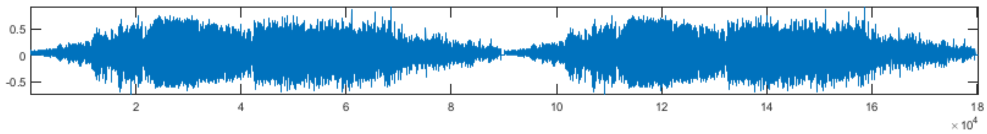
12. Convolutional Neural Networks

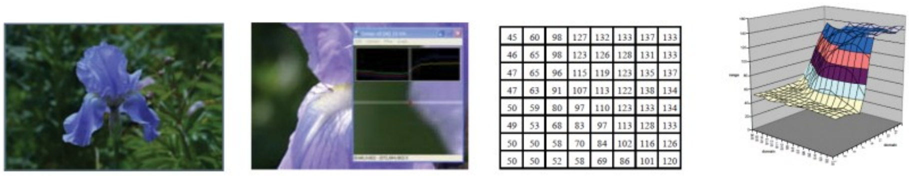
12.1 Convolution

Audio signal – vector (tensor – vectors of vectors) of variable length

Audio



...	0.0468	0.0468	0.0468	0.0390	0.0390	0.0390	0.0546	0.0625	0.0625	0.0390	0.0312	0.0468	0.0625	...



Note: multi-channel 2D matrices (3D tensor)

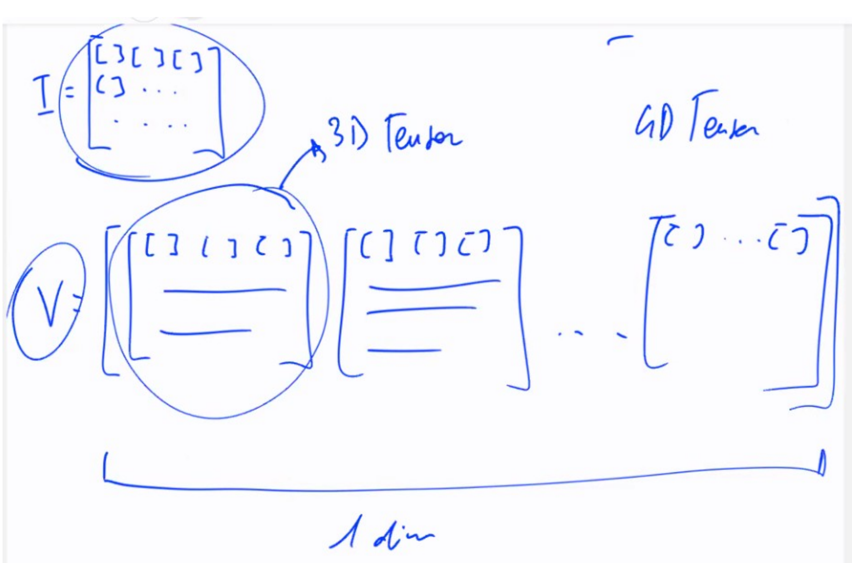


Figure 1: Example of tensor with image and video

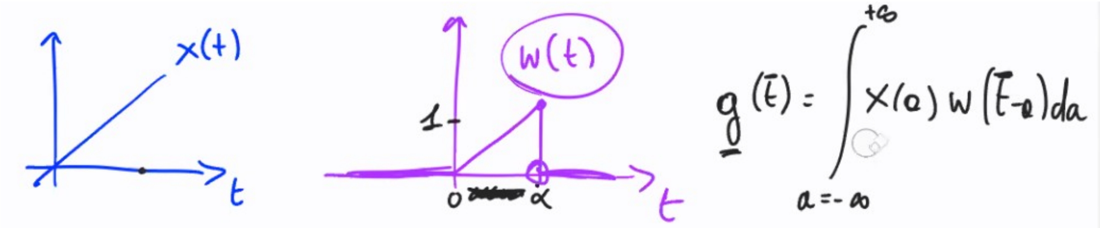
Continuous functions

Convolution operation is defined as:

$$(x * w)(t) \equiv \int_{a=-\infty}^{\infty} x(a)\, w(t-a)\, da$$

Discrete functions

$$(x * w)(t) \equiv \sum_{a=-\infty}^{\infty} x(a)\, w(t-a)$$



integral over a, so a variable and t constant

fix a certain t –> t^

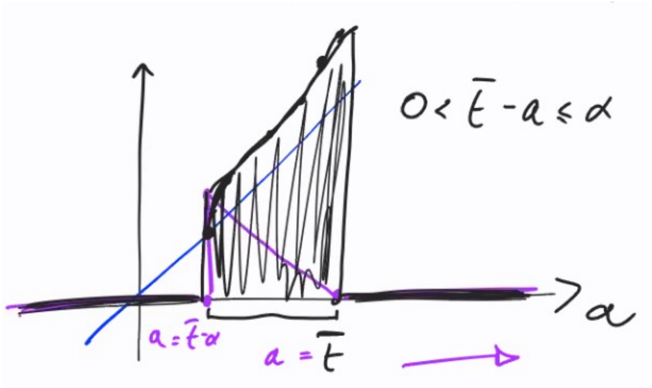
0 <= w(t^ - a) <= alfa

w(t^-a) = 0 <=> t^ = a

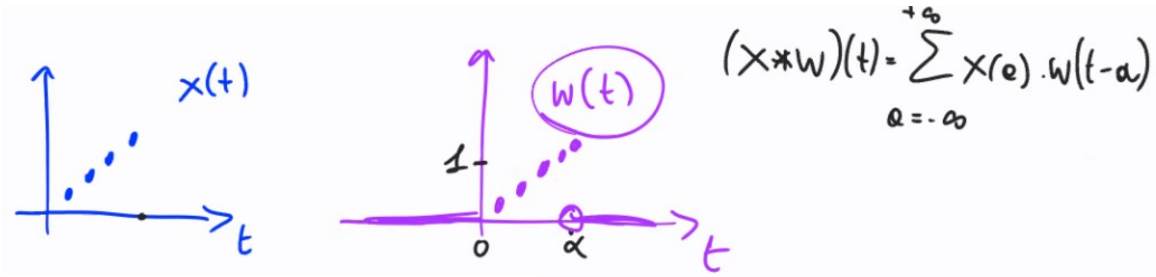
t^-alfa in [0,1]

con a = t^-alfa maximum

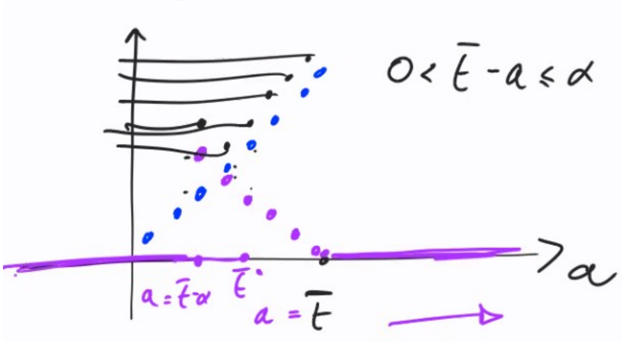
The value of g at t^ is this area →



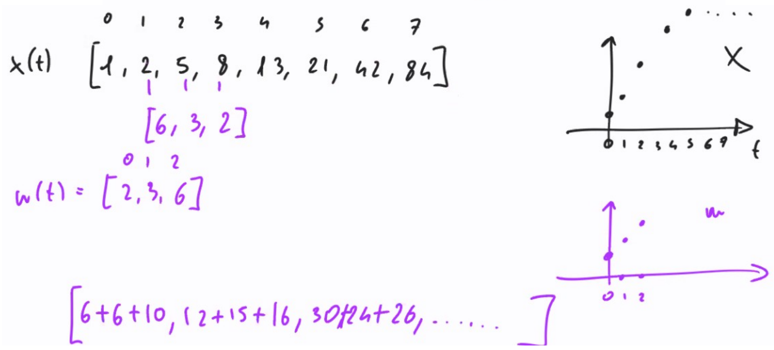
For discrete case:



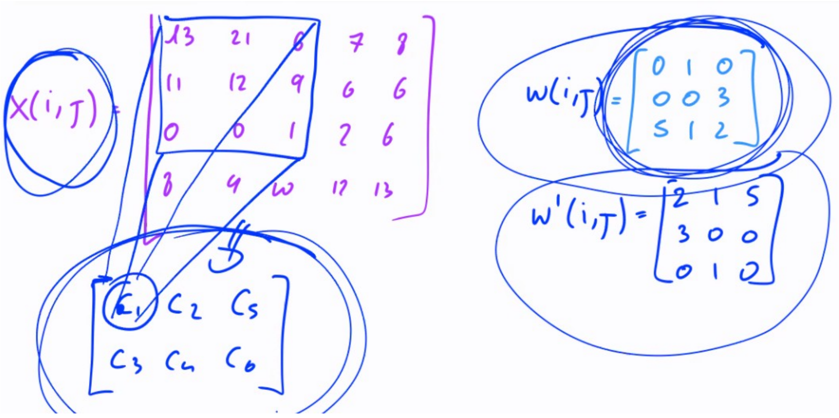
w(t-a) = flip w



Convolution: flip w, multiply w for x (if w points are less than x, slice it).



Multidimensional case:



With x less than w, we can’t do convolution

When you learn the wights you are learning a trasnformation of the input

obs: w is such a kernel function.

Discrete limited 2D functions:

$$(I * K)(i,j) \equiv \sum_{m \in S_1} \sum_{n \in S_2} I(m,n) K(i-m, j-n)$$

I: 2D input, K: 2D kernel, S_i: finite sets.

Discrete limited 3D functions:

$$(I * K)(i,j,k) \equiv \sum_{m \in S_1} \sum_{n \in S_2} \sum_{u \in S_3} I(m,n,u) K(i-m, j-n, k-u)$$

I: 3D input, K: 3D kernel, S_i: finite sets.

Commutative

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

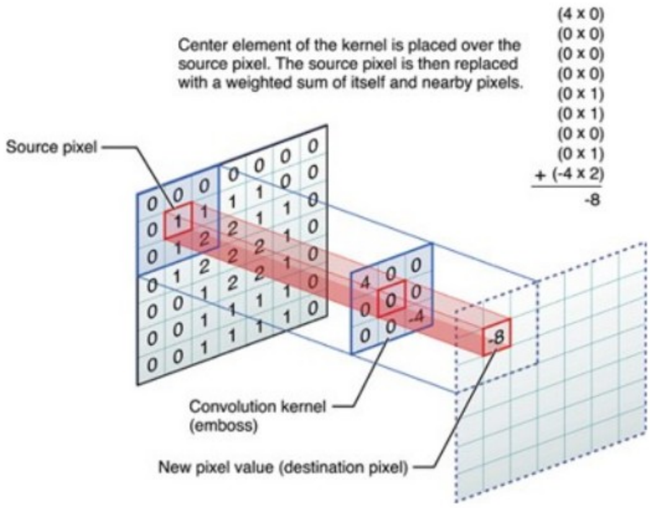
Cross-correlation

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

Also changing the operation, the learning algorithm will adapt.

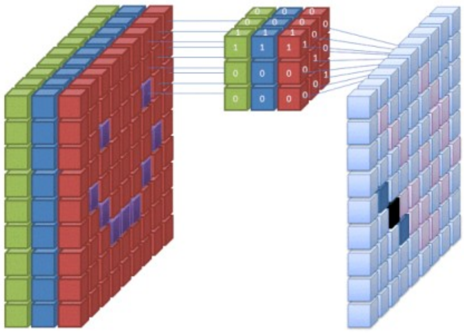
implemented in machine learning libraries (called convolution).

2D Convolution for 2D input image (gray scale) with 2D kernel

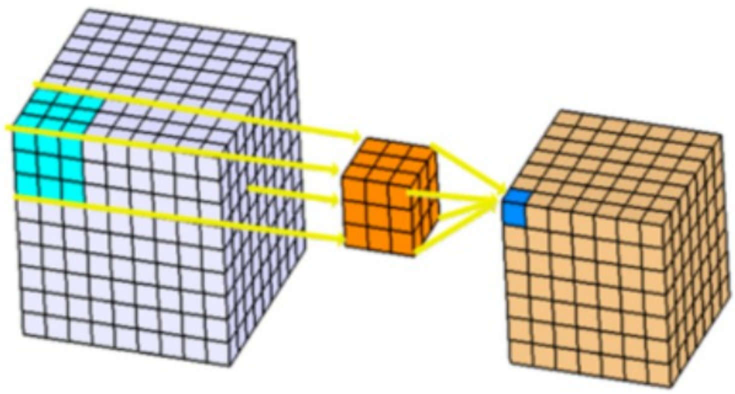


The transformation of that pixel depends on his neighbours.

2D Convolution for 3D input image (RBG channels) with 3D kernel (3 channels)



3D Convolution with 3D input and 3D kernel



Terminology

Input size (w in × h in × d in) dimensions of the input

Kernel size (w k × h k × d k) dimensions of the kernel

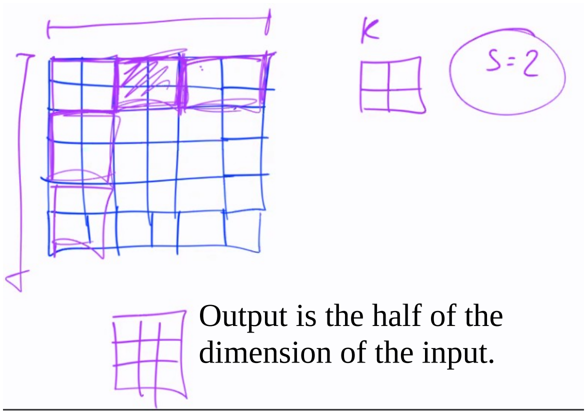
Feature map or Depth slice output of convolution between an input and one kernel

Depth (d) number of kernels (i.e., of feature maps)

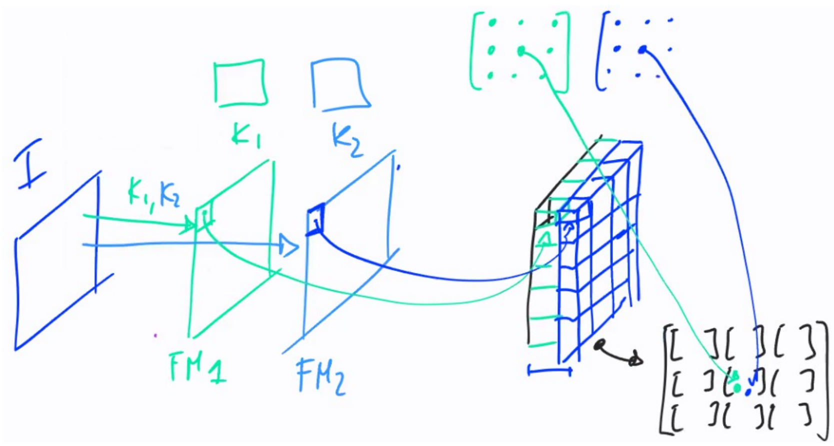
Padding (p) nr. of fillers for outer rows/columns (typically zeros)

Stride (s) step of sliding kernel (1 does not skip any pixel)

Receptive field region in the input space that a particular feature is looking at (i.e., is affected by)



Output is the half of the dimension of the input.



This with multiple kernels; if we consider this as a layer, the ouput will be the input of the next layer.

1 kernel applied to the input produces 1 feature map

Examples:
32 x 32 x 1 image * 5 x 5 x 1 kernel $\rightarrow$ 1 feature map 28 x 28
32 x 32 x 3 image * 5 x 5 x 3 kernel $\rightarrow$ 1 feature map 28 x 28

If we use d kernels, we can generate d feature maps (output of depth d)

Examples:
32 x 32 x 1 image * 6 kernels 5 x 5 x 1 $\rightarrow$ 6 feature maps 28 x 28
32 x 32 x 3 image * 6 kernels 5 x 5 x 3 $\rightarrow$ 6 feature maps 28 x 28

Note: 6 feature maps 28 x 28 are represented as a 28 x 28 x 6 tensor.

In general

Input: $w_{in} \times h_{in} \times d_{in}$

Kernels: d_{out} of size $w_k \times h_k \times d_k$ (with $d_k = d_{in}$)

Output: feature maps $w_{out} \times h_{out} \times d_{out}$

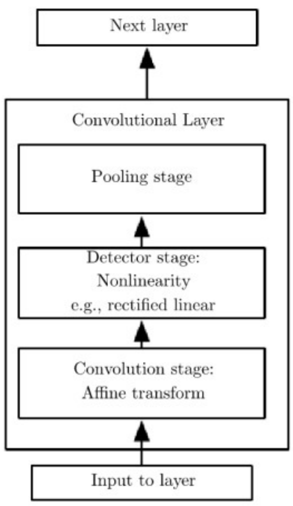
with w_{out}, h_{out} computed according to stride and padding (see next slides)

Number of kernel parameters: $w_k \cdot h_k \cdot d_k \cdot d_{out}$

Note: for 3D convolutions $d_{in} > d_k$ and output with multiple kernels is a 4D tensor $w_{out} \times h_{out} \times z_{out} \times d_{out}$, with z_{out} computed according to slide and padding in the third dimension.

Depends only on the size of the kernel

12.2 Convolutional Neural Networks Layers



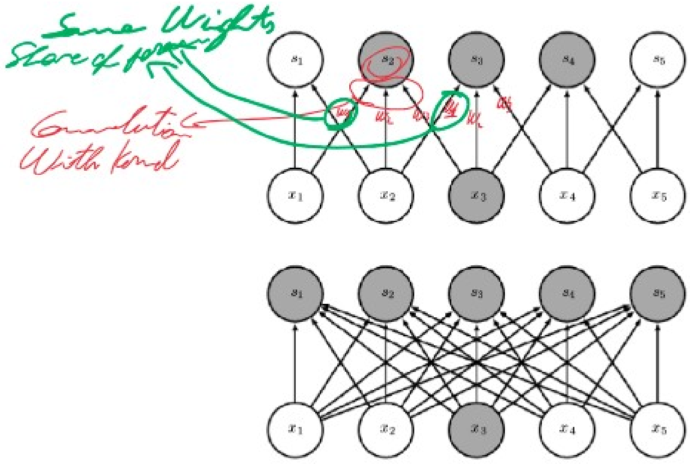
12.2.1 2D Convolution stage

We'll use 2D with several kernels

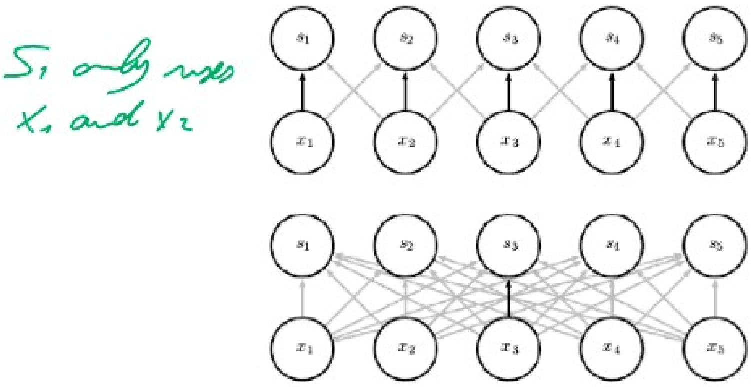
Note: It's better to have a deeper network than a wider one.

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

Sparse connectivity: outputs depend only on a few inputs



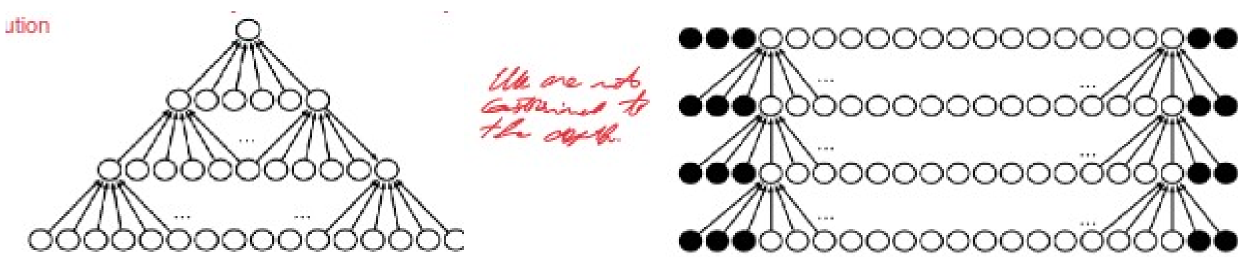
Parameter sharing: Learn only one set of parameters (for the kernel) shared to all the units. K parameters instead of m×n (note:k<<m).



Padding (Number of nodes we are adding to both sides of the network**):**

1. valid padding (p=0): only valid values (output depends on kernel size)

2. same padding(p=W_k/2): output has same size of input



12.2.2 Detector stage

Use non-linear activation functions.

- ReLU, tanh, etc.

12.2.3 Pooling stage

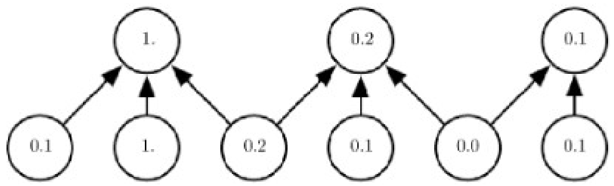
Implements invariance to local translations.

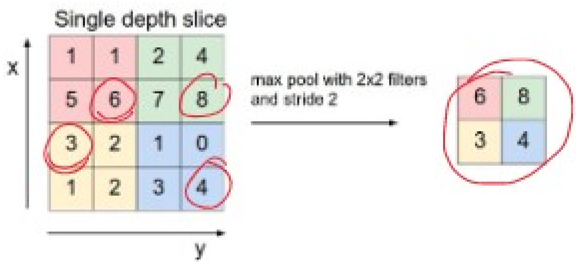
max pooling returns the maximum value in a rectangular region.

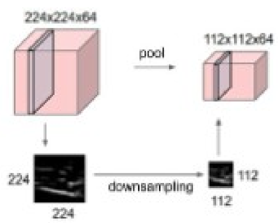
average pooling returns the average value in a rectangular region.

When applied with *stride*, it reduces the size of the output layer.

Example: max pooling with width 3 and stride 2







Consider input of size $w_{in} \times h_{in} \times d_{in}$, d_{out} kernels of size $w_k \times h_k \times d_{in}$, stride *s* and padding *p.* Dimensions of output feature map are given by:

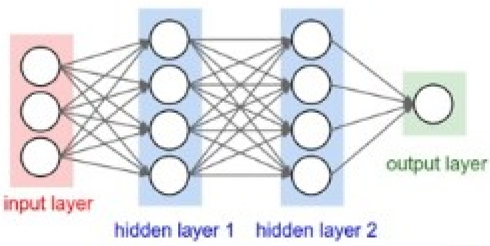
$$w_{out} = (w_{in} - w_k + 2p)/s + 1$$

$$h_{out} = (h_{in} - h_k + 2p)/s + 1$$

Number of trainable parameters of the convolutional layer is:

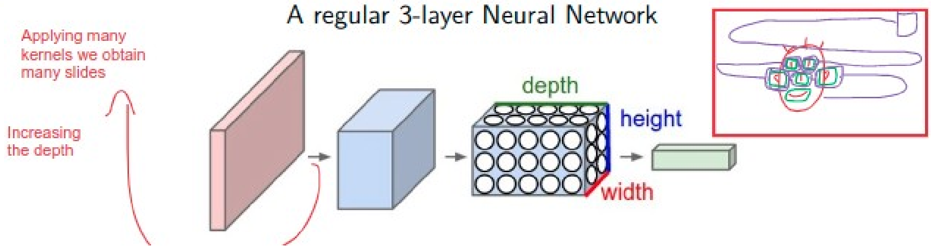
$$|\theta| = \underbrace{w_k \cdot h_k \cdot d_{in} \cdot d_{out}}_{\text{kernel weights}} + \underbrace{d_{out}}_{\text{bias}}$$

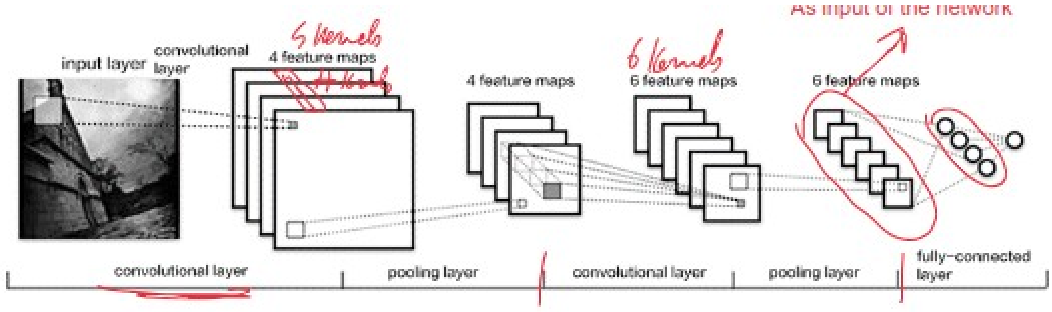
12.3 CNNs for images (2D input)

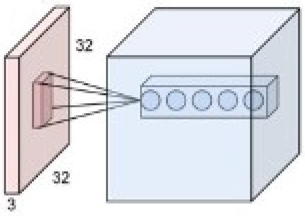


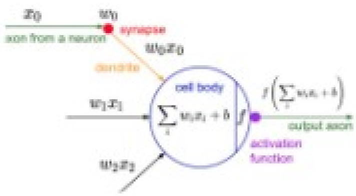
Why we use CNN for images? If you take a kernel and you slide it all ovr the image, you will detect a zone you need, e.g.: ears, nose, etc.

A regular 3-layer Neural Network



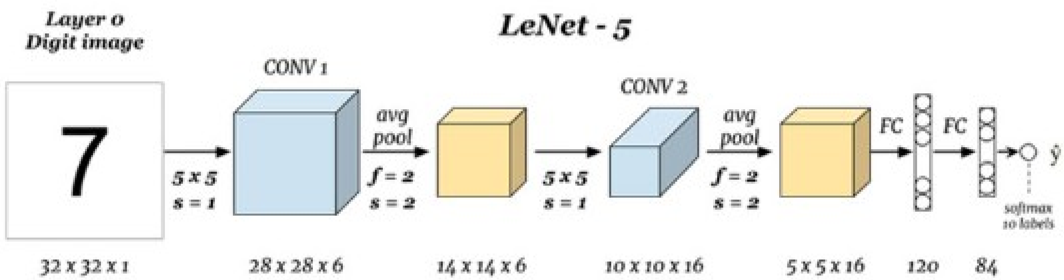


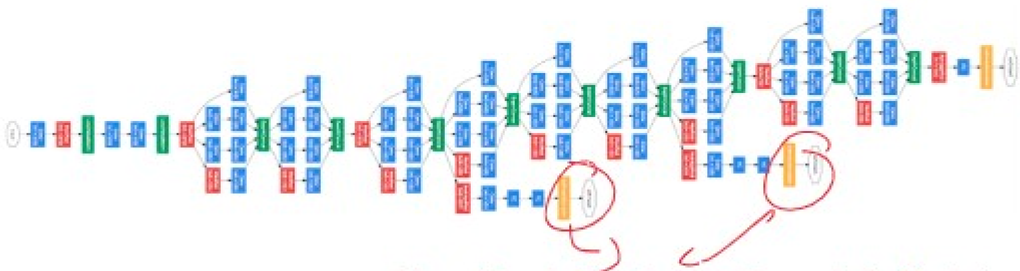




Each neuron is connected to a local 'horizontal' region of the input volume, but to **all** channels (depth)

The neurons still compute a dot product of their weights with the input followed by a non-linearity





Intermediate outputs, just to reduce the complexity of the train

12.4 Common uses of famous CNNs

Train a new model on a dataset

Use pre-trained models (e.g., trained on ImageNet) to:

- predict ImageNet categories for new images

- extract features to train another model (e.g., SVM)

Refine pre-trained models on a new dataset (new set of classes)

12.5 Transfer Learning

Definitions

- D is a *domain* defined by data points $\mathbf{x}_i \in \mathbf{X}$ distributed according to $\mathcal{D}(\mathbf{x})$
- T is a *learning task* defined by labels $\mathbf{y} \in \mathbf{Y}$, a target function $f : \mathbf{X} \to \mathbf{Y}$, and distribution $P_{\mathcal{D}}(\mathbf{y}|\mathbf{x})$

Given

- D_S and T_S a source domain and learning task

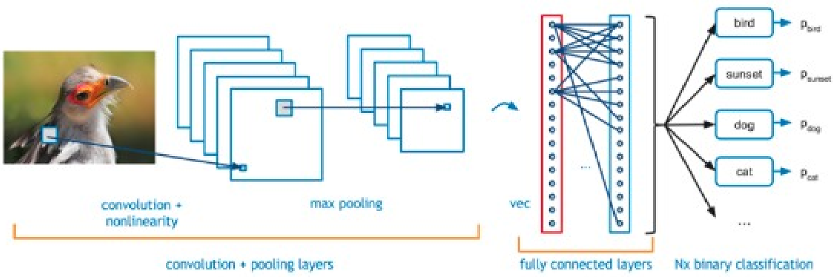
- D_T and T_T a target domain and learning task

In general, D_S != D_T and T_S != T_T

Goal: improve learning of f_T:$X_T \to Y_T$ using knowledge in D_S and T_S(i.e.,after training f_S:$X_S \to Y_S$)

12.5.1 Examples

Image classification using a CNN

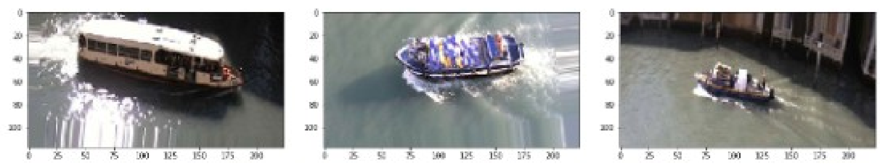


CNN pre-trained on Imagenet

Image classification using a CNN

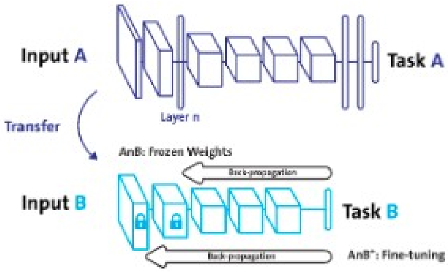
Use a pre-trained model for a different domain and/or learning task
E.g. Boat recognition in ARGOS dataset:

- thousands of boat images (target domain)
- classification of 20 boat classes (target learning task)



1st solution - Fine-tuning

- use same network architecture with pre-trained model
- network parameters 'copied' from the pre-trained model
- no random initialization



Strategies

- training of all network parameters
- 'freeze' parameters of some layers (usually the first ones)

Pro Full advantage of the CNN!
Con 'Heavy' training

2nd solution - CNN as feature extractor

1. extract features at a specific layer of CNN, usually:
   - last convolutional layer (flattened)
   - dense layers
2. collect extracted features $\mathbf{x}'$ of training/validation split and associate corresponding labels t in a new dataset $D' = \{(\mathbf{x}'_1, t_1), \ldots, (\mathbf{x}'_N, t_n)\}$
3. train a new classifier C' using dataset D', e.g.
   - ANN (extreme case of fine-tuning)
   - SVM
   - linear classifier
   - ...
4. classify extracted features of test set using the classifier C'

Pro No need to train the CNN!
Con Cannot modify features, source and target domains should be as 'compatible' as possible