Lesson 5

- Size, stride, padding, depth
- Examples of real CNNs
- Transfer Learning

Next argument

Tensors for 2D processing

Tensors

Convolutional Networks process Tensors. A Tensor is just an multidimensional array of floating numbers.

The typical tensor for 2D images has **four** dimensions:

$$batchsize \times width \times height \times channels$$

Features maps are **stacked** along the channel dimension.

At start, for a color image, we just have 3 channels: r,g,b.

How do kernel operate along the channel dimension?

Tensors

Convolutional Networks process Tensors. A Tensor is just an multidimensional array of floating numbers.

The typical tensor for 2D images has **four** dimensions:

$$batchsize \times width \times height \times channels$$

Features maps are **stacked** along the channel dimension.

At start, for a color image, we just have 3 channels: r,g,b.

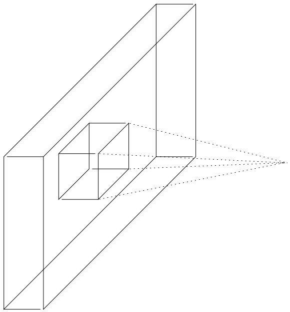
How do kernel operate along the channel dimension?

Dense processing along channel axis

Unless stated differently (e.g. in separable convolutions), a filter operates on all input channels in parallel.

So, if the input layer has depth D, and the kernel spatial size is NxM, the actual dimension of the kernel will be

$$NxMxD$$



The convolution kernel is tasked with simultaneously mapping cross-channel correlations and spatial correlations

Spatial dimension of the resulting feature map

Each kernel produces a single feature map.
Feature maps produced by different kernels are stacked along the channel dimension: the number of kernels is equal to the channel-**depth** of the next layer.

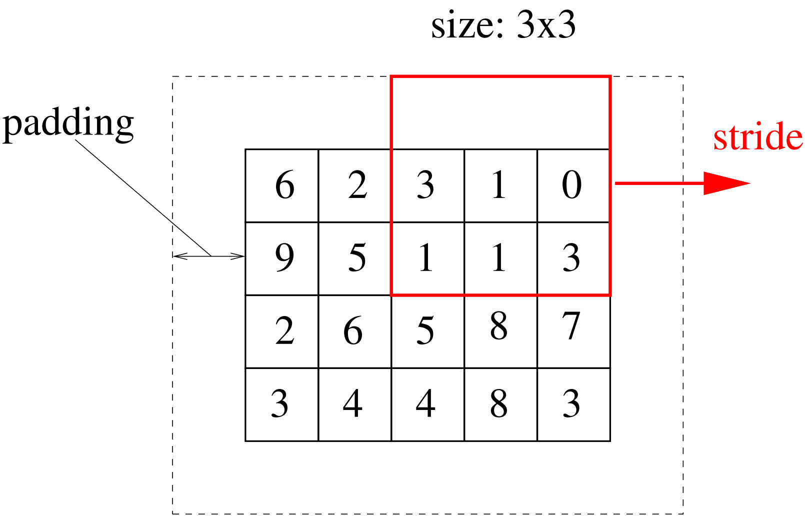
The **spatial** dimension of the feature map depends from two configurable factors:

- **paddding**: extra space added around the input
- **stride**: kernel deplacement over the input during convolution

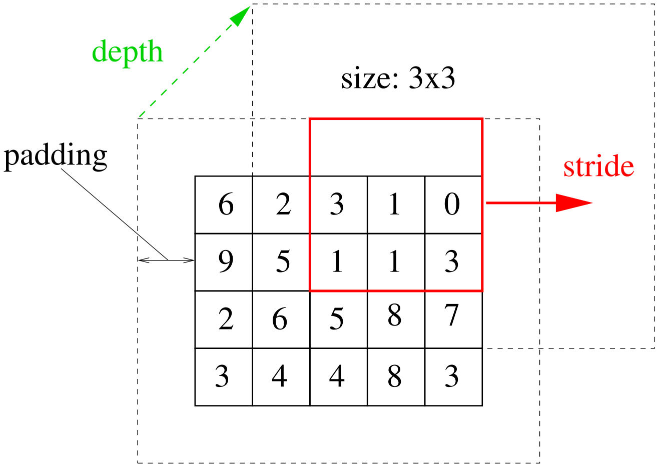
Relevant parameters for convolutional layers

- **kernel size**: the dimension of the linear filter.
- **stride**: movement of the linear filter. With a low stride (e.g. unitary) receptive fields largely overlap. With a higher stride, we have less overlap and the dimension of the output get smaller (lower sampling rate).
- **padding** Artificial enlargement of the input to allow the application of filters on borders.
- **depth**: number of features maps (stacked along the so called channel axis) that are processed in parallel.
  The depth of the output layer depends from the number of different kernels that we want to synthesize (each producing a different feature map).

Configuration params for conv2D layers



Configuration params for conv2D layers



Input-output spatial relation

Along each axes the dimension of the output is given by the following formula

$$\frac{W + P - K}{S} + 1$$

where:

W = dimension of the input
P = padding
K = Kernel size
S = Stride

An example

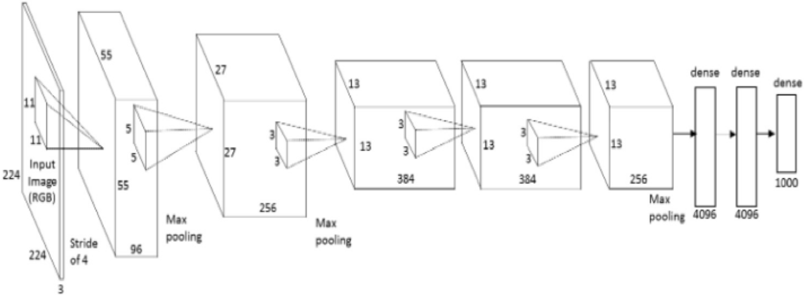
[DEMO]

Next argument

Important networks

AlexNet

AlexNet Architecture (Krizhevsky, Sutskever e Hinton), winner of a NIPS contest in 2012.



Pooling

In deep convolutional networks, it is common practice to alternate convolutional layers with pooling layers, where each neuron simply takes the mean or maximal value in its receptive field.

This has a double advantage:

- it reduces the dimension of the output
- it gives some tolerance to translations

Max Pooling example

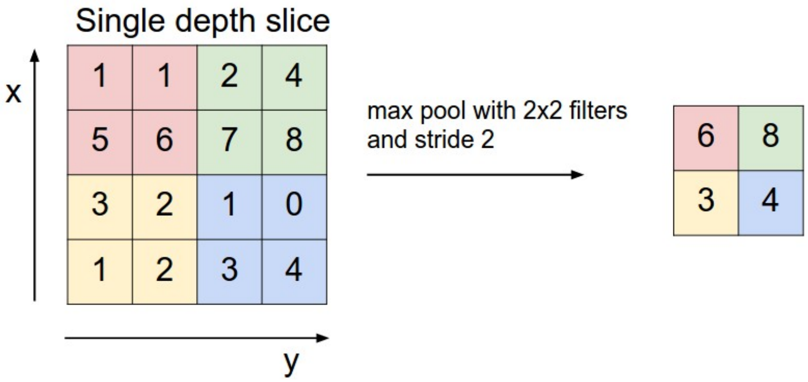
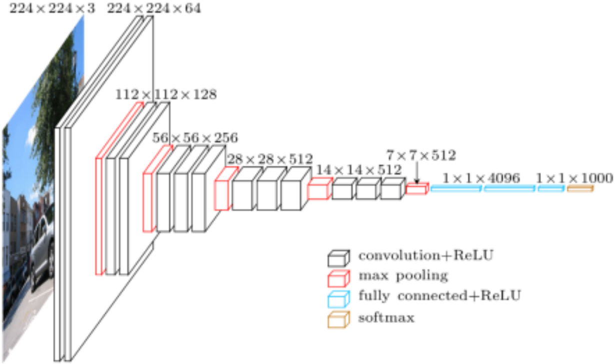


Immagine tratta da
`http://cs231n.github.io/convolutional-networks/`

VGG

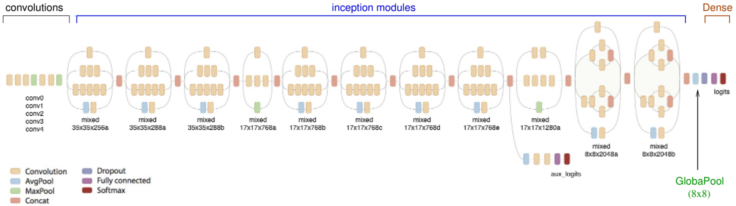
VGG 16 (Simonyan e Zisserman). 92.7 accuracy (top-5) in ImageNet (14 millions images, 1000 categories).



Picture by Davi Frossard: VGG in TensorFlow

Inception V3

Inception V3

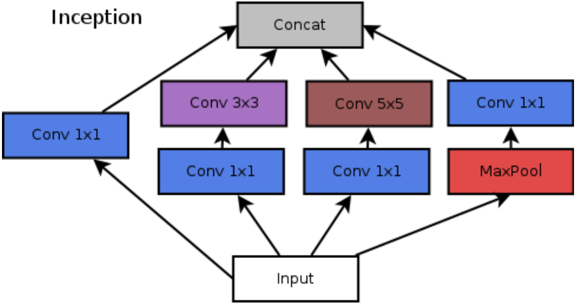


The convolutional part is a long composition of

inception modules

Inception modules

The networks is composed of inception modules (towers of nets):

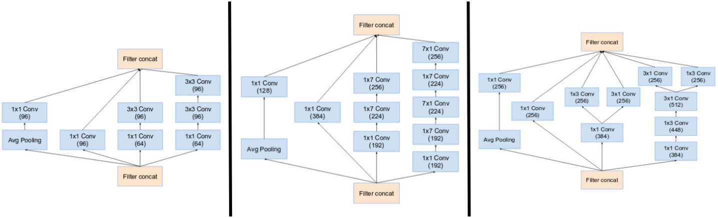


Video from the Udacity course "Deep Learning"

Variants

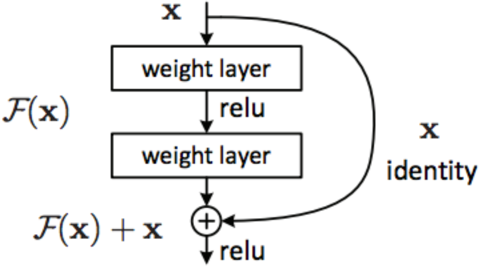
The point is to induce the net to learn different filters.

Many variants proposed and used over years:



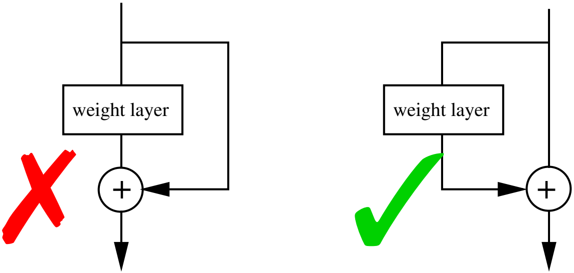
Residual Learning

Another recent topic is residual learning.

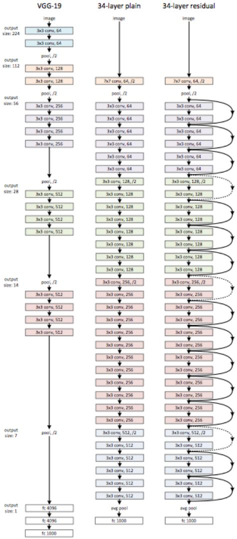


Instead of lerning a function $\mathcal{F}(x)$ you try to learn $\mathcal{F}(x) + x$.

The right intuition



Residual networks



you add a residual shortcut connection every 2-3 layers

Inception Resnet is an example of a such an architecture

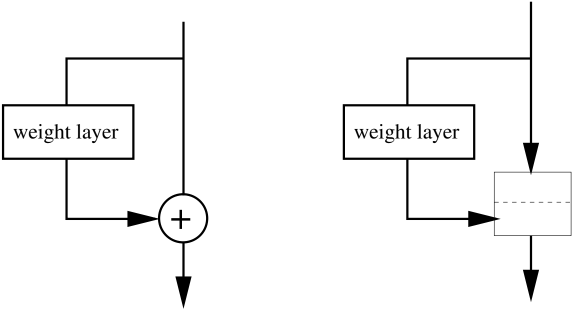
Why Residual Learning works?

Not well understood yet.

The usual explanation is that during back propagation, **the gradient at higher layers can easily pass to lower layers**, withouth being mediated by the weight layers, which may cause vanishing gradient or exploding gradient problem.

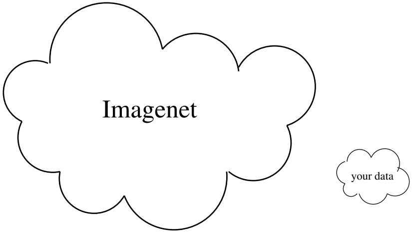
Sum or concatenation?

The "sum" operation can be interpreted in a liberal way.
A common variant consists in concatenating instead of adding (usually along the channel axis):



Next arguments

Transfer Learning



Reusing Knowledge

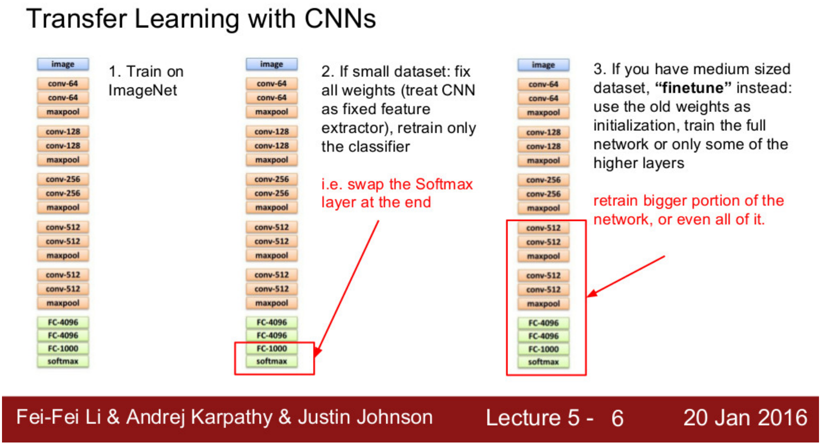
We learned that the first layers of convolutional networks for computer vision compute feature maps of the original image of growing complexity.

The filters that have been learned (in particular, the most primitive ones) are likely to be independent from the particular kind of images they have been trained on.

They have been trained on a huge amount of data and are probably very good.

It is a good idea to try to *reuse them* for other classification tasks.

Transfer Learning with CNNs



When Transfer Learning makes sense

transferring knowledge from problem A to problem B makes sense if

- the two problems have “similar” inputs
- we have much more training data for A than for B

What we may expect

Faster and more accurate training