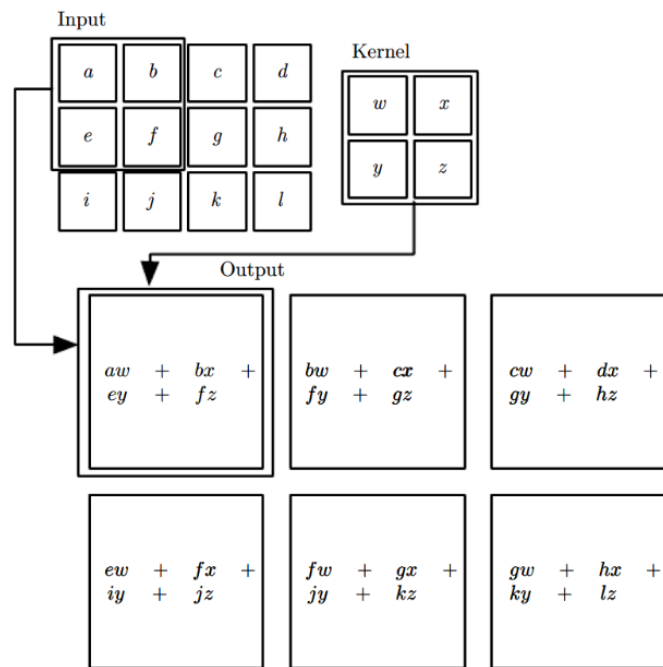


CNN operations cheatsheet

Matteo Lugli

December 14, 2023

1 Basic Idea



2 Notation Summary

The following table provides a summary of the variables used in the example described in 3.

Explanation	Symbol
P	padding layers
X	input image
H	kernel
Y	output
N	filters applied in a layer
w	input width
h	input height
k	kernel size

Table 1: Summary of symbols used

To understand the formulation it is really important to clarify a couple of important things:

- The following formula, given the plain kernel, it flips it and applies it to the image, performing the so called **convolution** (if you don't flip it, it's just a correlation);
- For the kernel matrix, we use indexes in such a way that the central element has coordinates $[0,0]$. This means that we will end up with some negative indices.
- The input matrix needs to be considered with **zero padding**. The first row and column of the resulting matrix will have index -P;
- The output matrix has standard coordinates, so starting from $[0,0]$;

3 Example

We are going to use the following formula, Input and Kernel:

$$Y[m, n] = \sum_{i=-P+m} \sum_{j=-P+n} X[i, j] H[m-i, n-j] \quad (1)$$

i, j	-1	0	1	2	3
-1	0	0	0	0	0
0	0	1	2	3	0
1	0	4	5	6	0
2	0	7	8	9	0
3	0	0	0	0	0

Table 2: Input X

i, j	-1	0	1
-1	-1	-2	-3
-0	0	0	0
1	1	2	3

Table 3: Kernel H

As you can see padding is already applied in the input matrix, applying 1 layer of zeroes means that we are going to start counting indexes at -1.

$$\begin{aligned}
Y[0,1] &\Rightarrow m=0, n=1, P=-1 \\
i &= -1 \\
X[-1,0] \cdot H[1,1] &= 0 \cdot 3 + \\
X[-1,1] \cdot H[1,0] &= 0 \cdot 2 + \\
X[-1,2] \cdot H[1,-1] &= 0 \cdot 1 + \\
i &= 0 \\
X[0,0] \cdot H[0,1] &= 1 \cdot 0 + \\
X[0,1] \cdot H[0,0] &= 2 \cdot 0 + \\
X[0,2] \cdot H[0,-1] &= 3 \cdot 0 + \\
i &= 1 \\
X[1,0] \cdot H[-1,1] &= 4 \cdot -3 + \\
X[1,1] \cdot H[-1,0] &= 5 \cdot -2 + \\
X[1,2] \cdot H[-1,-1] &= 6 \cdot -1 + \\
&= -28
\end{aligned} \tag{2}$$

As you can see in table 4, we flipped the kernel on both axis and overlapped it with the correct portion of the input.

i, j	-1	0	1	2	3
-1	0	$0^{(3)}$	$0^{(2)}$	$0^{(1)}$	0
0	0	$1^{(0)}$	$2^{(0)}$	$3^{(0)}$	0
1	0	$4^{(-3)}$	$5^{(-2)}$	$6^{(-1)}$	0
2	0	7	8	9	0
3	0	0	0	0	0

Table 4: Convolution to compute element $Y[0,1]$ of output matrix

Let's write also the calculations made to compute $Y[1,2]$

$$Y[1, 2] \Rightarrow m = 1, n = 2, P = -1$$

$$i = 0$$

$$X[0, 1] \cdot H[1, 1] = 2 \cdot 3 +$$

$$X[0, 2] \cdot H[1, 0] = 3 \cdot 2 +$$

$$X[0, 3] \cdot H[1, -1] = 0 \cdot 1 +$$

$$i = 1$$

$$X[1, 1] \cdot H[0, 1] = 5 \cdot 0 +$$

$$X[1, 2] \cdot H[0, 0] = 6 \cdot 0 +$$

$$X[1, 3] \cdot H[0, -1] = 0 \cdot 0 +$$

$$i = 2$$

$$X[2, 1] \cdot H[-1, 1] = 8 \cdot -3 +$$

$$X[2, 2] \cdot H[-1, 0] = 9 \cdot -2 +$$

$$X[2, 3] \cdot H[-1, -1] = 0 \cdot -1 +$$

$$= -30$$

(3)

i, j	-1	0	1	2	3
-1	0	0	0	0	0
0	0	1	$2^{(3)}$	$3^{(2)}$	$0^{(1)}$
1	0	4	$5^{(0)}$	$6^{(0)}$	$0^{(0)}$
2	0	7	$8^{(-3)}$	$9^{(-2)}$	$0^{(-1)}$
3	0	0	0	0	0

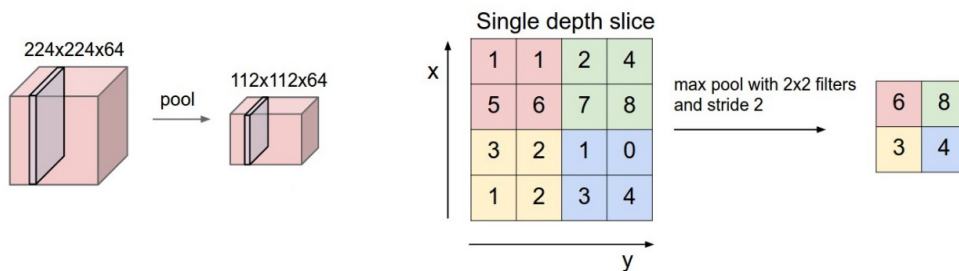
Table 5: Convolution to compute element $Y[1, 2]$ of output matrix

4 Convolution Layers

In most cases CNNs are used with RGB images. As the name says, such images are made of 3 **channels** (Red, Green and Blue). You can imagine them as $h \times w \times 3$ cubes, or as 3 $h \times w$ matrixes stacked. This means that we have to imagine our kernels as $k \times k \times 3$ dimensional cubes as well! Now it's easy to imagine why for each convolutional layer there are $k \times k \times c \times N + N$ learnable parameters, where c represents the number of channels of the input data (3 in case of RGB images), and N is the number of filters for that layer. It's really important to remember that the number of channels for the next layer becomes N : if in the first layer we use 10 kernels to process our plain image, the input that needs to be processed by the next layer will have 10 channels!

5 Pooling Layers

Pooling layers are used as noise reduction layers or to perform dimensionality reduction. The most common types of pooling are *max pooling* and *average pooling*, which either compute the maximum value among the overlapped elements or the average value. The figure below should be clear enough.



The only 2 hyperparameters that need to be defined are

- Pool size k , in this case equal to 2;
- Pool stride s , in this case equal to 2 as well;

6 Output volume size

A trick that is usually done when implementing CNNs and not mess up with the dimensions, is to use *kernel size* = 3 with a 1 layer padding. An example of such a convolution has already been presented in section 4. If you notice, by using such kernel size and padding, you can overlap the center of the kernel with the top left element of the input image, and make the kernel slide until it overlaps the top right element. Considering that each overlap produces a single element, for each row that we process this way we get w elements. If we do that for each single row (so by applying a complete convolution) the output size of the matrix will be exactly $h \times w$, the same as the size of the input. One of the benefits of doing so is that we can stack how many convolutional layers as we want and we are sure that the height and width of the output will not change until we apply a pooling layer.

Type of layer	Output size
Convolutional	$w_2 = \frac{w_1 - k + 2p}{s} + 1$
	$h_2 = \frac{h_1 - k + 2p}{s} + 1$
	$c_2 = N$
Pooling	$w_2 = \frac{w_1 - k}{s} + 1$
	$h_2 = \frac{h_1 - k}{s} + 1$
	$c_2 = c_1$

Table 6: Output size cheatsheet