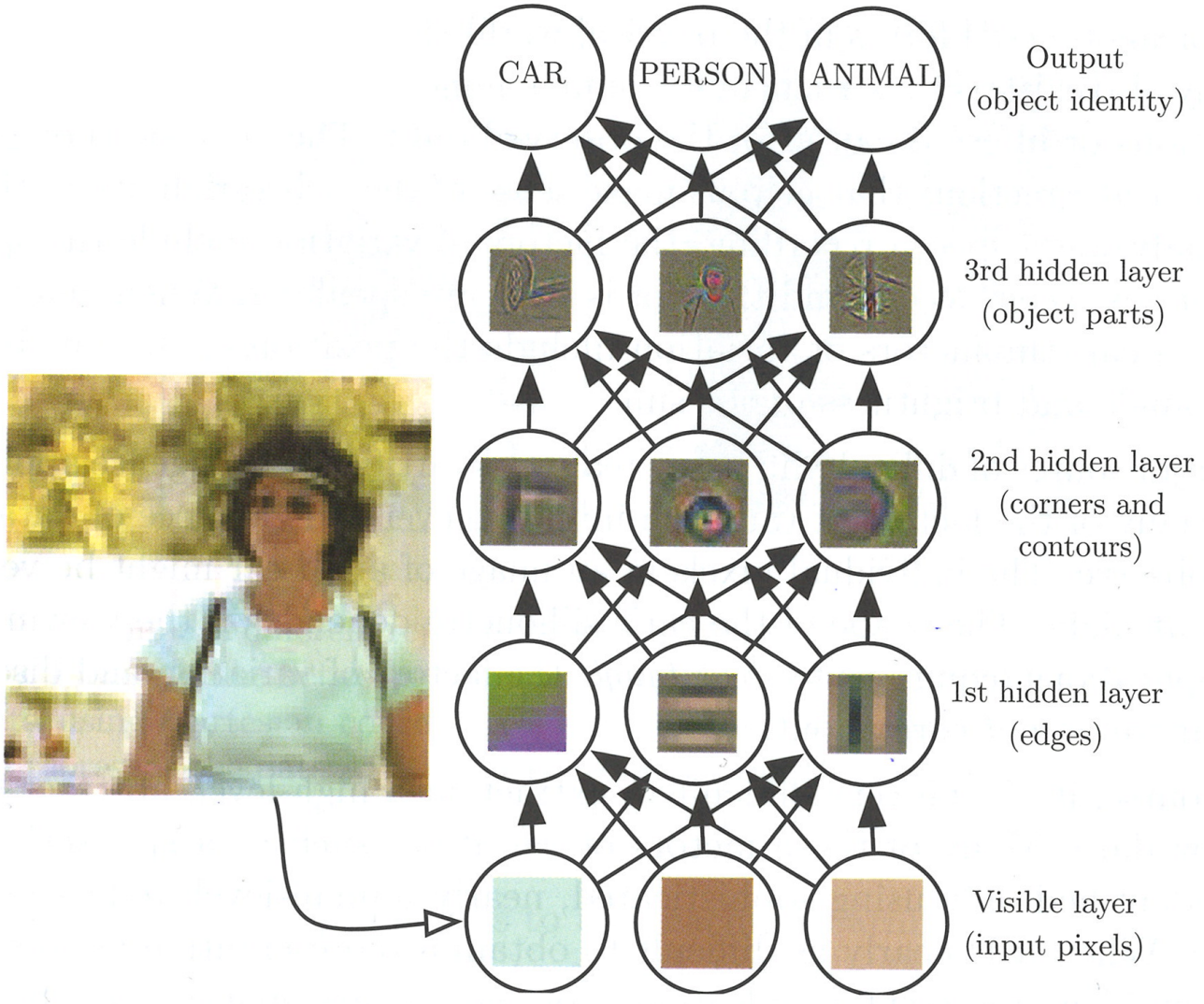
# Advanced course in machine learning
# 582744
# Lecture 11

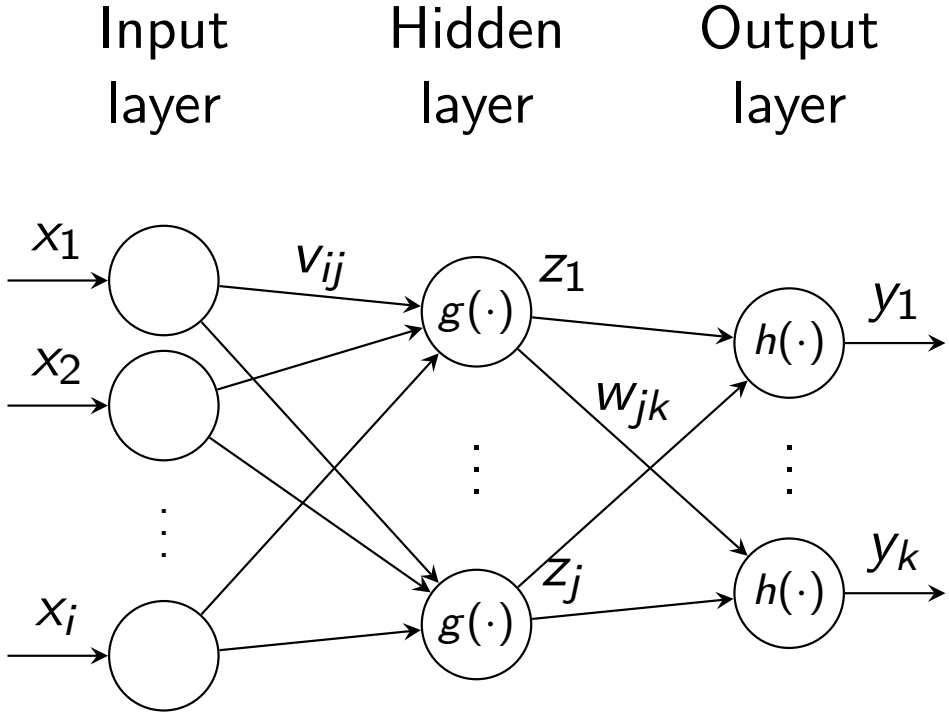
Mats Sjöberg

# Previous lecture: deep learning




http://www.deeplearningbook.org/

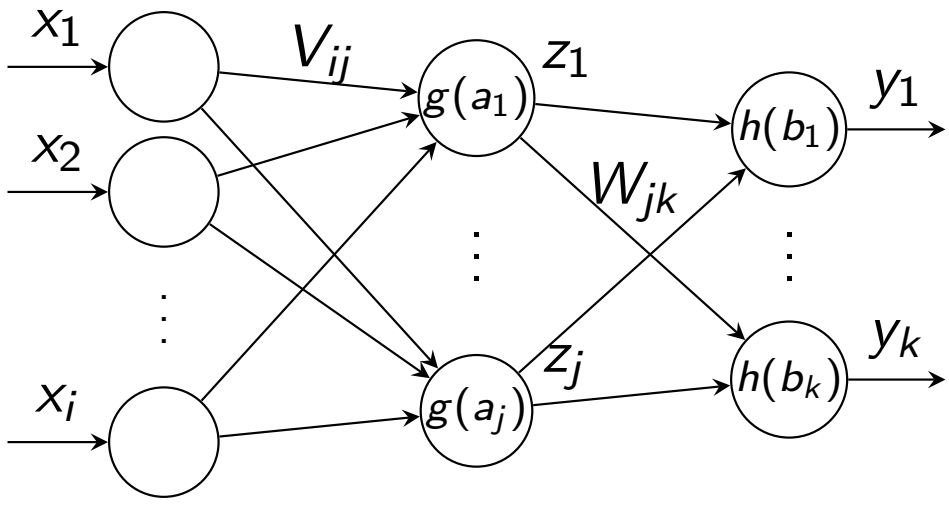
# Previous lecture: multilayer perceptron (MLP)




$$\mathbf{z} = g(\mathbf{V}\mathbf{x}) \qquad \mathbf{y} = h(\mathbf{W}\mathbf{z})$$

$$\mathbf{y} = h(\mathbf{W}g(\mathbf{V}\mathbf{x}))$$

# Previous lecture: backpropagation




$$\frac{\partial L}{\partial W_{kj}} = \delta_k z_j$$

$$\frac{\partial L}{\partial V_{ji}} = \delta_j^{(z)} x_i$$

where

$$\delta_j^{(z)} = \sum_k \delta_k W_{kj} g'(a_j)$$

# Previous lecture: regularization of MLPs

MLPs are flexible models and hence prone to overfitting

Regularization by

- Early stopping – don't let the model fit too well to the training set
- Weight decay – $L_2$ regularization on the weights
- **Weight sharing – reduce number of parameters by forcing some nodes to use the same weights**
- Dataset augmentation, addition of noise
- Dropout - removing non-output nodes with some probability $p$

# Example: image recognition

Instead of fully connected MLPs we typically want to encode knowledge about the problem domain into the network

One standard example is image recognition, where we know the input vector is actually a two-dimensional array where neighboring pixels are related

The standard model for this is the *convolutional neural network*, which means any network that uses convolutions as part of the model

# Convolution

Convolution is the integral of the product of two functions where one is reversed and shifted:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Discrete convolution is defined as

$$(f * g)(t) = \sum_{m} f(t-m)g(m),$$

where $g(\cdot)$ typically is non-zero for a finite area. Thus the convolution can be interpreted as a weighted average of $f(t)$ in a finite surrounding.

In convolutional neural networks $f(\cdot)$ is sometimes referred to as the **input**, $g(\cdot)$ as the **kernel**, and the result the **feature map**.

# Convolution

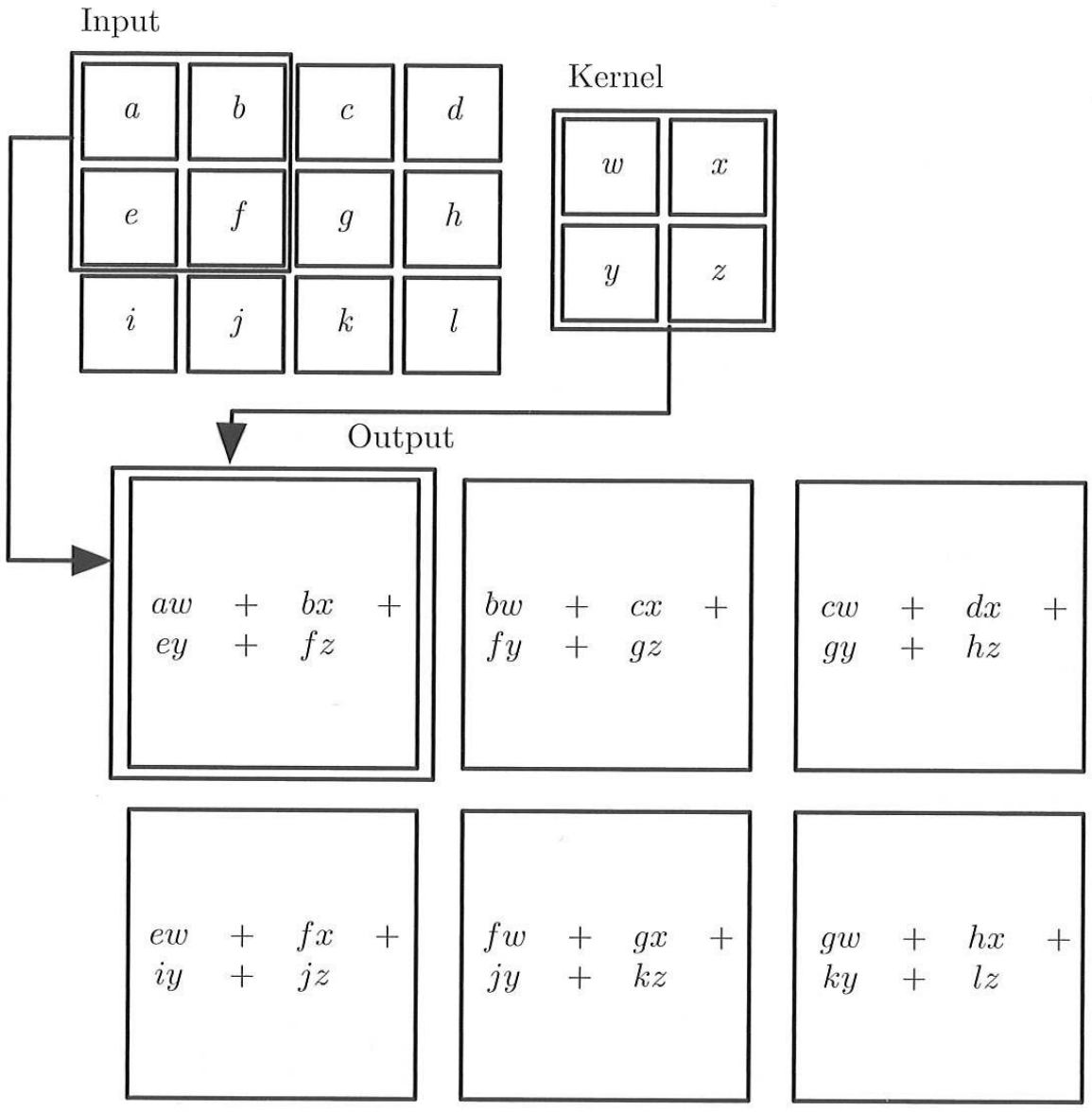
The convolution is easily extended to more dimensions. For a 2-D image $I$, we need a 2-D kernel $K$ as well:

$$(I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n)$$

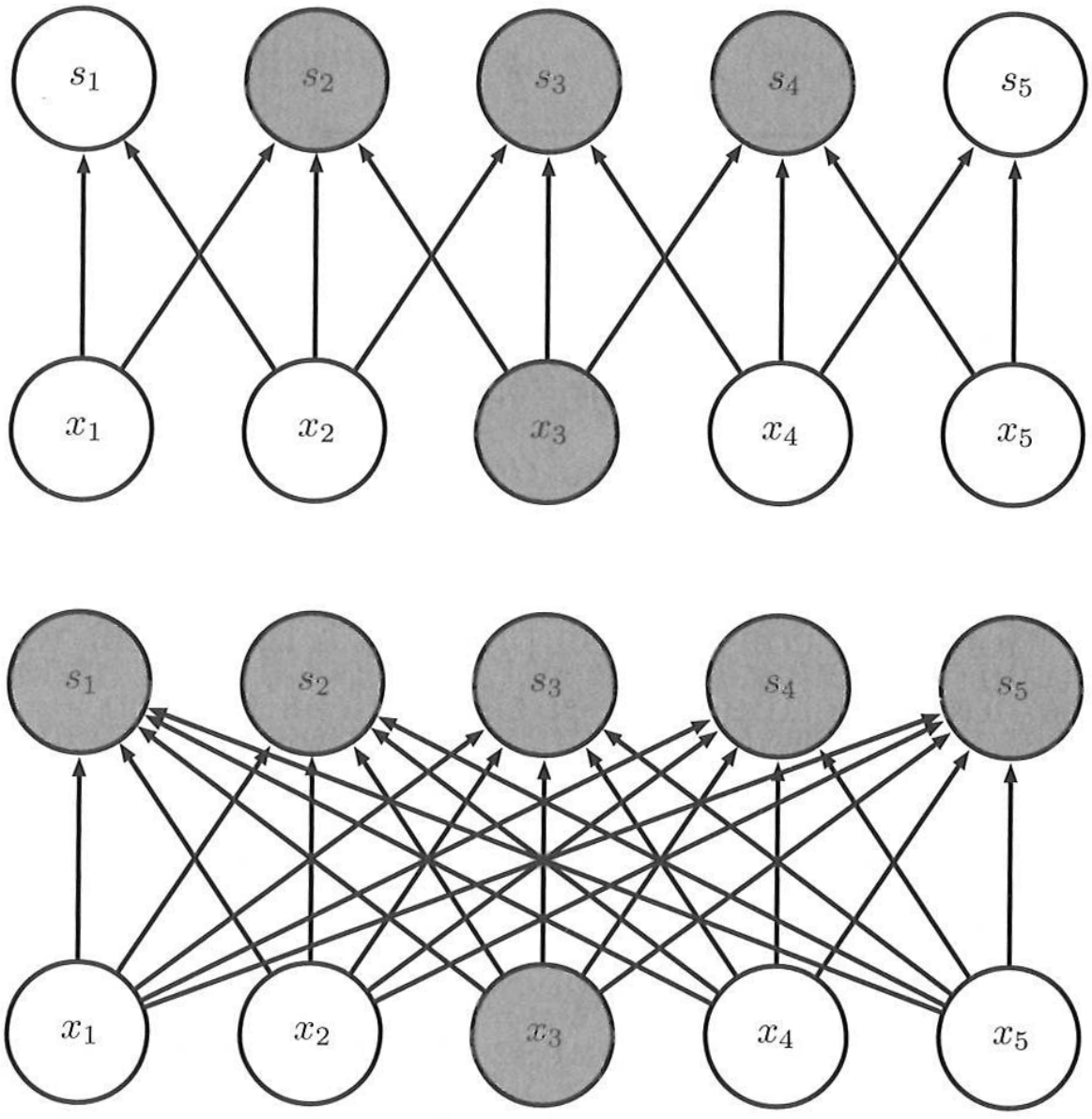
In practice, many deep learning libraries actually implement the **cross-correlation** instead (and still, incorrectly, call it convolution):

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

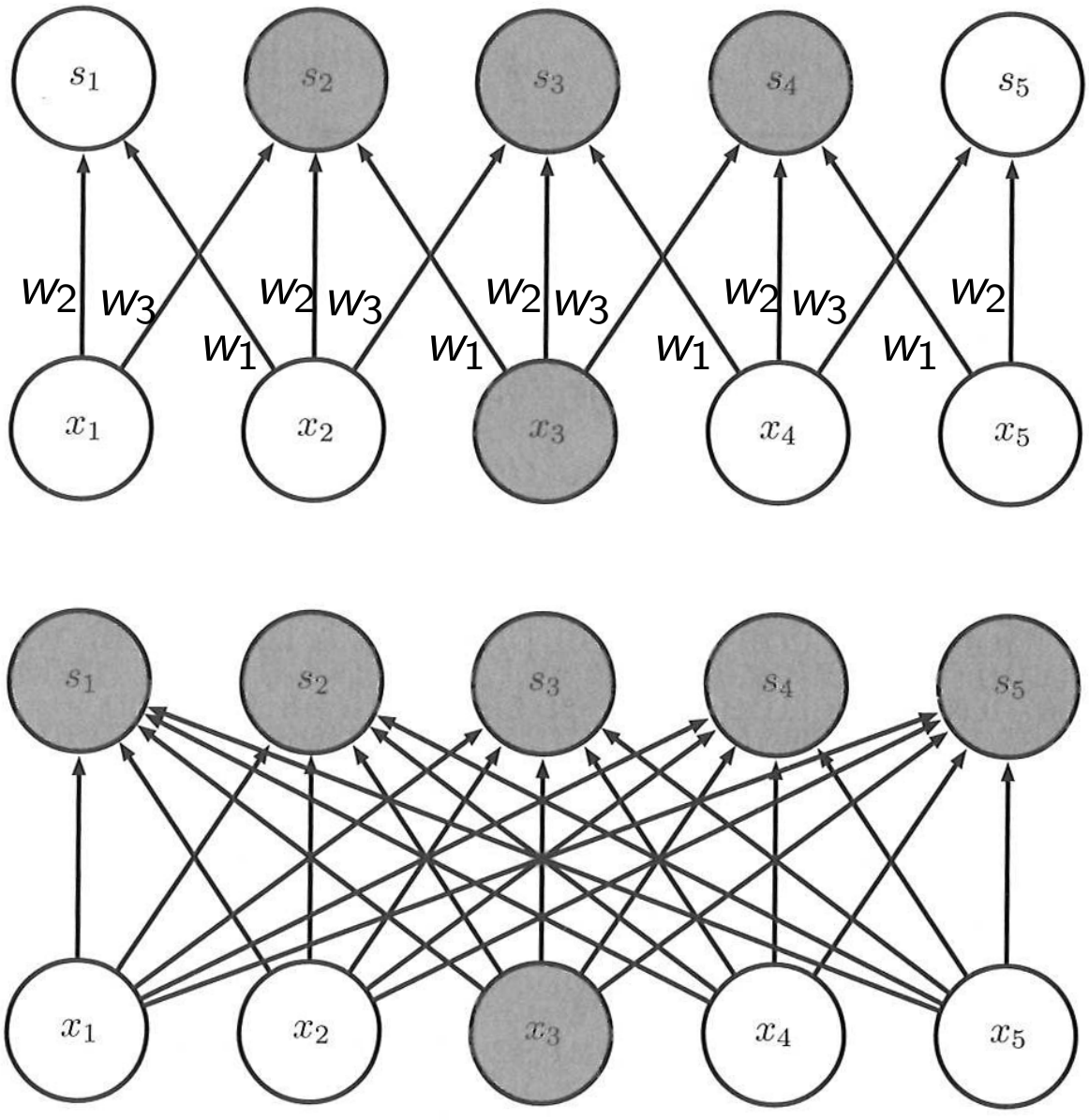
# Convolution



# Sparse connectivity



# Parameter sharing



# Convolutional layer

A convolutional layer in a neural network computes the convolution over the image

In other terms: Each unit looks at a *local receptive field* and averages the inputs using the weights corresponding to some filter

The weights for each unit are identical, so they correspond to applying the same filter for each area of the image

A filter with learned weights is also called a feature, as it detects a particular feature (e.g. a horizontal edge)

Typically we have several such filters per layer, as we wish to detect several different features

# Convolutional layer

For an image the input is a 2-D matrix, or 3-D for a color image (e.g. RGB value)

$\mathbf{V}$ where $V_{i,j,k}$ is the value for channel $i$ at row $j$ and column $k$

Then the kernel is a 4-D tensor $\mathbf{K}$, and the convolution is:

$$\mathbf{Z}_{i,j,k} = \sum_{l,m,n} \mathbf{V}_{l,j+m-1,k+n-1} \mathbf{K}_{i,l,m,n}$$

Often we don't convolve at every point, but only at every $s$ pixels:

$$\mathbf{Z}_{i,j,k} = \sum_{l,m,n} \mathbf{V}_{l,(j-1)s+m,(k-1)s+n} \mathbf{K}_{i,l,m,n}$$

$s$ is called the **stride**

# Training via backpropagation

Given a cost funtion $J(\mathbf{V}, \mathbf{K})$ we will receive a gradient from the previous layer:

$$\mathbf{G}_{i,j,k} = \frac{\partial}{\partial \mathbf{Z}_{i,j,k}} J(\mathbf{V}, \mathbf{K})$$

For updating this layer we need:

$$\frac{\partial}{\partial \mathbf{K}_{i,j,k,l}} J(\mathbf{V}, \mathbf{K}) = \sum_{m,n} \mathbf{G}_{i,m,n} \mathbf{V}_{j,(m-1)s+k,(n-1)s+l}$$

For the next layer we can calculate:

$$\frac{\partial}{\partial \mathbf{V}_{i,j,k}} J(\mathbf{V}, \mathbf{K}) = \sum_{\substack{l,m \\ \text{s.t.} \\ (l-1)s+m=j}} \sum_{\substack{n,p \\ \text{s.t.} \\ (n-1)s+p=k}} \sum_{q} \mathbf{K}_{q,i,m,p} \mathbf{G}_{q,l,n}$$

# Convolutional neural networks

Typically after the convolution we apply a non-linearity (e.g. ReLU), and after that **pooling**

Pooling calculates a summary statistic of a neighborhood, often also downsampling

Most common are **max pooling** and **average pooling**

Reduces the number of parameters, provides translation invariance

# Convolutional neural networks

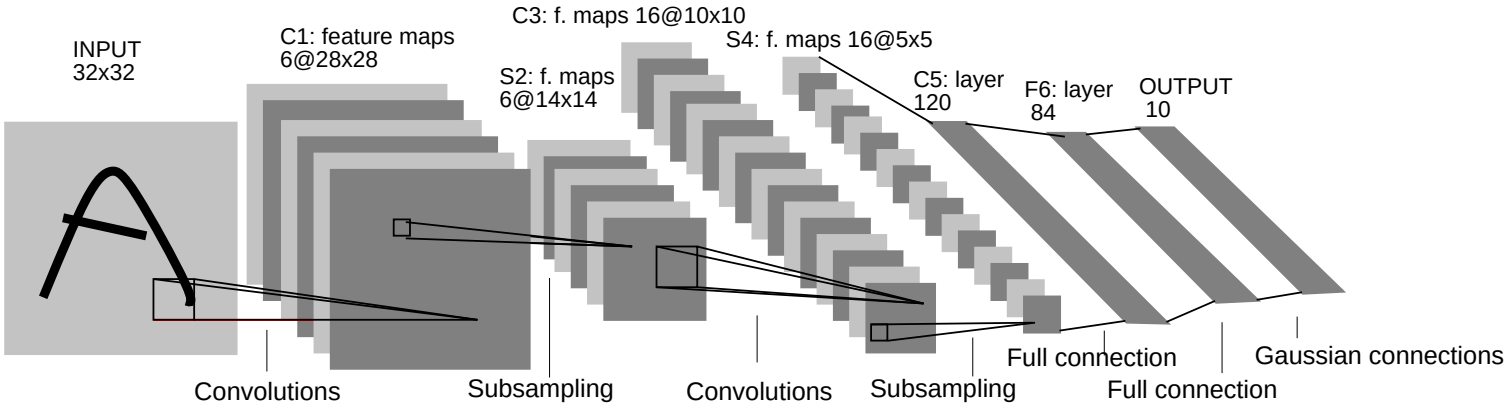
Convolutional neural networks (CNNs) typically has one or more layers of convolution (followed by ReLU and pooling)

Not only images: 1-D for text, or audio, 3-D video data.

A fully-connected layer for predicting class labels (like normal MLP)

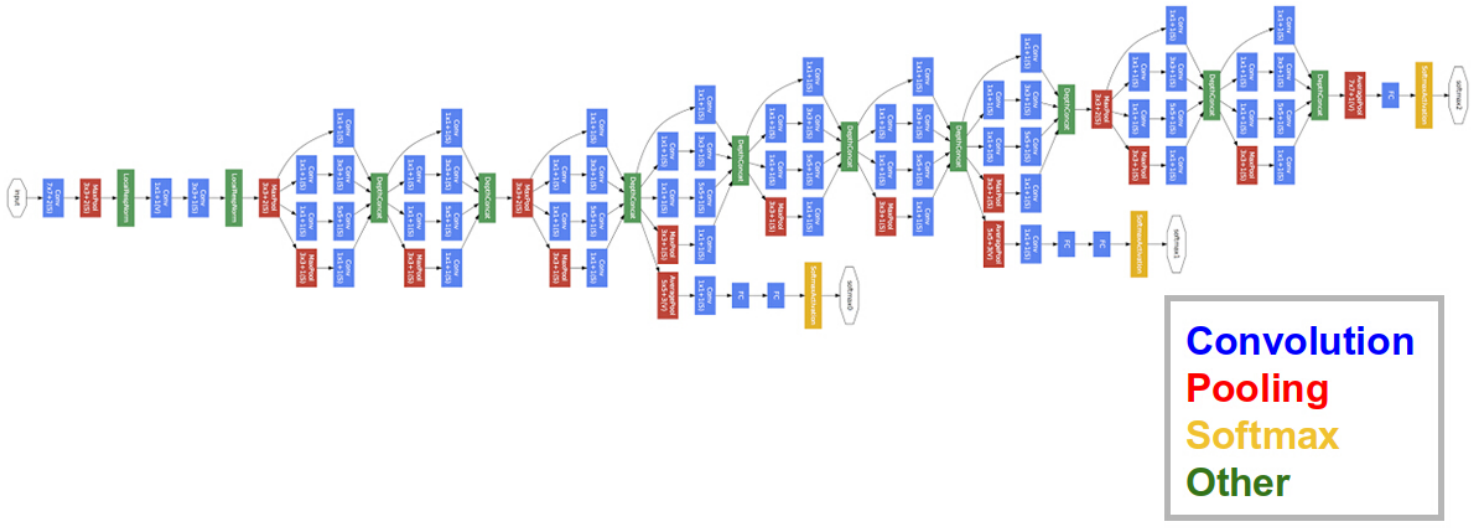
Structured outputs: use the tensor output, instead of predicting a class - e.g. a label for each pixel

# Convolutional neural networks




LeNet5 (LeCun et al. 1998)

# Convolutional neural networks




GoogLeNet (Szegedy et al. 2014)

# Convolutional neural networks

Training the individual features is computationally expensive

Some work-arounds:

- Random weights (works surprisingly well!)
- Greedy layer-wise training
- Use pre-trained layers. Today you can find online, e.g., trained on ImageNet.
  - Replace final fully-connected layer
  - Fine tune for your own data

# CNNs in image recognition

ImageNet Large Scale Visual Recognition Challenge (ILSVRC):

- First time won by a CNN in 2012 (Krizhevsky et al)
- wide margin: top-5 error rate from 26.1% to 15.3%
- since then consistently won by deep CNNs
- top-5 error rate down to 3.0% in 2016

In theory performance is close to that of humans . . .

- still worse on some images (thin objects, filter distortions)
- better on fine-grained categories (e.g. dog breeds)
- but: this is pixel statistics, no real understanding of the image contents

# Modern practices for deep feedforward nets

- Use enough data and GPU computation (via TensorFlow or a similar tool)
- Rectified linear units the default choice for hidden layers
- Convolutions + max pooling for images and other spatial data
- The structure: Pretty much still trial and error; need not even be a chain of layers (skip connections); parameter sharing
- Backpropagation with SGD and momentum, adaptive step-length
- Dropout

# Computational graphs

TensorFlow, Theano and Torch provide another layer of computational abtraction, representing neural networks as *computational graphs*

Programming a deep learning model consists of specifying the computational nodes and their relationships

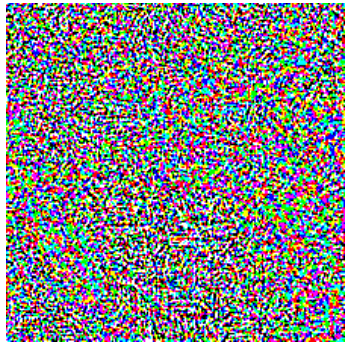
Learning procedures of computing a gradient or updating the weights are simply another set of computational instructions

- Automatic differentiation
- State-of-the-art optimization algorithms
- Parallel computation, GPU support
- Existing implementations for typical models

Note: The same tools work for many other ML models too!

# Adversarial training

Szegdy et al. (2014) found out that deep learning models for image classification can be fooled by constructing adversarial inputs

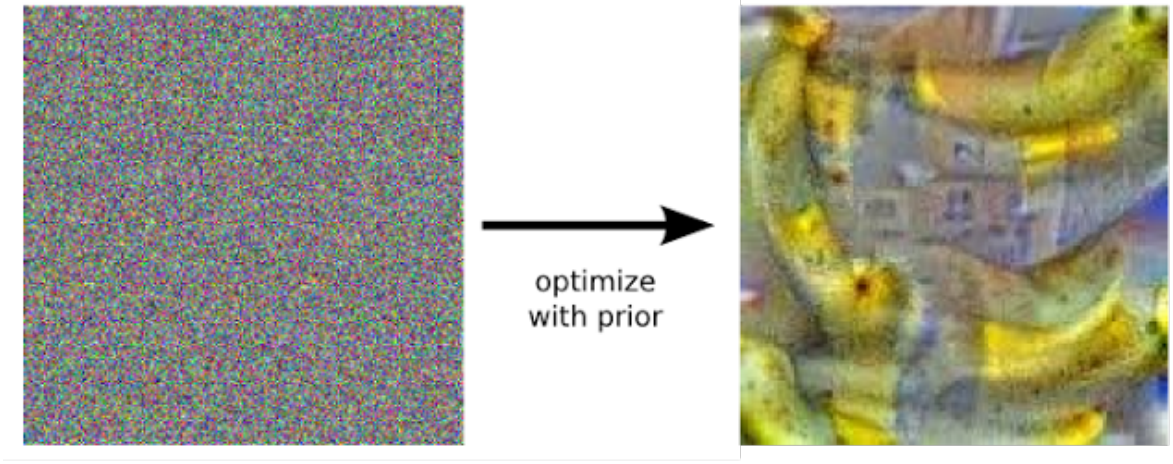



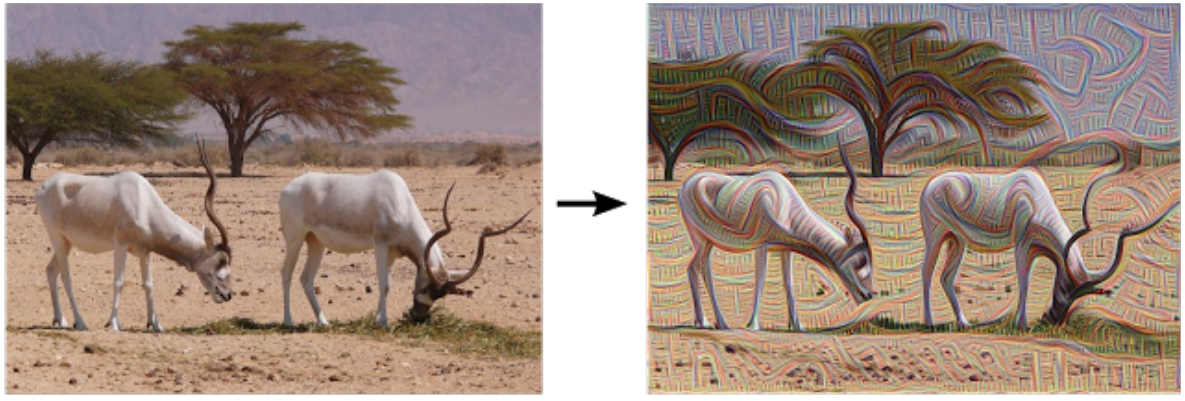
If the scale of the perturbation that looks like random noise is small enough compared to the scale of the original pixel values, humans would not notice a difference, but flexible enough classifier will change its decision

Find input $\mathbf{x}'$ so that it visually looks exactly like $\mathbf{x}$ but has different class prediction – we can use gradient-based optimization wrt to $\mathbf{x}$ instead of the weights to force the output to any class

# Inceptionism

The same trick can be used to make deep networks "dream": Use noise as input for the network and then modify the input until it corresponds to a given class label



Images from http://googleresearch.blogspot.fi/2015/06/inceptionism-going-deeper-into-neural.html

# An aside: human visual system

Hubel and Wiesel in the 1960: basic understanding of mammalian vision system (Nobel prize 1981)

**Primary visual cortex** or **V1** has an actual 2-D structure mirroring the image on the retina

V1 contains **simple cells** performing a linear function on a spatially localized **visual receptive field** (similar to convolutional layers!)

V1 also has **complex cells** respond to same functions, but are invariant to small shifts in position (similar to pooling!)

Finally there are know to be specific cells corresponding to particular concepts such as well-known persons, "grandmother cells" or "Halle Berry neurons"

Of course also many important differences . . .

# Next lectures

Thursday: Recurrent neural networks (RNN): by adding feedback-loops we can learn temporal behaviour: speech recognition, machine translation, image captioning (together with CNNs!)...
Briefly some of the frontiers in deep learning

Next week: **no lecture on Tuesday** (May 2),
"recapitulation lecture" on Thursday (May 4)

Exercise 7 will be to get bonus points