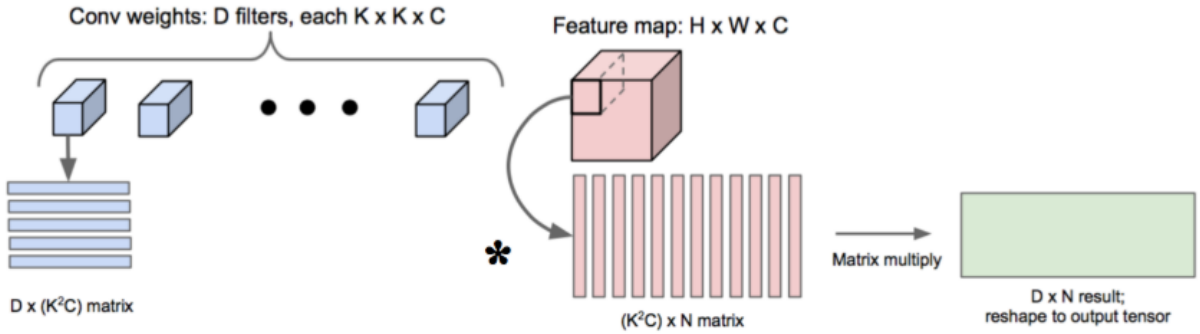
Im2col

As shown on previous source code, we use a lot for for-loops to implement the convolutions, while this is useful for learning purpose, it's not fast enough. On this section we will learn how to implement convolutions on a vectorized fashion.
First, if we inspect closer the code for convolution is basically a dot-product between the kernel filter and the local regions selected by the moving window, that sample a patch with the same size as our kernel. What would happens if we expand all possible windows on memory and perform the dot product as a matrix multiplication. Answer 200x or more speedups, at the expense of more memory consumption.



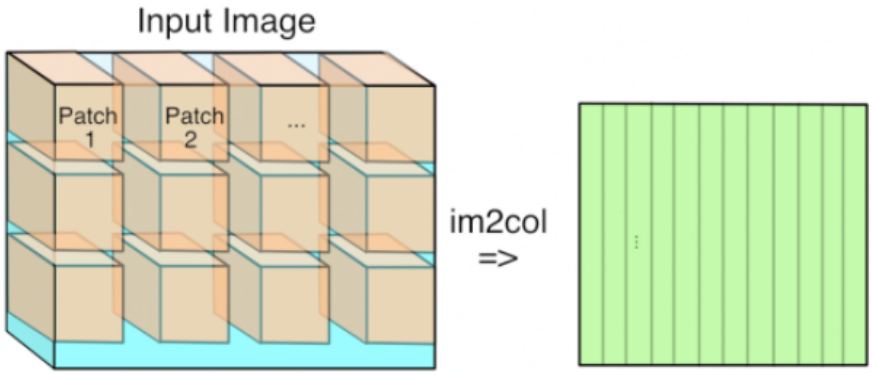
For example, if the input is [227x227x3] and it is to be convolved with 11x11x3 filters at stride 4 and padding 0, then we would take [11x11x3] blocks of pixels in the input and stretch each block into a column vector of size $11 * 11 * 3$ = 363.

Calculating with input 227 with stride 4 and padding 0, gives ((227-11)/4)+1 = 55 locations along both width and height, leading to an output matrix X_col of size [363 x 3025].
Here every column is a stretched out receptive field (patch with depth) and there are 55*55 = 3025 of them in total.

To summarize how we calculate the im2col output sizes:

```
[img_height, img_width, img_channels] = size(img);
newImgHeight = floor(((img_height + 2*P - ksize) / S)+1);
newImgWidth = floor(((img_width + 2*P - ksize) / S)+1);
cols = single(zeros((img_channels*ksize*ksize),(newImgHeight * newImgWidth)));
```

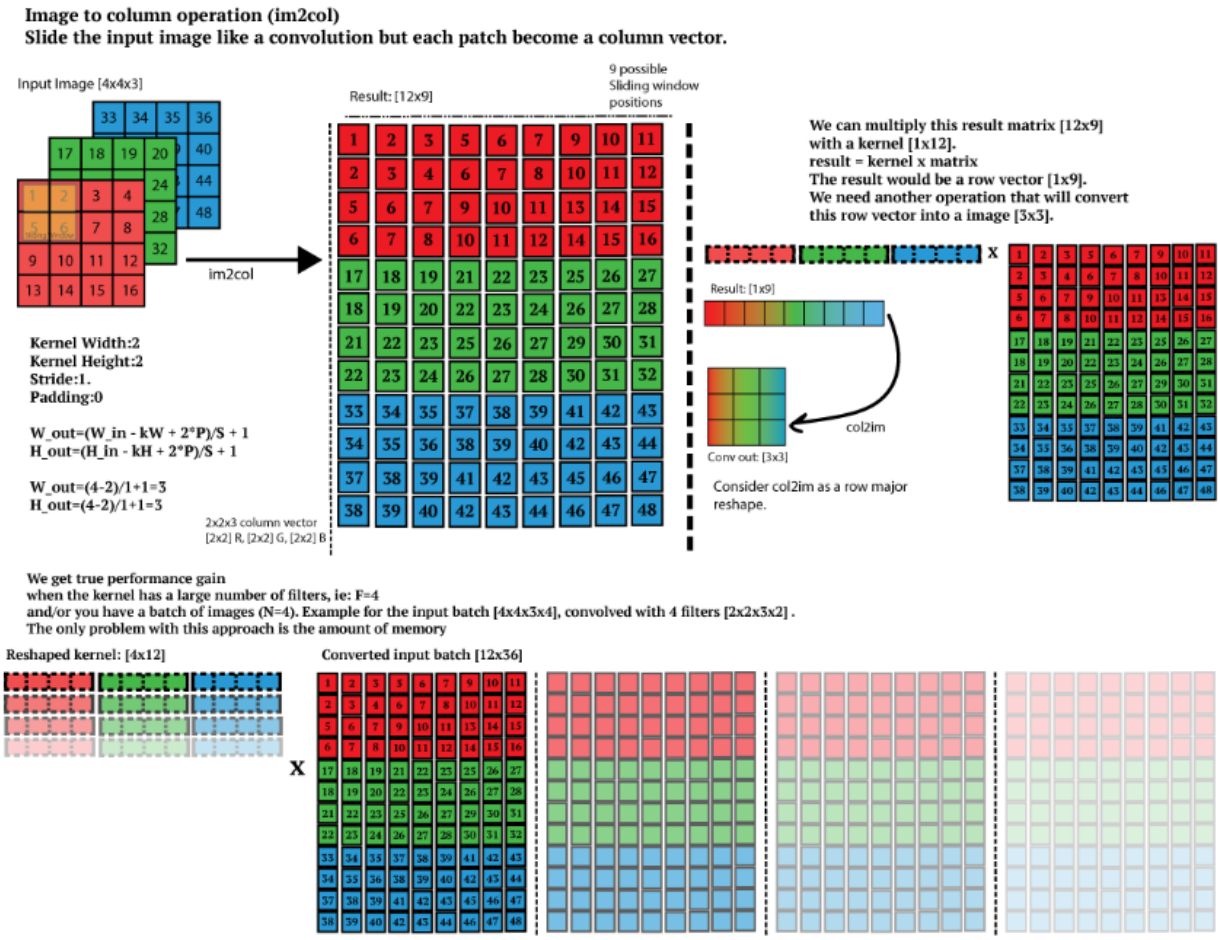
The weights of the CONV layer are similarly stretched out into rows. For example, if there are 96 filters of size [11x11x3] this would give a matrix W_row of size [96 x 363], where 11x11x3=363



After the image and the kernel are converted, the convolution can be implemented as a simple matrix multiplication, in our case it will be W_col[96 x 363] multiplied by X_col[363 x 3025] resulting as a matrix [96 x 3025], that need to be reshaped back to [55x55x96].

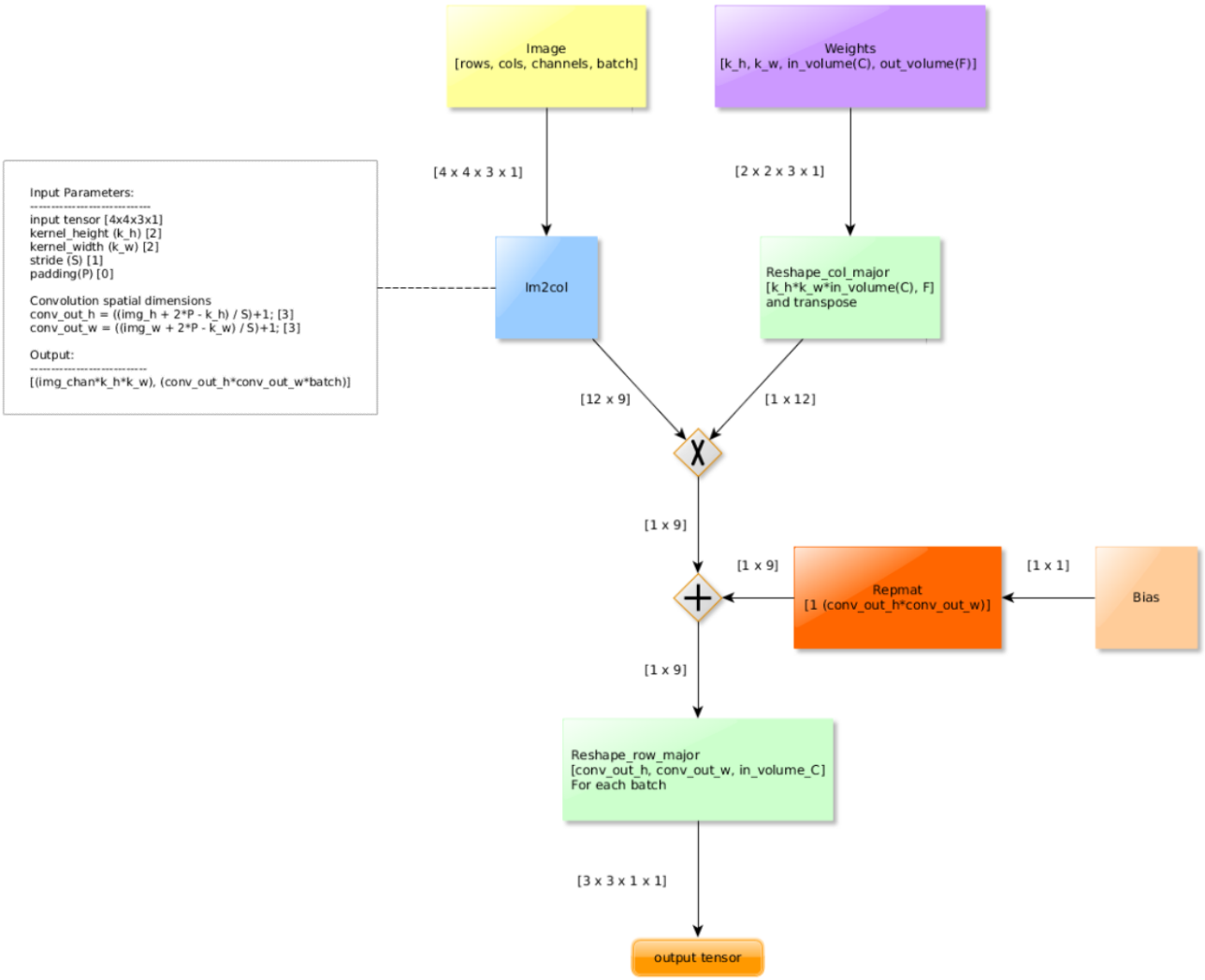
This final reshape can also be implemented as a function called col2im.

Notice that some implementations of im2col will have this result transposed, if this is the case then the order of the matrix multiplication must be changed.



Forward graph

In order to help the usage of im2col with convolution and also to derive the back-propagation, let's show the convolution with im2col as a graph. Here the input tensor is single a 3 channel 4x4 image. That will pass to a convolution layer with S:1 P:0 K:2 and F:1 (Output volume).



Backward graph

Using the im2col technique the computation graph resembles the FC layer with the same format $f(x, \theta, \beta) = (x.\theta^T) + \beta$, the difference that now we have a bunch of reshapes, transposes and the im2col block.

About the reshapes and transposes during back propagation you just need to invert their operations using again another reshape or transpose, the only important thing to remember is that if you use a reshape row major during forward propagation you need to use a reshape row major on the backpropagation.

The only point to pay attention is the im2col backpropagation operation. The issue is that it cannot be implemented as a simple reshape. This is because the patches could actually overlap (depending on the stride), so you need to sum the gradients where the patches intersect.