Ex2 - TPA
deadline : 09/06/2024 20:00 +0200

Write a parallel code in C++ 11 and OpenMP that solves efficiently the main operation in a Convolution Neural Network (CNN): Convolution

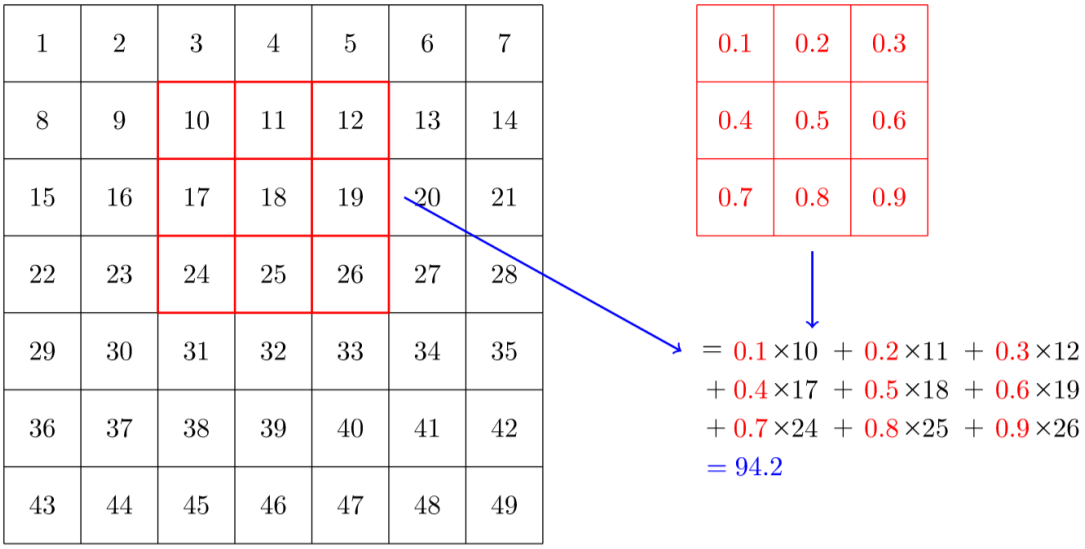
The operation acts as follows:
Convolution: this is the most interesting and complex operation in a CNN. With convolution the CNN extracts information (feature map) from the input image that is then used to detect and classify objects.

Examples of filters can be found here: https://en.wikipedia.org/wiki/Kernel_(image_processing)

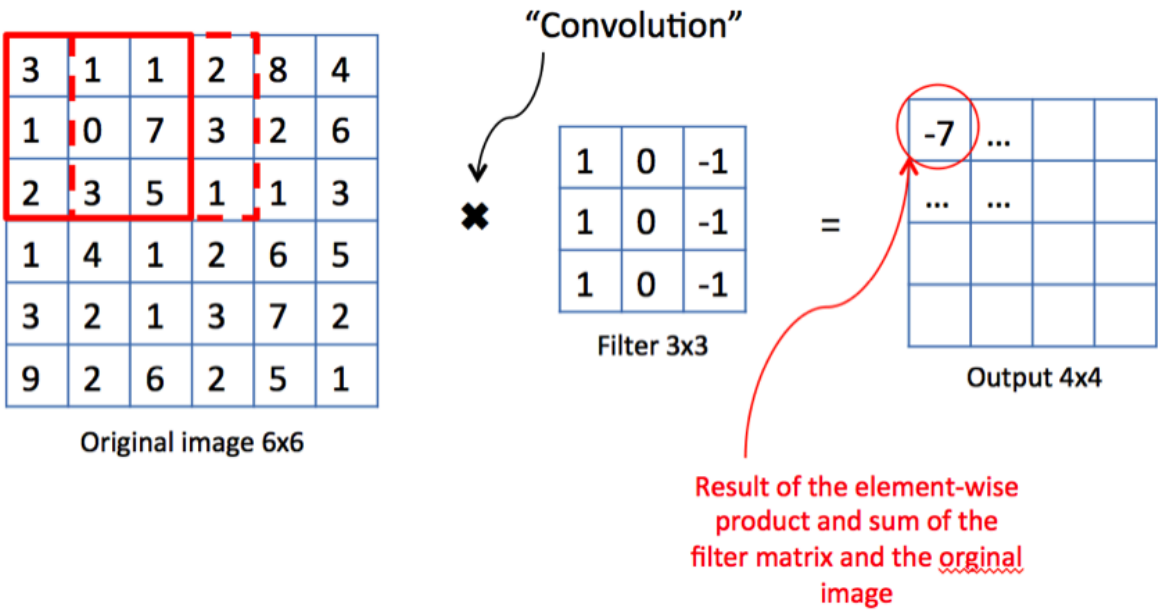
You will implement a convolution on a matrix of float rather than on an image (working on an image is possible, but requires to deal with heathers and descriptors that are well out of the scope of this course. You could try with a raw image of RGB values, just for fun .bmp, . jpg,. png, or any other format requires understanding the compression: *if you want to try it let me know*).

The convolution is performed as **multiplication and accumulation** between a **kernel** (4x4 matrix of fixed values) and a 4x4 submatrix of the input (the input matrix will be square, bigger than 4x4, and with sizes multiple of 4). The result of the multiplication and accumulation is **one floating value** obtained accumulating (adding) all the elements of the multiplication. In other words:
1 - you instantiate conv_tmp as a floating value to 0 (remember to instante to 0, otherwise…).
2 - you perform the sub_matrix[i,j] * kernel[i,j] multiplication, which is a floating point value x.
3 - you accumulate the result in conv_tmp += x.

*** the pics have kernels 3x3, we will work on 4x4 kernels ***



You need to do so for all the rows and columns of the 4x4 kernel and submatrix.
The convolution is performed sliding the kernel in the input matrix as follows:



To simplify the code it is **not** necessary to add zero-padding to the edge of the original image, thus the output matrix will be **smaller** (you need to understand the size of the output matrix) than the input matrix.

Implementation details:
Inputs: file with input matrix (**you choose the size**) and kernel (**fixed size 4x4**)
matrici quadrate e multipli di 4
The goal is to first have a working sequential code for the four operations. Then, parallelize the operations that can be efficiently parallelized. Pay special attention to *data races* (more threads requesting the same input) and to *concurrency/conflicts* (multiple threads updating the same output). As a general suggestion, to achieve the best performance you should group in one thread (or in multiple threads executed in the same CPU) all the operations that work on the same input data. This avoids costly data copy to multiple locations.

There are multiple ways to parallelize the code. You can parallelize the single convolution or you can parallelize the convolutions (each thread executes a 4x4 convolution). Please discuss the benefit of each solution and evaluate the performance of both.

Suggestion: when you parallelize convolutions pay attention that if multiple threads take subsequent sliding convolutions they all will need the same part of the input data, thus….

You need to create **an OpenMP file with the implementation of the convolution and a main file** for testing the function. The main will:
- read the input matrix from a text file (matrix.txt) - randomly generated or static, you choose
- read the kernel from a text file (kernel.txt) - fixed 4x4 size, you choose the values
- apply convolution and save the result in a file

You need to present a **performance report** where you show the *measurements of the execution time of the sequential implementation* (to simply, simply set the number of threads to 1) *and of various parallel implementations* (degree of parallelism, threads distribution, threads grouping etc…). Write your consideration in a **PDF document to add to the submission.**

********** extra points **********

To gain extra points, you can:
1 - consider the "zero padding", by performing convolution in the whole input matrix, till the last column and the last row. Please do not add 3 extra rows and 3 extra columns of zeros in the input matrix but try more smart solutions. The output matrix will have the same size as the input matrix.

2 - consider bigger input matrix sizes and discuss if/why the performance improves.

ciclo di un centinaio di esecuzioni per vedere il tempo totale poi fare la media. Scartare le prime due (conviene cit Paolo Rech)

scrivere la performace in base al tipo di macchina valutare: core, cpu, frequenza, numero di threads

relazione mostrare le scelte che abbiamo fatto