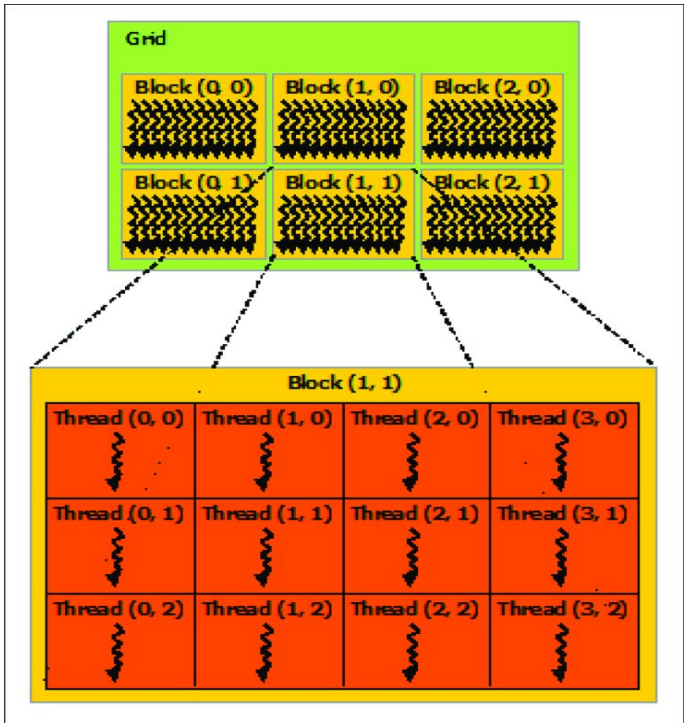
Parallel CNN with Cuda

This is a brief summary of the current work i’m doing regarding the implementation of a parallel CNN for 2D inputs and filters arrays. This is currently only for the convolution operation.

CUDA



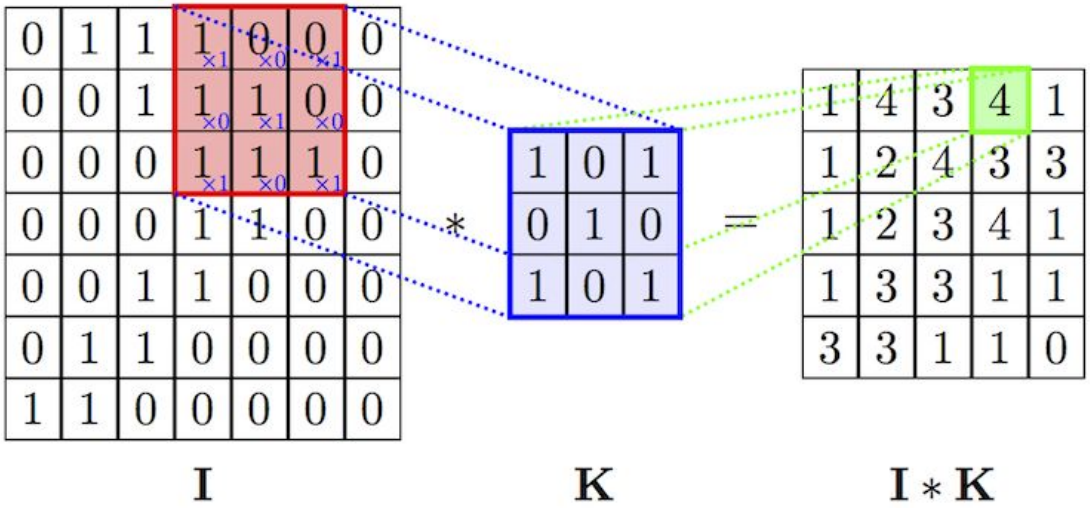
In Cuda a kernel is launched with a grid that contains blocks on up to 3 dimensions, x,y,z.

Each block with contains threads and can have up to 3 dimensions, (x,y,z)

This dimensions are generally chosen according to our parameters.

As we will see later, the kernel function (@cuda) is executed on each individual thread.

2D Convolution



Typically in a 2D convolution the output dimensions are:

Ix - Kx + 1 and Iy-Ky + 1.

Dimensions

Let’s consider the following:

Input: 2 images of 28x28

Filters: 5 filters of 3x3

We get an output dimension of 26x26 and 10 images (5 for each input).

This mean we have a total of 26x26x10 output elements so in theory if we have that number of threads each can compute his output in parallel.

Block and grid sizes

Now that we know how many nodes we need to compute we can choose sizes, let’s say we use a constant block size of 8x8.

To calculate the dimension of the grid we can do the following:

X: should be greater or equal to the number of input image

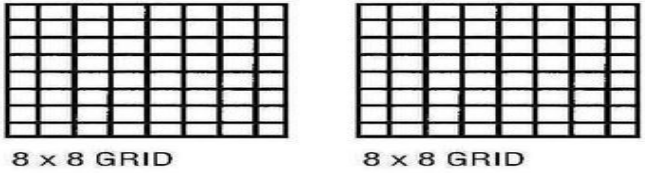
Y: should be greater or equal to output height

Z: should be greater or equal to output width

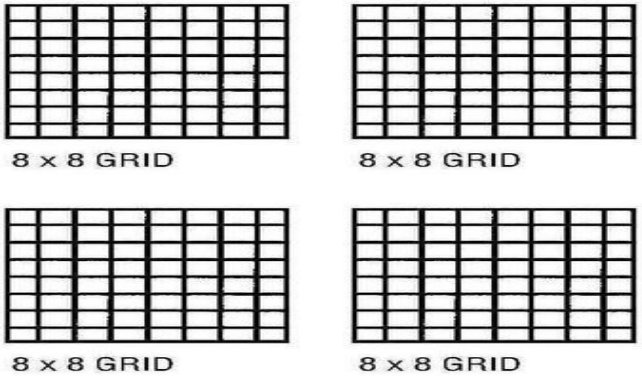
Block and grid sizes. cont

Here are some examples of grid sizes:

(2,1,1): 128 threads 16 x-axis, 8 y-axis.



(2,2,1): 256 threads 16 x-axis, 16 y-axis.



Block and grid sizes. cont

Let’s compute the size for our previous example: 2 images, output size is 26x26.

For simplification Block size will be constant 8x8.

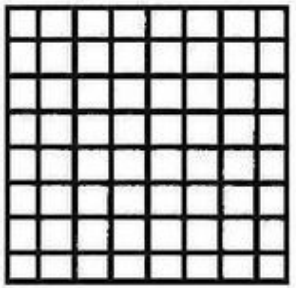
Grid size will be as follow:

x=1 because one block have 8 thread on the x-axis and 8 > 2.

y=4 because one block has 8 thread on the y-axis and 8*4 > 26

z=26 because we don’t have any thread on the z-axis 26*1 >=26.

The goal is to make sure we cover our whole output.



8 x 8 GRID

Block and grid sizes. cont

We can illustrate our grid like this (1,4,26):

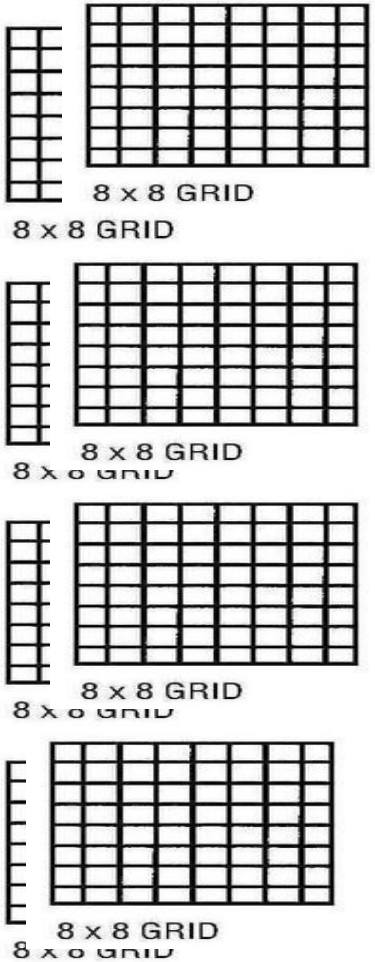
Each of these block will also have 26 layers

In the z-axis. (only 2 z-layers are shown here)

With this structure each thread can compute

the corresponding output value for every image

and filter in parallel.



Example

In cuda we can get the coordinates of a thread as follow: `x, y, z = cuda.grid(3)`

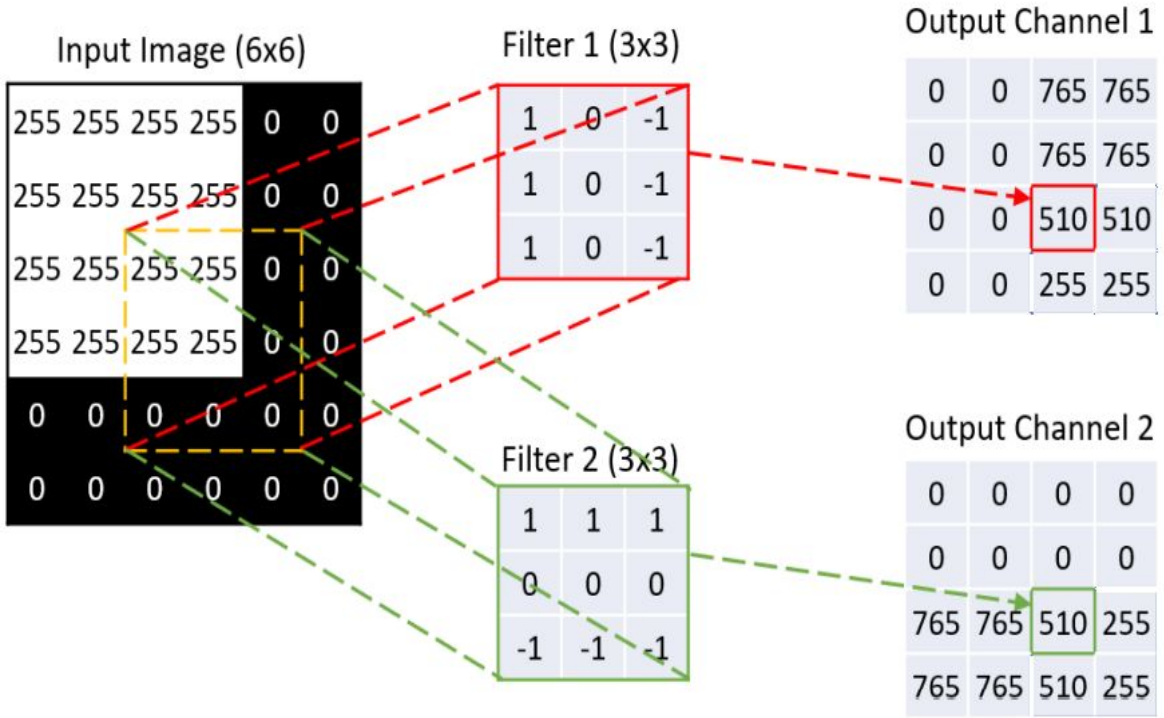
In our example this will represent the output position to compute,

For example x=0,y=3,z=15 this means first image 4th row 16th column, we will compute that position for the first image and each filter.

So we will loop through those coordinates for each filter and update the corresponding output node in each output channel.

Example. cont

We can say that each thread is responsible for computing a single output node for all filters as illustrated.In our example we have 5 filters so we would have to compute a single node in five output channels.



Details

With this implementation we can drastically reduce the computation time, For instance let’s consider 10000 arrays of 28x28 and 32 filters of 3x3.

Output dimension is 10000,26,26, 32, this is number of images, dimensions and channels respectively. Our grid size will be (1250, 4, 26) meaning we have a total of 130.000 threads running in parallel and each is computing a single output node for all 32 filters.

Benchmark (Execution time):

GPU: 1.5 seconds

CPU: 660 seconds (11 min)