Deep Learning and Temporal Data Processing

2 - Convolutional Neural Networks

Andrea Palazzi

June 21th, 2017

University of Modena and Reggio Emilia

Agenda

Introduction

Architecture

Credits

References

Introduction

Convolutional Neural Networks are very similar to ordinary Neural Networks.

- They are made up of neurons that have learnable weights and biases.
- Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity.
- The whole network still expresses a single differentiable function.

However, **CNNs make the explicit assumption that inputs are images**.

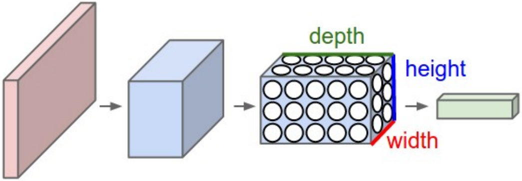
- This architecture constraint paves the way to more efficient implementation, better performance and a vastly reduced amount of learnable parameters *w.r.t.* fully-connected deep networks.

Most important peculiarities of CNNs are presented in the following slides.

Architecture

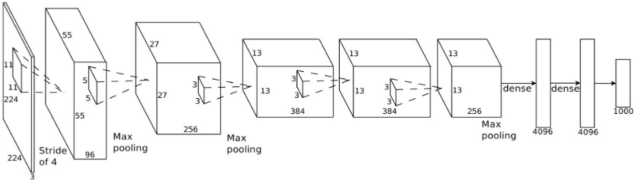
CNN Architecture

Unlike a regular neural network, CNN layers have neurons arranged in 3 dimensions: width (W), height (H) and depth (C).



Achtung: in the following we'll refer to the word *depth* to indicate the number of channels of an activation volume. This has nothing to do with the depth of the whole network, which usually refers to the total number of layers in the network.

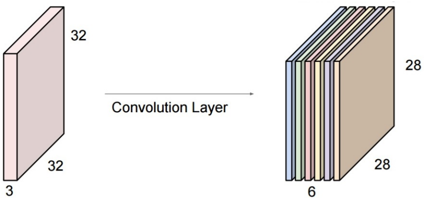
An "real-world" CNN is made up by a whole bunch of layers stacked one on the top of the other.



Every layer has a simple API: *it transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.*

The **Convolutional Layer** is the core building block of convolutional neural networks.

Intuition: every convolutional layer is equipped with a set of learnable filters. During the forward pass, each filter is convolved with the input volume thus producing a 2D activation map. One map for each filter is produced. The output volume is then made up by stacking all activation maps produced one on the top of the other.



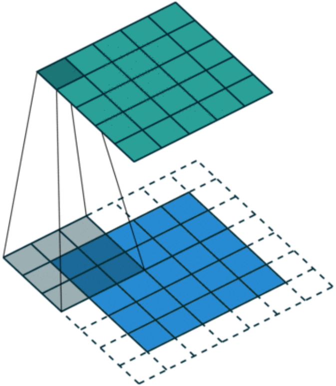
e.g. Result of $N = 6$ filters of kernel size $K = 5\text{x}5$ convolved on input image.

Each convolutional layer has three main hyperparameters:

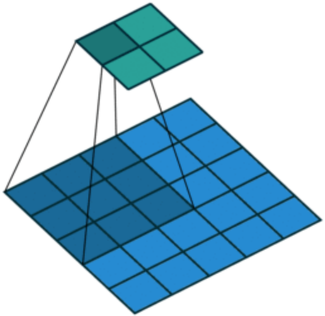
- Number of filters N
- Kernel size K, the spatial size of the filters convolved
- Filter stride S, factor by which to downscale

The presence and amount of spatial padding P on the input volume may be considered an additional hyperparameter. In practice padding is usually performed to avoid headaches caused by convolutions "eating the borders".

Convolution 2D, half padding, stride $S = 1$.

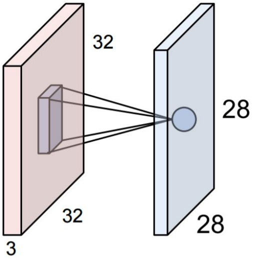


Convolution 2D, no padding, stride $S = 2$.



Looking closer, neurons in a CNN perform the very same operation of the neurons we already know from DNN.

$$\sum_i w_i x_i + b$$

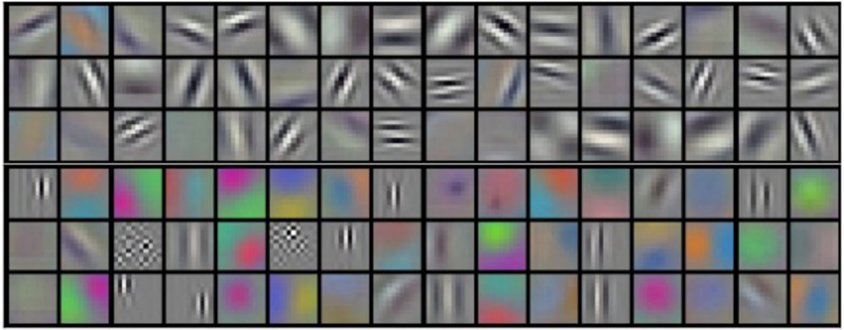


However, in convolutional layers neurons are only locally connected to the input volume. The small region that each neuron "sees" of the previous layer is usually referred to as the *receptive field* of the neuron.

Convolutional Layers: Parameter Sharing

Assumption: if a feature is useful to compute at some spatial location (x, y), then it should be useful to compute also at different locations (x_i, y_i). **Thus, we constrain the neurons in each depth slice to use the same weights and bias.**

If all neurons in a single depth slice are using the same weight vector, then the forward pass of the convolutional layer can *in each depth slice* be computed as a convolution of the neuron's weights with the input volume (hence the name). This is why it is common to refer to each set of weights as a filter (or a kernel), that is convolved with the input.



Example of weights learned by [4]. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images.

Given an input volume of size H_1 x W_1 x C_1, the number of **learnable parameters** of a convolutional layer with N filters and kernel size KxK is:

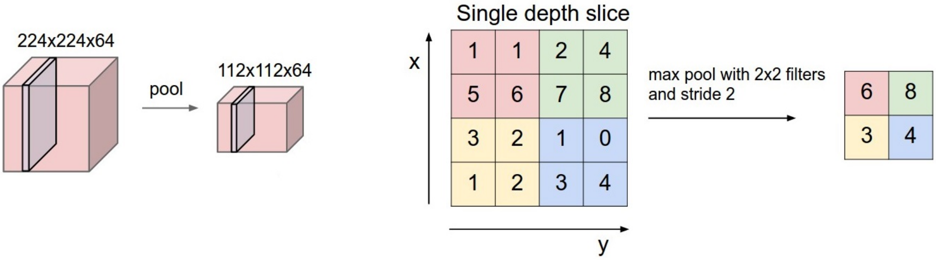
$$tot\_learnable = N * K * K * C_1 + N$$

Explanation: there are N filters which convolve on input volume. The neural connection is local on width and height, but extends for the full depth of input volume, so there are $K * K * C_1$ parameters for each filter. Furthermore, each filter has an additive learnable bias.

Pooling Layers: overview

Pooling layers spatially subsample the input volume.

Each depth slice of the input is processed independently.



Two hyperparameters:

- Pool size K, which is the size of the pooling window
- Pool stride S, which is the factor by which to downscale

The pooling function may be considered an additional hyperparameter.

In principle, many different functions could be used.

In practice, the **max** pooling is by far the most common

$$h_i^n(x, y) = max_{\bar{x}, \bar{y} \in N(x,y)} h_i^{n-1}(\bar{x}, \bar{y})$$

Another common pooling function is the **average**

$$h_i^n(x, y) = \frac{1}{K} \sum_{\bar{x}, \bar{y} \in N(x,y)} h_i^{n-1}(\bar{x}, \bar{y})$$

Pooling Layers: why

Pooling layers are widely used for a number of reasons:

- Gain robustness to exact location of the features
- Reduce computational (memory) cost
- Help preventing overfitting
- Increase receptive field of following layers

Most common configuration: pool size $K = 2\text{x}2$, stride $S = 2$.
In this setting 75% of input volume activations are discarded.

Pooling Layers: why not

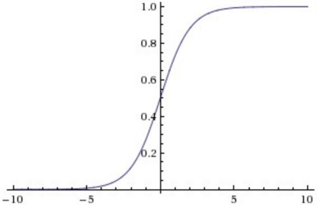
The loss of spatial resolution is not always beneficial.

e.g. semantic segmentation

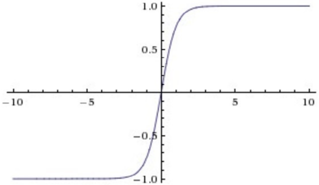
There's a lot of research on getting rid of pooling layers while mantaining the benefits (*e.g.* [6, 8]). We'll see if future architecture will still feature pooling layers.

Activation Layers

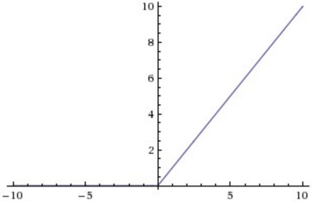
Activation layers compute non-linear activation function elementwise on the input volume. The most common activations are **ReLu**, **sigmoid** and **tanh**.



Sigmoid



Tanh



ReLu

Nonetheless, more complex activation functions exist [2, 1].

ReLu wins

ReLu was found to greatly accelerate the convergence of SGD compared to sigmoid/tanh functions [4]. Furthermore, ReLu can be implemented by a simple threshold, *w.r.t.* other activations which require complex operations.

Why using non-linear activations at all?

Composition of linear functions is a linear function. Without nonlinearities, neural networks would reduce to 1 layer logistic regression.

Computing Output Volume Size

Convolutional layer: given an input volume of size H_1 x W_1 x C_1, the output of a convolutional layer with N filters, kernel size K, stride S and zero padding P is a volume with new shape H_2 x W_2 x C_2, where:

- $H_2 = (H_1 - K + 2P)/S + 1$
- $W_2 = (W_1 - K + 2P)/S + 1$
- $C_2 = N$

Computing Output Volume Size

Pooling layer: given an input volume of size H_1 x W_1 x C_1, the output of a pooling layer with pool size K and pool stride S is a volume with new shape H_2 x W_2 x C_2, where:

- $H_2 = (H_1 - K)/S + 1$
- $W_2 = (W_1 - K)/S + 1$
- $C_2 = C_1$

Activation layer: given an input volume of size H_1 x W_1 x C_1, the output of an activation layer is a volume with shape H_2 x W_2 x C_2, where:

- $H_2 = H_1$
- $W_2 = W_1$
- $C_2 = C_1$

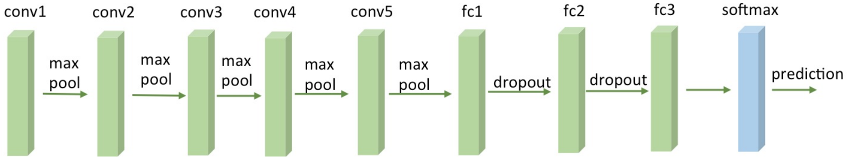
More complex CNN architectures have recently been demonstrated to perform better than the traditional `conv -> relu -> pool` stack architecture.

These architectures usually feature different graph topologies and much more intricate connectivity structures (*e.g.* [3, 7]).

However, these advanced architectures are out of the scope of these lectures.

Computational Footprint

If we consider a standard convolutional network like VGG [5] we can make a couple of considerations on the computational footprint.



- Most of the memory is taken by the very first layers
- Most of parameters (70%) are condensated in the last fully-connected layers

Can you justify why?

Credits

These slides heavily borrow from a number of awesome sources. I'm really grateful to all the people who take the time to share their knowledge on this subject with others.

In particular:

- Stanford CS231n Convolutional Neural Networks for Visual Recognition
  http://cs231n.stanford.edu/
- Deep Learning Book (GoodFellow, Bengio, Courville)
  http://www.deeplearningbook.org/
- Convolution arithmetic animations
  https://github.com/vdumoulin/conv_arithmetic

- Andrej Karphathy personal blog
  http://karpathy.github.io/
- WildML blog on AI, DL and NLP
  http://www.wildml.com/
- Michael Nielsen Deep Learning online book
  http://neuralnetworksanddeeplearning.com/

References

[1] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio.
Maxout networks.
arXiv preprint arXiv:1302.4389, 2013.

[2] K. He, X. Zhang, S. Ren, and J. Sun.
Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.
In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton.
Imagenet classification with deep convolutional neural networks.
In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] K. Simonyan and A. Zisserman.
Very deep convolutional networks for large-scale image recognition.
arXiv preprint arXiv:1409.1556, 2014.

[6] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller.
Striving for simplicity: The all convolutional net.
arXiv preprint arXiv:1412.6806, 2014.

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna.
Rethinking the inception architecture for computer vision.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[8] F. Yu and V. Koltun.
Multi-scale context aggregation by dilated convolutions.
arXiv preprint arXiv:1511.07122, 2015.