Sapienza University of Rome

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

Machine Learning

A.Y. 2019/2020

Prof. L. Iocchi, F. Patrizi, V. Ntouskos

12. Convolutional Neural Networks

L. Iocchi, F. Patrizi, V. Ntouskos

Overview

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

References

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

Motivation

Up to now we treated inputs as general feature vectors

In some cases inputs have special structure:

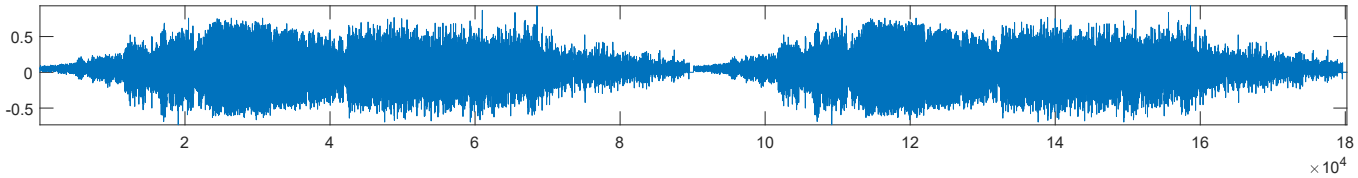
- Audio
- Images
- Videos

Signals: Numerical representations of physical quantities

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

Motivation - Examples

Audio

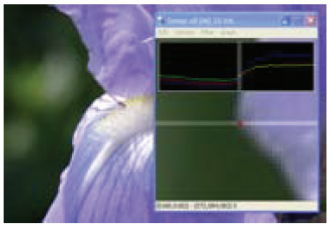


...	0.0468	0.0468	0.0468	0.0390	0.0390	0.0390	0.0546	0.0625	0.0625	0.0390	0.0312	0.0468	0.0625	...

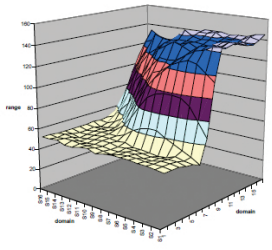
Note: variable length 1D vectors (1D tensor)

Motivation - Examples

Images



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

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

Discrete functions

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

Discrete limited 2D functions:

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

I: input, K: kernel

Convolution

Commutative

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

Cross-correlation

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

implemented in machine learning libraries (called convolution).

Image Convolutions

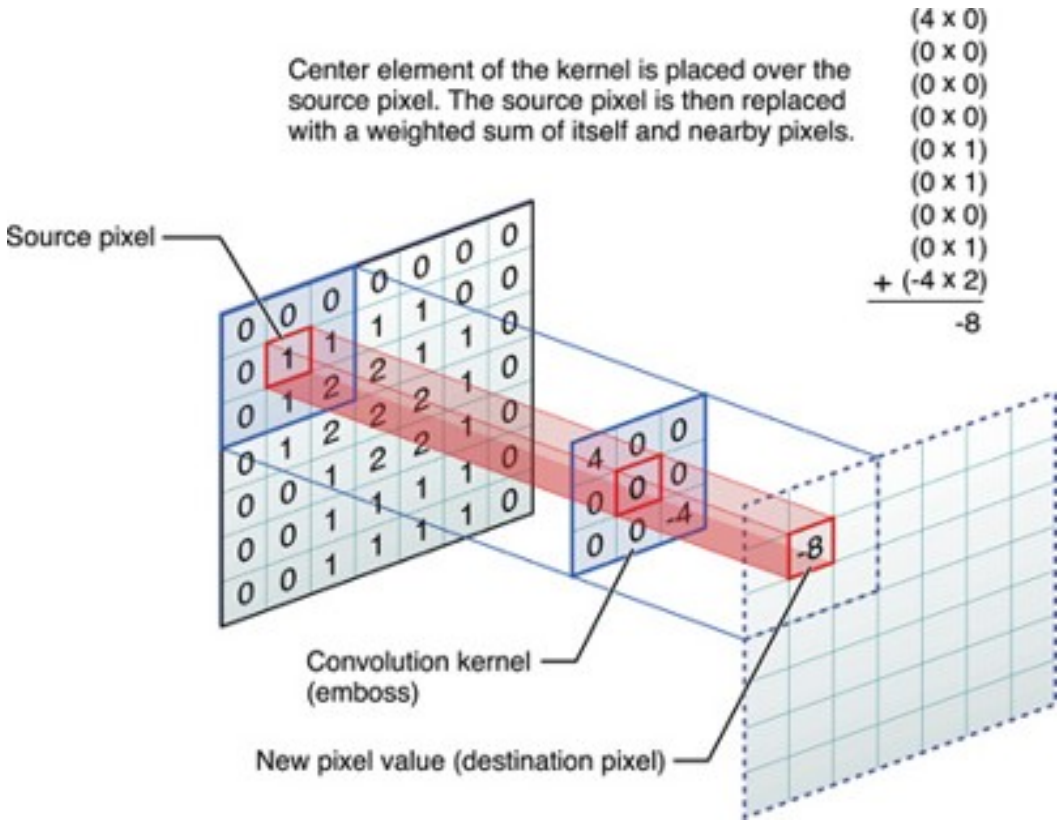


Image Convolutions

Image filtering is based on convolution with special kernels

Original Emboss

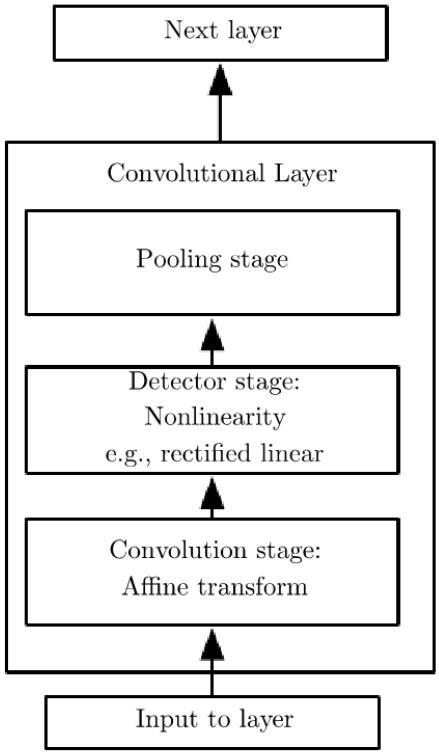
Interactive example: `http://setosa.io/ev/image-kernels/`

Convolutional Neural Networks

FNN with one or more convolutional layers.

Convolutional layer
Three stages:

- convolutions between input and kernel
- non-linear activation function (detector)
- pooling



Convolutional layer: Convolution

1. Convolution stage

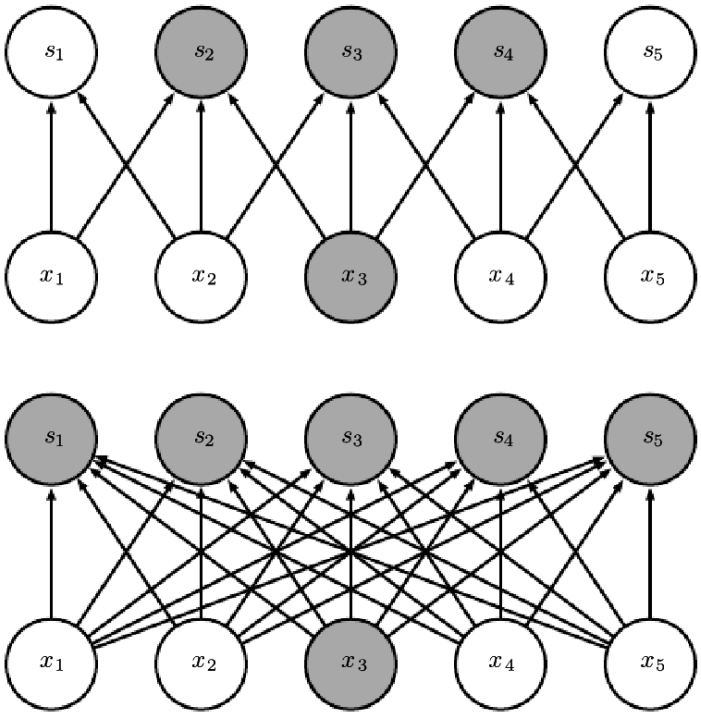
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Very efficient in terms of memory requirements and generalization accuracy.

- Sparse connectivity
- Parameter sharing

Sparse connectivity

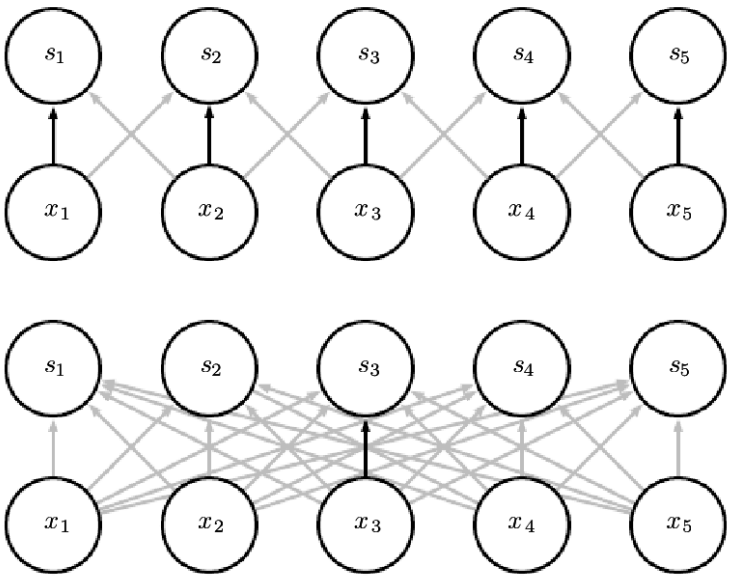
sparse interactions/ sparse connectivity: outputs depend only on a few inputs (as kernel is usually much smaller than input)



Parameter sharing

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

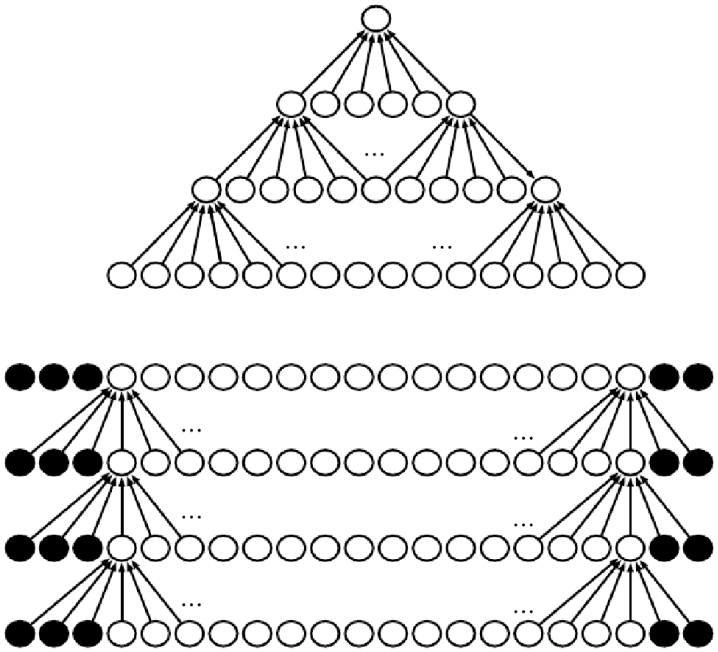
k parameters instead of $m \times n$ (note: $k \ll m$)



Padding

valid padding: output contains only valid values (depends on kernel size)

same padding: input layer is padded with zeros, output size independent of kernel size.



Convolutional layer: Detector

2. Detector stage

Use non-linear activation functions.

- ReLU
- tanh
- ...

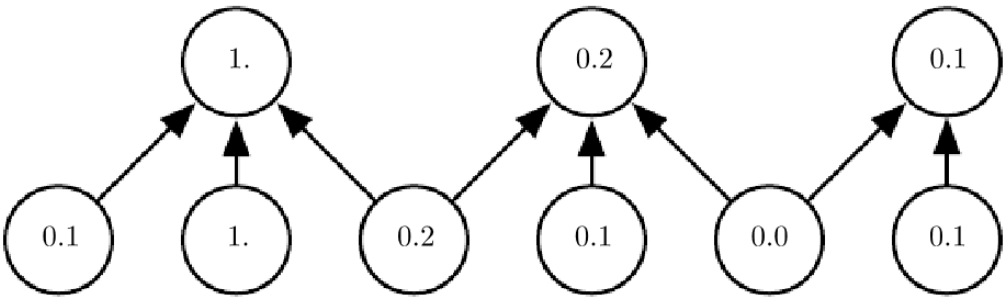
Convolutional layer: Pooling

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.

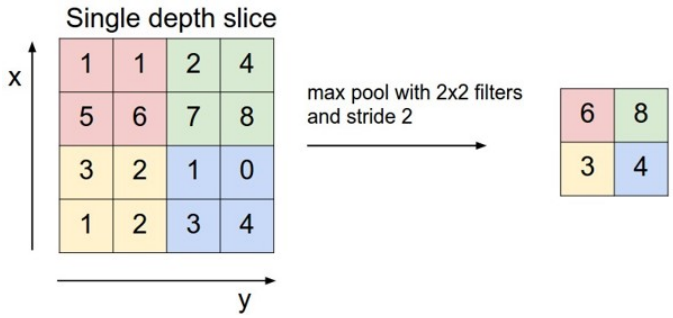
When applied with *stride*, it reduces the size of the output layer.

Example: pool width 3 and stride 2

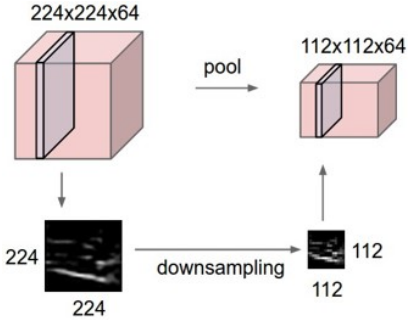


Convolutional layer: Pooling

Example of **max pooling***

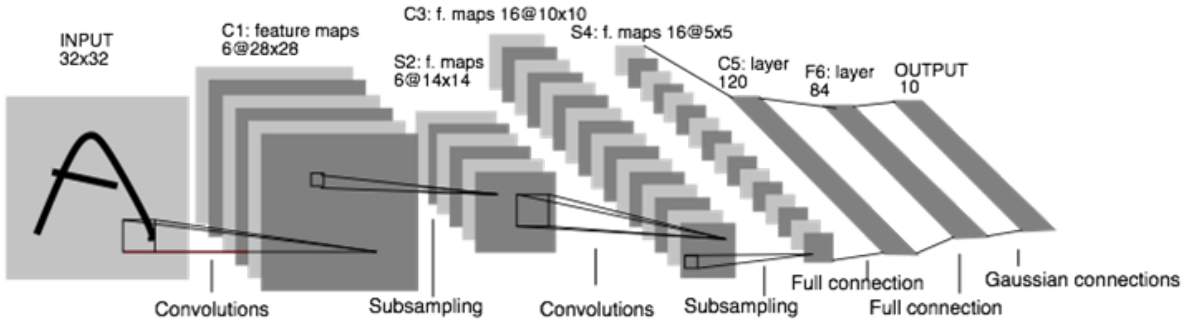


Introduces subsampling:



*How does back-propagation work?

LeNet



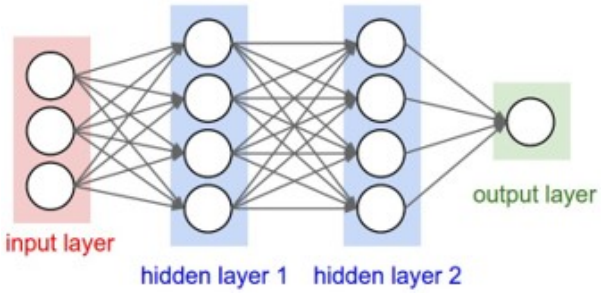
Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998

Handwritten recognition with LeNet

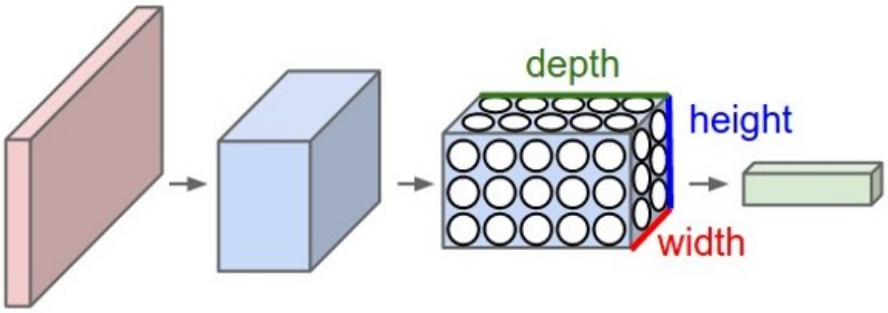
MNIST database

Accuracy up to 98 %.

CNNs for Images (2D input)



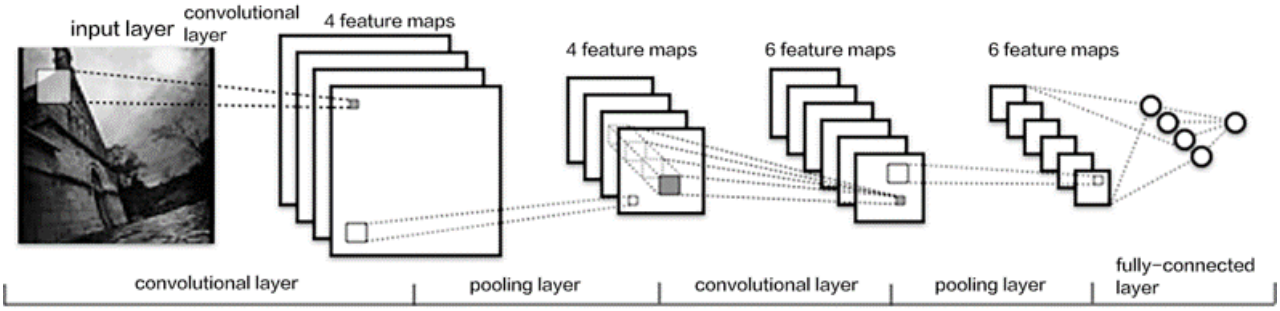
A regular 3-layer Neural Network

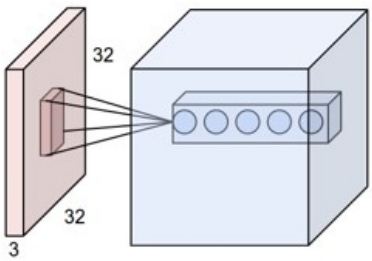


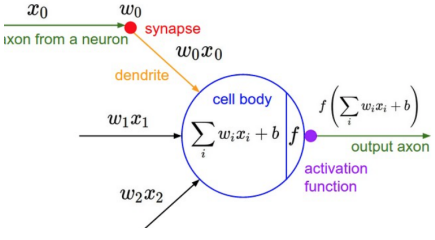
Every convolutional layer of a CNN transforms the 3D input volume to a 3D output volume of neuron activations.

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

Terminology

Kernel matrix used for convolution / filtering

Kernel size 2D (horizontal) dimensions of the kernel ($w_k \times h_k$)

Depth number of feature maps / filters (d)

Padding filled addition outer rows/columns (typically zeros) (p)

Stride step of sliding kernel (s) - e.g. value 1 does not skip any pixels

Depth slice a single feature map

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

Feature map size - Number of parameters

Let $w_{in} \times h_{in}$ the dimensions of the feature map at the previous convolutional layer and d_{in}, d_{out} be the input and output depth, respectively

Dimensions of output feature map are given by:

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

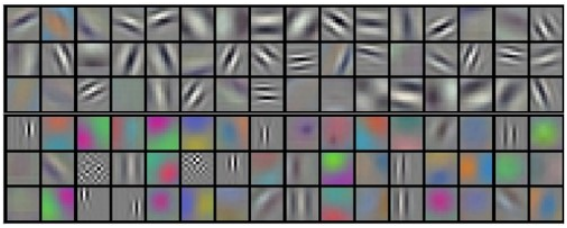
$$h_{out} = \frac{h_{in} - h_k + 2p}{s} + 1$$

The number of parameters of the convolutional layer is:

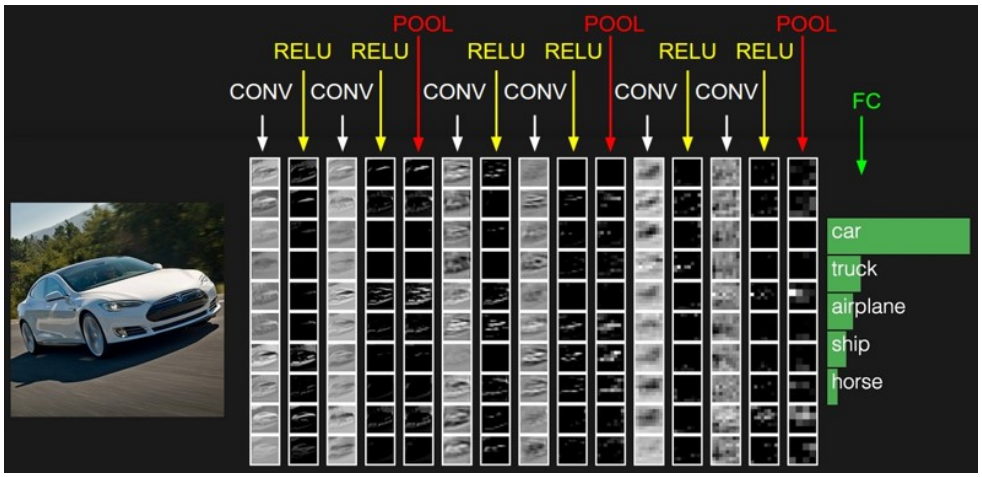
$$|\theta| = \underbrace{w_k \cdot y_k \cdot d_{in} \cdot d_{out}}_{\text{kernel weights}} + \underbrace{d_{out}}_{\text{bias}}$$

Kernels and Feature maps - Example

Visualization of learned kernels of the first layer:



Computed activations during forward pass:



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

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)

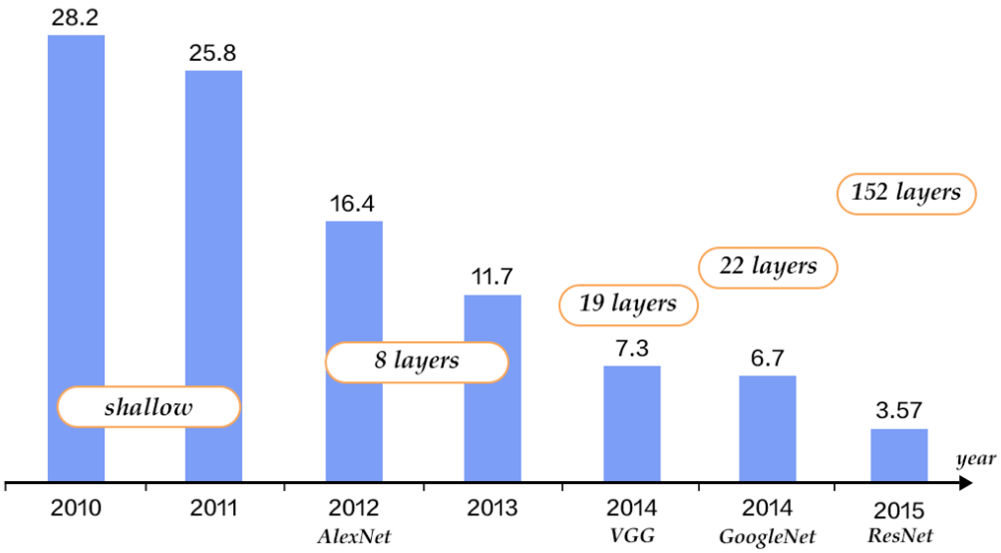
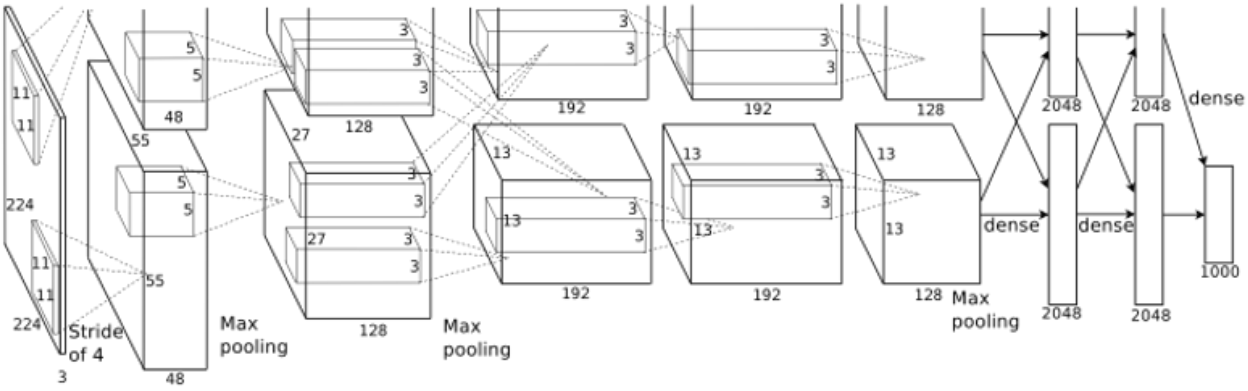


Image top-5 classification error rate on ILSVRC (%)

"Famous" CNNs

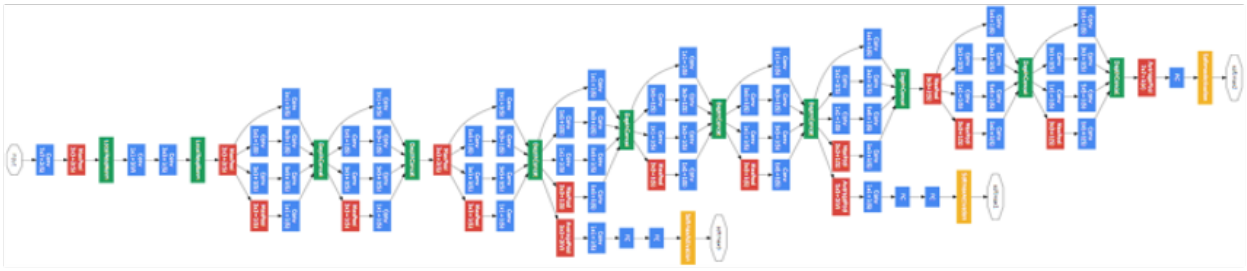
AlexNet - Winner of ILSVRC 2012



- 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

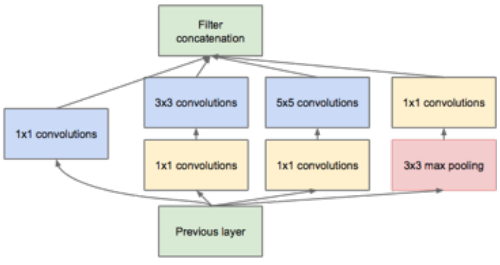
"Famous" CNNs

GoogLeNet/Inception - Winner of ILSVRC 2014



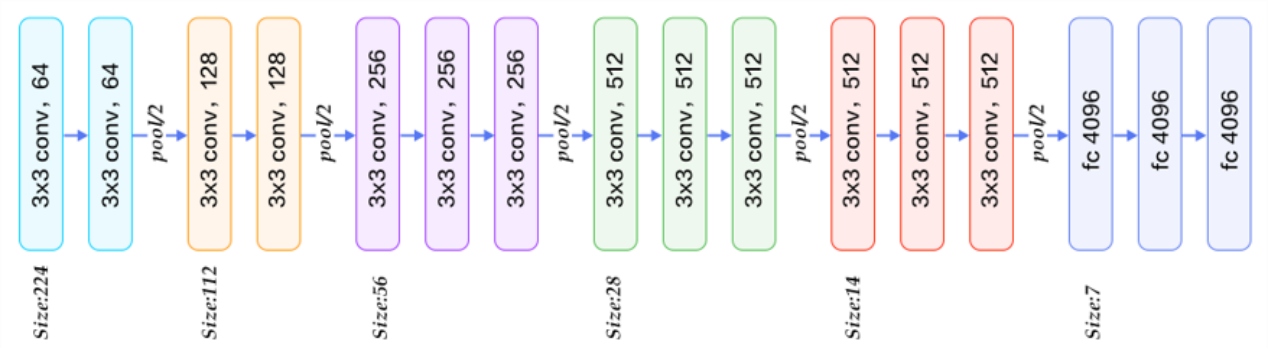
Inception module:

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



"Famous" CNNs

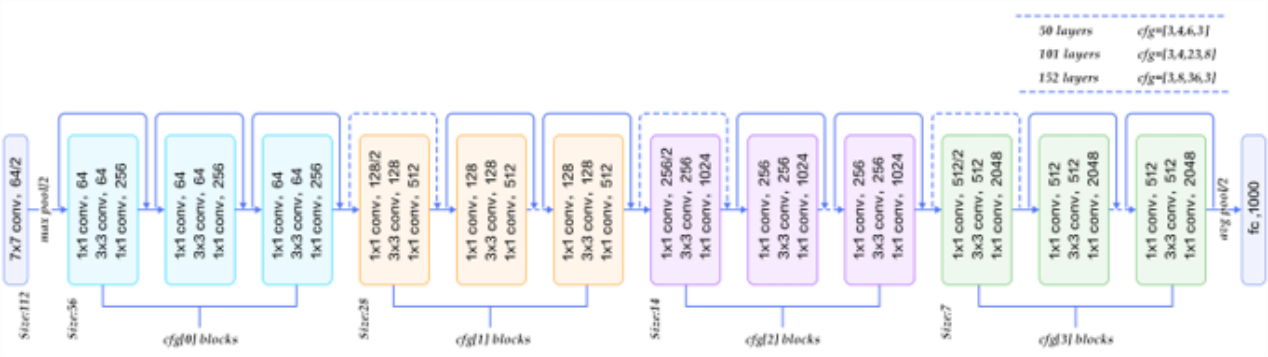
VGG - 1st Runner-Up of ILSVRC 2014



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

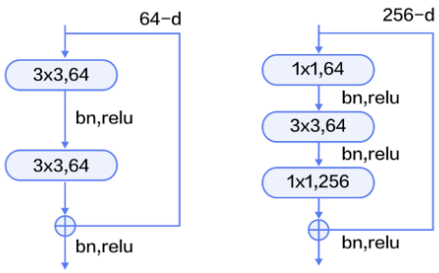
"Famous" CNNs

ResNet - Winner of ILSVRC 2015



- Intuition: Zero function is easier to learn with respect to the identity
- 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



(Use of skip connections)

Common uses of famous CNNs

- Train the model on a new 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 set of classes

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

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

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.