Deep Learning

Marc Lelarge, Jill-Jenn Vie, and Kevin Scaman

Class notes by Antoine Groudiev

Last modified 21st June 2024

Contents

Abstract

This document is Antoine Groudiev's class notes while following the class *Deep Learning* at the Computer Science Department of ENS Ulm. It is freely inspired by the lectures of Marc Lelarge, Jill-Jênn Vie, and Kevin Scaman.

5 Convolutional Neural Networks

5.1 Introduction

Convolutional Neural Networks (CNNs) is a class of models widely used in computer vision. While Fully Connected Neural Networks are very powerful machine learning models, they do not respect the 2D spatial structure of the input images. For instance, training a Multilayer Perceptron on a dataset of 32×32 images required the model to start with a `Flatten` layer, that reshaped matrix images of size `(32, 32)` to flattened vectors of size `(1024, 1)`. Similarly, different color channels were handled separately, reshaping tensor images of dimensions `(32, 32, 3)` to `(3072, 1)`.

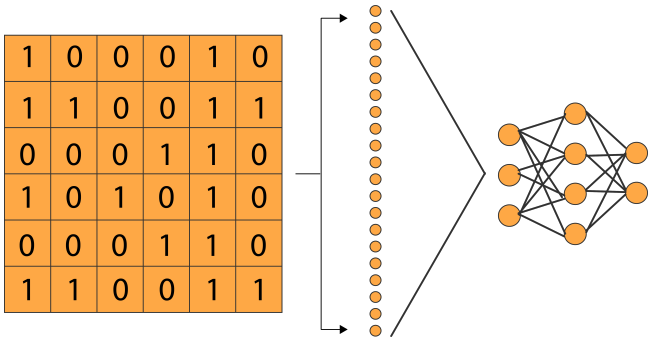

Figure 5.1: Flatten layer breaking the spatial strucutre of input data

CNNs introduce new operators taking advantage of the spatial structure of the input data, while remaining compatible with automatic differentiation. While MLPs build the basic blocks of Deep Neural Networks using Fully-Connected Layers and Activation Layers, this chapter will introduce three new types of layers: *Convolution Layers*, *Pooling Layers*, and *Normalization*.

5.2 Convolution Layers

Similarly to Fully-Connected Layers, *Convolution Layers* have learnable weights, but also have the particularity to respect the spatial information.

5.2.1 Input shape

A Fully-Connected layer receives some flattened vector and outputs another vector:

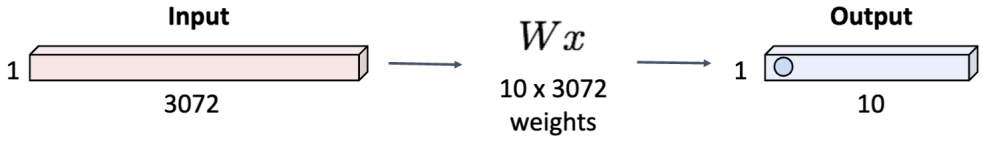

Figure 5.2: Fully-Connected Layer

Instead, a CNN takes as an input a 3D volume: for instance, an image can be represented as a tensor of shape $3 \times 32 \times 32$, the first dimension being the number of channels (red, green, blue), and the other two being the width and height of the image.

5.2.2 Kernels

The convolutional layer itself consists of small kernels (also called filters) used to *convolve* with the image, that is sliding over it spatially, and computing the dot products at each possible location.

Definition (Kernel)**.** A *kernel* (or *filter*) is a tensor of dimensions $D \times K \times K$, where D is the number of channels (or "depth") of the input, and K is a parameter called *kernel size*.

Definition (Convolution of two matrices)**.** Given two matrices $A = (a_{i,j})_{i,j}$ and $B = (b_{i,j})_{i,j}$ in $\mathscr{M}_{m,n}(\mathbb{R})$, the *convolution of A and B*, noted $A * B \in \mathbb{R}$, is the following:

$$A * B = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{(m-i+1),(n-j+1)} \cdot b_{i,j} \tag{5.2.1}$$

This corresponds to the dot product in the space $\mathscr{M}_{m,n}(\mathbb{R})$.

Definition (Kernel convolution)**.** An input of shape $D \times H \times W$ can be processed by a kernel of shape $D \times K \times K$ by computing at each possible spatial position the convolution between the kernel and the submatrix of the input.

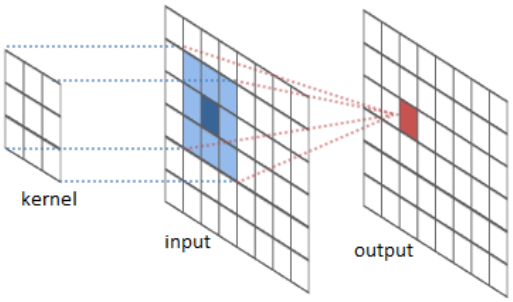

Figure 5.3: Kernel convolution

The output of this operation in an *activation map* of dimension $1 \times (H - K + 1) \times (W - K + 1)$ representing for each pixel the convolution between the kernel and corresponding chunk of the image.

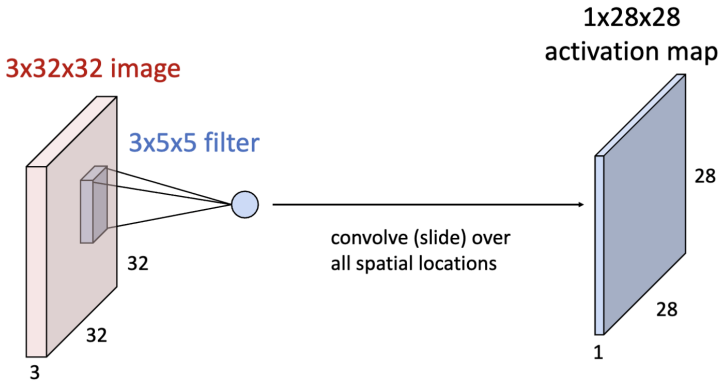

Figure 5.4: Input and output of the convolution operation

Intuitively, the result of the kernel convolution tells us for each pixel *how much the neighbourhood of the input pixel corresponds to the kernel.*

Example (Gaussian blur)**.** Let $G \in \mathscr{M}_3(\mathbb{R})$ be the following kernel:

$$G := \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 1 \\ 1 & 2 & 1 \end{bmatrix}$$

Each coefficient of this matrix is an approximation of the Gaussian distribution. Applying this kernel to an image produces a smoothed version of the input.

Example (Sobel operator)**.** Let S_x and $S_y \in \mathscr{M}_3(\mathbb{R})$ be the following kernels:

$$S_x := \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad S_y := S_x^\top = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The convolution between these operators and an image produces horizontal and vertical derivatives approximations of the image pixels.

Input image | Sobel operator applied to the image

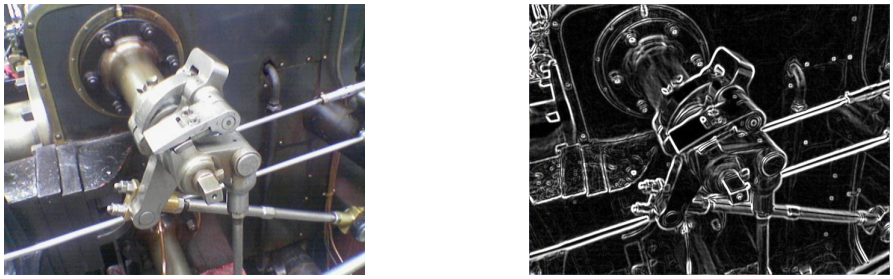

Figure 5.5: Effect of the Sobel operator on an image

These two examples show that kernels used in convolutional layers express meaningful transformations of the input, justifying their use in CNNs. For instance, one could hardcode different kernels (gaussian blur, Sobel operator, vertical/horizontal lines extraction) to extract interesting features from an image, and plug these features into an MLP to obtain an improved classifier compared to a basic, flattening MLP. We will see that instead, CNNs have learnable kernel weights, allowing the model to choose the kernels that it considers bests.

5.2.3 Multiple kernels

In Figure 5.4, we used simply one kernel to compute one activation map. In practice, we repeat this process multiple times: we consider a set (or *bank*) of filters having different weights values, and for each kernel of the set, we compute its activation map.

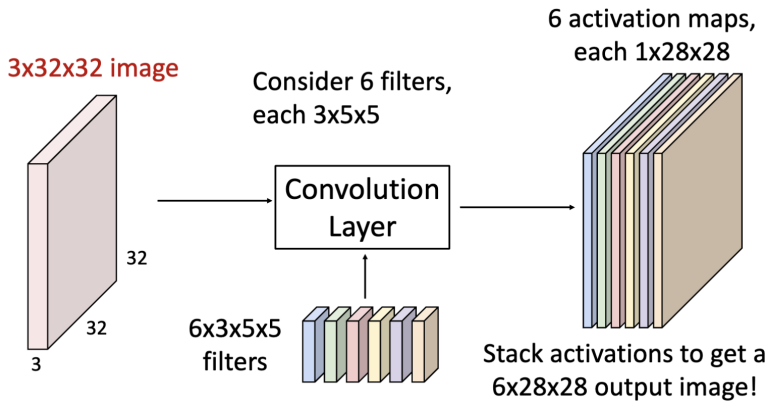

Figure 5.6: Convolutional Layer using 6 kernels