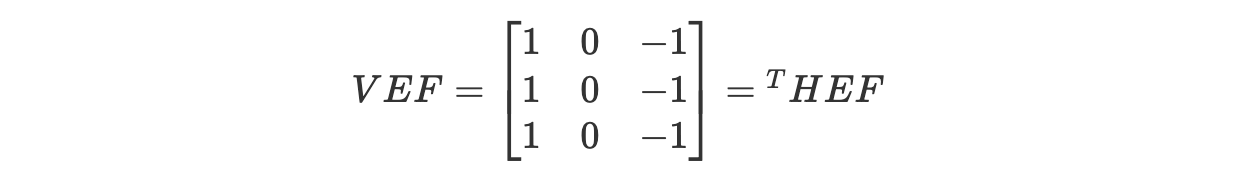
**Convolutional Neural Networks’ mathematics**

<https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0>

Computer vision is a subfield of deep learning which deals with images on all scales. It allows the computer to process and understand the content of a large number of pictures through an automatic process.   
The main architecture behind Computer vision is the convolutional neural network which is a derivative of feedforward neural networks. Its applications are very various such as image classification, object detection, neural style transfer, face identification,… If you have no background on deep learning in general, I recommend you to first read my [post](https://medium.com/swlh/deep-learnings-mathematics-f52b3c4d2576) about feedforward neural networks.

**1- Filter processing**

The first processing of images was based on filters that allowed, for instance, to get the edges of an object in an image using the combination of vertical-edge and horizontal-edge filters.   
Mathematically speaking, the vertical edge filter, VEF, if defined as follows:



Where HEF stands for the horizontal edge filter.

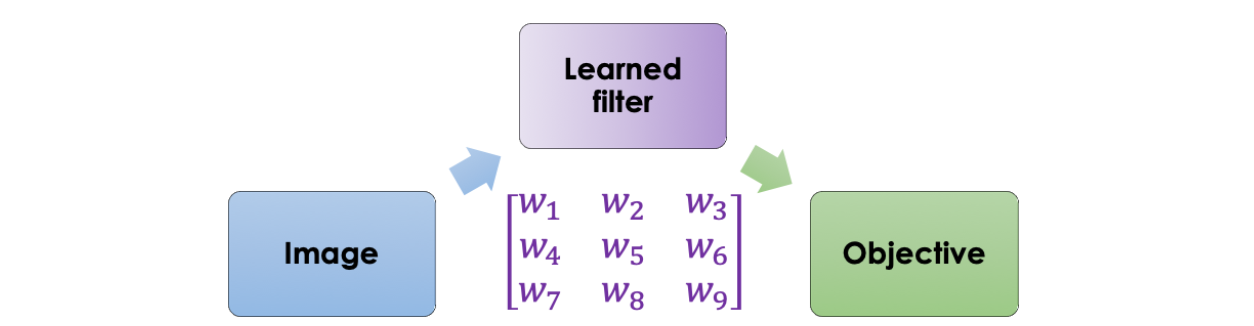
For the sake of simplicity, we consider grayscale 6x6 image A, a 2D matrix where the value of each element represents the amount of light in the corresponding pixel.   
In order to extract the vertical edges from this image, we carry out a **convolutional product (⋆)** which is basically **the sum of the elementwise product in each block**:

We carry out the elementwise multiplication on the first 3x3 block of the image then we consider the following block on the right and do the same thing until we have covered all the potential blocks.

We can sum up the following process in:



Given this example, we can think of using the same process for any objectivewhere the filter is learned by neural network as follows:



The main intuition is to set a [neural network](https://medium.com/swlh/deep-learnings-mathematics-f52b3c4d2576) that takes the image as an input and outputs a defined target. The parameters are learned using [backpropagation](https://medium.com/swlh/deep-learnings-mathematics-f52b3c4d2576).

# 2- Definition

A convolutional neural network is a serie of convolutional and pooling layers which allow extracting the main features from the images responding the best to the final objective.

## **Convolution product**

Before we explicitly define the convolution product, we will first start by defining some basic operations such as the padding and the stride.

**Padding**

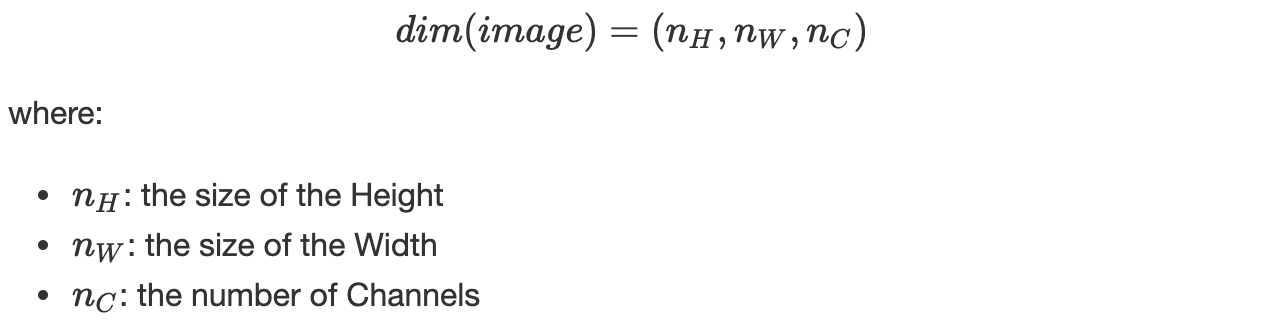
As we have seen in the convolutional product using the vertical-edge filter, the pixels on the corner of the image (2D matrix) are less used than the pixels in the middle of the picture which means that the information from the edges is thrown away.   
To solve this problem, we often add padding around the image in order to take the pixels on the edges into account. In convention, we padde with zeros and denote with p the padding parameter which represents the number of elements added on each of the four sides of the image.

**Stride**

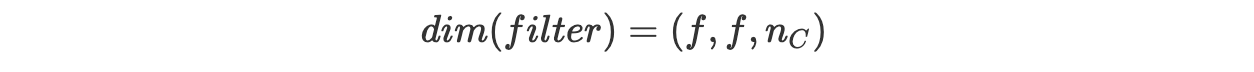
The stride is the step taken in the convolutional product. A large stride allows to shrink the size of the output and vice-versa. We denote s the stride parameter.   
The following image illustrates a convolutional product (sum of element-wise element per block) with s=1:

**Convolution**

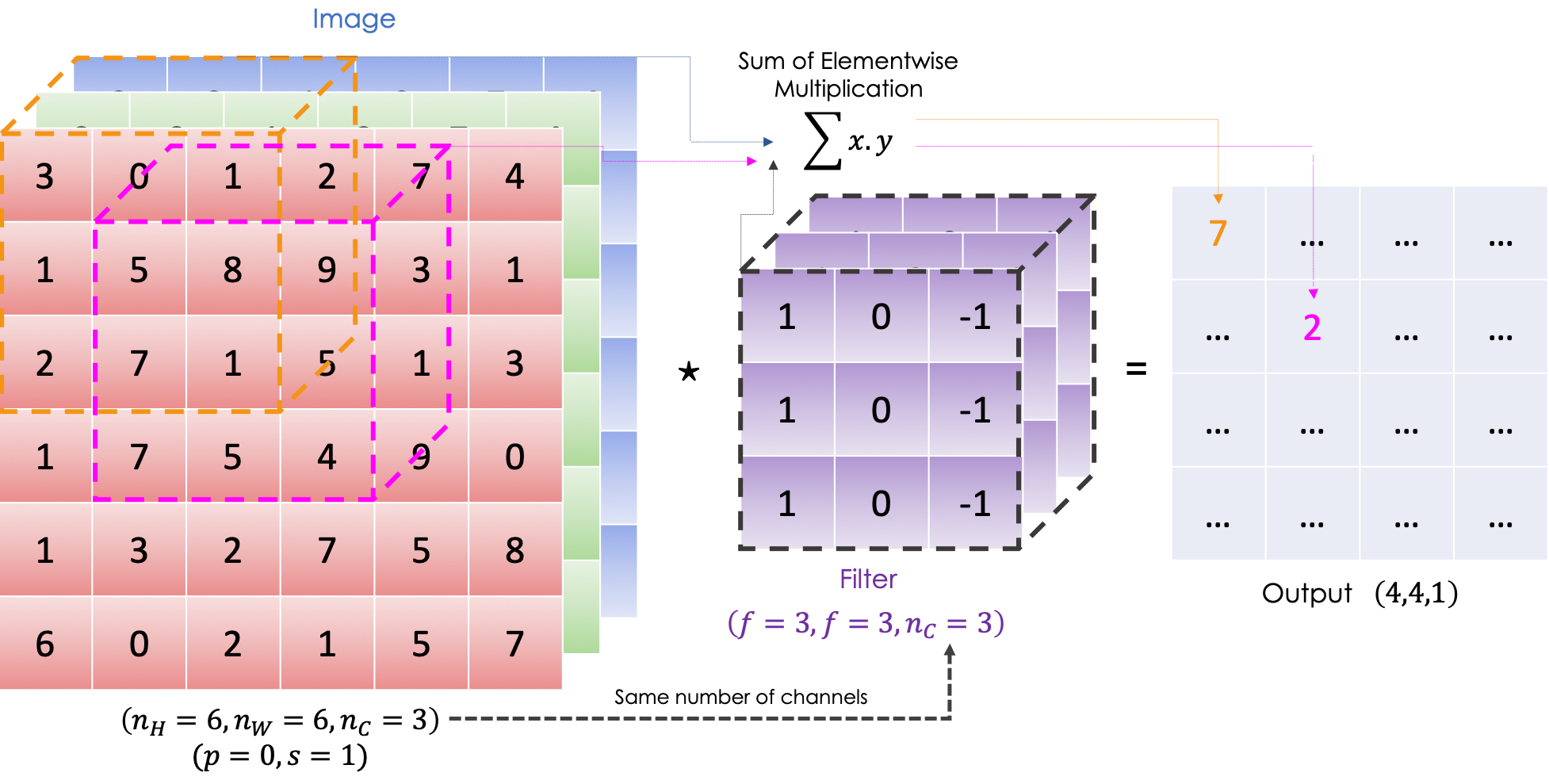
Once we have defined the stride and the padding we can define the convolution product between a tensor and a filter.   
After previously defining the convolution product on a 2D matrix which is the sum of the element-wise product, we can now formally define the convolution product on a volume.   
An image, in general, can be mathematically represented as a tensor with the following dimensions:



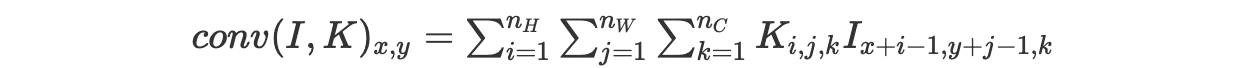
In the case of an RGB image, for instance, n\_C=3, we have, Red, Green and Blue. In convention, we consider the filter *K* to be squared and to have an odd dimension denoted *f,*which allows each pixel to be centered in the filter and thus consider all the elements around it.   
When operating the convolutional product, the filter/kernel *K* must have the same number of channels as the image, this way we apply a different filter to each channel. Thus the dimension of the filter is as follows:



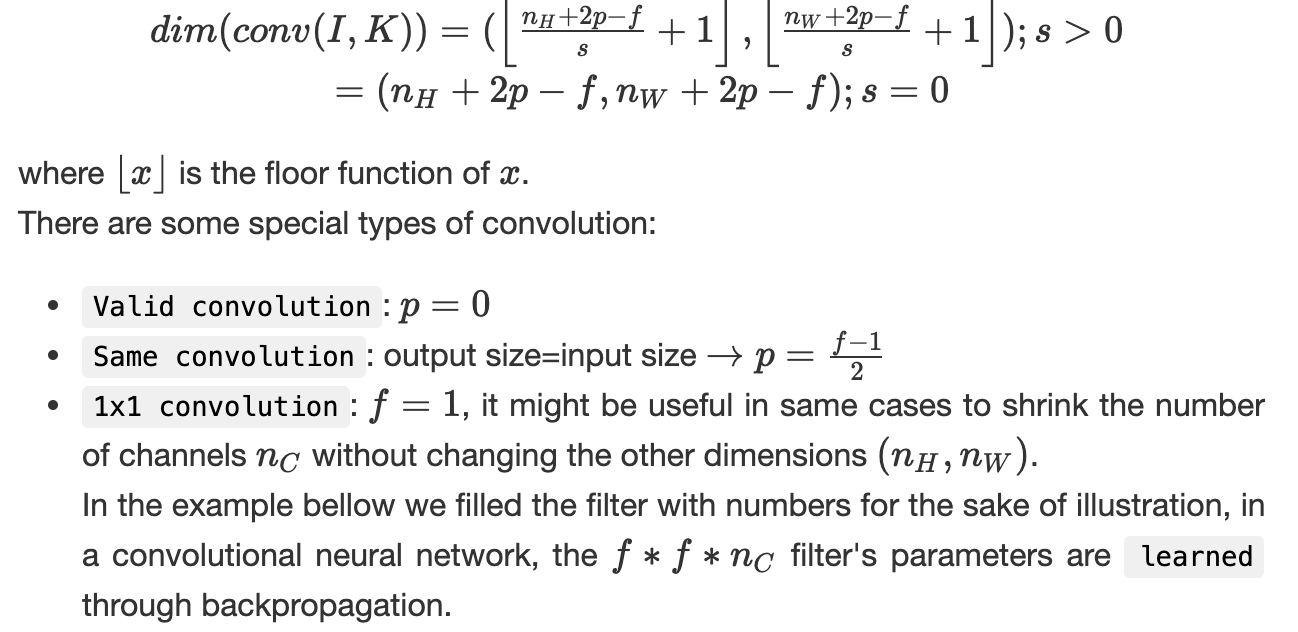
The convolutional product between the image and the filter is a 2D matrixwhere each element is the **sum of the elementwise multiplication** of the cube (filter) and the subcube of the given image as illustrated below:



Mathematically speaking, for a given image and filter we have:

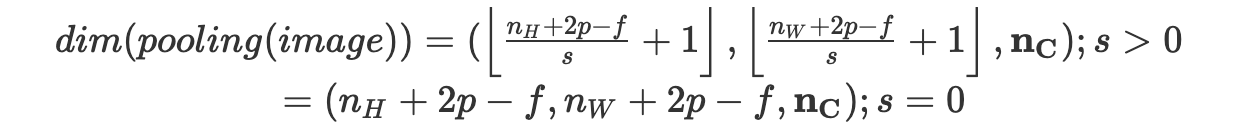


Keeping the same notations as before, we have:



**Pooling**

It is the step of downsampling the image’s features through summing up the information. The operation is carried out through each channel and thus it only affects the dimensions (n\_H, n\_W)and keeps n\_C intact.   
Given an image, we slide a filter, with no parameters to learn, following a certain stride, and we apply a function on the selected elements. We have:



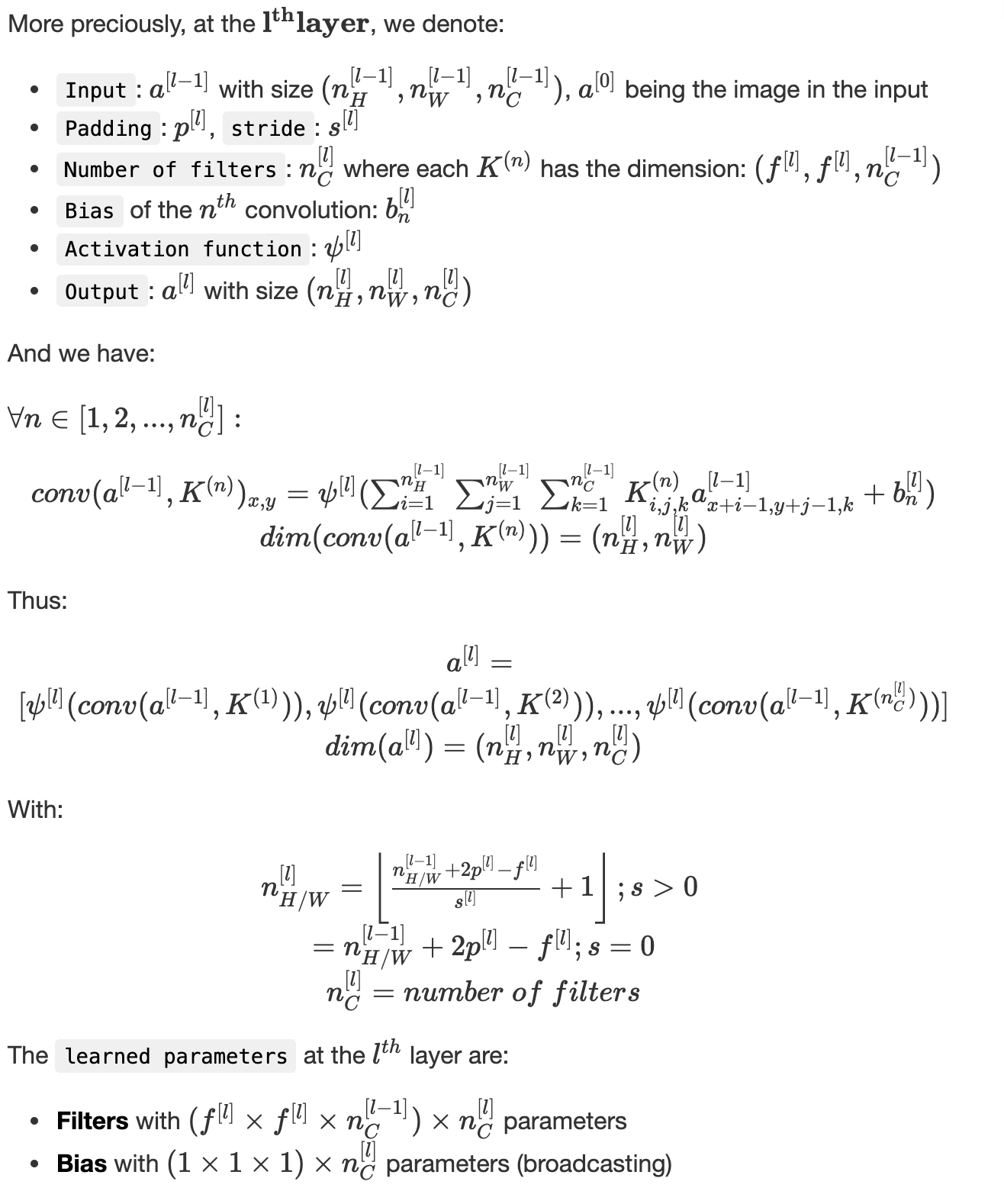
In convention, we consider a squared filter with size *f* and we usually set *f*=2 and consider *s*=2.

We often apply:

* Average pooling: we average on the elements present on the filter
* Max pooling: given all the elements in the filter, we return the maximum

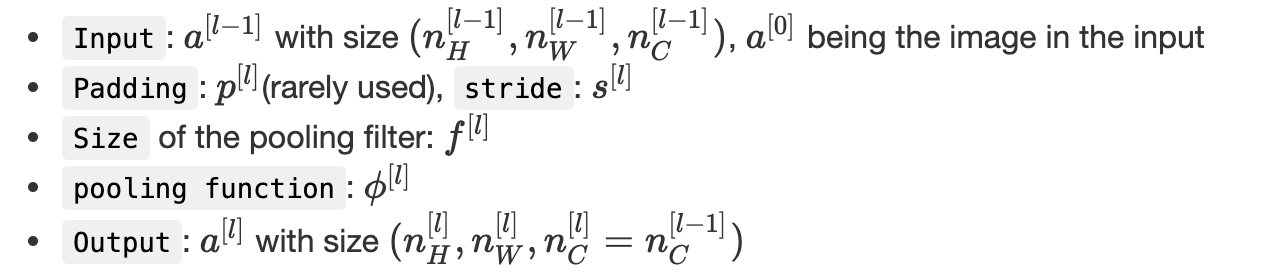
## **• Convolutional layer**

As we have seen before, at the convolutional layer, we apply convolutional product**s**, using many filters this time, on the input followed by an activation function ψ.

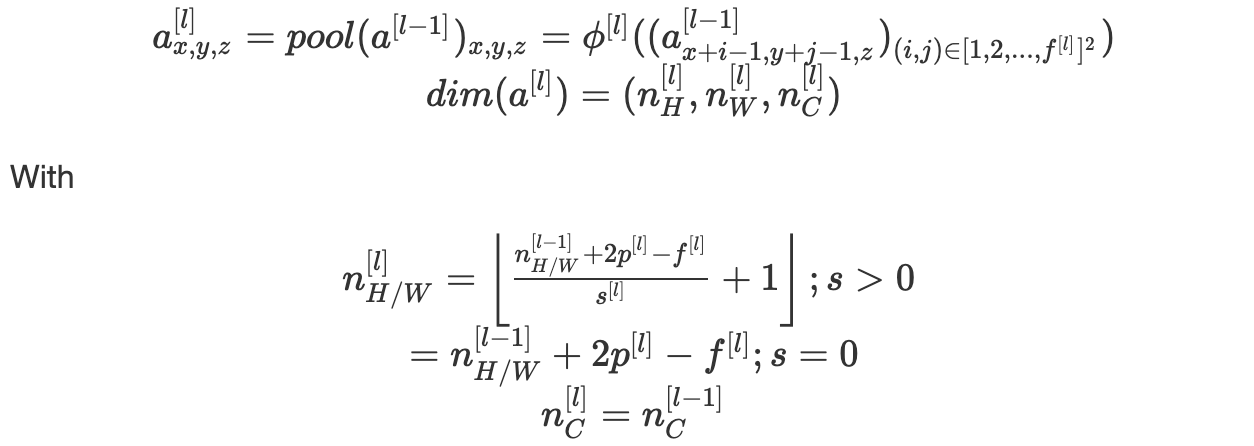


## **• Pooling layer**

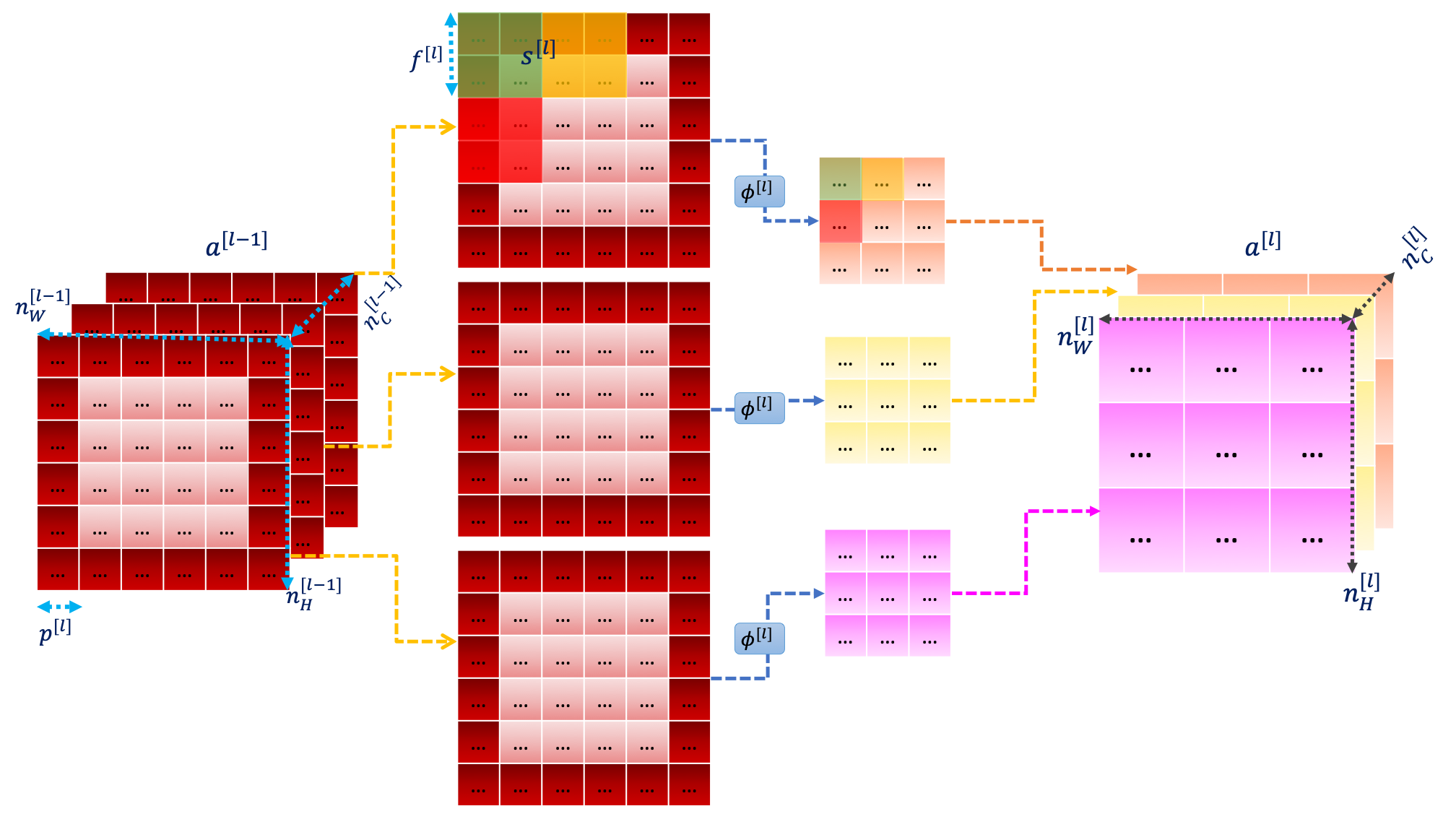
As mentioned before, the pooling layer aims at downsampling the features of the input without impacting the number of the channels.  
We consider the following notation:



We can assert that:

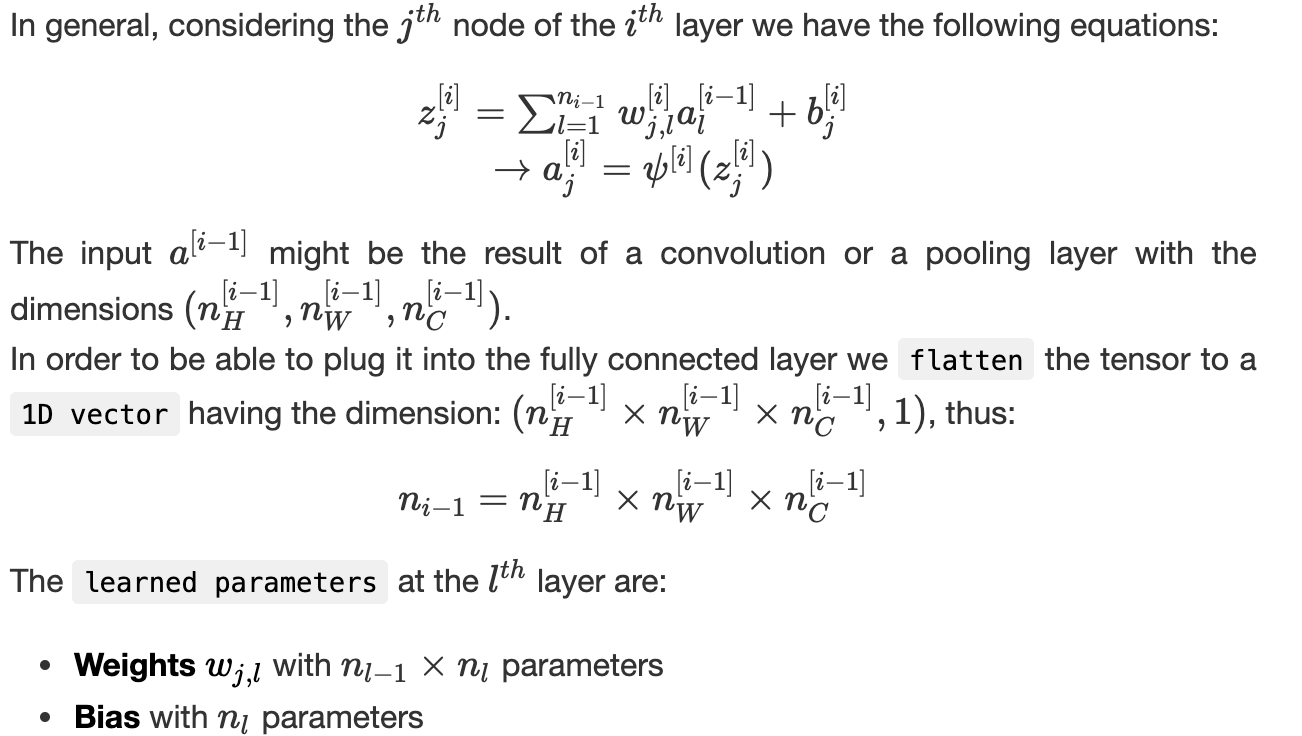


The pooling layer has no parameters to learn.



## **Fully connected layer**

A fully connected layer is a finite number of neurons which takes in input a vector and returns another one.



After repeating a serie of convolutions followed by activation functions, we apply a pooling and repeat this process a certain number of time. These operations allow to extract features from the image that will be fed to a neural network described by the fully connected layers which are regularly followed by activation functions as well.   
The main idea is to decrease n\_H & n\_W and increase n\_C when going deeper through the network.

## **Why do CNN work efficiently?**

Convolutional neural networks enable the state of the art results in image processing for two main reasons:

* **Parameter sharing**: a feature detector in the convolutional layer which is useful in one part of the image, might be useful in other ones
* **Sparsity of connections**: in each layer, each output value depends only on a small number of inputs