Sapienza University of Rome

Master in Artificial Intelligence and Robotics
Master in Engineering in Computer Science

Machine Learning

A.Y. 2020/2021

Prof. L. Iocchi, F. Patrizi

Sapienza University of Rome, Italy - Machine Learning (2020/2021)

12. Convolutional Neural Networks

NN with convolutional layers

L. Iocchi, F. Patrizi

The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers

with contributions from Valsamis Ntouskos

Overview

- Convolution
- Convolutional layers
- Pooling
- Example: LeNet
- CNNs for Images
- "Famous" CNNs
- Transfer learning
- Resources

References

Ian Goodfellow and Yoshua Bengio and Aaron Courville. Deep Learning - Chapter 9. `http://www.deeplearningbook.org`

Motivation

are a specialized kind of neural network for processing data that has a known grid-like topology.

Up to now we treated inputs as general feature vectors

In some cases inputs have special structure:

- Audio
- Images
- Videos

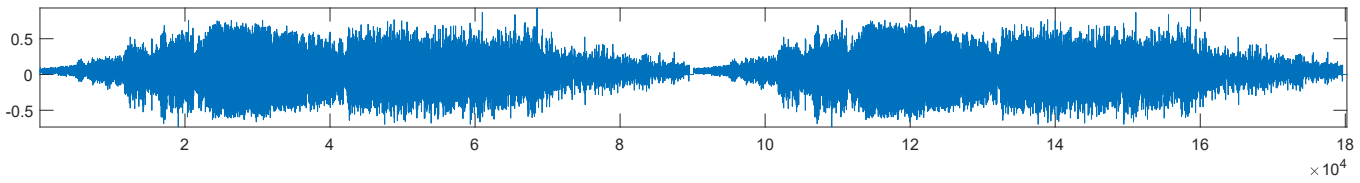
CNN important for this type of inputs

Signals: Numerical representations of physical quantities

Deep learning can be directly applied on signals by using suitable operators

Motivation - Examples

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	...

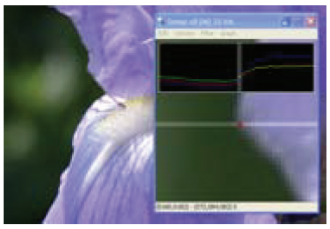
Note: variable length 1D vectors (1D tensor)

we can have physical measurements of the signal for ex. the amplitude of the sound signal

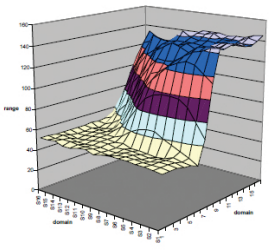
Motivation - Examples

Images

the third dimension is to represent colors



45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120



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

Video

A sequence of color images sampled through time
Note: sequence of multi-channel 2D matrices (4D tensor)

Convolution

Continuous functions

convolutional operator

functon shifted with respect to the first

function

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

Discrete functions

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

Convolution

the sum is done on a finite interval of values

Discrete limited 2D functions:

The output is sometimes referred to as the feature map

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

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.

Convolution

The commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases The only reason to flip the kernel is to obtain the commutative property. While the commutative property is useful for writing proofs, it is not usually an important property of a neural network implementation.

Commutative

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

Cross-correlation the main goal of the convolutional NN is to learn K (that is not given) or its flipped version

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

implemented in machine learning libraries (called convolution).

many neural network libraries implement a related function called the cross-correlation, which is the same as convolution but without flipping the kernel

Different types of Convolutions

1D Convolution always 1D output

Convolution kernel moves in one direction.

2D Convolution always 2D output

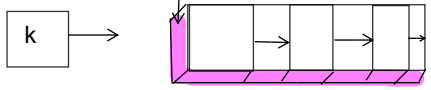
we are interested in this type for images

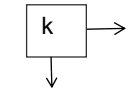
Convolution kernel moves in two directions.

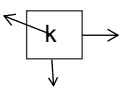
3D Convolution

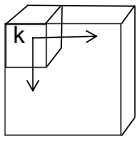
Convolution kernel moves in three directions.

we can also have a 3D input and a 1d kernel so the kernel moves in 1 direction









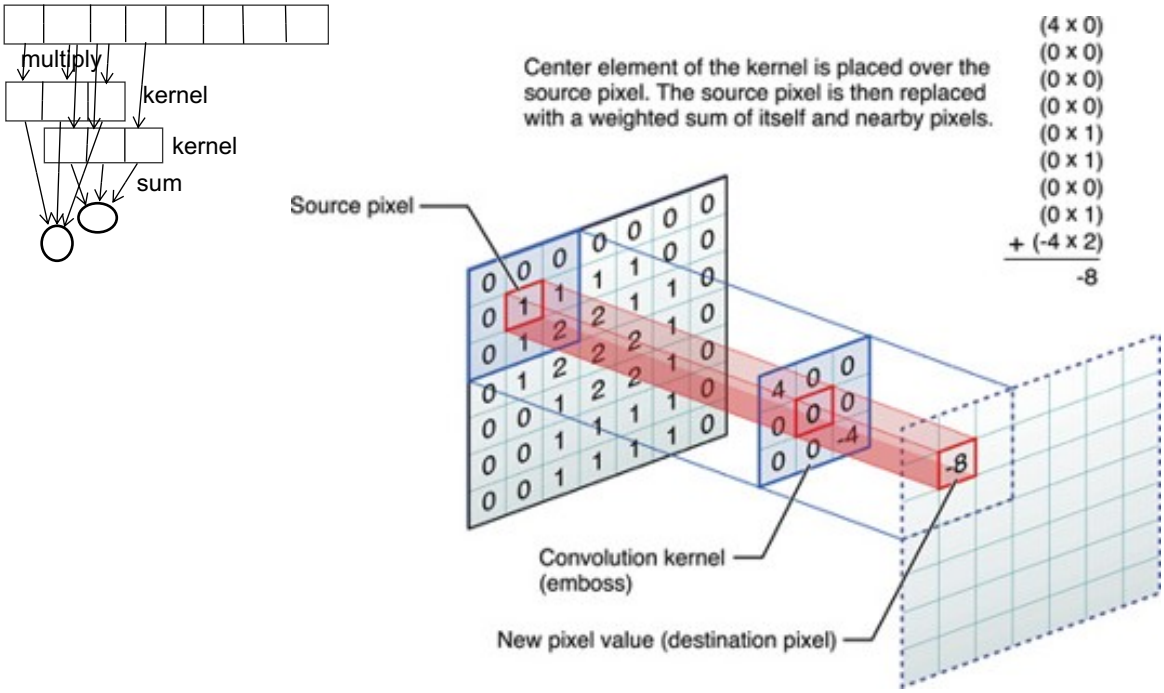
3D input, 3D kernel but in this case the kernel is equal to the depth of the image, if it is smaller it can move in 3 directions

Note: 1D/2D/3D Convolutions does not refer to the dimensions of the input and of the kernel.

We can have 1D Convolutions for 2D input and kernels, 2D Convolutions for 3D input and kernels, etc.

Image 2D Convolutions

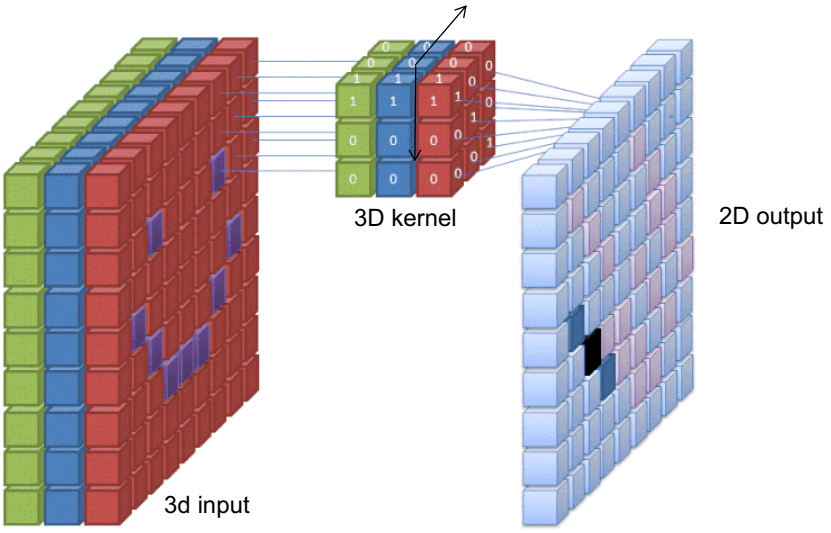
2D Convolution for 2D input image (gray scale) with 2D kernel



Interactive example: `http://setosa.io/ev/image-kernels/`

Image 2D Convolutions

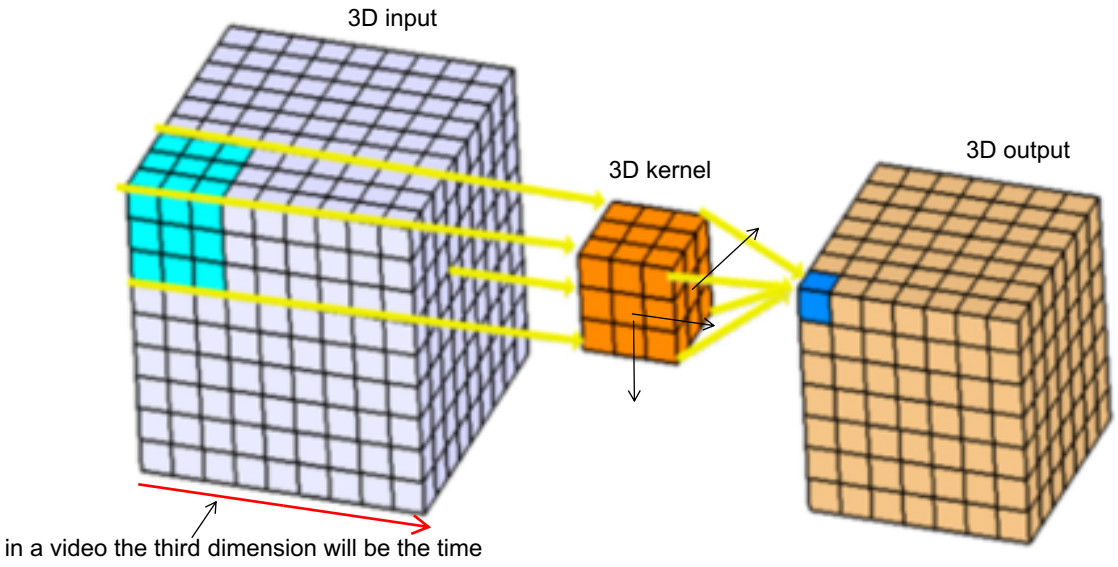
2D Convolution for 3D input image (RBG channels) with 3D kernel (3 channels)



Source: https://commons.wikimedia.org/wiki/File:Convolutional_Neural_Network_with_Color_Image_Filter.gif

3D Convolutions

3D Convolution with 3D input and 3D kernel



Source: https://www.kaggle.com/shivamb/3d-convolutions-understanding-use-case

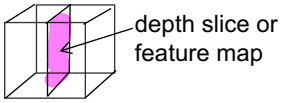
Terminology

rgb=3, gray scale=1

Input size $(w_{in} \times h_{in} \times d_{in})$ dimensions of the input

Kernel size $(w_k \times h_k \times d_k)$ 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)

2D Convolutions with multiple kernels

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

only 1 feature map becuase we use 1 kernel and the kernel and the image have the same depth

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.

2D Convolutions with multiple kernels

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.

Convolutional Neural Networks

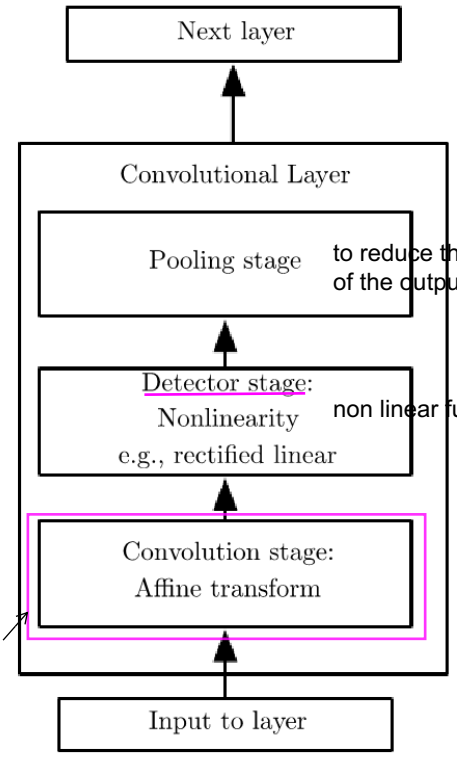
convolution is a linear operation

FNN with one or more convolutional layers.

Convolutional layer

Three stages:

- convolutions between input and kernel
  In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations
- non-linear activation function (detector)
  In the second stage, each linear activation is run through a nonlinear activation function
- pooling
  In the third stage, we use a pooling function to modify the output of the layer further



to reduce the dimension of the output

non linear function f

the only trainable parameters are here and the other layers not introduce additional parameters to be trained (the only trainable parameters are in the kernel)

it means that the dimension of the output depends only on the input because detection and pooling do not affect the depth

Convolutional layer: Convolution

1. 2D Convolution stage

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

Very efficient in terms of memory requirements and generalization accuracy.

Convolutional networks, however, typically have sparse interactions (also referred to as sparse connectivity or sparse weights) (kernel<input)

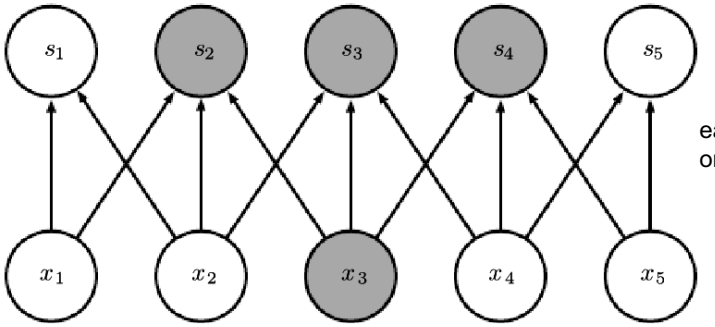
- Sparse connectivity
- Parameter sharing

they permits to reduce the number of trainable parameters

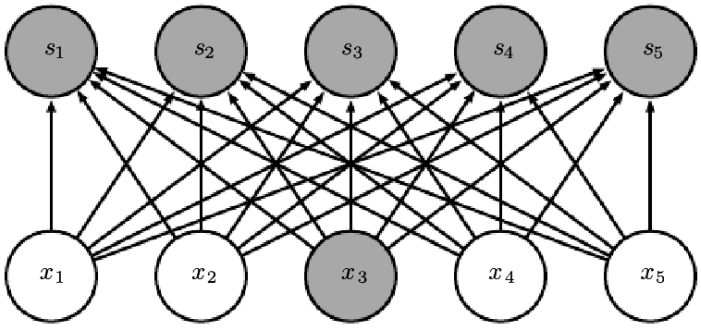
refers to using the same parameter for more than one function in a model.

Sparse connectivity

sparse interactions/ sparse connectivity: outputs depend only on a few inputs (as kernel is usually much smaller than input)



each input node can be connected only to a subset of output nodes



in a fully connected each input nodes (dim N) is connected to all the output nodes (dim M). The number of connections is M*N

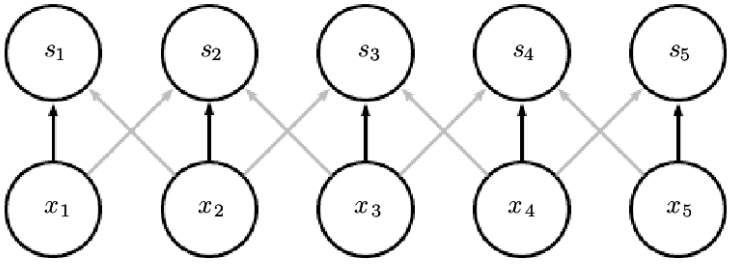
Parameter sharing

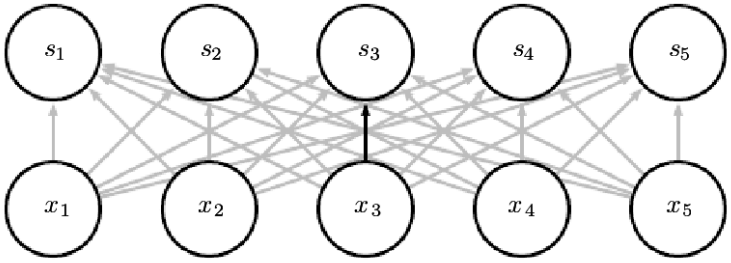
because the value of the kernel is the same in multiple applications over the input

Learn only one set of parameters (for the kernel) shared to all the units.

k parameters instead of $m \times n$ (note: $k \ll m$)

The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all input locations.





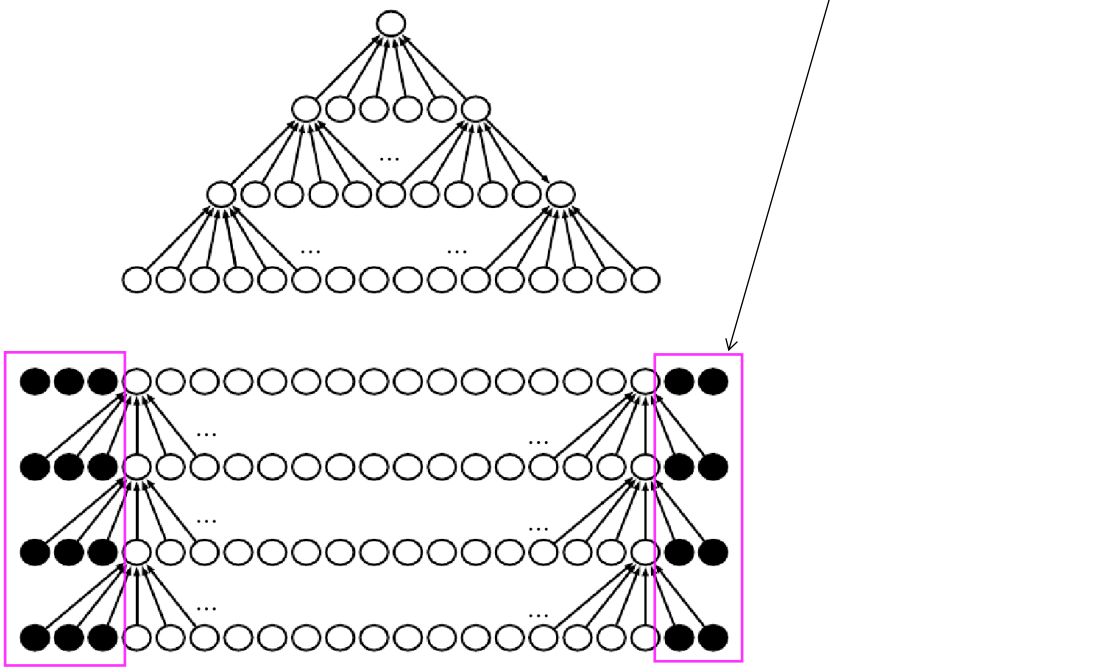
The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing, so the parameter is used only once

Black arrows indicate the connections that use a particular parameter in two different models.

Padding

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

same padding $(p = w_k/2)$: output has same size of input



Convolutional layer: Detector

2. Detector stage

Use non-linear activation functions.

- ReLU
- tanh
- ...

Convolutional layer: Pooling

3. Pooling stage

Implements *invariance to local translations*.

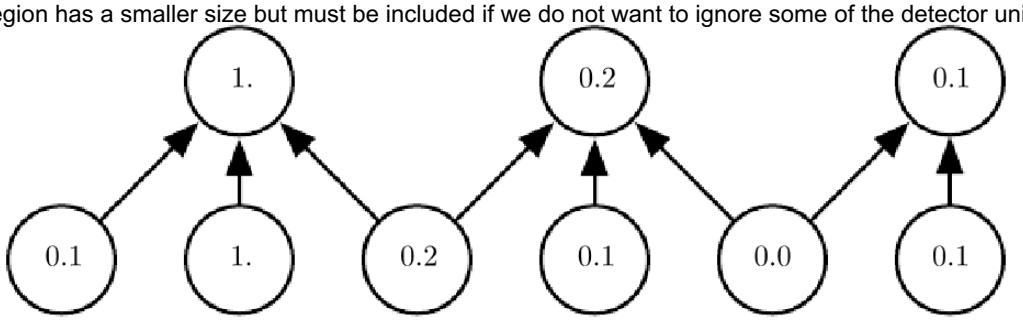
In all cases, pooling helps to make the representation approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.

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.

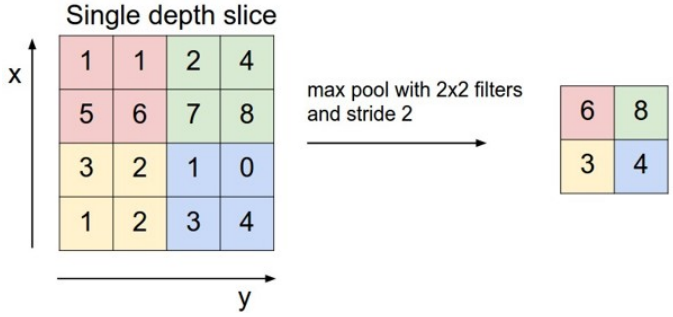
Example: max pooling with width 3 and stride 2 higher the stride smaller the output

This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer. Note that the rightmost pooling region has a smaller size but must be included if we do not want to ignore some of the detector units.

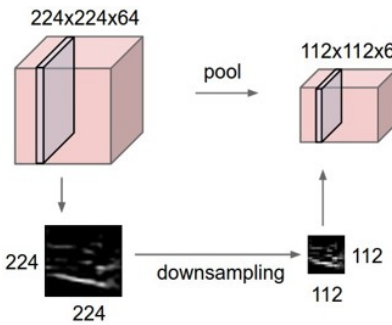


Convolutional layer: Pooling

Example of **max pooling**



Introduces subsampling:



Feature map size - Number of parameters

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:

$$w_{out} = \frac{w_{in} - w_k + 2p}{s} + 1$$

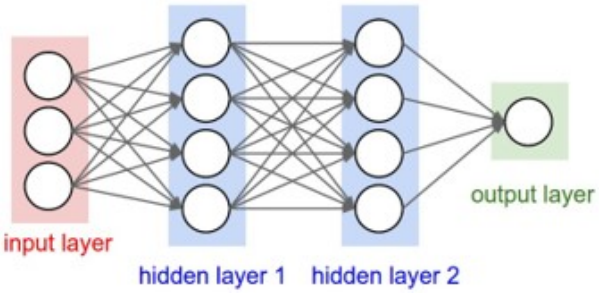
$$h_{out} = \frac{h_{in} - h_k + 2p}{s} + 1$$

Number of trainable parameters of the convolutional layer is:

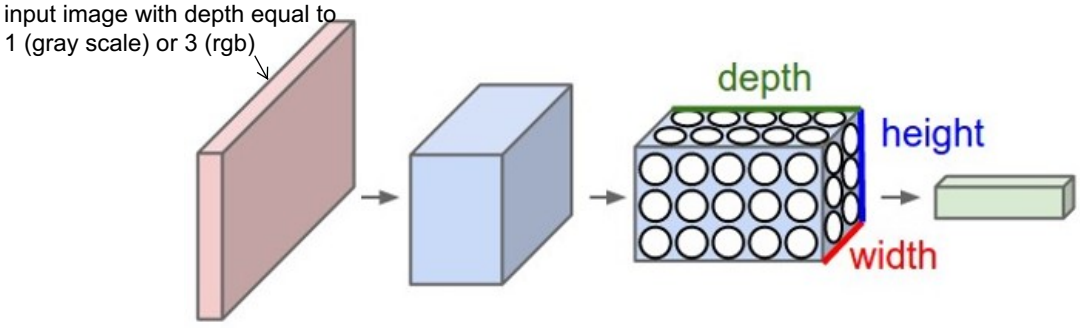
$$|\theta| = \underbrace{w_k \cdot h_k \cdot d_{in} \cdot d_{out}}_{\text{kernel weights}} + \underbrace{d_{out}}_{\text{bias}}$$

num of kernels

CNNs for Images (2D input)



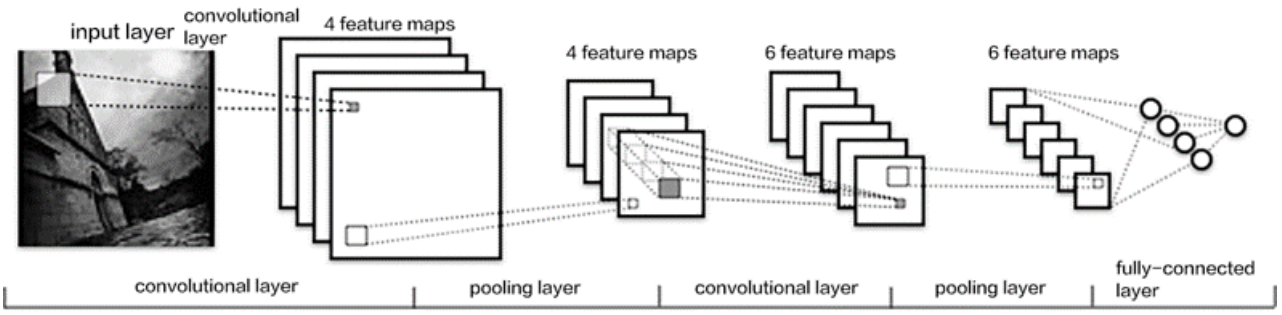
A regular 3-layer Neural Network

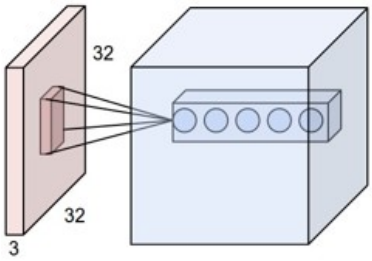


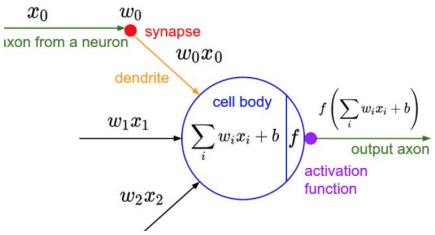
Every convolutional layer of a CNN transforms the 3D input volume to a 3D output volume of neuron activations.

Material from Fei-Fei's group

CNNs for Images (2D input)







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

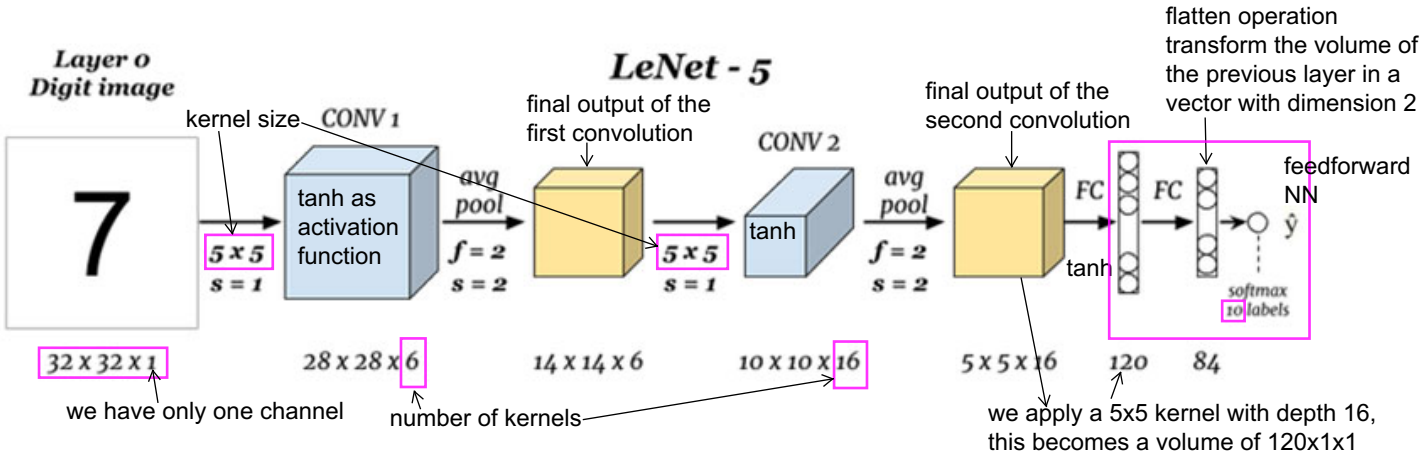
The neurons still compute a dot product of their weights with the input followed by a non-linearity

Material from Fei-Fei's group

Example: LeNet

simple but very good performance (fo ex. with MNIST dataset)

it's not very deep (only 4 layers)



Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998

the number of params usually is very small at the beginning and increases with the depth of the network

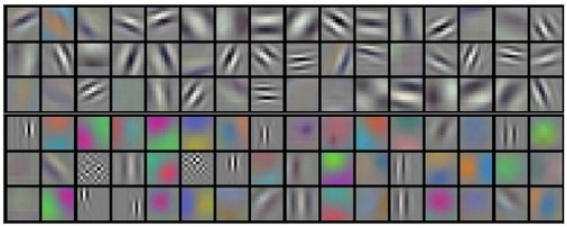
Handwritten recognition with LeNet

MNIST database

Accuracy up to 98 %.

Kernels and Feature maps - Example

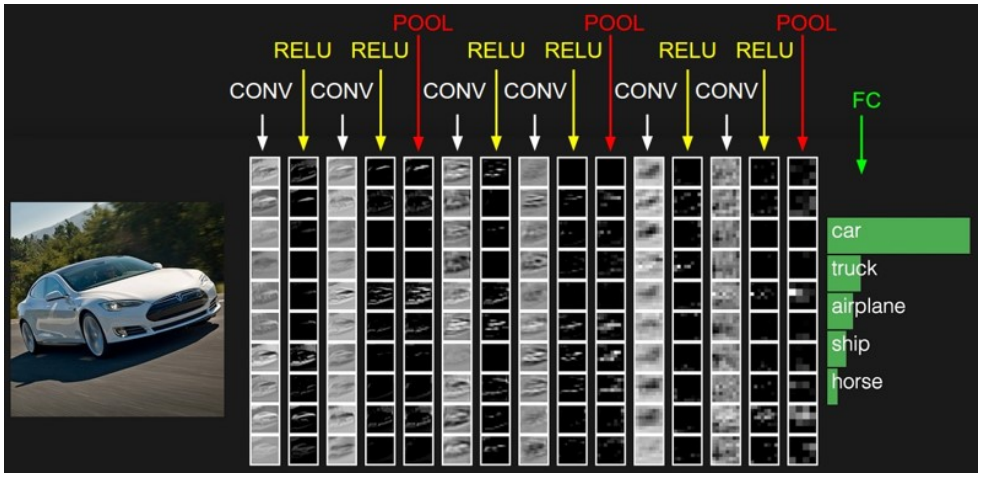
Visualization of learned kernels of the first layer:



visualization of some feature maps

Computed activations during forward pass:

we can see CNN layers as an authomatic generation of feature extraction



Material from Fei-Fei's group

Recent success of CNNs

- Theoretical advancements:
  - Dropout
  - ReLUs
  - Batch Normalization
  - Skip connections
  - ...
- Massively Parallel Computing (GPUs/TPUs)
- Very large training sets (ImageNet/MS COCO)
- International competitions (ILSVRC 2010-2017)
- Developing Frameworks (Keras, Theano, Tensorflow, PyTorch, ...)

ImageNet and ILSVRC

largest image dataset to train a NN

ImageNet Huge dataset of images
over 14 M labelled high resolution images
about 22 K categories

ILSVRC Competitions of image classification at large scale (since 2010)
1.2 M images in 1 K categories
5 guesses about image label

`http://image-net.org`

"Famous" CNNs

- AlexNet (2012)
- GoogLeNet / Inception (2014)
- VGG (2014)
- ResNet (2015)

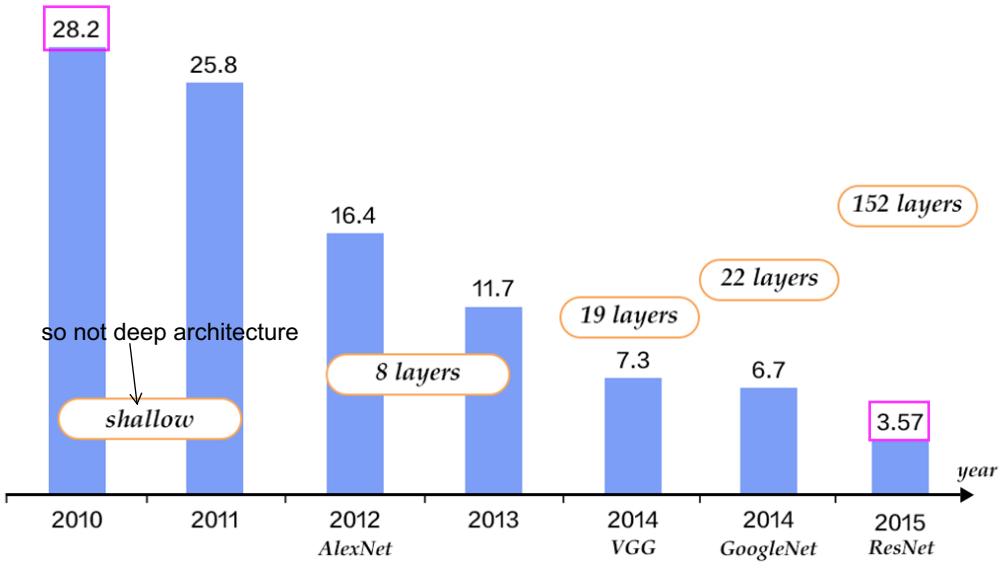
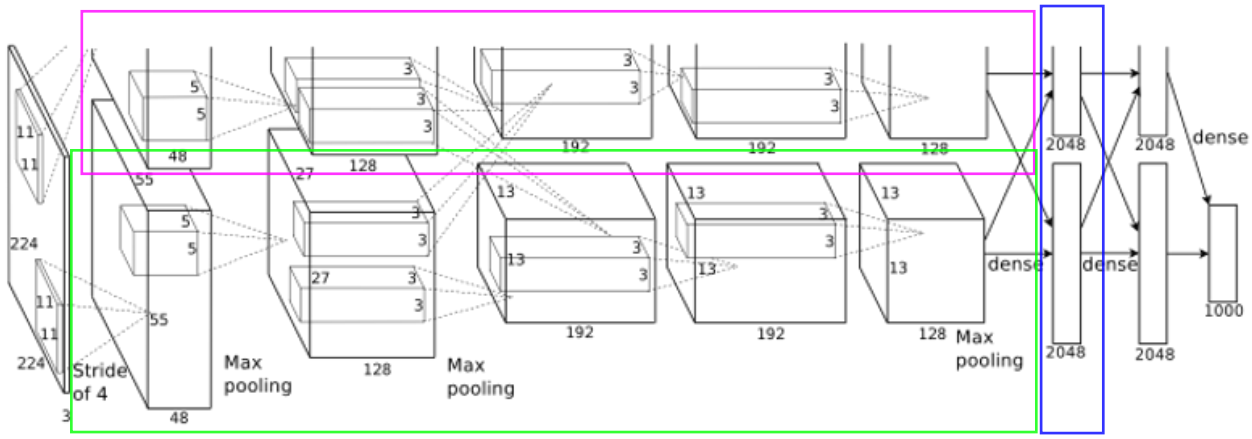


Image top-5 classification error rate on ILSVRC (%)

best performance increasing the depth of the network

"Famous" CNNs

AlexNet - Winner of ILSVRC 2012



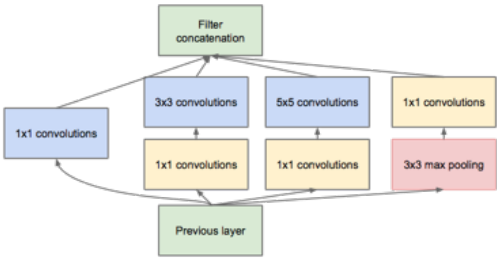
- Reached 16.4% top-5 image classification error on ILSVRC 2012
- Large gap from 25.8% top-5 error, best result of ILSVRC 2011
- Not commonly used anymore

"Famous" CNNs

GoogLeNet/Inception - Winner of ILSVRC 2014

many more layers

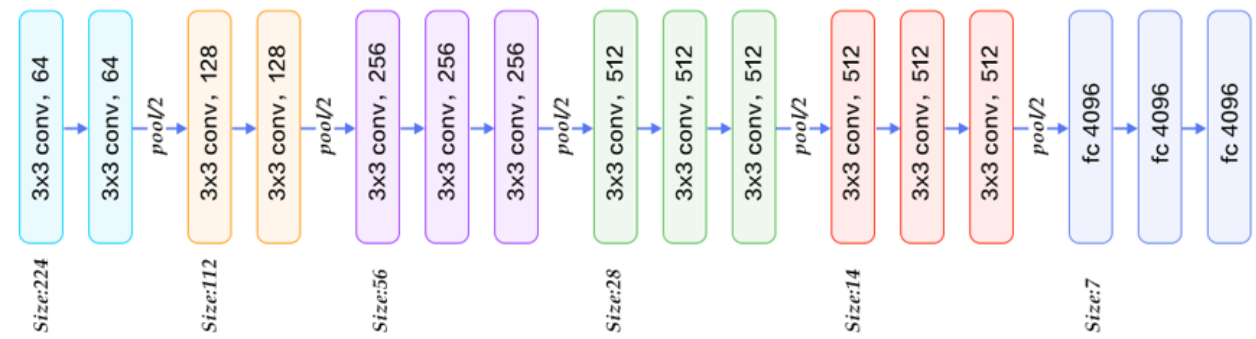
Inception module:



- Reached 6.7% top-5 image classification error on ILSVRC 2012
- Updated versions commonly used also today (Inception v3 and v4)

"Famous" CNNs

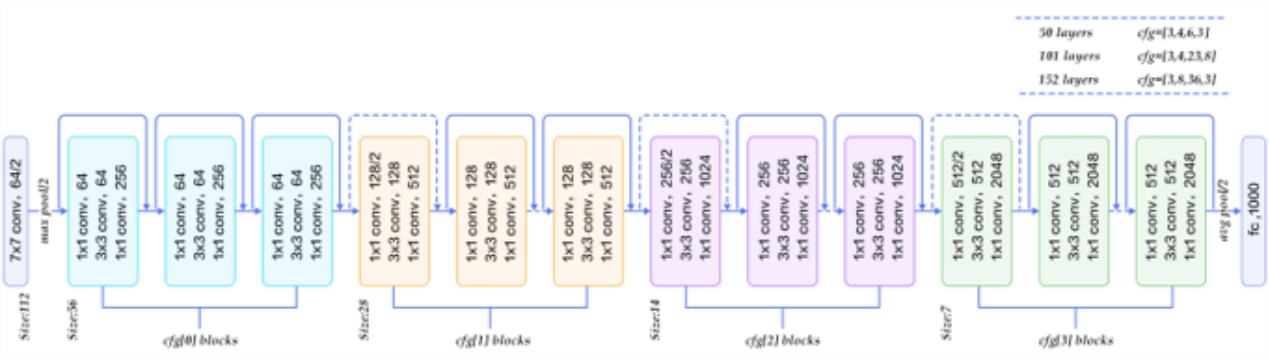
VGG - 1st Runner-Up of ILSVRC 2014



- Reached 7.3% top-5 image classification error on ILSVRC 2012
- Commonly used also today

"Famous" CNNs

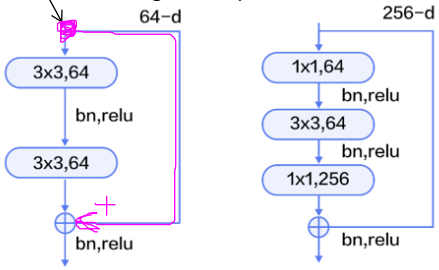
ResNet - Winner of ILSVRC 2015



- Reached 3.6% top-5 image classification error on ILSVRC 2012
- Commonly used also today (various versions e.g. 50, 101, 152 layers)

Residual Layers

this allow to speed up the process and increasing the depth



(Use of skip connections)

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)

Examples: `https://keras.io/applications/`

Transfer Learning

the idea is to understand how can we exploit the Learning on a different task for having a new task

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

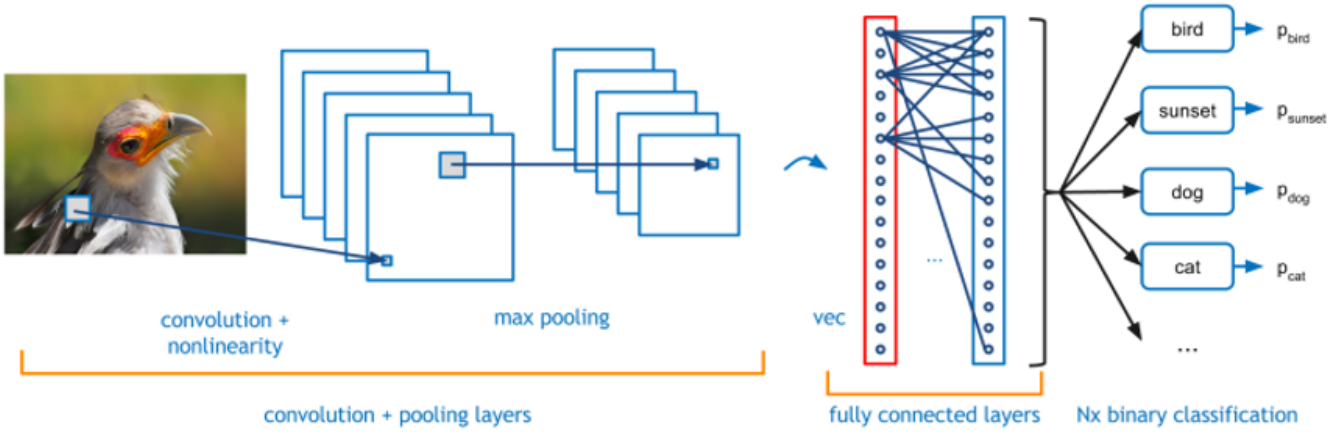
- D_S and T_S a source domain and learning task
- D_T and T_T a target domain and learning task
- In general, $D_S \neq D_T$ and $T_S \neq T_T$

Goal

improve learning of $f_T : \mathbf{X}_T \to \mathbf{Y}_T$ using knowledge in D_S and T_S (i.e., after training $f_S : \mathbf{X}_S \to \mathbf{Y}_S$)

Transfer Learning - Example

Image classification using a CNN



CNN pre-trained on Imagenet

- millions of images (source domain)
- classification of 1000 classes (source learning task)

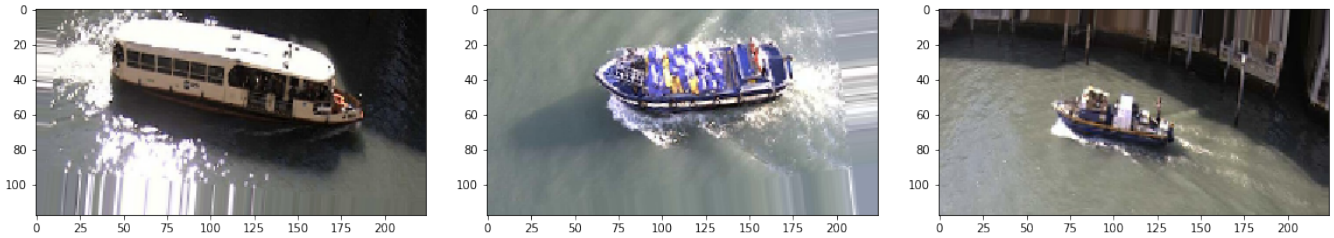
Slide from from Caffe framework tutorial @CVPR2015

Transfer Learning - Example

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:

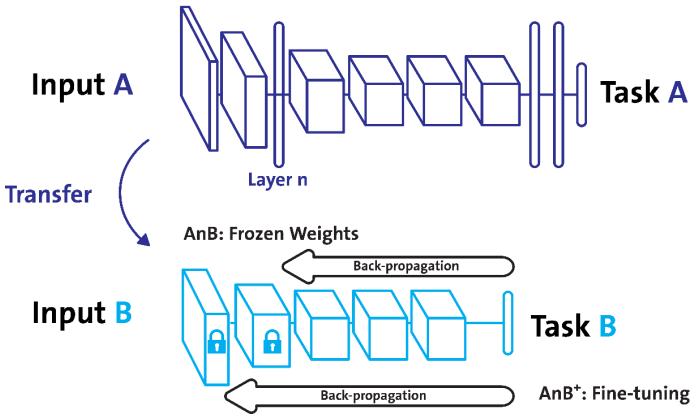
- thousands of boat images (target domain)
- classification of 20 boat classes (target learning task)



Transfer Learning - Example

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

Image from P. Gudikandula's blog

Transfer Learning - Example

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

Sapienza University of Rome, Italy - Machine Learning (2020/2021)

Transfer Learning - Example

2nd solution - CNN as feature extractor

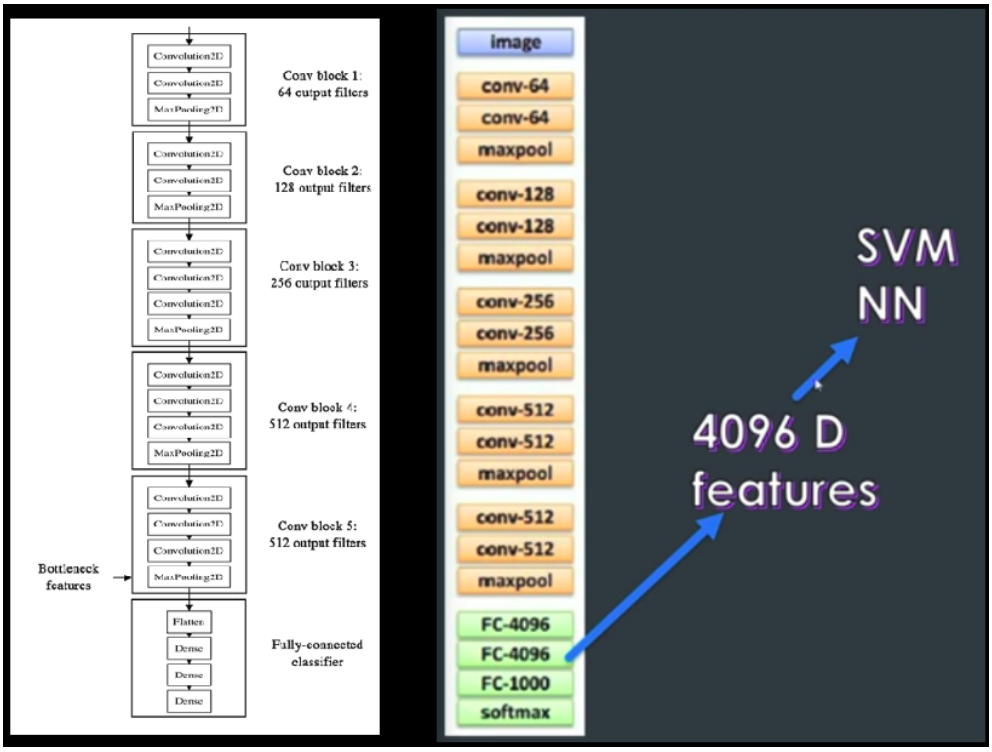


Image from P. Gudikandula's blog

Resources

Frameworks

- Caffe/Caffe 2 (UC Berkeley) - C/C++, Python, Matlab
- TensorFlow (Google) - C/C++, Python, Java, Go
- Theano (U Montreal) - Python
- CNTK (Microsoft) - Python, C++ , C#/.Net, Java
- Torch/PyTorch (Facebook) - Lua/Python
- MxNet (DMLC) - Python, C++, R, Perl,
- Darknet (Redmon J.) - C

High-level application libraries and models

- Keras
- TFLearn

Datasets

- `https://www.tensorflow.org/datasets`
- `https://www.kaggle.com/`

Summary

- CNNs are deep networks using convolution
- Very efficient (memory and generalization accuracy)
- Many successful examples
- Requires very large amount of data and very high computational resources