More convolutional nets (part 1 only)

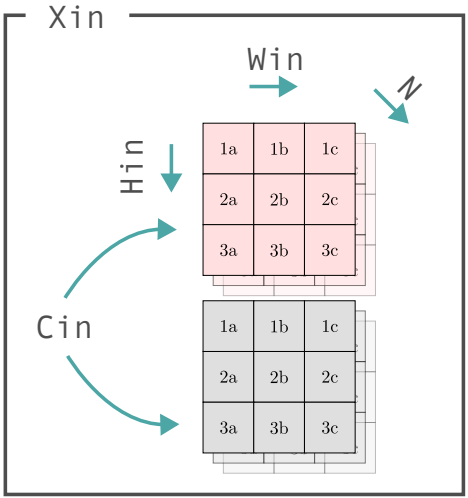
Lecture 09 — CS 577 Deep Learning

Instructor: Yutong Wang

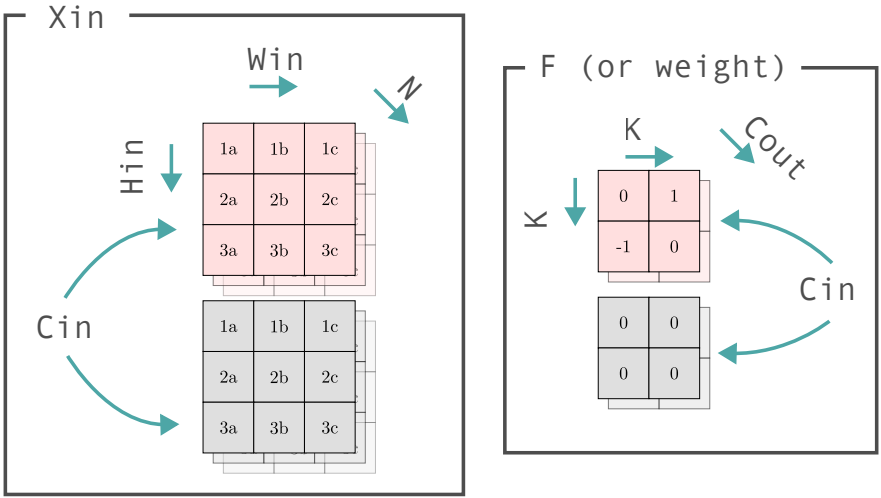
Computer Science
Illinois Institute of Technology

October 16, 2024

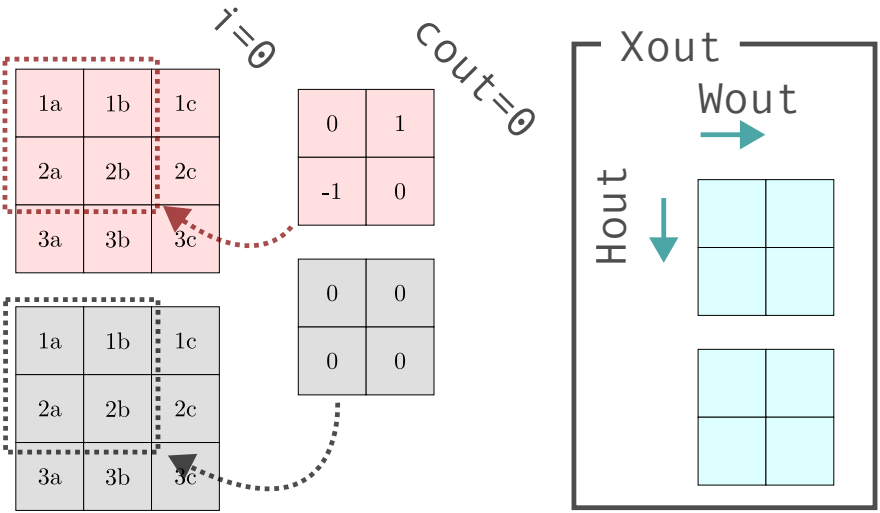
Input image tensor with shape $(N, C_{\text{in}}, H_{\text{in}}, W_{\text{in}})$



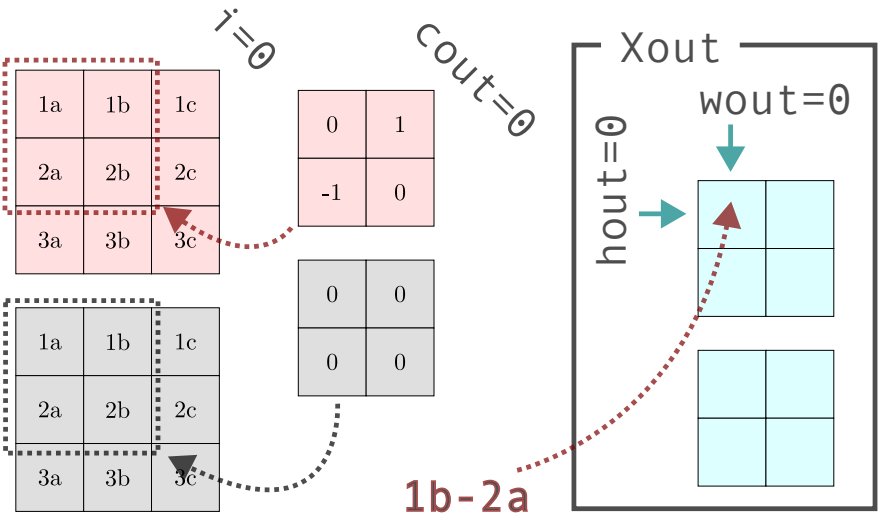
A single convolution layer $(C_{\text{out}}, C_{\text{in}}, K, K)$



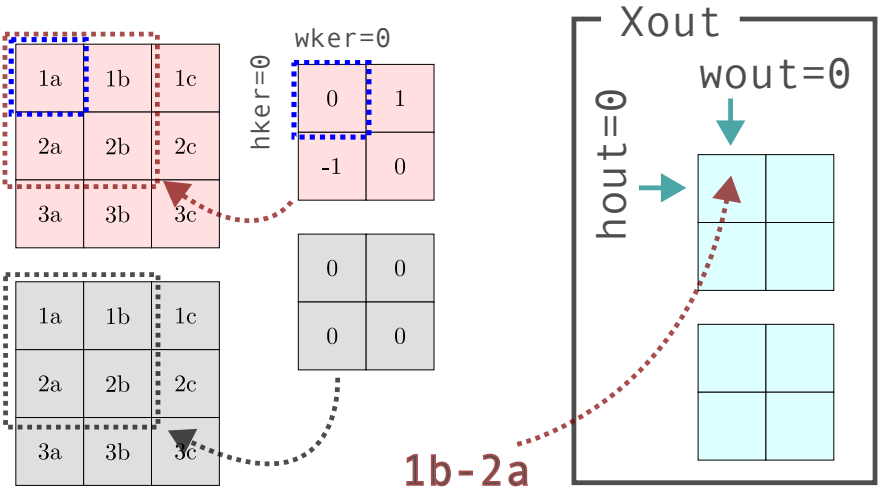
Output: `Xout` with shape $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$



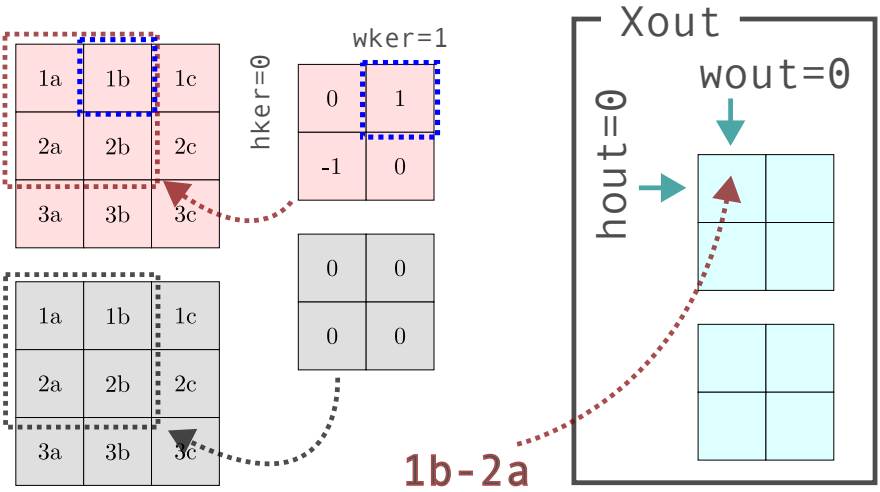
patch_idx = 0


i=0
cout=0
Xout
wout=0
hout=0
1a
1b
1c
2a
2b
2c
3a
3b
3c
0
1
-1
0
0
0
0
0
1b-2a

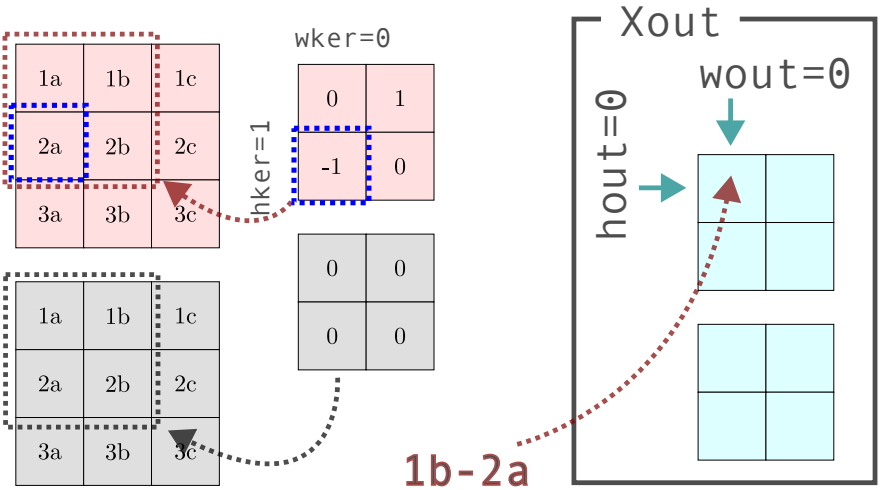
patch_idx = 0


1a
1b
1c
2a
2b
2c
3a
3b
3c
hker=0
wker=0
0
1
-1
0
0
0
0
0
1a
1b
1c
2a
2b
2c
3a
3b
3c
Xout
wout=0
hout=0
1b-2a

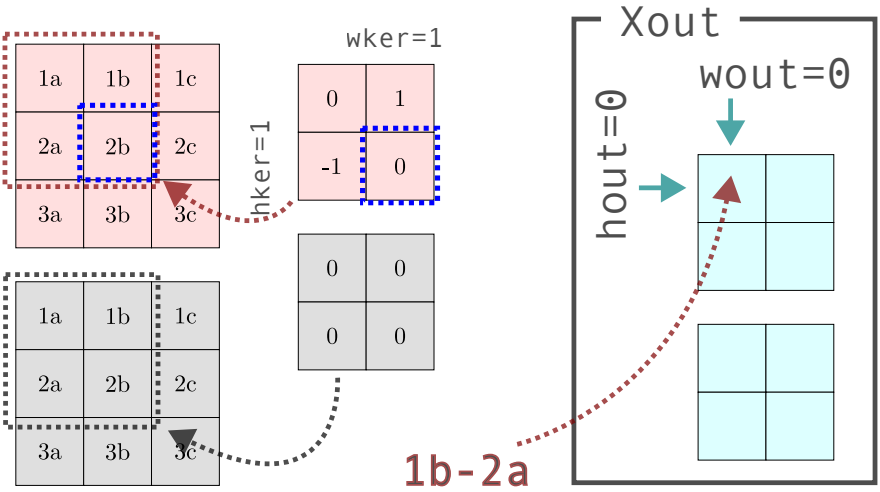
patch_idx = 0


1a
1b
1c
2a
2b
2c
3a
3b
3c
hker=0
wker=1
0
1
-1
0
0
0
0
0
1a
1b
1c
2a
2b
2c
3a
3b
3c
Xout
wout=0
hout=0
1b-2a

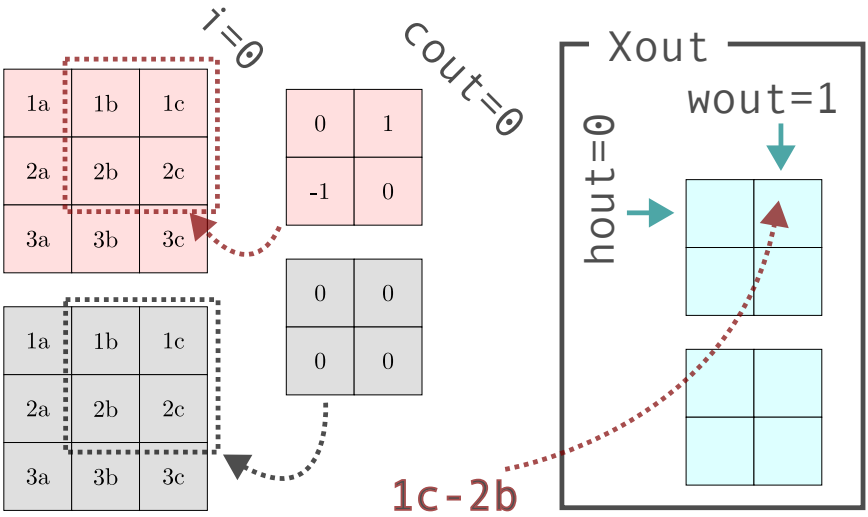
patch_idx = 0


1a
1b
1c
2a
2b
2c
3a
3b
3c
wker=0
hker=1
0
1
-1
0
0
0
0
0
1a
1b
1c
2a
2b
2c
3a
3b
3c
Xout
wout=0
hout=0
1b-2a

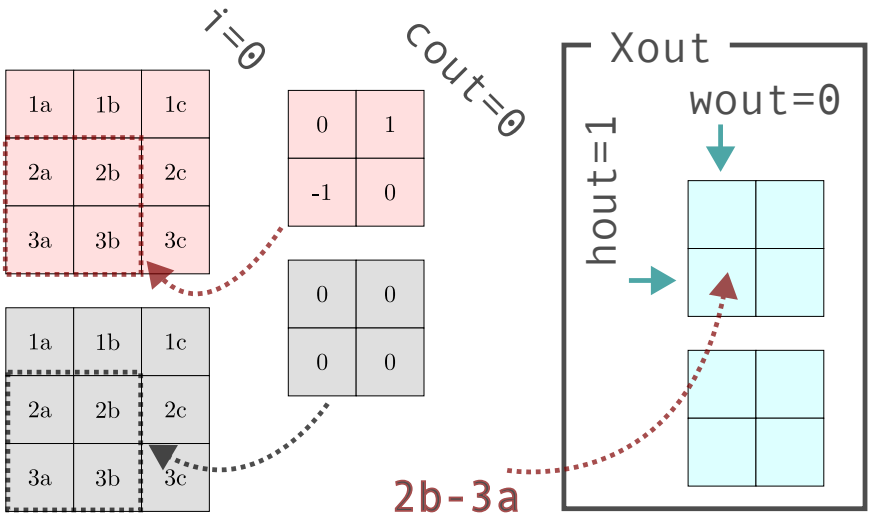
patch_idx = 0


1a
1b
1c
2a
2b
2c
3a
3b
3c
wker=1
hker=1
0
1
-1
0
0
0
0
0
1a
1b
1c
2a
2b
2c
3a
3b
3c
Xout
wout=0
hout=0
1b-2a

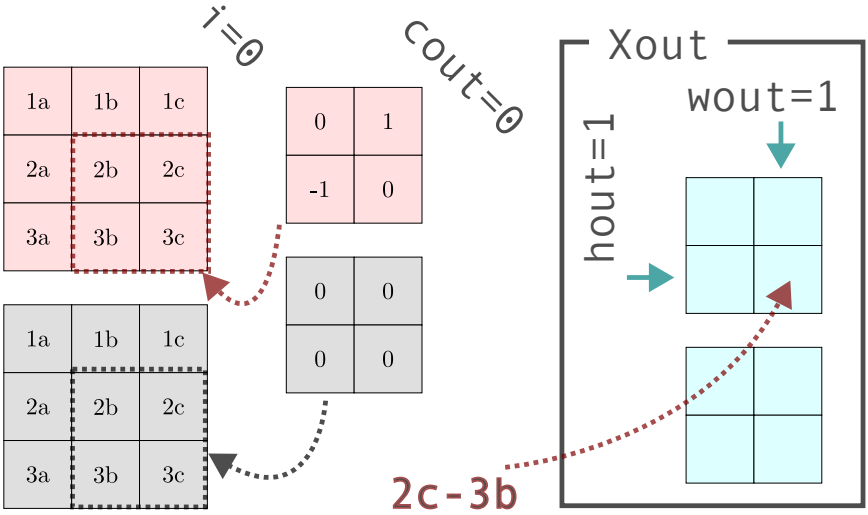
patch_idx = 1


i=0
cout=0
Xout
wout=1
hout=0
1a
1b
1c
2a
2b
2c
3a
3b
3c
0
1
-1
0
0
0
0
0
1c-2b

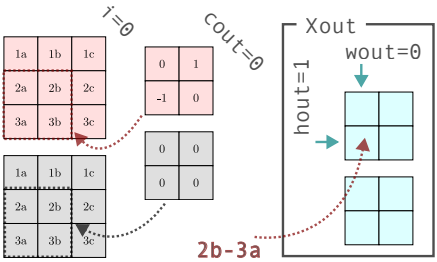
patch_idx = 2


i=0
cout=0
Xout
wout=0
hout=1
1a 1b 1c
2a 2b 2c
3a 3b 3c
0 1
-1 0
0 0
0 0
1a 1b 1c
2a 2b 2c
3a 3b 3c
2b-3a

patch_idx = 3


i=0
cout=0
Xout
wout=1
hout=1
1a 1b 1c
2a 2b 2c
3a 3b 3c
0 1
-1 0
0 0
0 0
1a 1b 1c
2a 2b 2c
3a 3b 3c
2c-3b

Sliding window (simple but slow)




```
# Suppose i, c_out, h_out, w_out are already defined, and
# X_out is initialized to all zeros
for cin in range(Cin):  # Input channels
    for hker in range(K):  # Kernel height
        for wker in range(K):  # Kernel width
            Xout[i, cout, hout, wout] += (
                F[cout, cin, hker, wker] *
                Xin[i, cin, hout + hker, wout + wker])
```

There's gotta be a better way! "`im2col`" (Next)

“im2col”

Main idea (in theory)

- Keep the window fixed in one spot
- Move the image tensor

Main idea (in theory)

- Create patches of the data
- Each patch has the same shape as the convolution filter
- Do matrix multiplication between the (flattened) patches and the (flattened) convolutional filter weights

Flatten the weights

1a	1b	1c
2a	2b	2c
3a	3b	3c

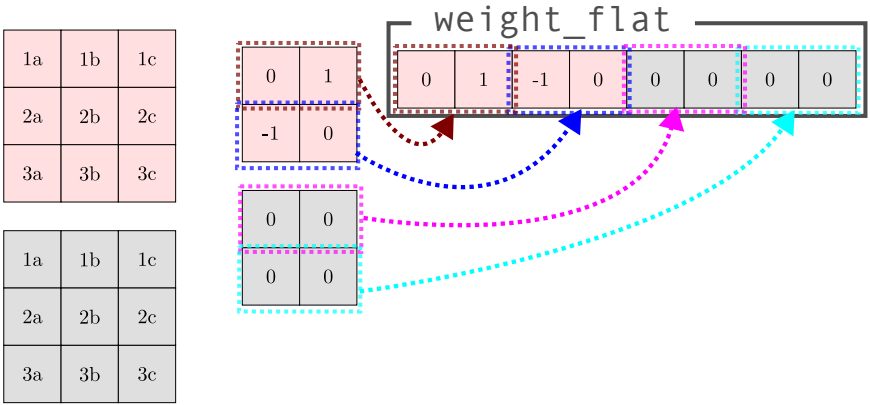
0	1
-1	0

0	0
0	0

1a	1b	1c
2a	2b	2c
3a	3b	3c

weight_flat

Flatten the weights



Flatten the weights

1a	1b	1c
2a	2b	2c
3a	3b	3c

0	1
-1	0

0	0
0	0

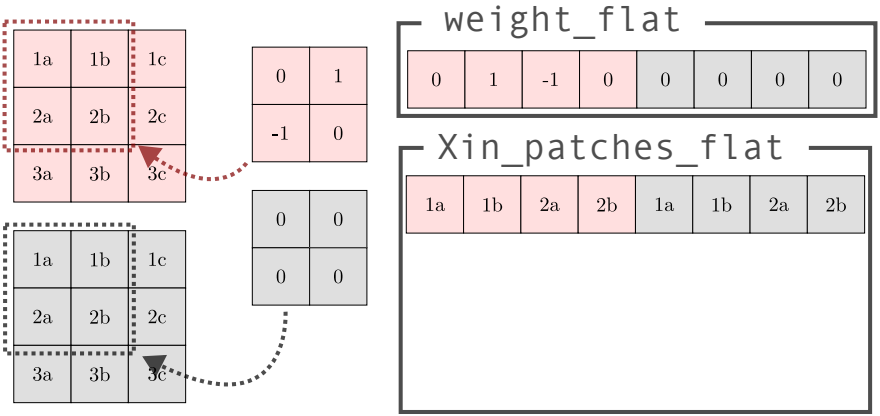
1a	1b	1c					
2a	2b	2c					
3a	3b	3c					

weight_flat

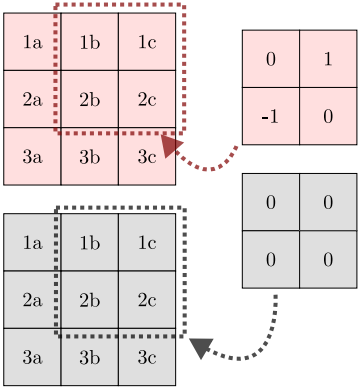
0	1	-1	0	0	0	0	0

Xin_patches_flat

Flatten the weights



Flatten the weights



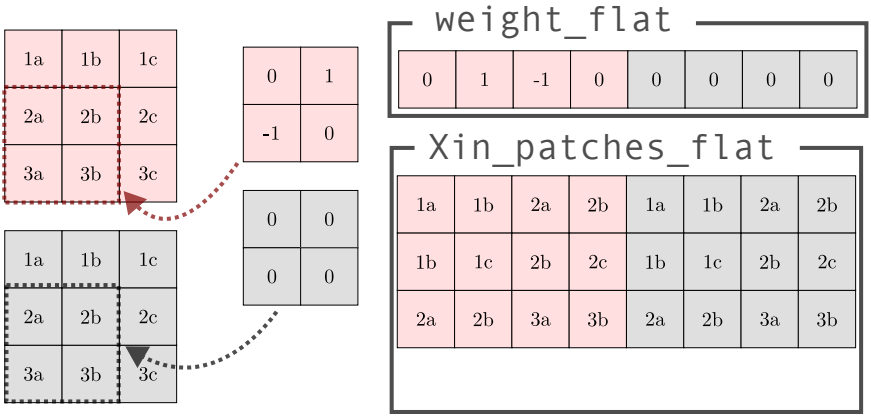
weight_flat

0	1	-1	0	0	0	0	0

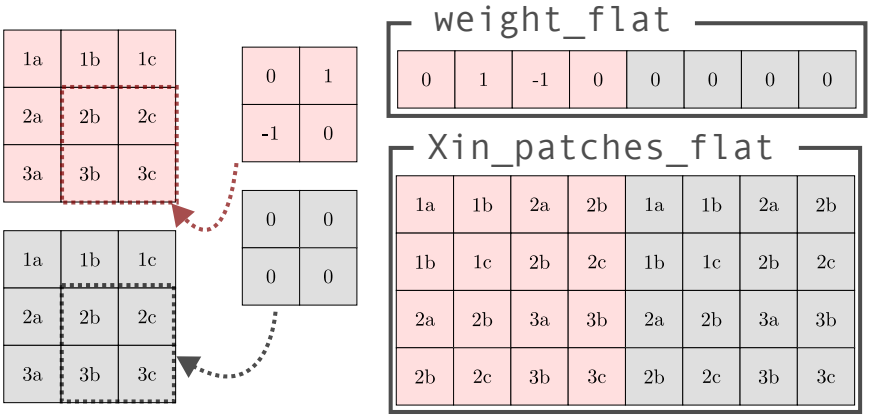
Xin_patches_flat

1a	1b	2a	2b	1a	1b	2a	2b
1b	1c	2b	2c	1b	1c	2b	2c

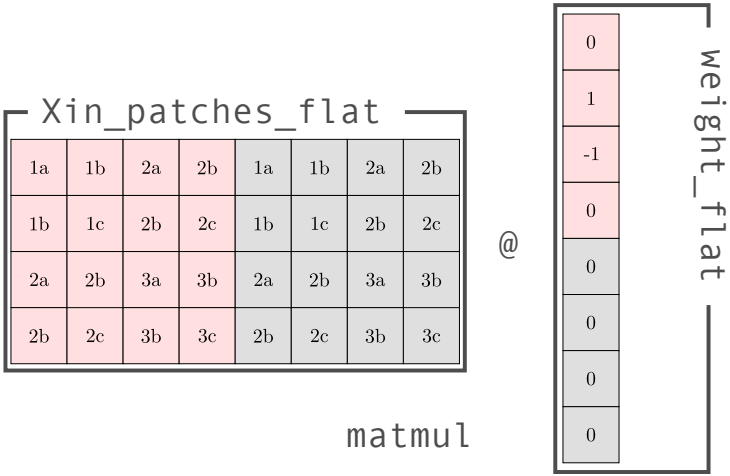
Flatten the weights



Flatten the weights



Sliding window via matmul



Sliding window via matmul

Xin_patches_flat

1a	1b	2a	2b	1a	1b	2a	2b
1b	1c	2b	2c	1b	1c	2b	2c
2a	2b	3a	3b	2a	2b	3a	3b
2b	2c	3b	3c	2b	2c	3b	3c

@

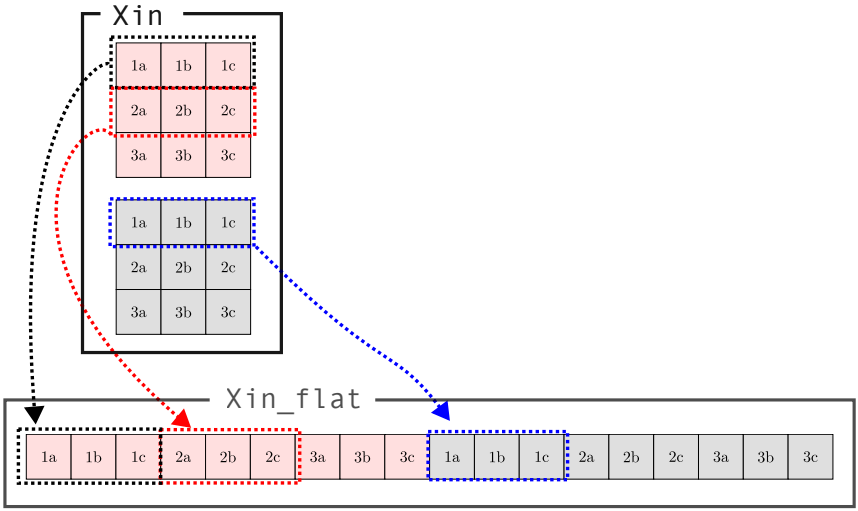
weight_flat

0	0
1	0
-1	0
0	0
0	1
0	0
0	-1
0	0

matmul

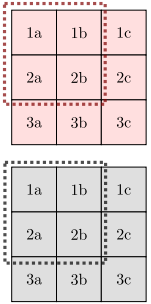
Cout

Flattening `Xin` into `Xin_flat`

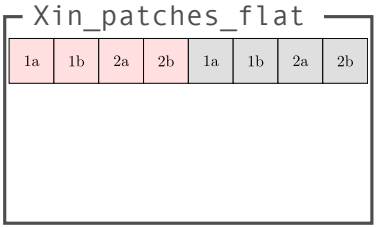


Flattening `Xin` into `Xin_flat`

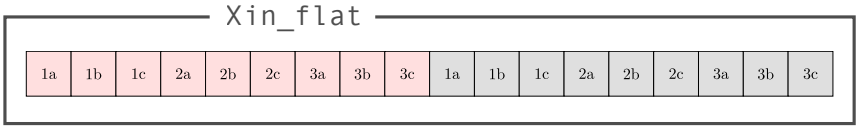
Recall `patch_idx=0`:



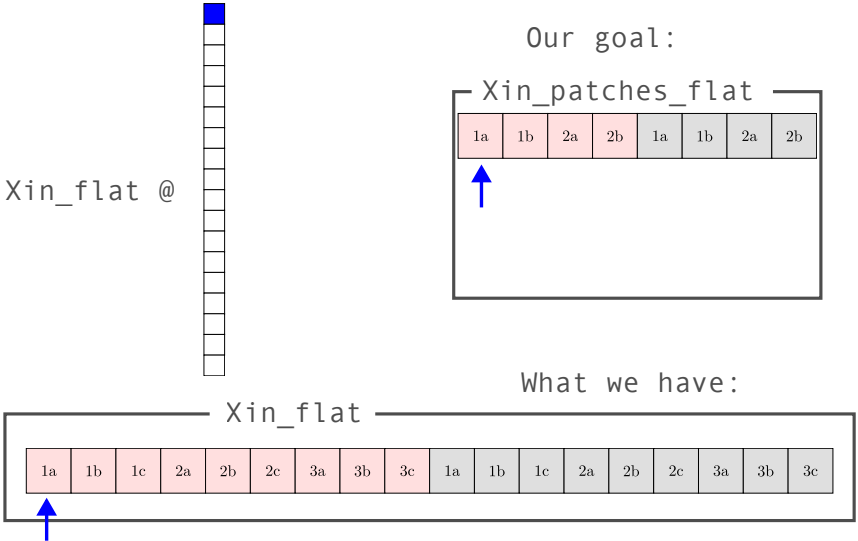
Our goal:



What we have:

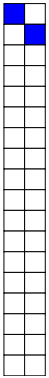


The “im2col” matrix



The “im2col” matrix

Xin_flat @



Our goal:

Xin_patches_flat

1a	1b	2a	2b	1a	1b	2a	2b

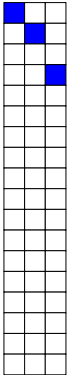
What we have:

Xin_flat

1a	1b	1c	2a	2b	2c	3a	3b	3c	1a	1b	1c	2a	2b	2c	3a	3b	3c

The “im2col” matrix

Xin_flat @



Our goal:

Xin_patches_flat

1a	1b	2a	2b	1a	1b	2a	2b

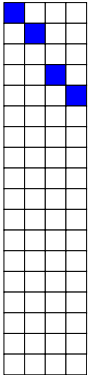
What we have:

Xin_flat

1a	1b	1c	2a	2b	2c	3a	3b	3c	1a	1b	1c	2a	2b	2c	3a	3b	3c

The “im2col” matrix

Xin_flat @



Our goal:

Xin_patches_flat

1a	1b	2a	2b	1a	1b	2a	2b

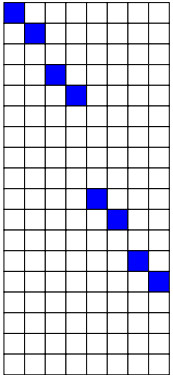
What we have:

Xin_flat

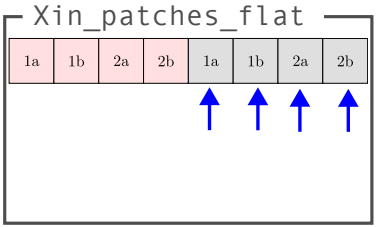
1a	1b	1c	2a	2b	2c	3a	3b	3c	1a	1b	1c	2a	2b	2c	3a	3b	3c

The “im2col” matrix

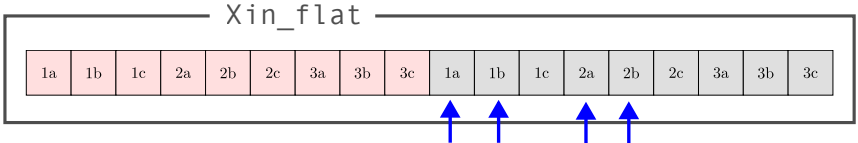
Xin_flat @



Our goal:

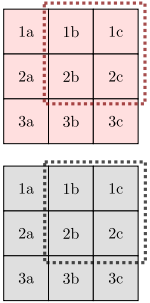


What we have:

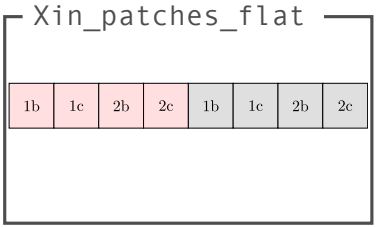


The “im2col” matrix

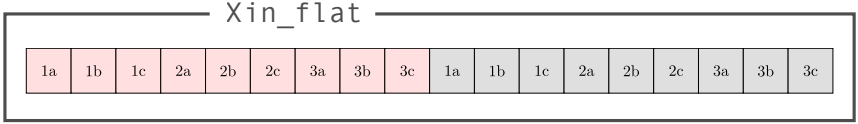
Recall patch_idx=1:



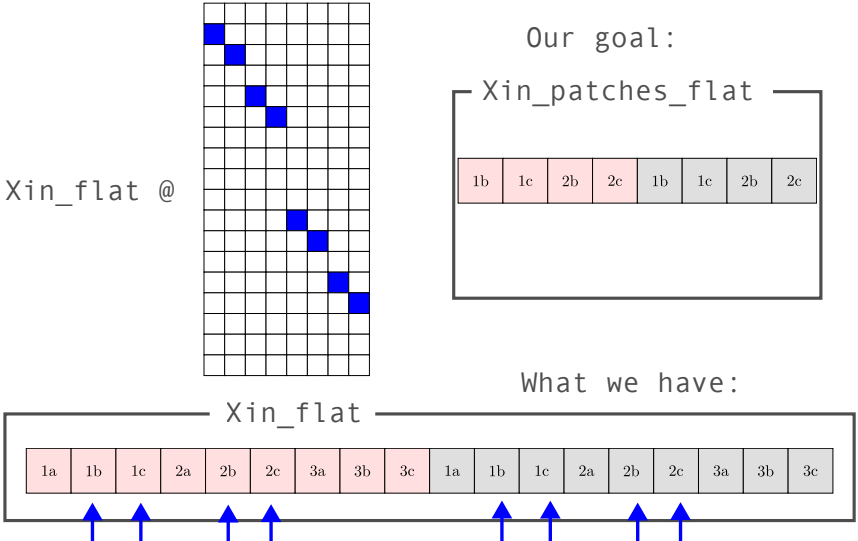
Our goal:



What we have:

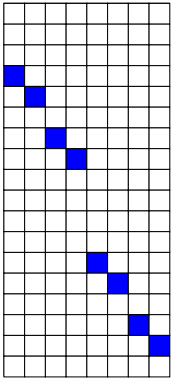


The "im2col" matrix



The “im2col” matrix

Xin_flat @



Our goal:

Xin_patches_flat

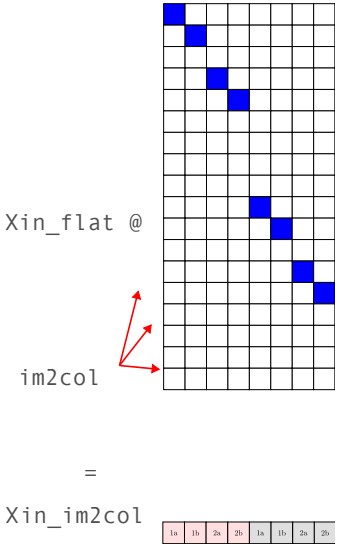
2a	2b	3a	3b	2a	2b	3a	3b

What we have:

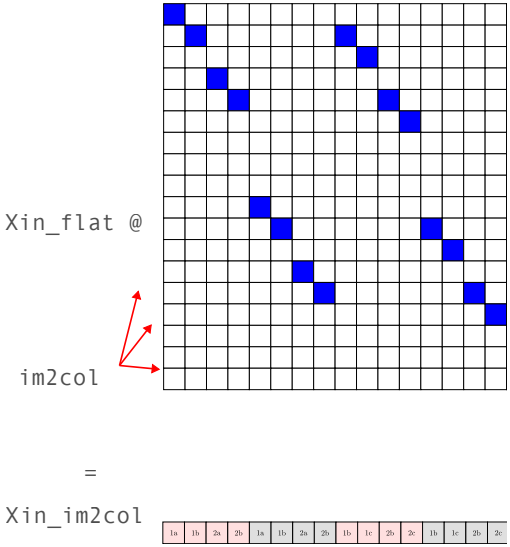
Xin_flat

1a	1b	1c	2a	2b	2c	3a	3b	3c	1a	1b	1c	2a	2b	2c	3a	3b	3c

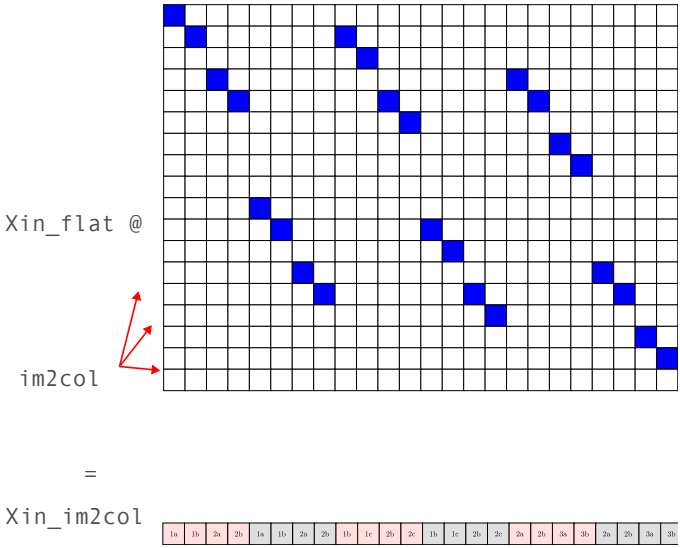
The “im2col” matrix



The “im2col” matrix

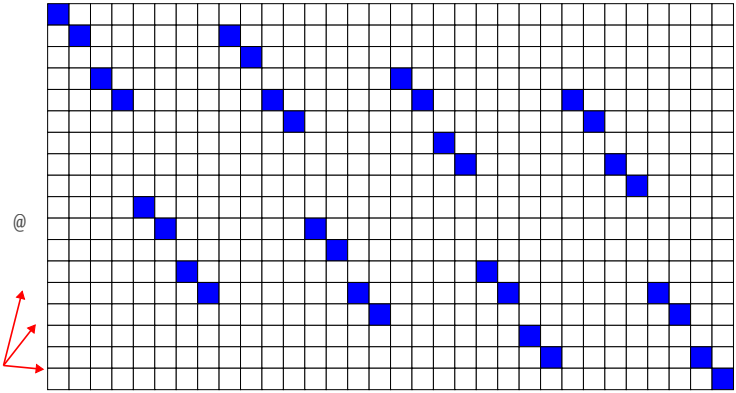


The “im2col” matrix



The “im2col” matrix

Xin_flat @



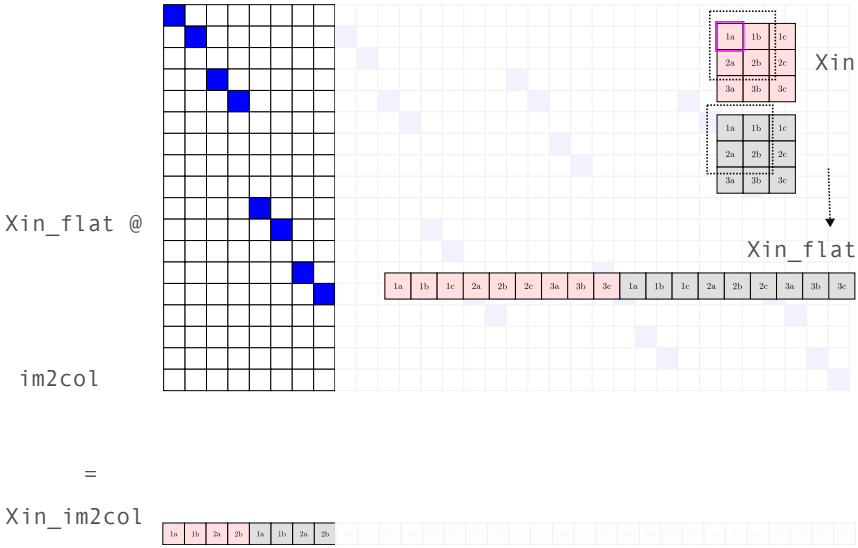
im2col

=

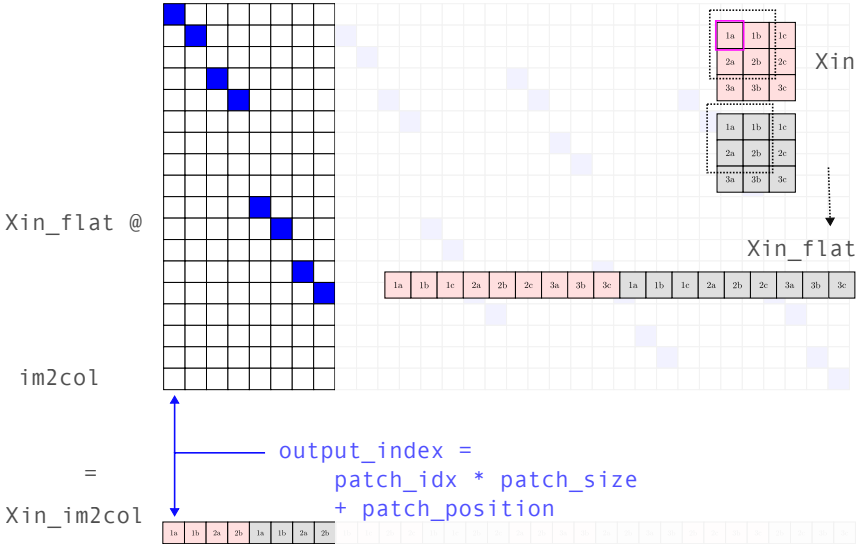
Xin_im2col

1a	1b	2a	2b	1a	1b	2a	2b	1b	1c	2b	2c	1b	1c	2b	2c	2a	2b	3a	3b	2a	2b	3a	3b	2b	2c	3b	3c	2b	2c	3b	3c

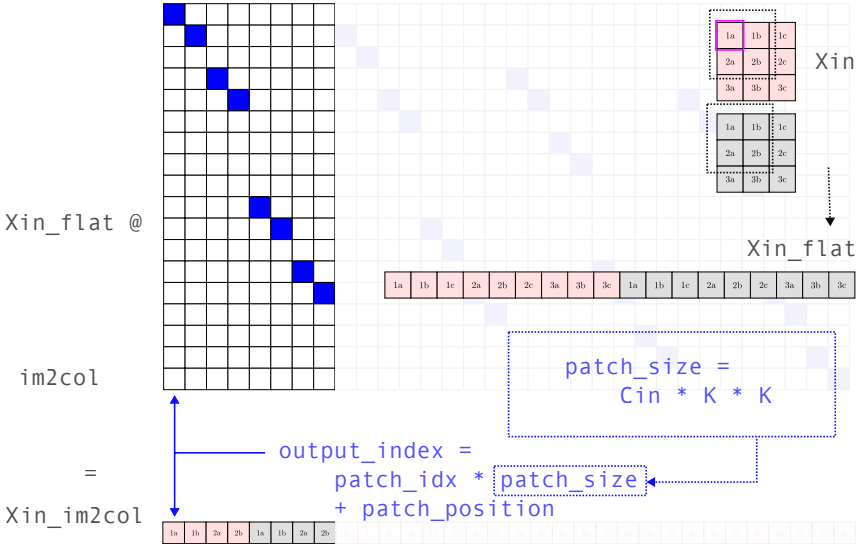
The “im2col” matrix



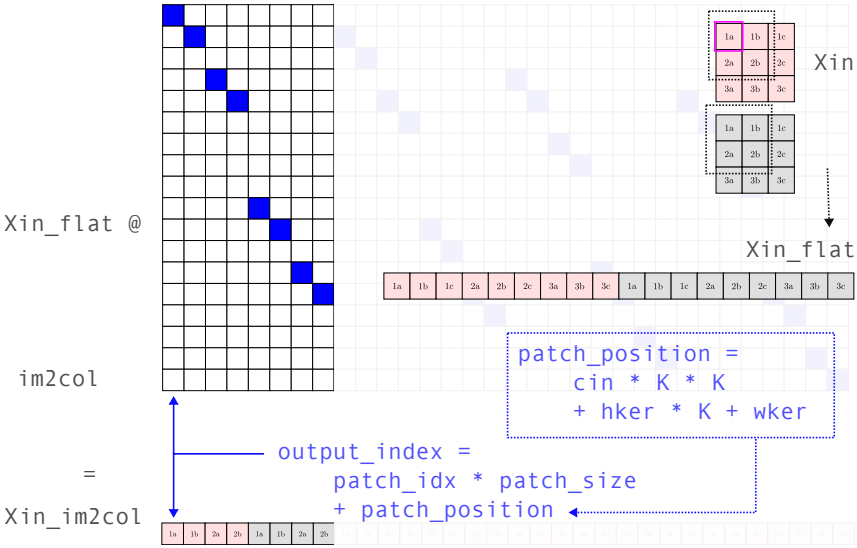
The “im2col” matrix



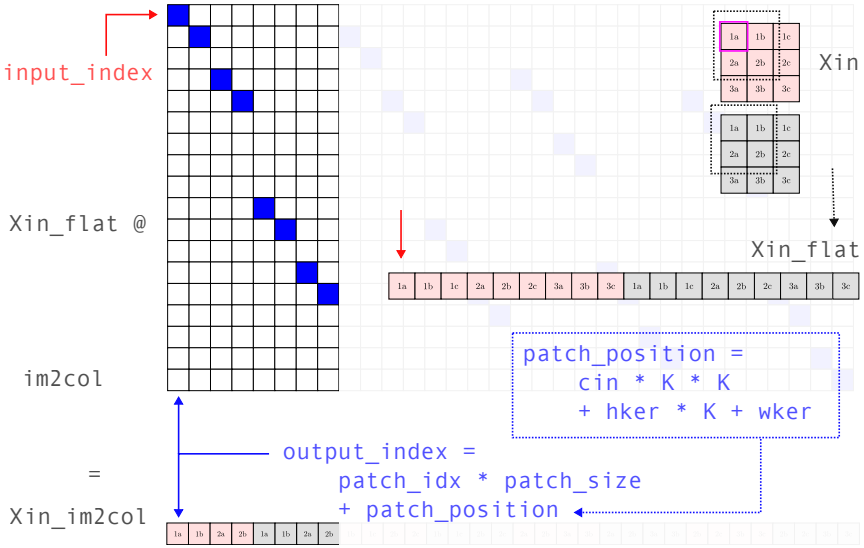
The “im2col” matrix



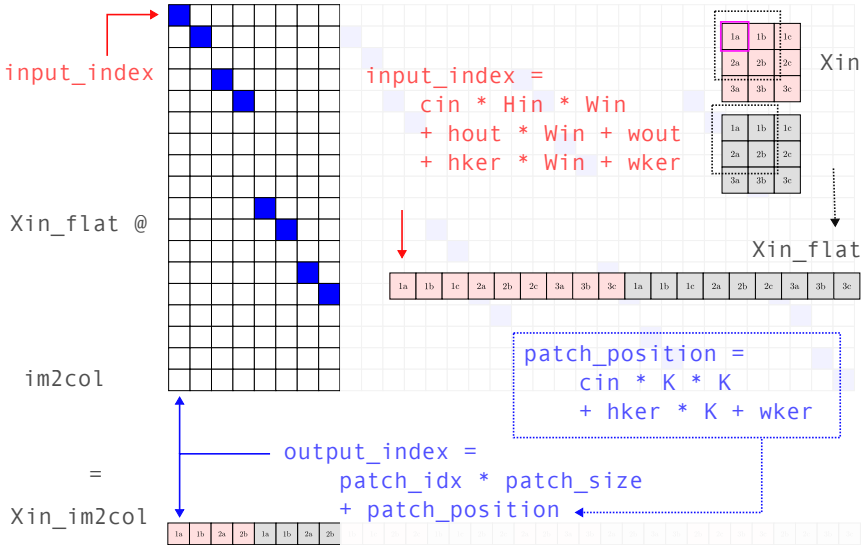
The “im2col” matrix



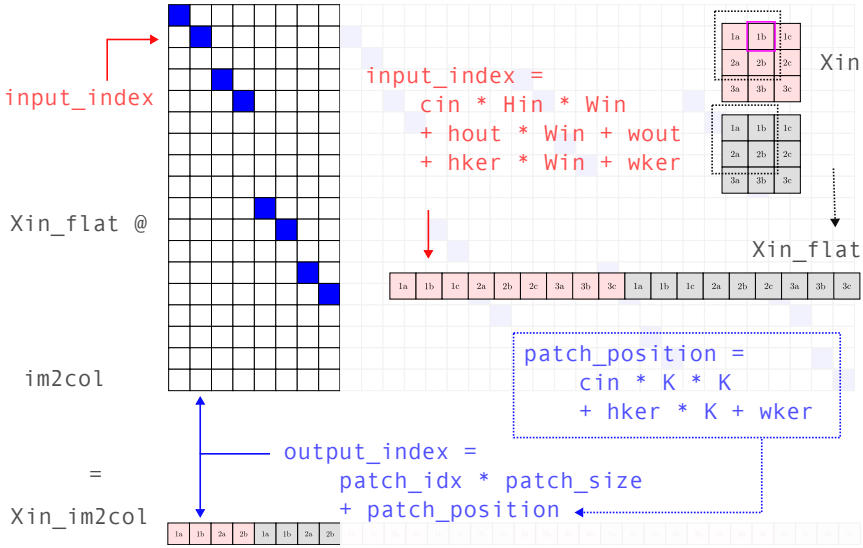
The “im2col” matrix



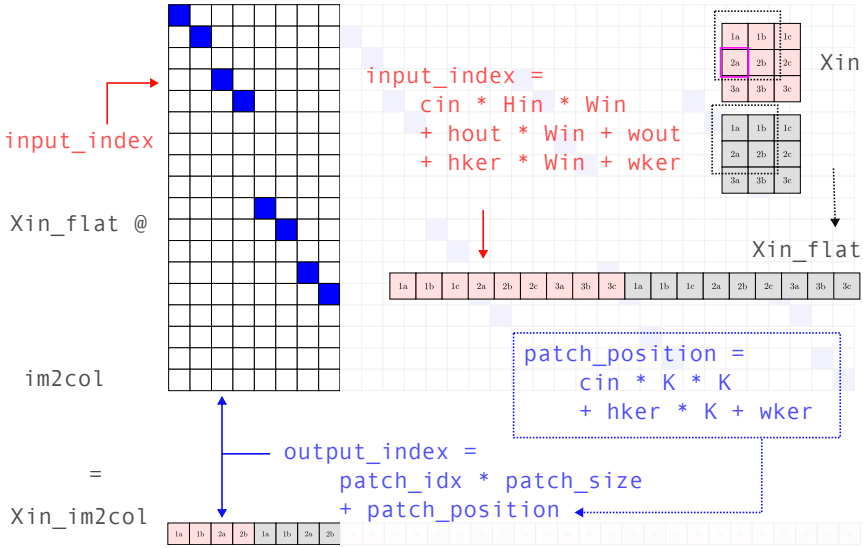
The “im2col” matrix



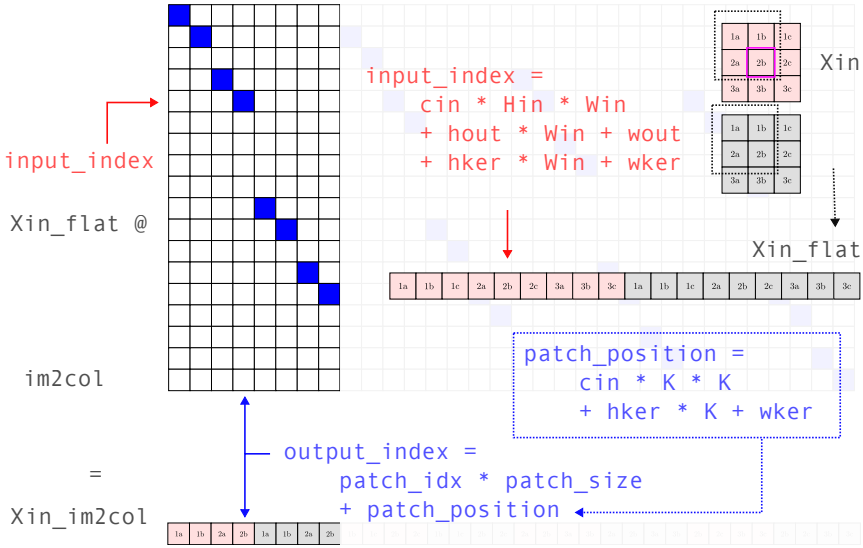
The “im2col” matrix



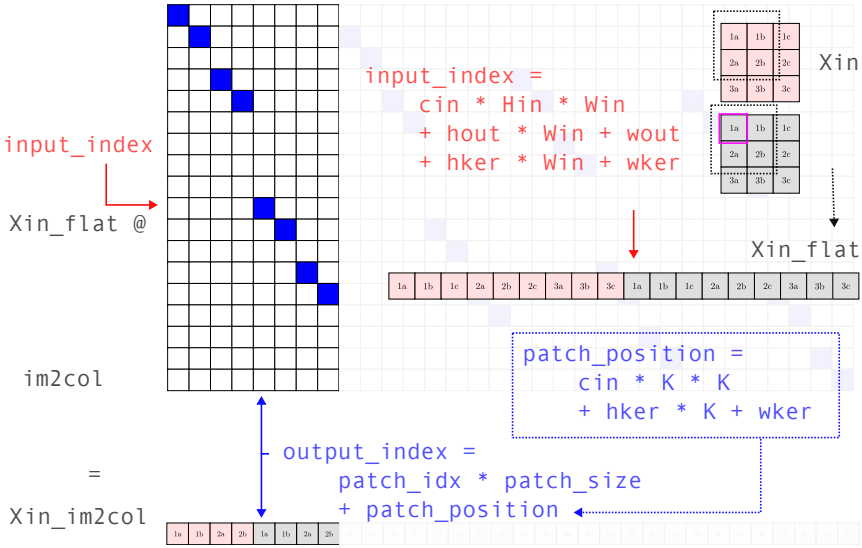
The “im2col” matrix



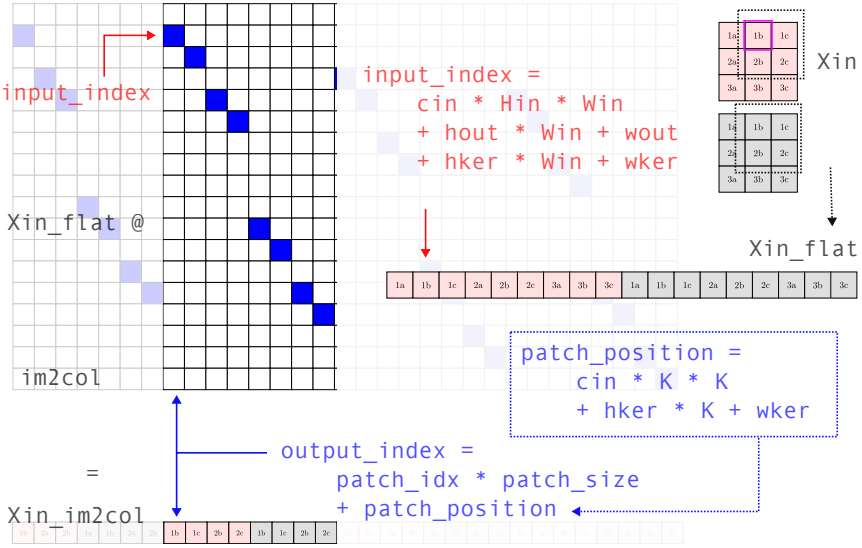
The “im2col” matrix



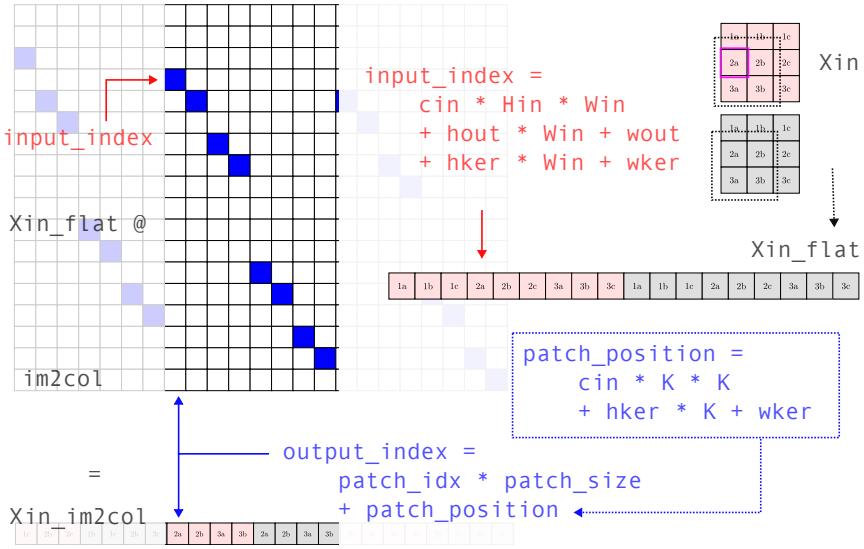
The “im2col” matrix



The “im2col” matrix



The “im2col” matrix



Putting it all together

```python
def im2col_matrix_dense(Xin, K, S=1):
    N, Cin, Hin, Win = Xin.shape
    Hout, Wout = Hin - K + 1, Win - K + 1
    P = Hout * Wout  # Total number of patches per image
    patch_size = Cin * K * K # Size of each flattened patch
    im2col_mat_dense = np.zeros((Cin * Hin * Win, P * patch_size))

    # [main loop on next slide...]

    return im2col_mat_dense
```

Putting it all together

```
    # [continued from previous slide...]
    patch_idx = 0
    for hout in range(Hout):
        for wout in range(Wout):
            for cin in range(Cin):
                for hker in range(K):
                    for wker in range(K):
                        input_index = cin * Hin * Win + hout * Win + wout + hker * Win + wker
                        patch_position = cin * K * K + hker * K + wker
                        output_index = patch_idx * patch_size + patch_position
                        im2col_mat_dense[input_index, output_index] = 1
            patch_idx += 1
```

Stride = 2

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e
3a	3b	3c	3d	3e
4a	4b	4c	4d	4e
5a	5b	5c	5d	5e

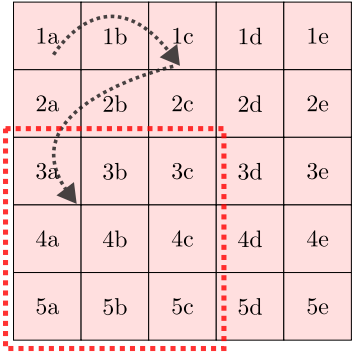
1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

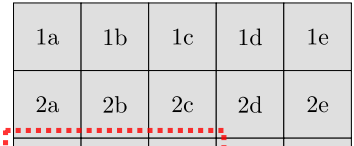
Stride = 2

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e
3a	3b	3c	3d	3e
4a	4b	4c	4d	4e
5a	5b	5c	5d	5e

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

Stride = 2


1a 1b 1c 1d 1e
2a 2b 2c 2d 2e
3a 3b 3c 3d 3e
4a 4b 4c 4d 4e
5a 5b 5c 5d 5e


1a 1b 1c 1d 1e
2a 2b 2c 2d 2e

Implement `im2col_matrix_dense` with stride

See `lec09-in-class-ex1-im2col.ipynb`
End of part 1.

References I