Deep Learning

Leonardo Biason

October 20, 2024

Chapter 1

Convolutional Neural Networks

When dealing with **Multilayer Perceptrons** (**MLP**s), we mostly used data that didn't have articulate structures. Even in the example of the MNIST digits dataset, we still consider the input image as a stream of flattened vectors. This approach, even though helps for making small examples, would not hold well with actual images, which are more complex in terms of number of channels, possible values of each pixel, and so on...

So how can we deal with that? How can we use images with neural networks, so that to still keep track of relevant informations that would be otherwise lost? The solution is given by the architectural model of the **Convolutional Neural Network**, CNNs for short. In this first chapter, we'll see how CNNs are made, and what components they have.

1.1 Image Filters

Many times in the fotography field we hear the term **filter**, but what *is* a filter to begin with? Why do we use it? What kind of filters can we apply? Let's first give a proper definition:

DEFINITION

Filter

A **filter** is the **application** of a **specific function** to a **local image patch** of a given dimension

Consider the following example: we have a patch of an image (suppose that the patch's dimensions are smaller than the image ones) and we apply a function which returns the mean of all the pixels adjacent to a selected pixel:

8	7	4
5	9	1
2	3	6

$\xrightarrow{f(x)}$

	5	

Image filtering is a technique that is widely used for various reasons: to **reduce noise**, to **fill in missing values** and even to **extract image features**, such as edges and/or corners. The simplest type of filter that we can have is a filter that replaces each pixel with a linear combination of its neighbours. We call this a **linear filter**. One of the most known linear filters is the **2D convolution**.

DEFINITION

Convolution

A **convolution** is a **linear filter** which slides a given **filter kernel** through the image and performs the **matrix multiplication** between the filter and the overlapped image patch, returning a filtered image.

A filtered image f is expressed as follows:

$$f[m, n] = I \otimes g = \sum_{k,l} I[m-k, n-l] \cdot g[k, l]$$

where I is the image, g is the kernel and m, n, k and l are indexes.

Let's make a quick example to show how convolutions work:

EXAMPLE

1.1.0

Suppose that we have the following image I and kernel g:

8	5	2
7	5	3
9	4	1

$I[k, l]$

−1	0	1
−1	0	1
−1	0	1

$g[k, l]$

How can we perform the convolution of I with the kernel g? Suppose that we want to perform the convolution at the center of the image. When using k and l, it's important to note that the coordinates work in the following way:

- the center of the kernel has coordinates [0, 0];
- if from a coordinate $[k, l]$ we move to the right, then we arrive at $[k, l-1]$, and viceversa if we go to the left we arrive to $[k, l+1]$;
- if from a coordinate $[k, l]$ we move upwards, then we arrive at $[k+1, l]$, and viceversa if we go downwards we arrive to $[k-1, l]$.

The following schema sums up this coordinate system:

[1, 1]	[1, 0]	[0, −1]
[0, 1]	[0, 0]	[0, −1]
[−1, 1]	[−1, 0]	[−1, −1]

Now, for $k = -1$ and $l = -1$, we would have that:

$$I[m+1, n+1] \cdot g[-1, -1] = 1 \cdot -1 = -1$$

For $k = -1$ and $l = 0$ we would have instead:

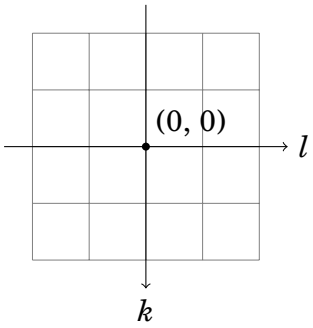
$$I[m+1, n+0] \cdot g[-1, 0] = 4 \cdot 0 = 0$$

Then, for $k = -1$ and $l = 1$ we would have:

$$I[m+1, n-1] \cdot g[-1, 1] = 9 \cdot 1 = 9$$

And so on and so forth for all the multiplications...

From this previous example we had a way to illustrate how the convolution works, but what if we had to code it? If we had to keep track of two different indexes, we would waste some computational memory. Let's try to find a quicker and more efficient method. We can start from the kernel: the multiplication $I \otimes g$ is made between items that are in mirrored positions with respect to some "invisible axes" l and k:



A simple way to make the computations easier is to flip the kernel along these axes, so that to align it to the image's axes. This way, we would just have to do the element-wise multiplication of the matrices and sum the resulting values.

A special case of convolution is the when we use a 1D filter on a 2D image.

Chapter 2

CNNs Architectures

2.1 AlexNet

do alexnet

2.2 VGG

do vgg