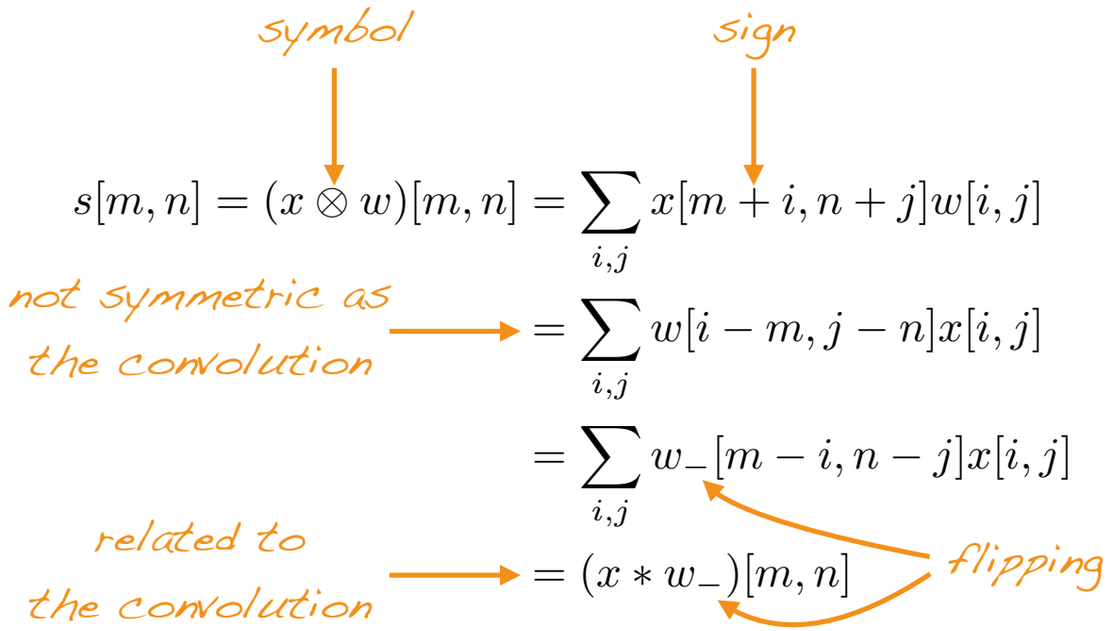
Let us first recall the fully connected layer. The main operation is a matrix product. Each entry of the output is an inner product between a row and a column (from the two matrices).

# The Convolution Operation



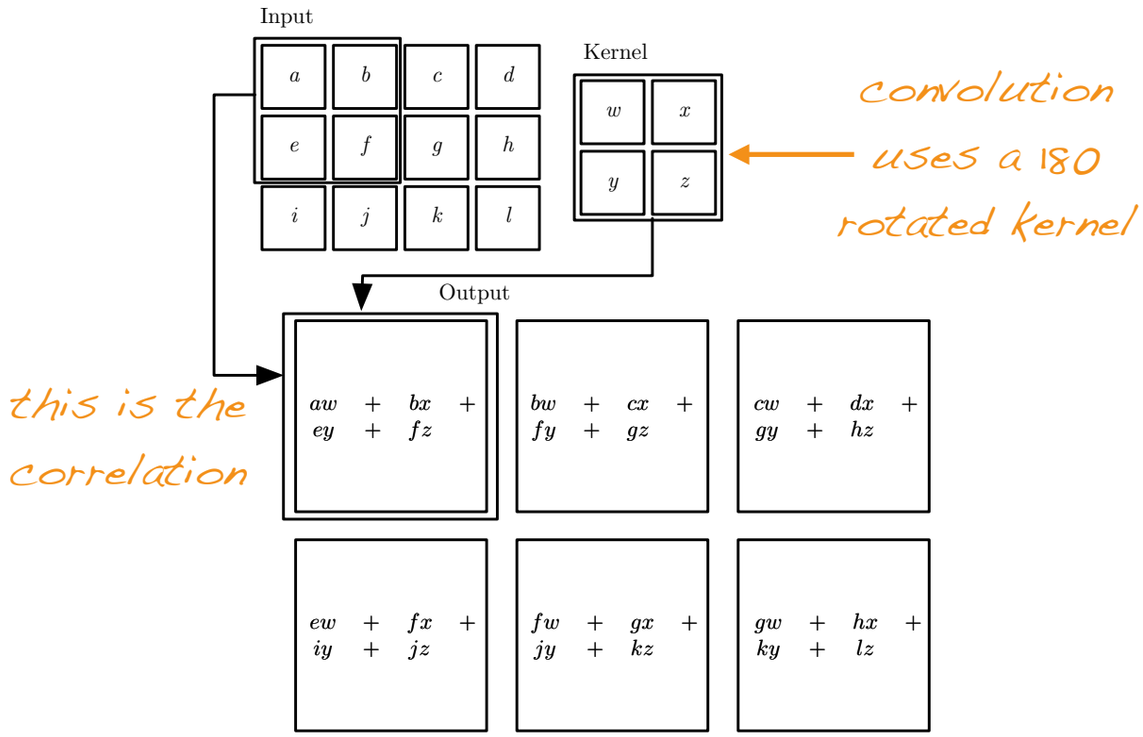

The convolution is an averaging operation. It satisfies several properties. One such property is that the input and the kernel can be interchanged. This can be easily shown with a change of variables.

# The Correlation Operation



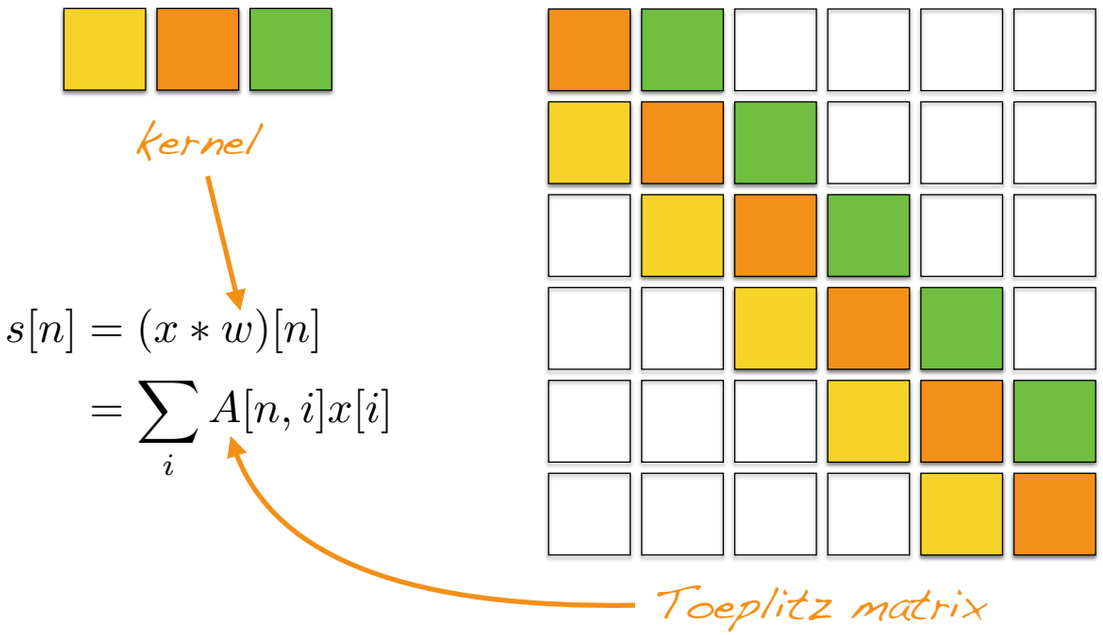

The correlation is a small variation over the convolution. Many libraries implement correlation but call it convolution!

# The Convolution Operation




A convolution instead uses the same kernel (1D, 2D, 3D, ...) and multiplies it to a matrix by sliding it. In the illustration the kernel is also flipped both along the x and y axes.

# Toeplitz Matrix




Gives the matrix-vector product version of the convolution.

# Motivation

- Convolutions leverage four ideas

  1. Sparse interaction

  2. Parameter sharing

  3. Equivariant representations

  4. It can equally handle inputs of different sizes

Note

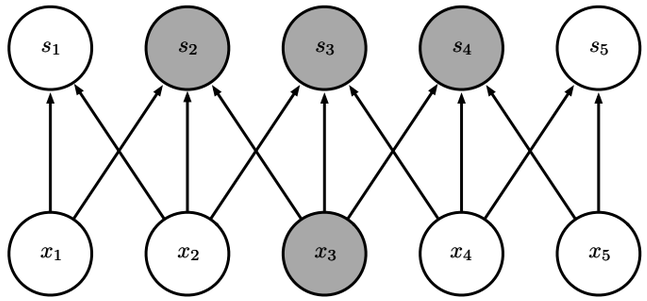
# Sparse Interaction

- In a fully connected layer every output can potentially depend on all inputs
- In a convolutional layer an output depends only on a small neighborhood of inputs (when the kernel is smaller than the input)
- The number of calculations is limited by the kernel size

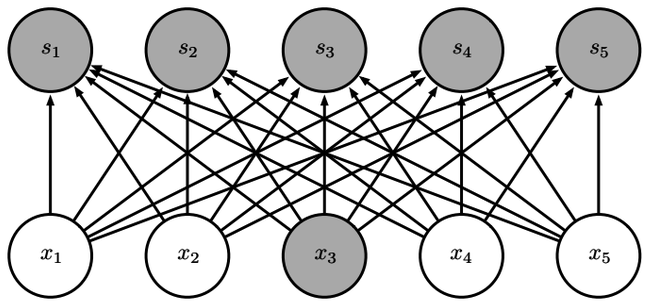
Note

# Sparse Interaction

Sparse connections due to small convolution kernel



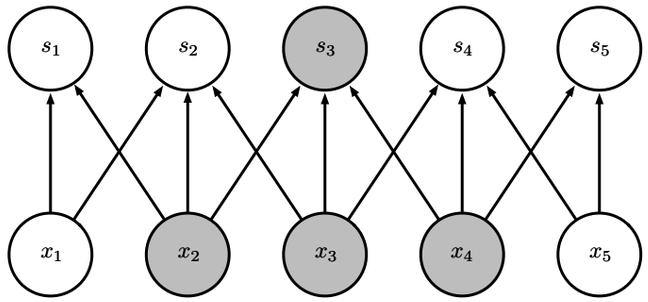

Dense connections



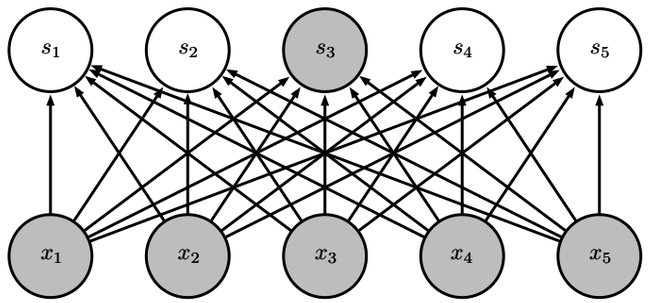

Here we see what entries a single input affects

# Sparse Interaction

Sparse connections due to small convolution kernel

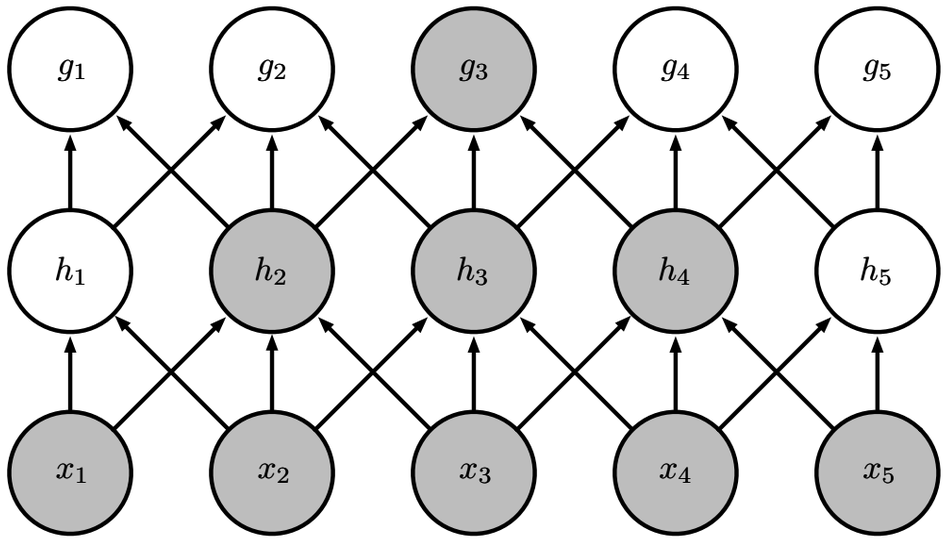



Dense connections




Here we see what inputs are combined to determine one intermediate output.

# Receptive Field




A **receptive field** is defined as all the units in a certain layer that affect a later unit

The receptive field of g3 on the first layer is of dimension 5. The receptive field of g3 on the second layer is of dimension 3.

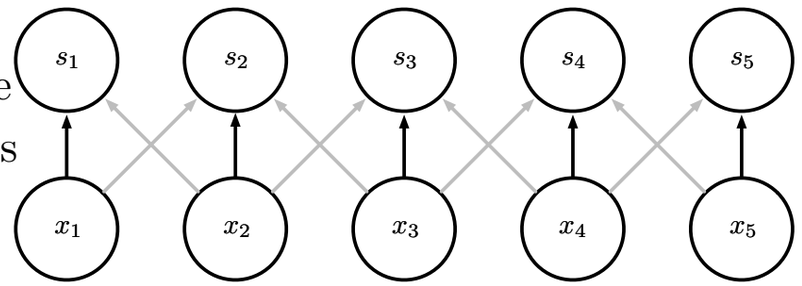
# Parameter Sharing

- Each output depends on the same parameters (the kernel weights)
- Exceptions are the boundaries (here padding and boundary assumptions are important)
- Fewer parameters means less storage

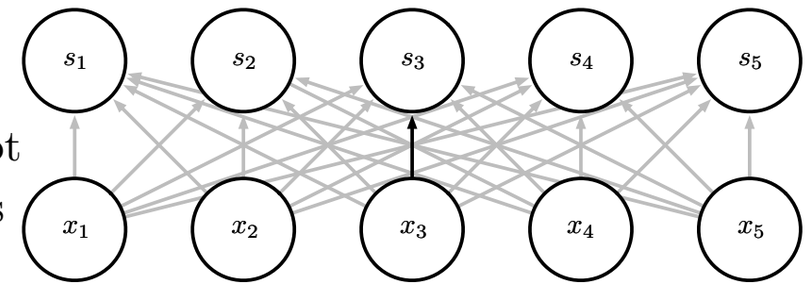
Note

# Parameter Sharing

Convolution shares the same parameters across all spatial locations



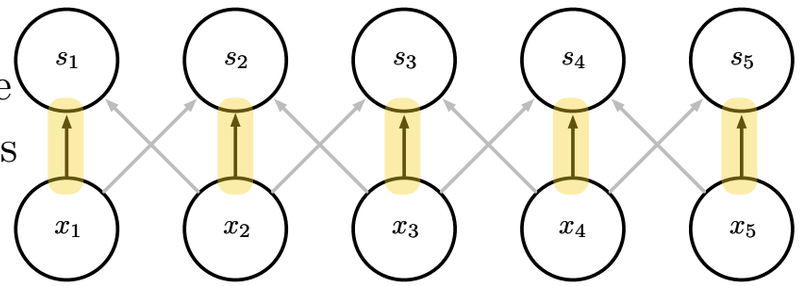

Traditional matrix multiplication does not share any parameters



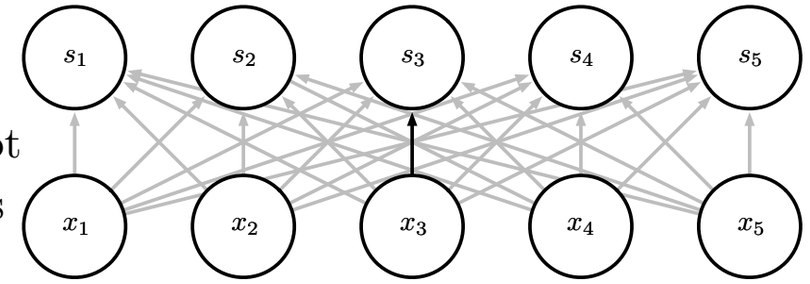

In yellow we see the spatial locations where all the parameters are the same (weights applied to the inputs). In contrast in red we have the spatial location in FC layers (different at each location).

# Parameter Sharing

Convolution shares the same parameters across all spatial locations

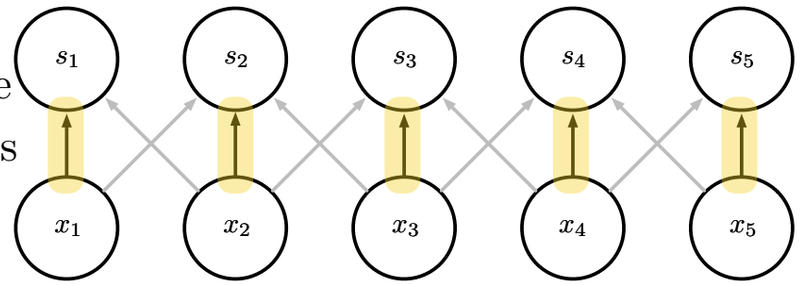



Traditional matrix multiplication does not share any parameters

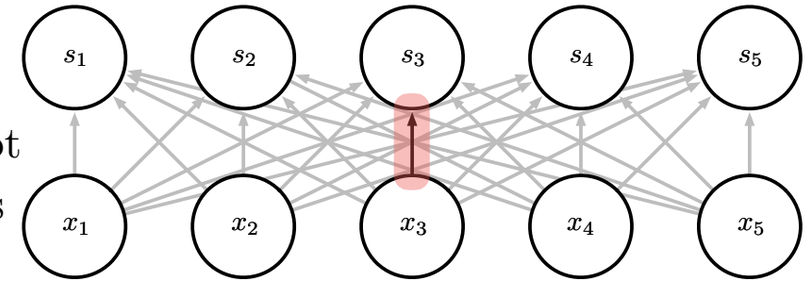


# Parameter Sharing

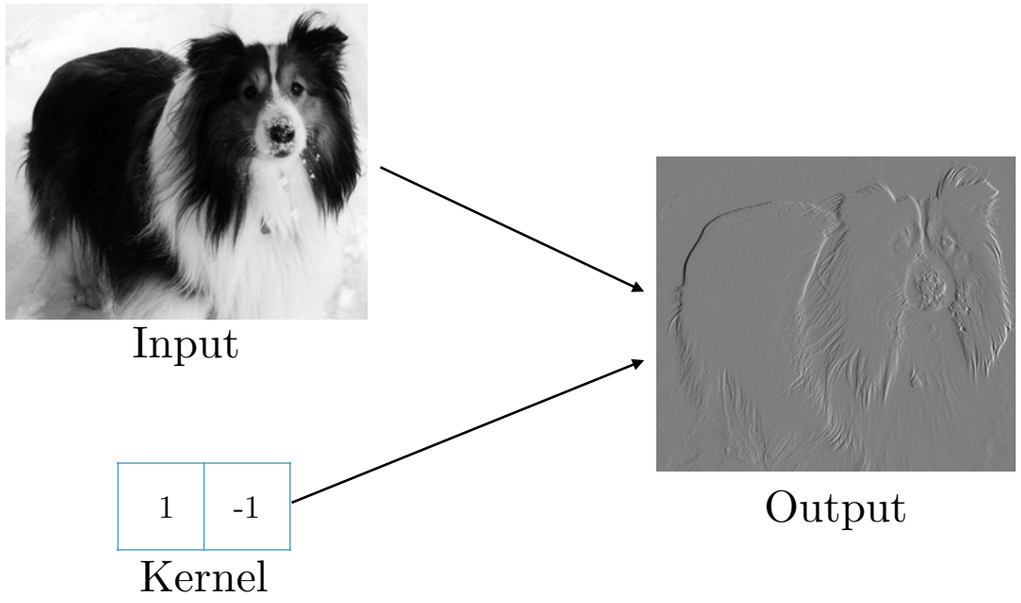
Convolution shares the same parameters across all spatial locations




Traditional matrix multiplication does not share any parameters



# Example




This gradient computation (to obtain edge information) is quickly and easily achieved via the convolutional layer with minimal computational effort. The same operation in a fully connected layer would have been quite challenging (computationally and due to the memory storage).

# Equivariant Representations

- An important property of the convolution is that it is shift-invariant (invariant to translation)
- In many signals this is the correct assumption: the same object might appear anywhere in the image and we should have the same (feature) response at any of these locations
- This concept is captured by **equivariance**

Note

# Equivariant Representations

- A function $f$ is **equivariant** to a function $g$ if $f(g(x)) = g(f(x))$
- If $f$ is the convolution and $g$ the translation, then $f$ is equivariant to $g$
- The convolution is not equivariant to rotation and scaling

Note

# Pooling

- Typical layers of convolutional neural networks consist of three stages: 1) a convolutional layer, 2) a nonlinear activation function (e.g., ReLU) and 3) a **pooling** function

- 1) and 2) are also called a **detector**

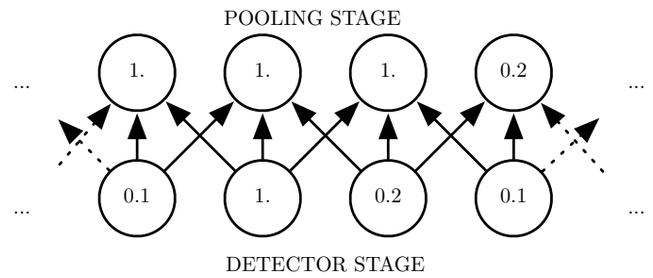
Note

# Pooling

- Pooling performs a calculation on a sliding window
- Some pooling operations: the **max** within the window, the **average** of the window, the **$L_2$ norm** of the window, the **weighted average** of the window
- Pooling gives a local (to a window) invariance to translation

To illustrate the local invariance, consider the max pooling and the response of a filter to a certain pattern. If the patter shifts within the window, the max pooling will always pick it and produce the same output on the same place.
This small invariance combined in a compositional manner leads to a useful alignment of detected patterns. This alignment makes it easier for the final classifier to take a decision (the complexity of the variation is small).

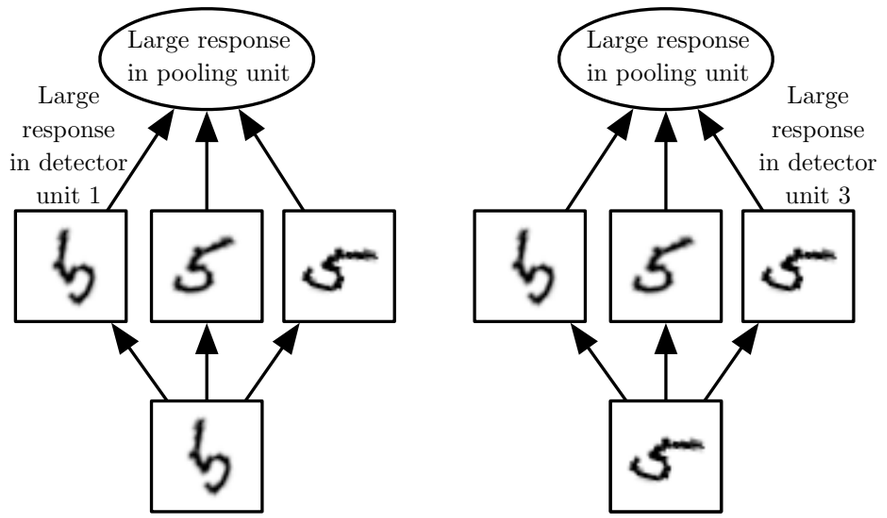
# Max Pooling




The top row in each plot shows the output of the max pooling (the previous layer is the output of the detector — convolution + nonlinear activation). At the bottom there is a shift of the values to the right. However, the max pooling output does not change much (especially when combined with a sampling stride matched to the pooling neighborhood).

# Learning Invariances

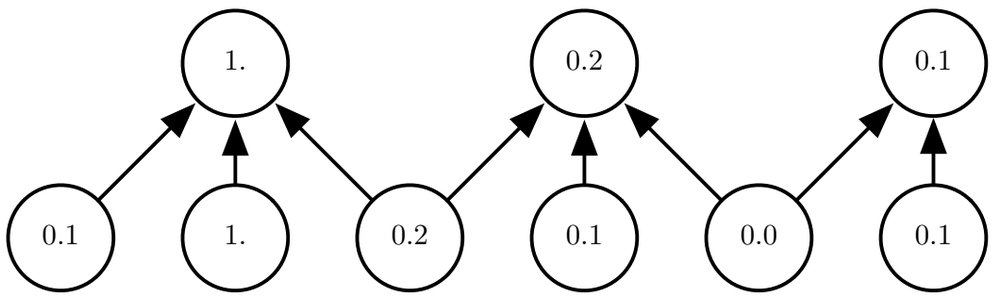
- Pooling applied across channels can learn invariances to transformations




In this example we consider max pooling. No matter which filter detects the digit 5, pooling can produce a “detected digit 5” output.

# Pooling

- Because pooling summarizes the responses over a neighborhood, it is sufficient to under sample the pooling output (e.g., by the size of the window)




Here the local shift invariance is more obvious (try shifting the lower layer to the right with a small new entry on the leftmost unit).

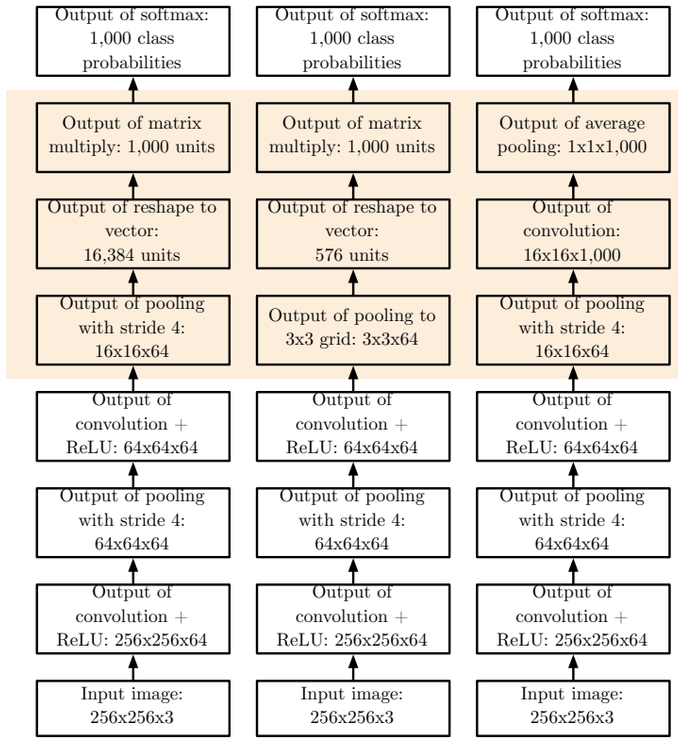
# Pooling

- In many tasks, this is useful to reduce the initial dimension of the input and eventually reach the desired output size

- With an adaptive neighborhood size it can produce the same output size regardless of the input size

For the last point, we can use as example the output of a pooling applied to the 4 quadrants of an image. The output has always 4 elements regardless of the input image.

# Classification Architectures




Some examples (not in use and not practical with real data) of classification networks that make use of different choices after the first 3 layers. The leftmost and rightmost networks depend on the input size. The middle network does not depend on the input size (if not trivially small). The rightmost network does not make use of fully connected layers. Also it associates each output to a channel.

# Variants

- Input data is typically a 4D tensor: 2 dimensions for the spatial domain, 1 dimension for the channels (e.g., colors), and 1 dimension for the batch
- The convolution (correlation) applies to the spatial domain only

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n}$$

output input kernel

Note

# Stride

- We can also skip outputs by defining a **stride** s larger than 1

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j\times s+m,k\times s+n} K_{i,l,m,n}$$

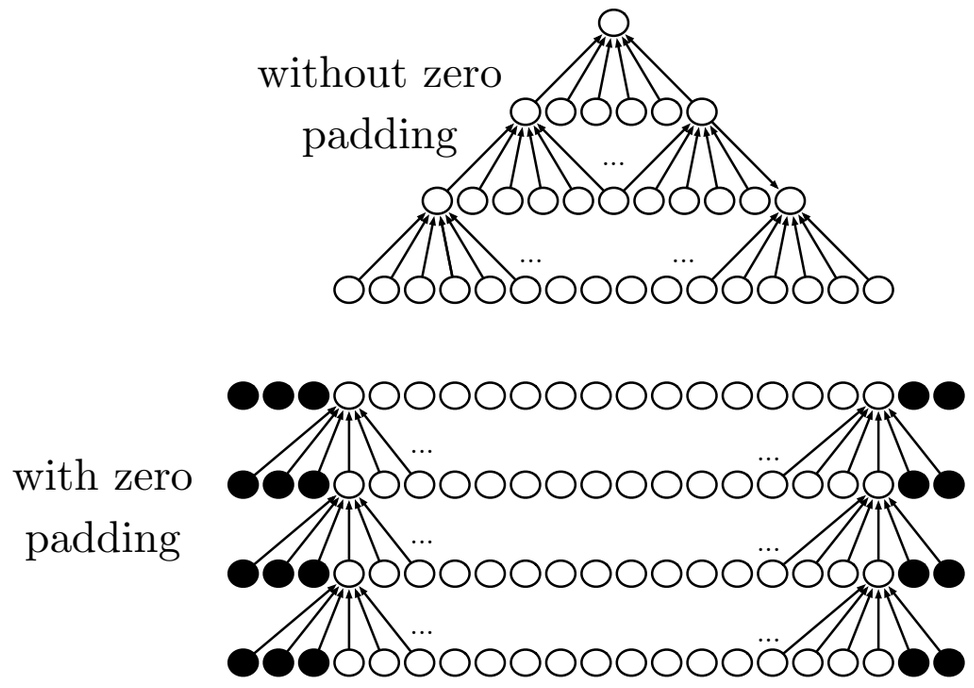
Note

# Padding

- The output of a convolution is valid as long as the summation uses available values
- In a convolution the valid output size is equal to:
  the input size - the size of the kernel + 1
- Unless we make boundary assumptions, a convolution will lead to a progressive shrinking of the input
- **Padding** is the assumption that outside the given domain the input takes some fixed values (e.g., zero)

Note

# Padding




Note

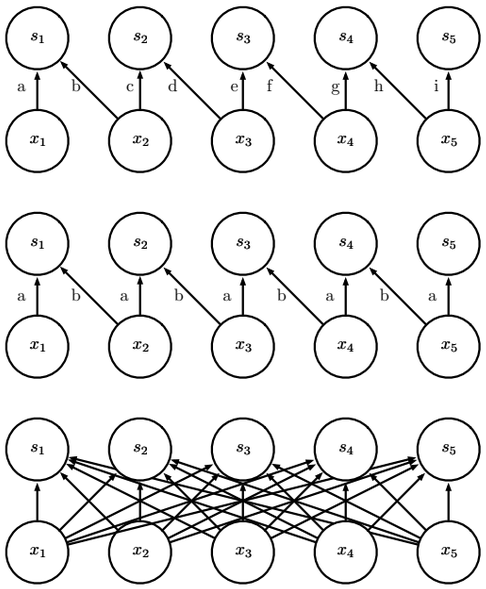
# Unshared Convolution

- Unshared convolutions use kernels whose weights change at every location, but only apply to a small neighborhood

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} w_{i,j,k,l,m,n}$$

Note

# Unshared Convolution




Note

# Tiled Convolution

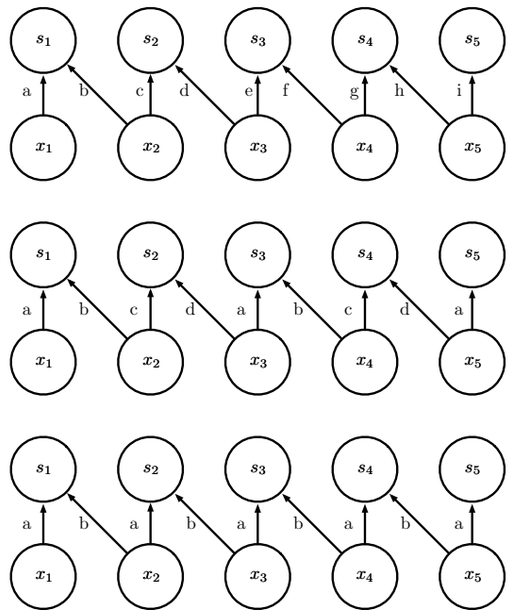
- A tradeoff between convolutions and locally connected layers
- The kernels repeat on a spatial lattice (locally they may be different)

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m,k+n} K_{i,l,m,n,j\%t,i\%t}$$

where $\%$ denotes the modulo operator

Note

# Tiled Convolution




local connection (no sharing)

tiled convolution (cycle between groups of shared parameters)

convolution (one group shared everywhere)

Note

# Structured Outputs

- Convolutional networks can output a high-dimensional structured object (e.g., a tensor)
- For example, one could output a 3D tensor with the probability of a given set of classes at each pixel
- This could be used for segmentation or pixel-wise labeling

Note

# Data Types

- Input data can be in different formats
- 1D: Audio waveforms (single channel) and skeleton animation data/motion (multi-channel)
- 2D: Audio data preprocessed via Fourier (single channel), color image data (multi-channel)
- 3D: Volumetric data such as CT scans (single channel), color video data (multi-channel)

Note

# Data Types

- Convolutional networks allow dealing with images of different sizes
- Also, we might design networks whose output is size-varying (the loss function must be able to handle it) — e.g., per pixel labeling

Note

# Efficient Implementations

- Convolutions can be efficiently implemented via parallel computing (GPU)
- One fast implementation (when kernels are large) is via the FFT (the convolution is a dot product in Fourier space)
- Another fast implementation is when a d-dimensional kernel is separable (it can be written as the outer product of d 1D kernels)

Note