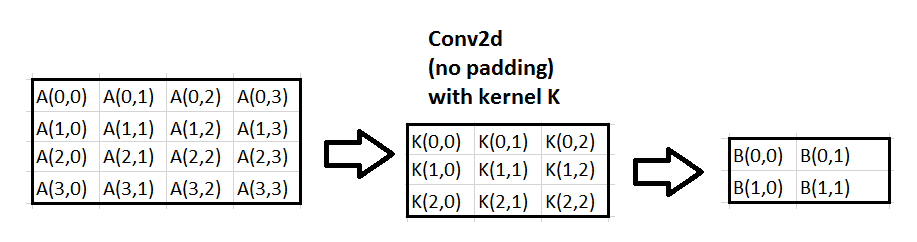
conv2d(filter,paddingSize,paddingVal)

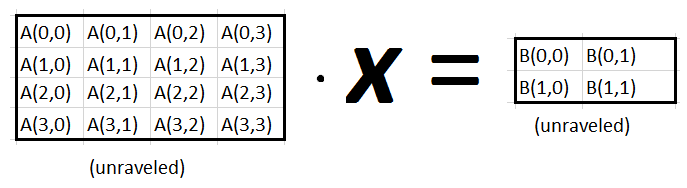
* filter (Matrix object with dimensions smaller than this.matrix) will step along the rows and columns this.matrix, producing a new array where each element is a sum of products of elements of the kernel and corresponding elements in this.matrix
* paddingSize (Matrix object of two elements) indicates how many rows and columns will be added to the edges of the matrix, to allow the filter to step outside the bounds of this.matrix
* paddingVal (number) indicates the values to place in the padding

Conv2d concept

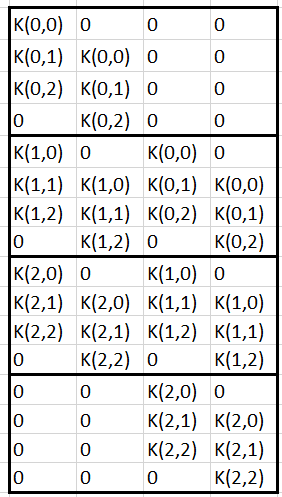
* It essentially produces a new matrix with values corresponding to how well the kernel “matches” this.matrix at a given step
* These operations can be equivalently done by a dot product and some simple matrix operations
* Known information: input matrix, kernel matrix
* Perform dot product by unraveling this.matrix ( transform from “y rows, x columns” to “x\*y rows, 1 column”), and performing a dot product with a sparse matrix created based on the kernel and the padding
* sparse matrix **column count** equals number of elements in resultant matrix AND number of steps the kernel will take overall
* sparse matrix **row count** equals number of elements in the input matrix, including padding



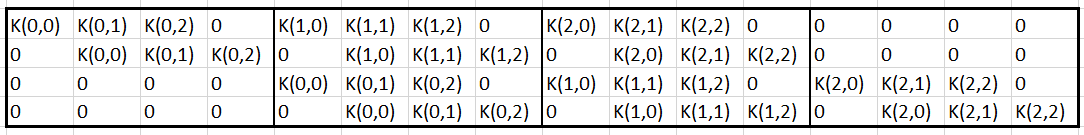
is equivalent to



where ***x*** **=**



and ***xT* =**



* Each column *n* of ***x*** represents a step in the path of the kernel in the convolution operation. The column’s elements will be multiplied through by the corresponding elements in the unraveled input matrix and then summed, resulting in the final value of *n* in the unraveled output matrix. These elements of ***x*** can be intuitively understood by visualizing the elements involved in the sum of multiplications involved in the convolution step; the row number of the elements of K correspond to which elements of A they will be multiplied by.
* In this way, transpose or “reverse” convolution can be achieved by taking an arbitrary matrix B and crossing it with the transpose of ***x*** and reshaping the output, although this is not implemented in the Matrix class