HW1 Recitation:
MLPs and CNNs

10707: Advanced Deep Learning

Chris Ki, Albert Liang, Jocelyn Tseng

Slides adapted from 10-707 Spring 2022 and 10-417/617 Fall 2022

Administrative

Homework 1 is due **Wednesday, Feb 15, 11:59 pm**

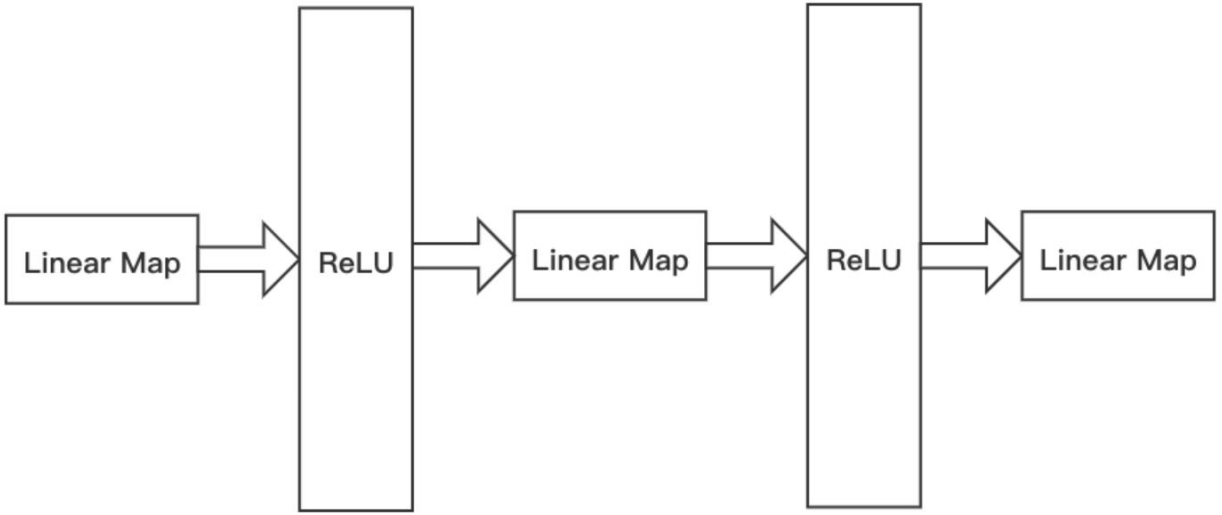
Please check Piazza for HW1 FAQs note for updates and questions!

MLPs

1. Overview
2. DropOut
3. BatchNorm

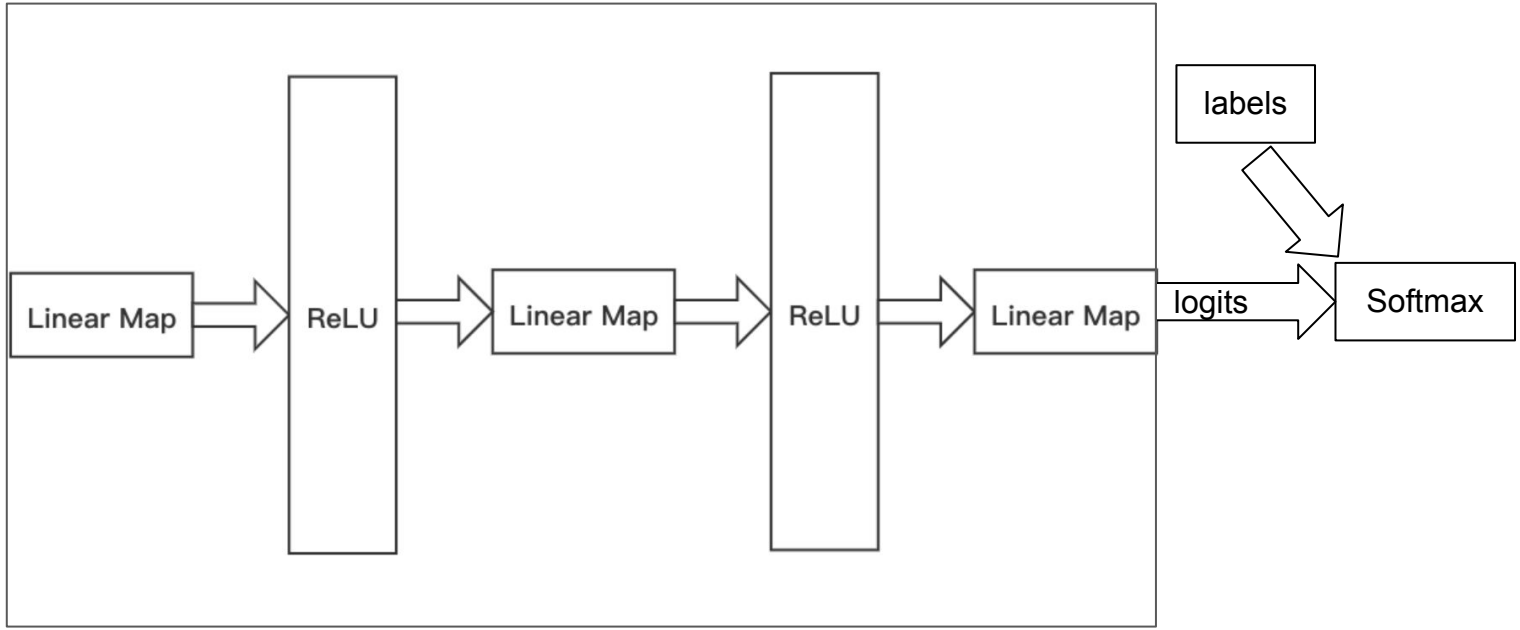
****Recommend doing problem 3 before coding MLP's and CNN's****

MLP Overview



Regularization goes before activation layer

Training: Forward

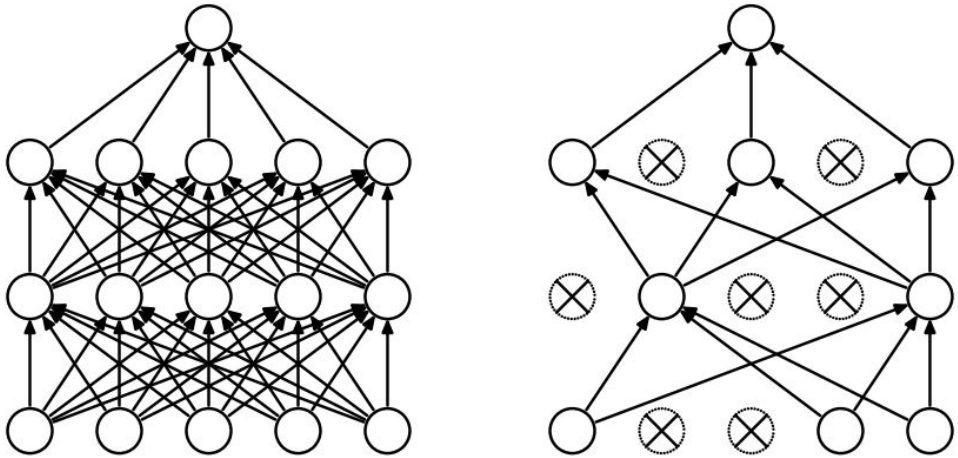


Training: Backward

- Softmax backward outputs grads
- grads => backpropagation
- step to apply gradient updates to weights
- remember to zerograd so gradients do not accumulate across epochs

Dropout

- Only perform dropout for training
- Generate mask based on probability p, dropping out with probability p
- use np.random.binomial



(a) Standard Neural Net

(b) After applying dropout.

image from: https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

Batchnorm

- GOAL: **prevent activation saturation** and/or exploding gradients when using sigmoid
- for each hidden layer of the currently training minibatch, you want to **normalize** the activations to a **standard normal distribution**
- then apply **linear transformation** with learned parameters

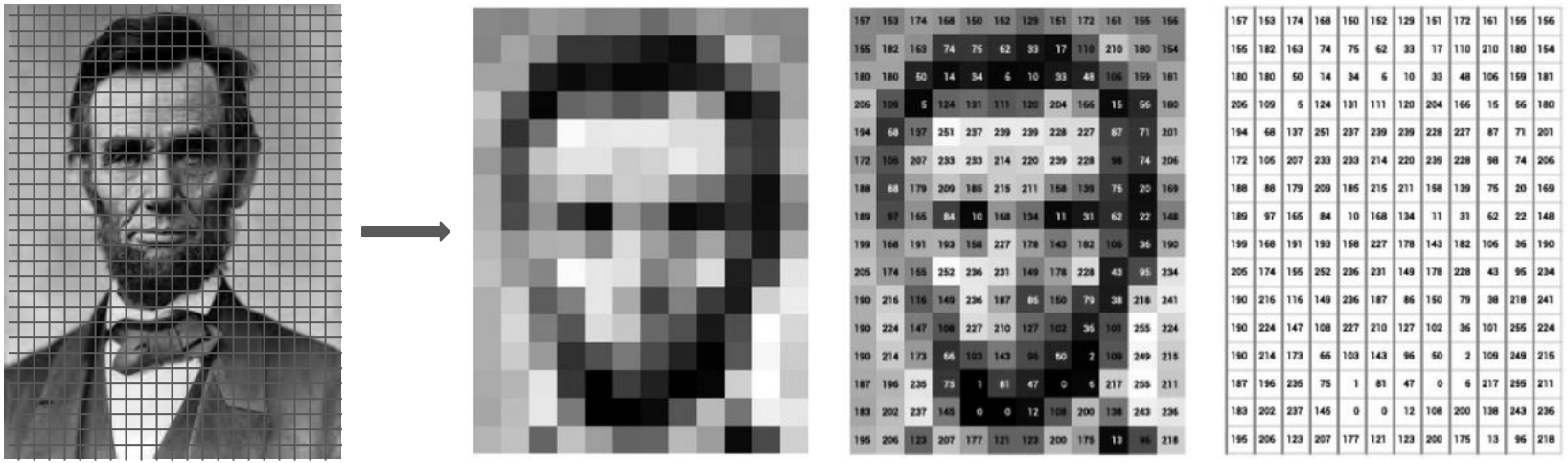
helpful material here (linked in the hw1 writeup)

CNNs

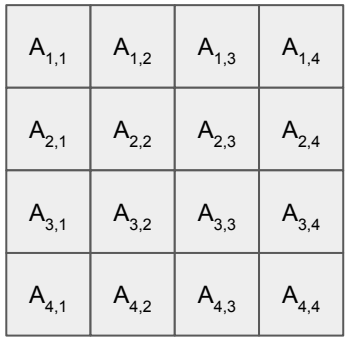
Some figures adapted from Simon Lucey's 16-720b slides and
Introduction to Deep Learning 11-785

What is an image?

A Matrix `I` of dimensions `(C,M,N)` with `I[c][i][j] = intensity(pixel(c,i,j))`



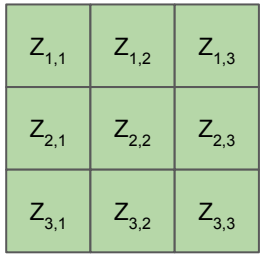
Applying a kernel to a channel of the image



Input - **A**

Kernel - **W**

Bias - **B**

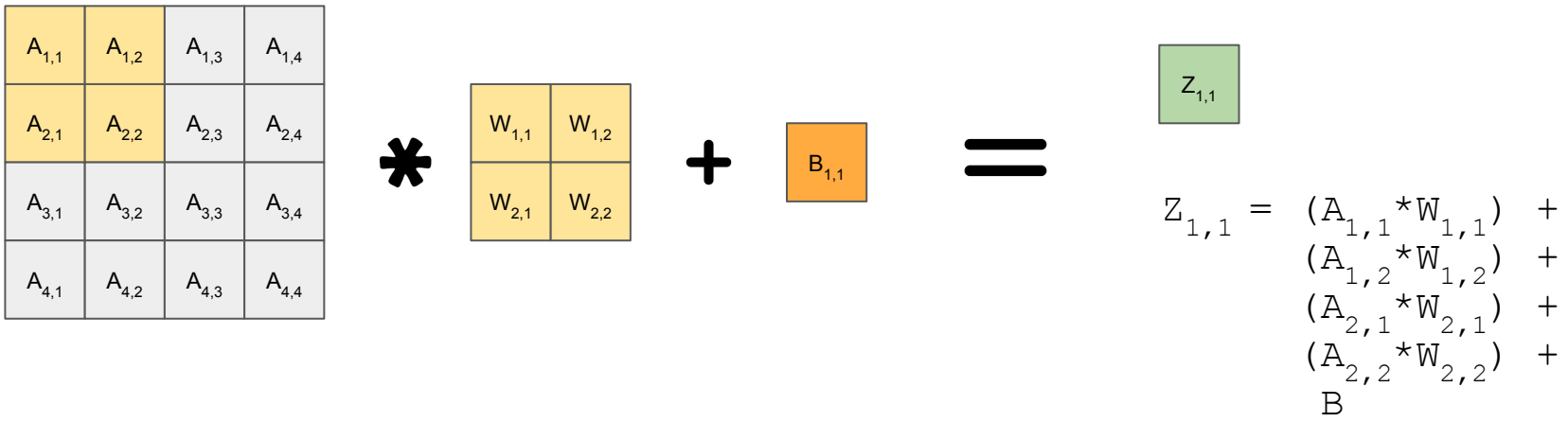


Output - **Z**

$$\mathbf{Z = (A \otimes W) + B}$$

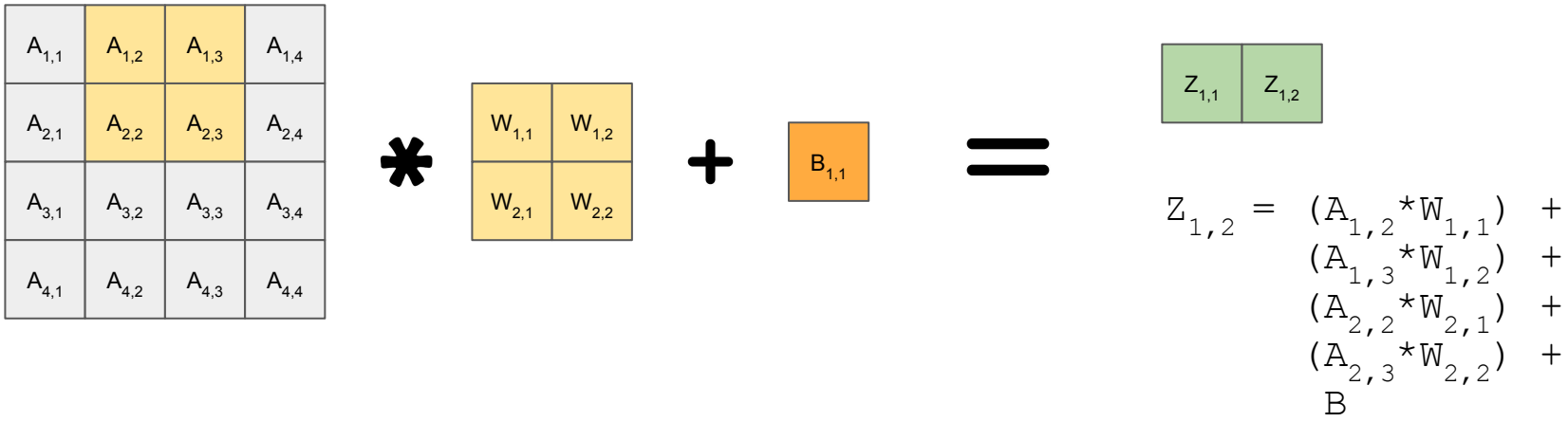
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations



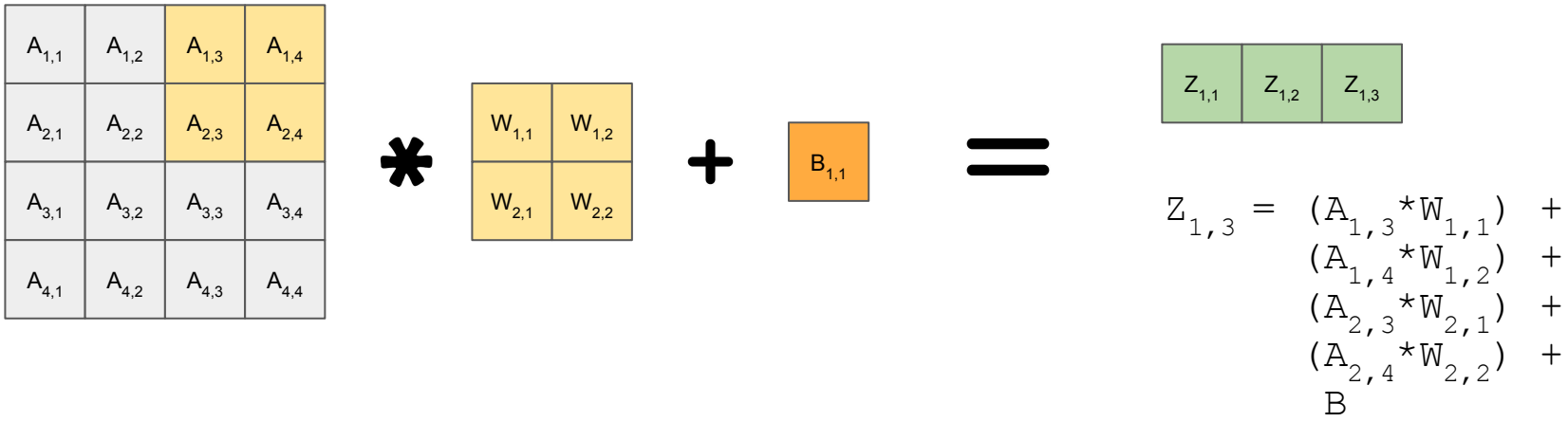
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations



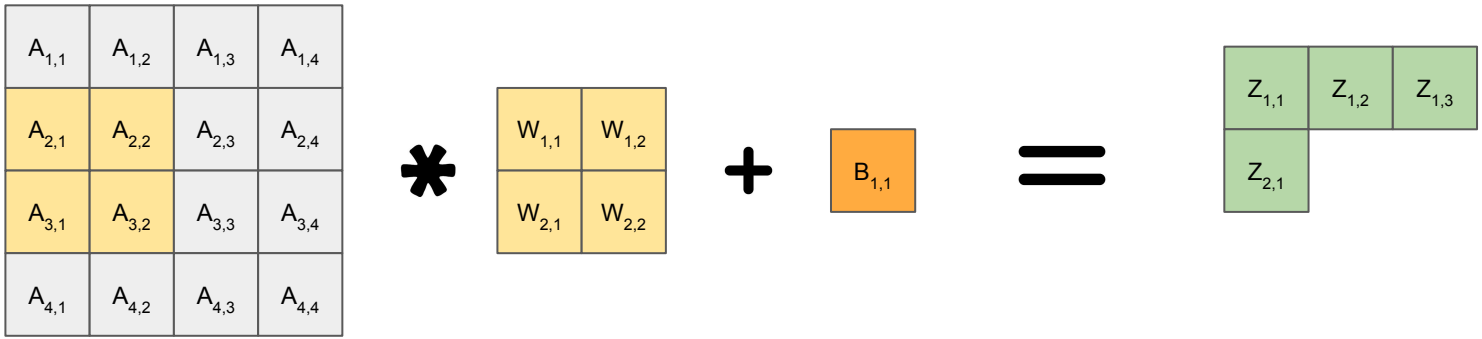
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations



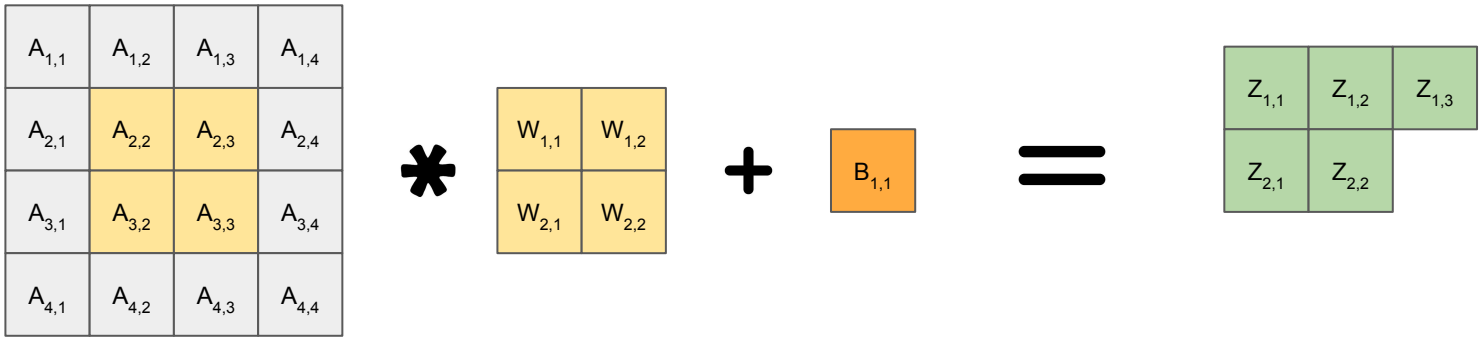
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations



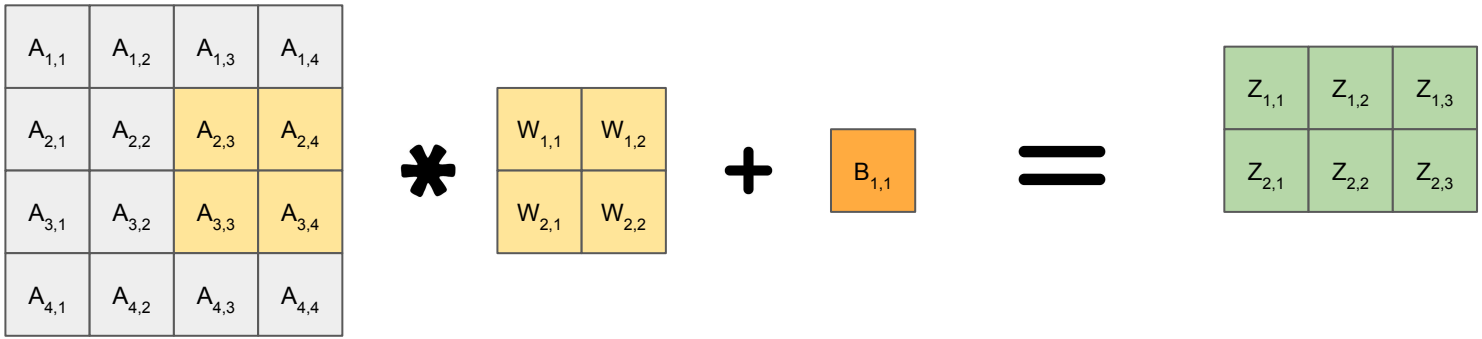
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations



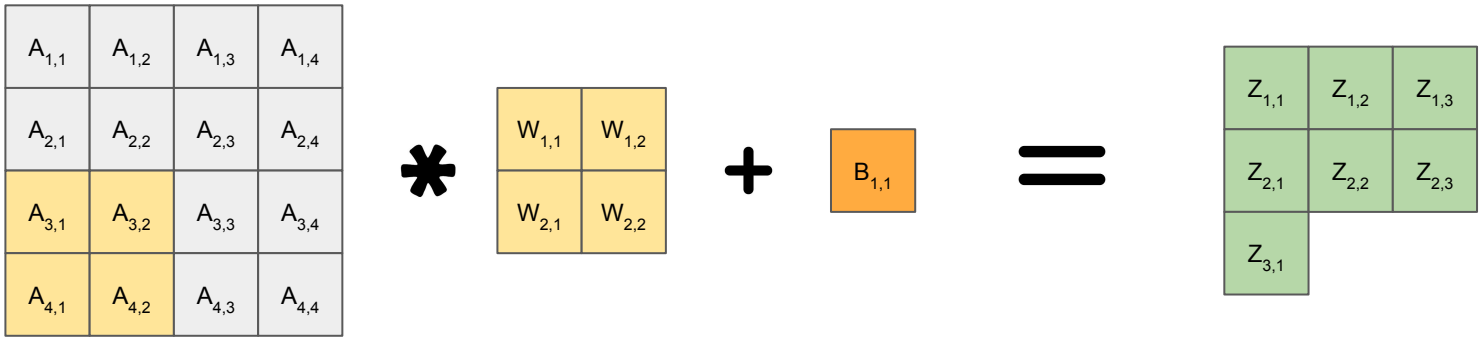
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations



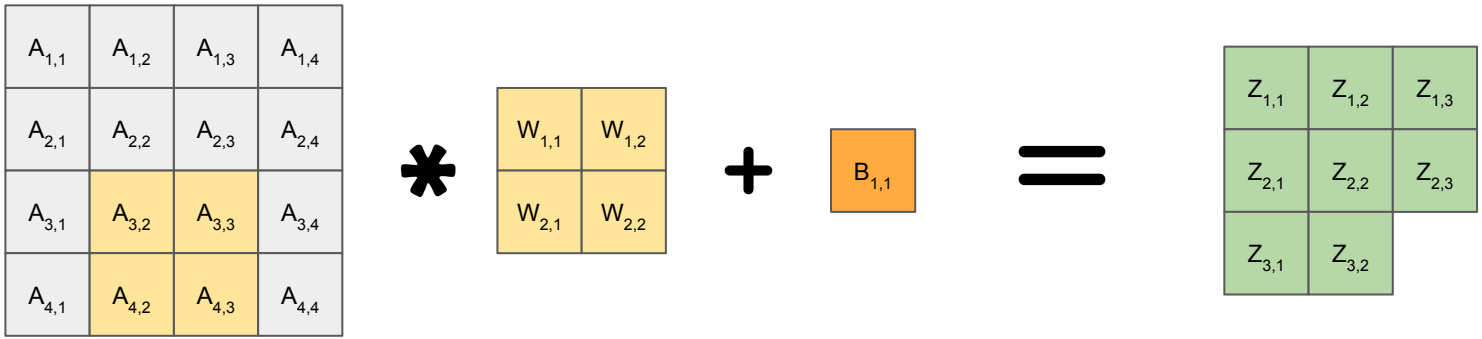
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations



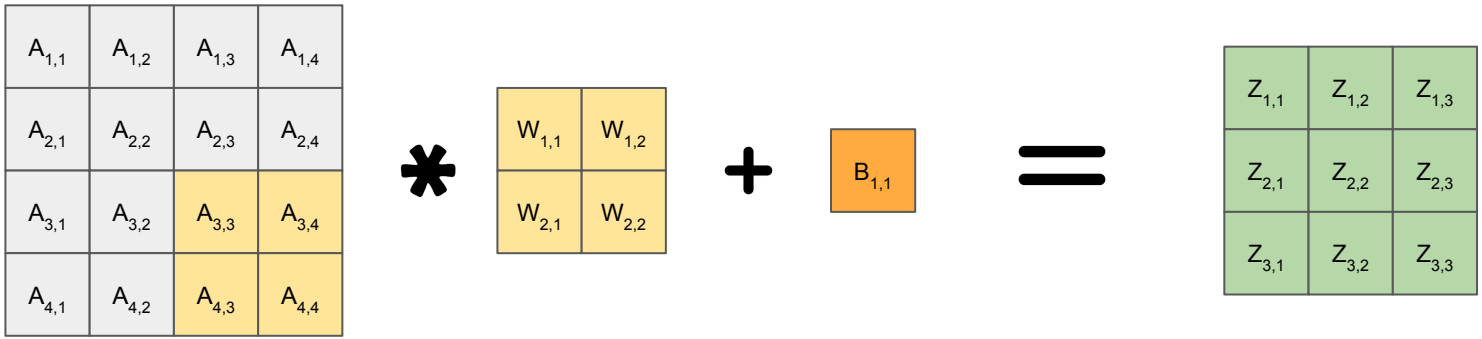
Applying a kernel to a channel of the image

Essentially element-wise multiplications and summations

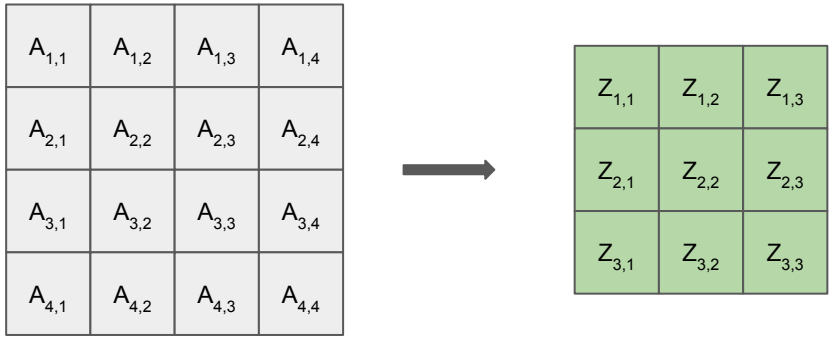


Applying a kernel to a channel of the image

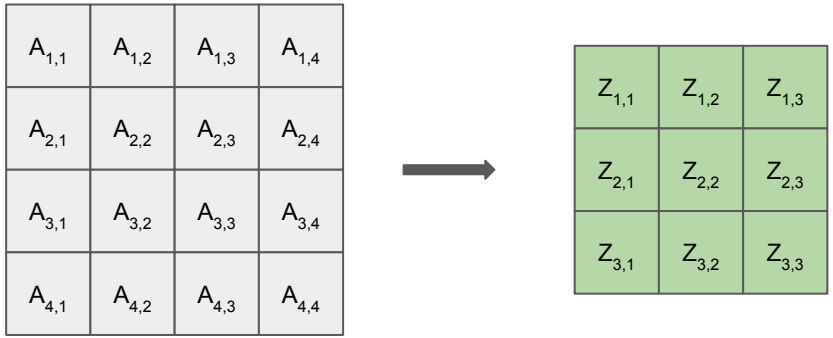
Essentially element-wise multiplications and summations



Formula of output size



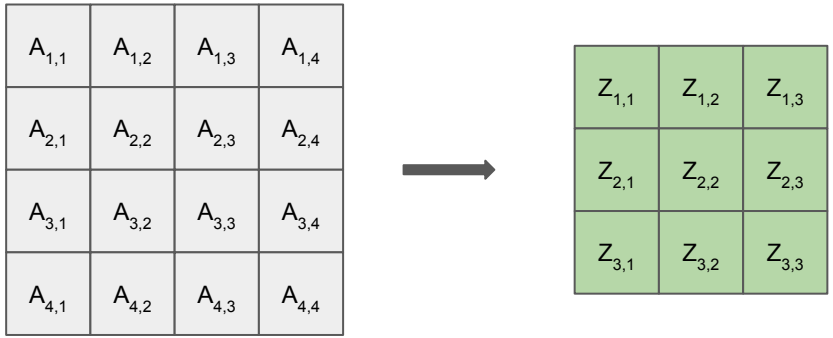
Formula of output size



$$\text{Output Width} = [(W_{in} - W_k + 2P) \,//\, (S)] + 1$$

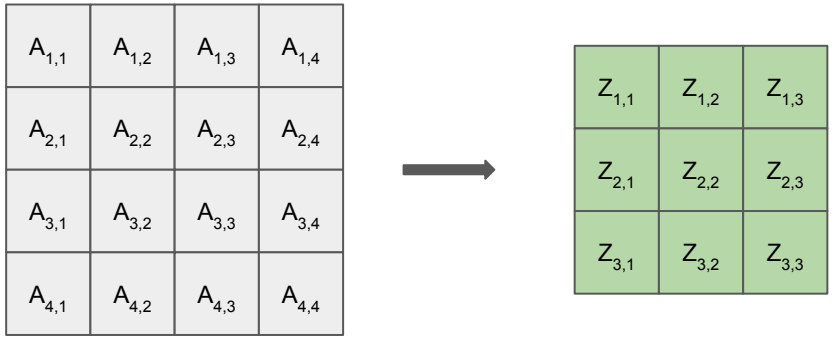
$$\text{Output Height} = [(H_{in} - H_k + 2P) \,//\, (S)] + 1$$

Formula of output size



$$\text{Output Width} = \left[(W_{in} - W_k + 2\mathbf{P}) \;//\; (\mathbf{S})\right] + 1$$

Formula of output size



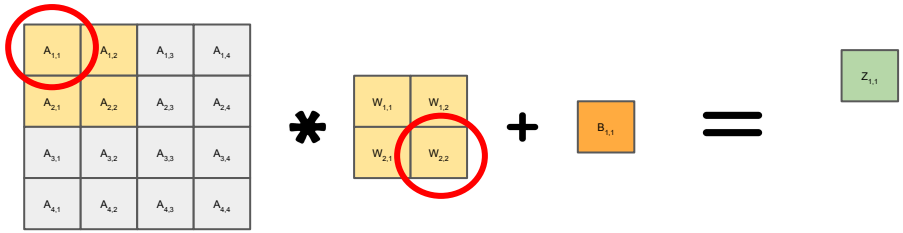
Output Width = $[(W_{in} - W_k + 2\mathbf{P}) // (\mathbf{S})] + 1$

P: Padding (here - 0)
S: Stride (here - 1)

Padding

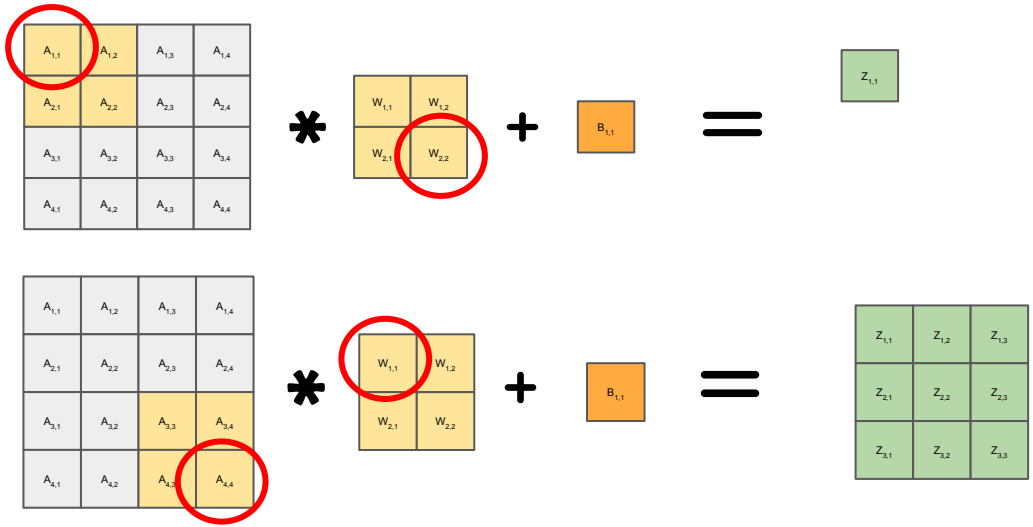
- Padding (i.e. attaching) zeros (usually) around inputs.
- Images can be padded to the left, right, top, and bottom.

One reason for padding

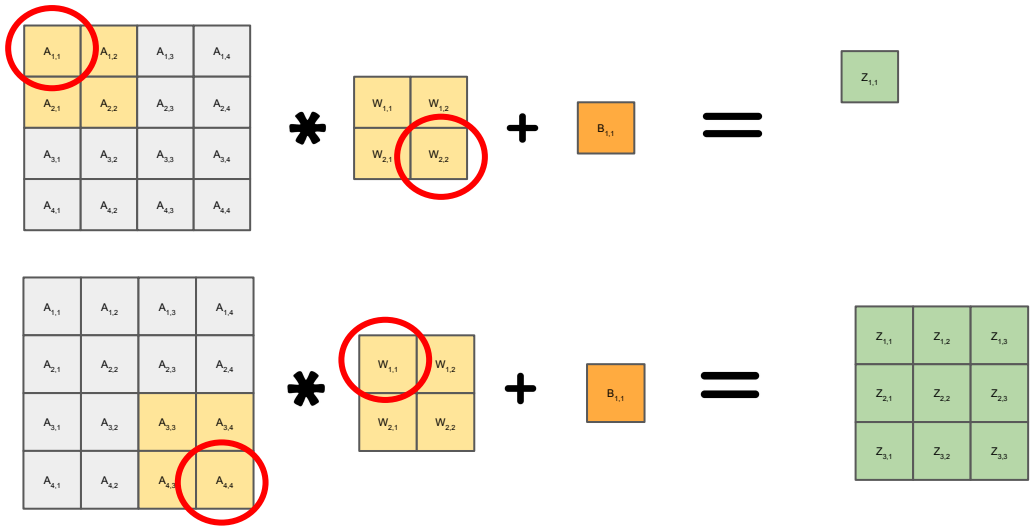


$$Z_{1,1} = (A_{1,1} * W_{1,1}) + (A_{1,2} * W_{1,2}) + (A_{2,1} * W_{2,1}) + (A_{2,2} * W_{2,2}) + B$$

One reason for padding

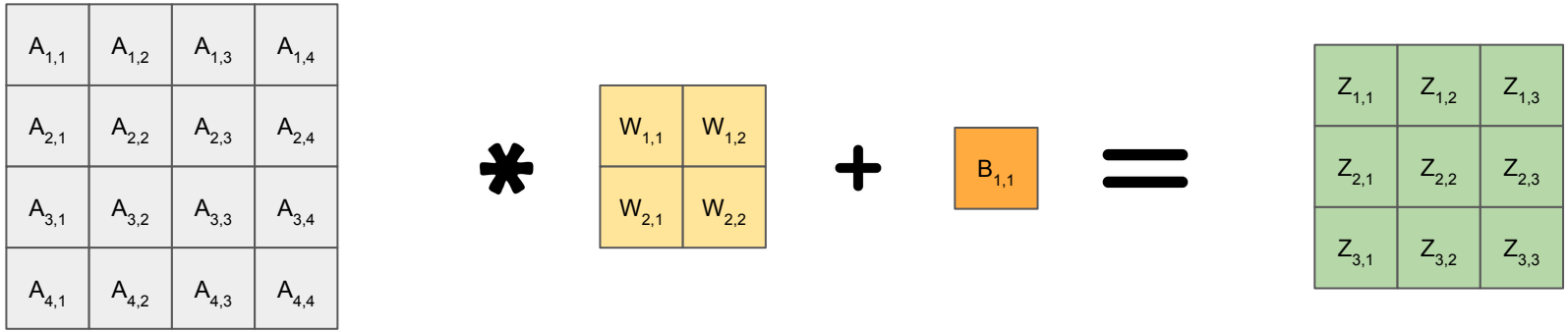


One reason for padding

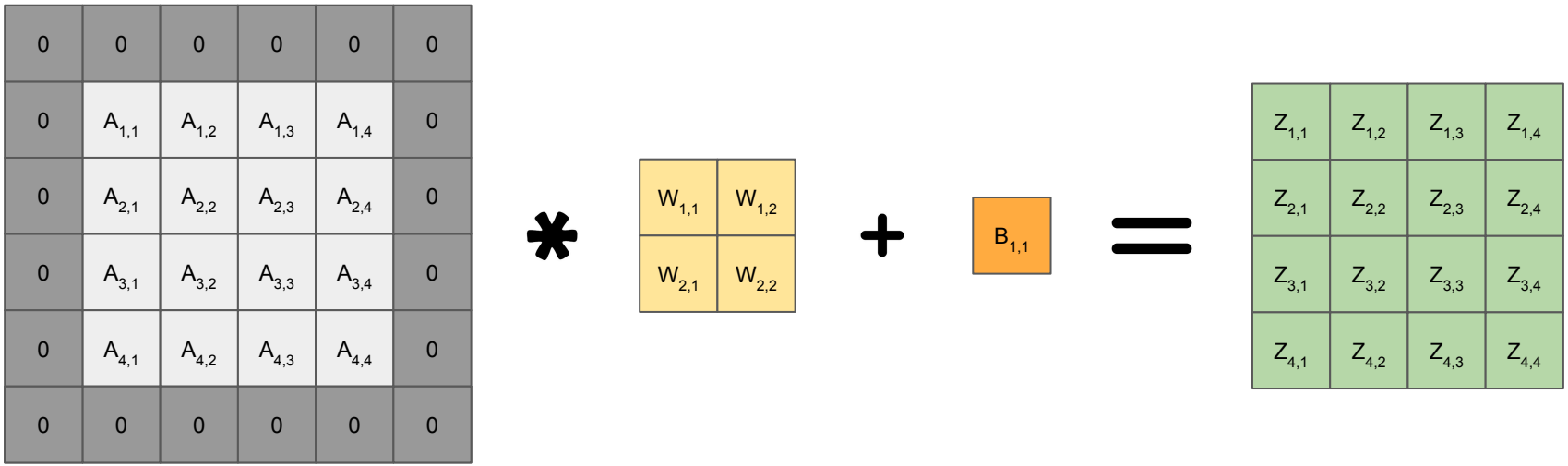


Never Meet...

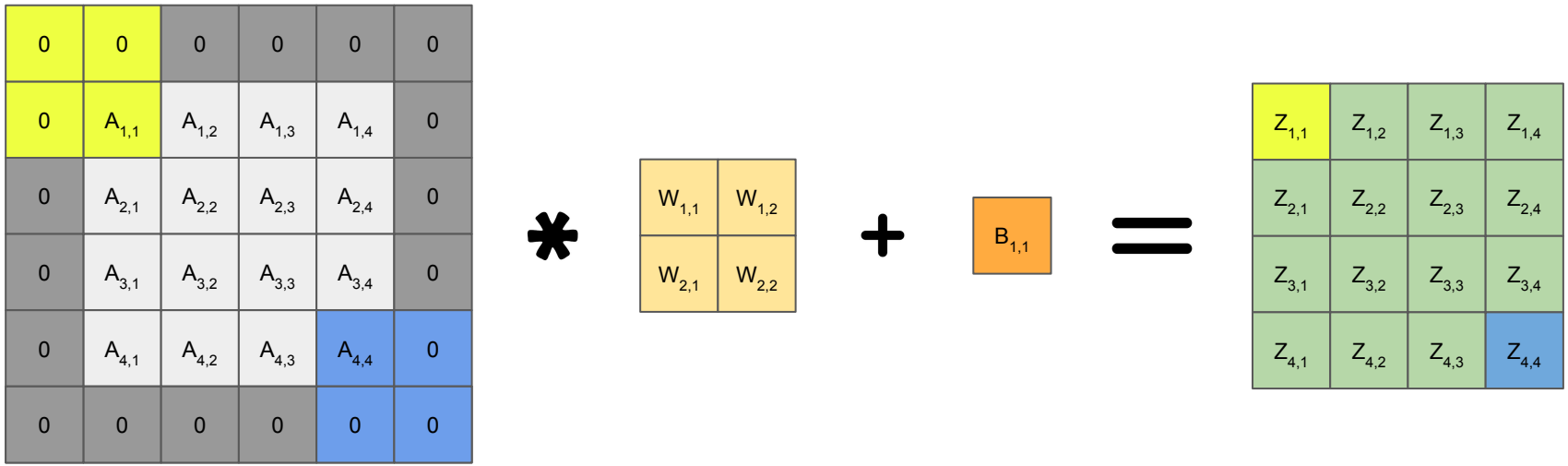
After padding



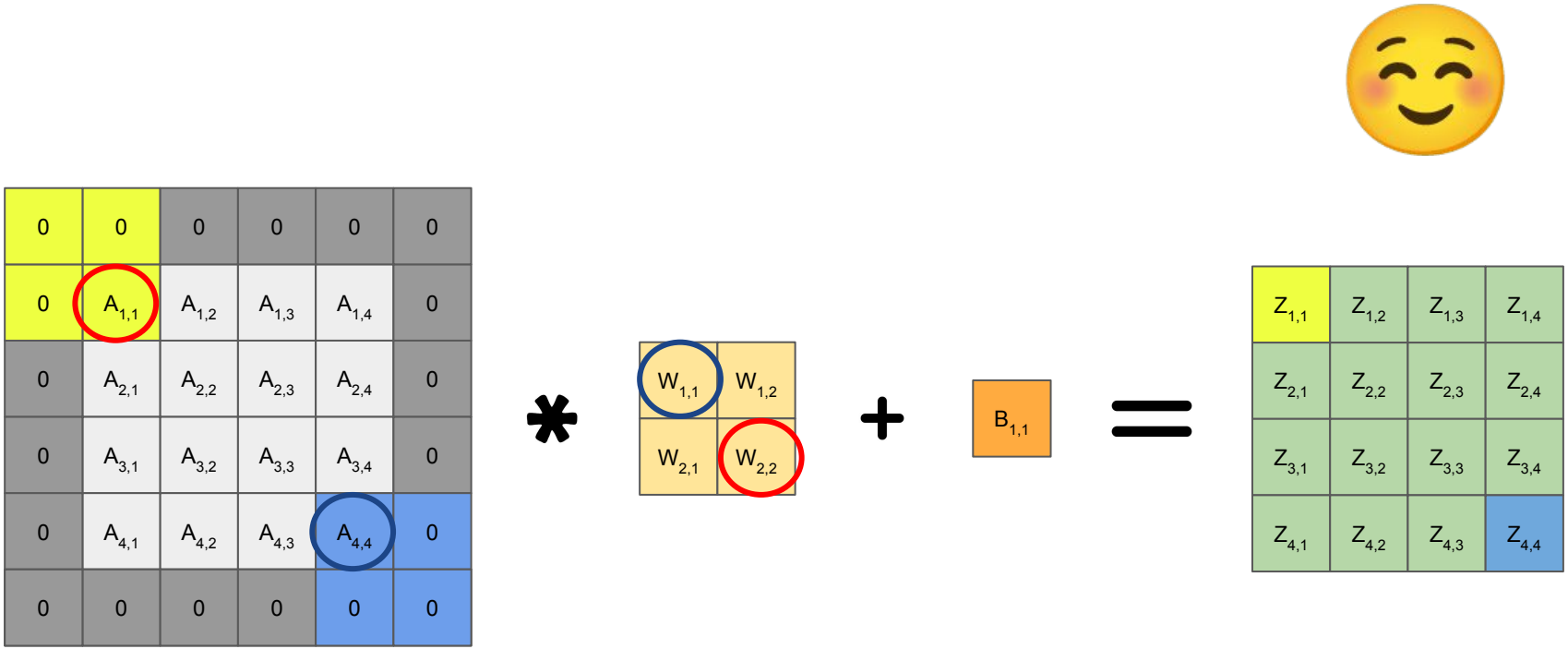
After padding



After padding



After padding



One reason for padding

Reason 1: increases the interaction between the input and the kernel

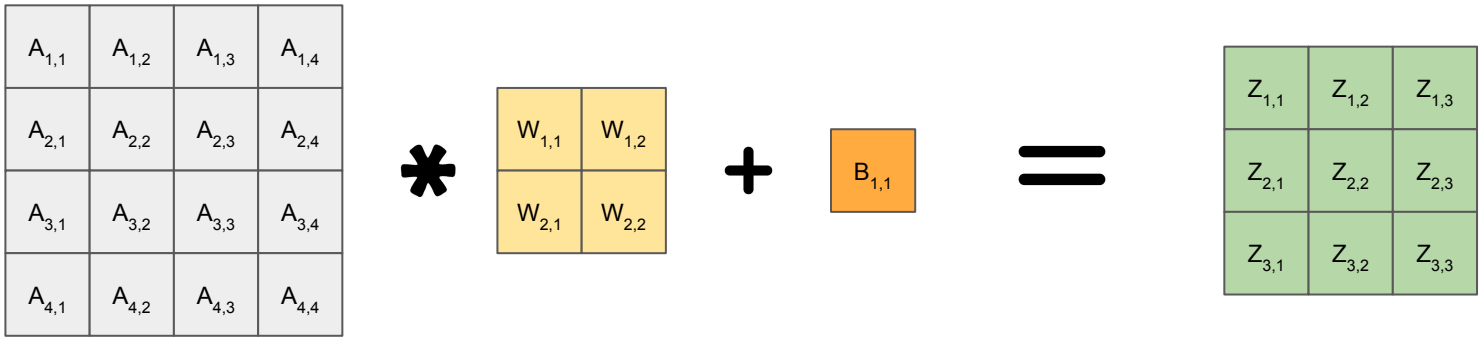
Reason 2: increase output size / preserves input size

Stride

How far the kernel moves in every iteration for each direction

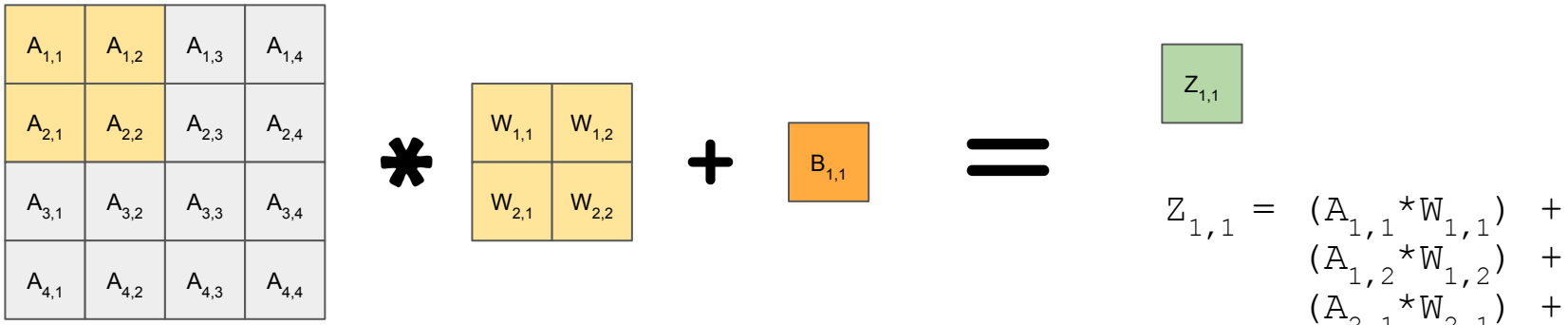
Stride = 1

In the previous slides: the kernel “moves” one pixel at a time.



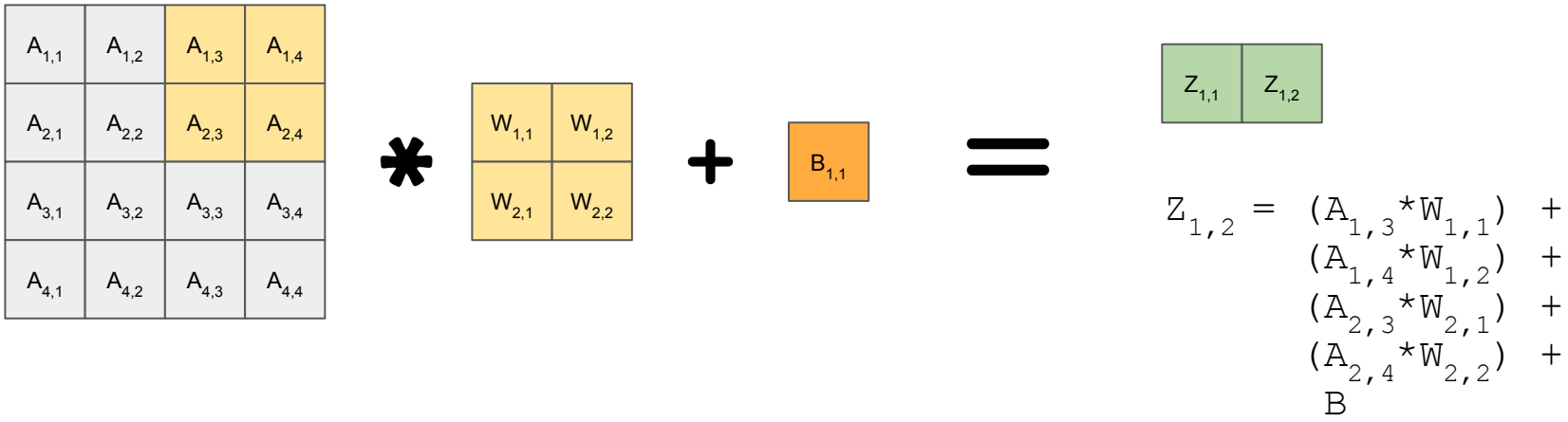
Stride = 2

Start at the same place



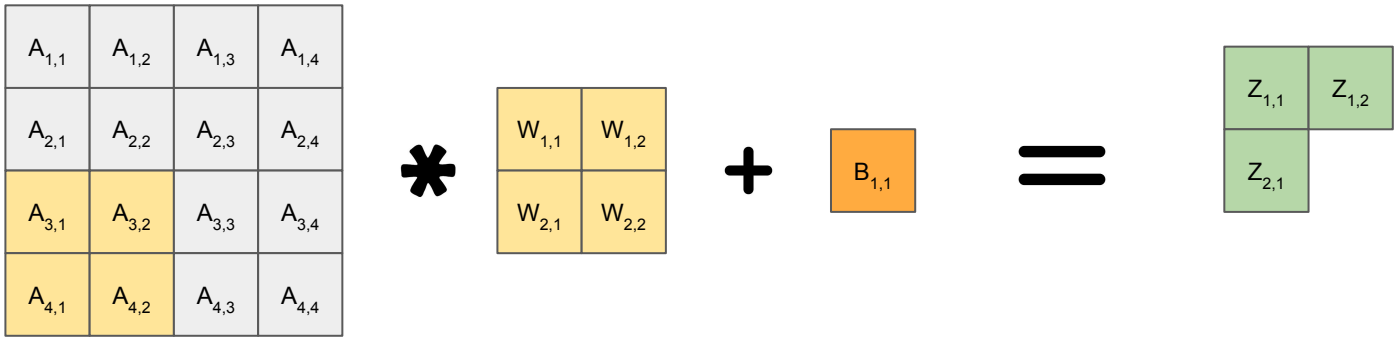
Stride = 2

Move two pixels to the right



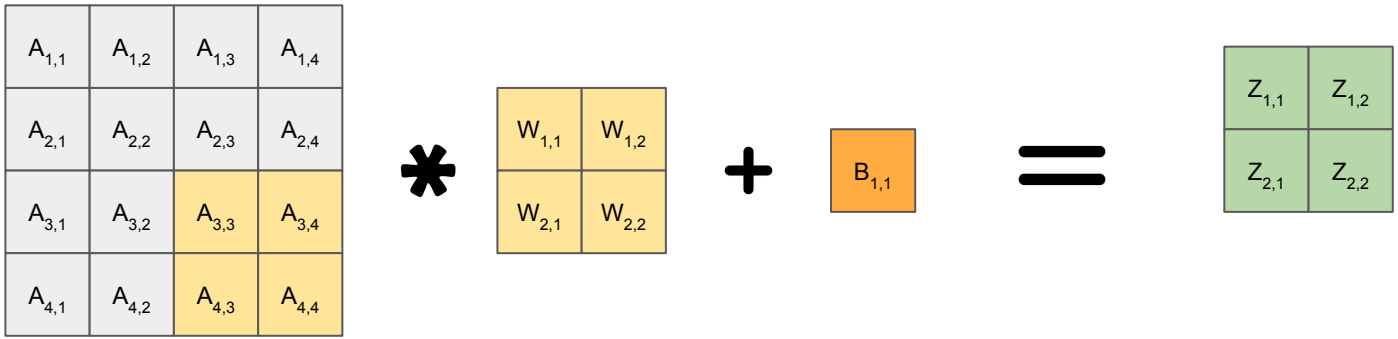
Stride = 2

Move two pixels down.

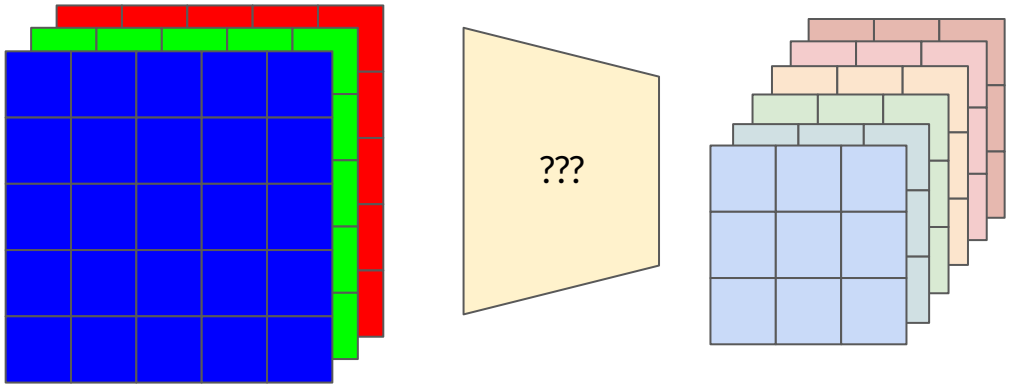


Stride = 2

Move two pixels to the right.



Multi-channel Convolution



Multi-channel Convolution

- Each filter has as many kernels as the number of channels in the input
- The c-th kernel of the filter convolves with c-th channel of the input.
- The number of output channels from the convolution = number of **filters**

C_{in} = Input channels

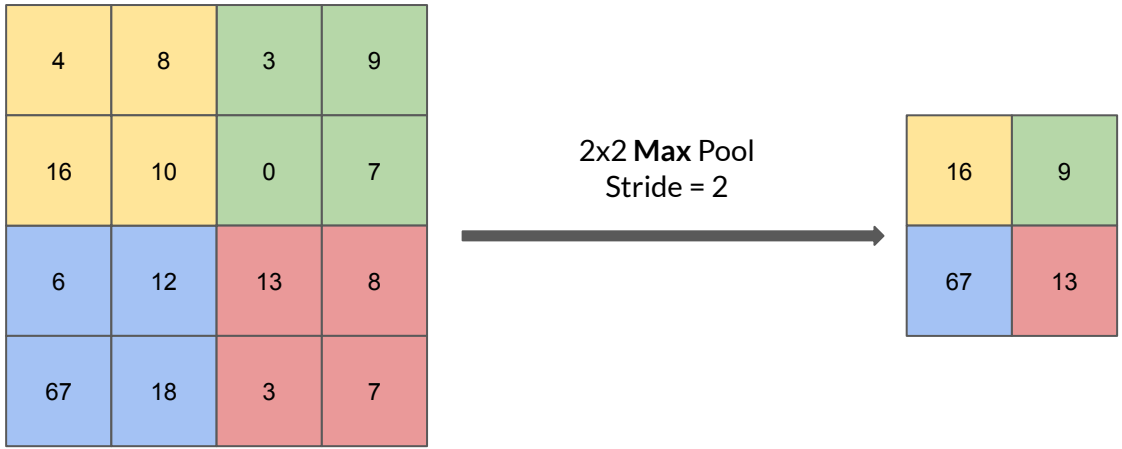
C_{filter} = Filter channels = C_{in}

F = Number of filters = C_{out} = Number of output channels

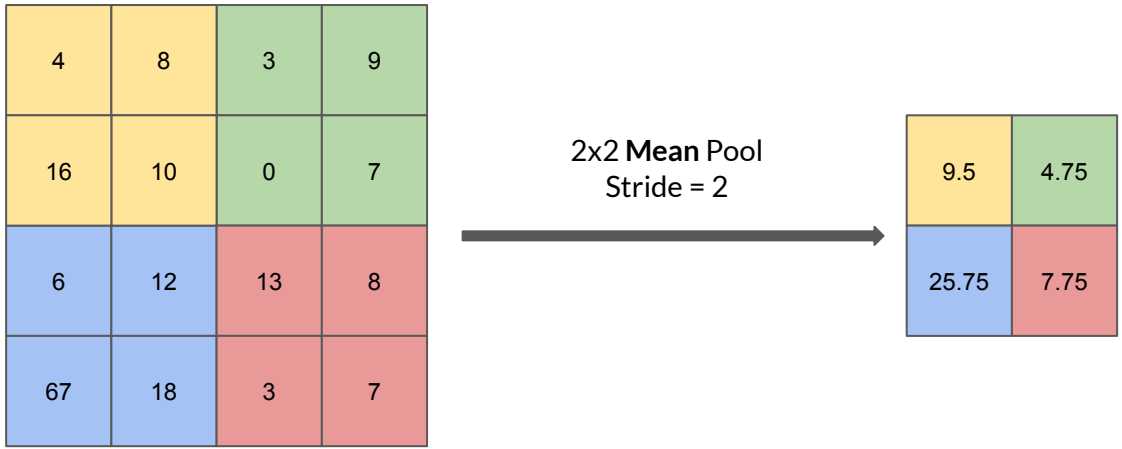
Pooling

- Usually applied after convolution
- Reduces height & width of the output
- Three common pooling operations:
    - Max Pooling
    - Mean Pooling
    - Min Pooling

Max Pooling



Mean Pooling



Convolutional Layer Implementation

- Earlier we said that a discrete convolution is a concatenation of inner products between the filter and receptive fields. This involves a lot of matrix multiplications!
- Parallelized matrix operations make this much faster than using a sliding filter.
  - `im2col`, `im2col_bw` are needed

THIS IS THE HARDEST PART OF THE PROGRAMMING. START EARLY!

Forward Pass for Conv Layer

Here are the following steps

1. Transform our input image into a matrix (im2col)
2. Reshape our kernel (flatten)
3. Perform matrix multiplication between reshaped input image and kernel

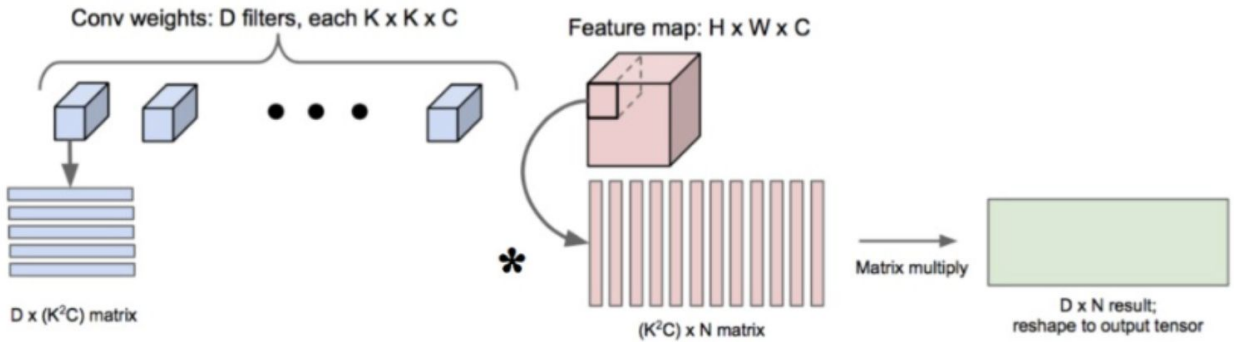


Figure 2: Illustration of im2col.

Forward Pass for Conv Layer

Here are the following steps

1. Transform our input image into a matrix (im2col).
2. Reshape our kernel (flatten).
3. Perform matrix multiplication between reshaped input image and kernel.

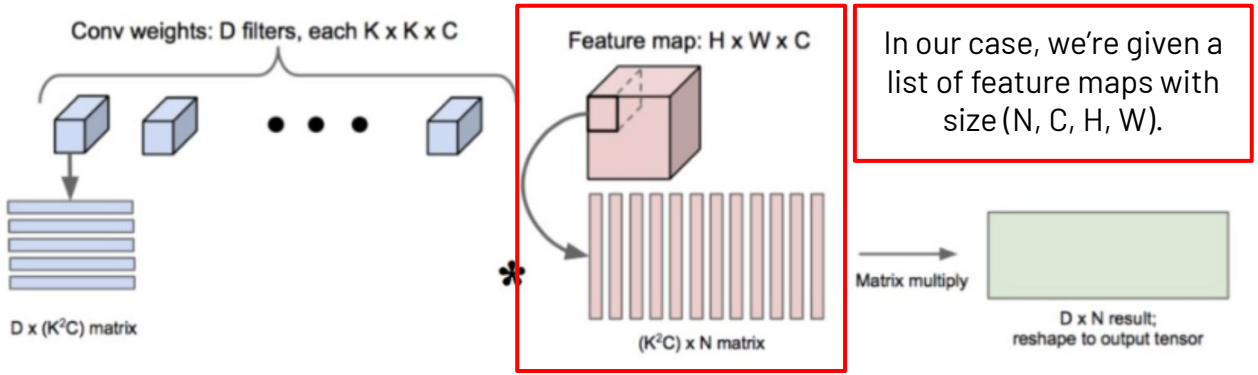


Figure 2: Illustration of im2col.

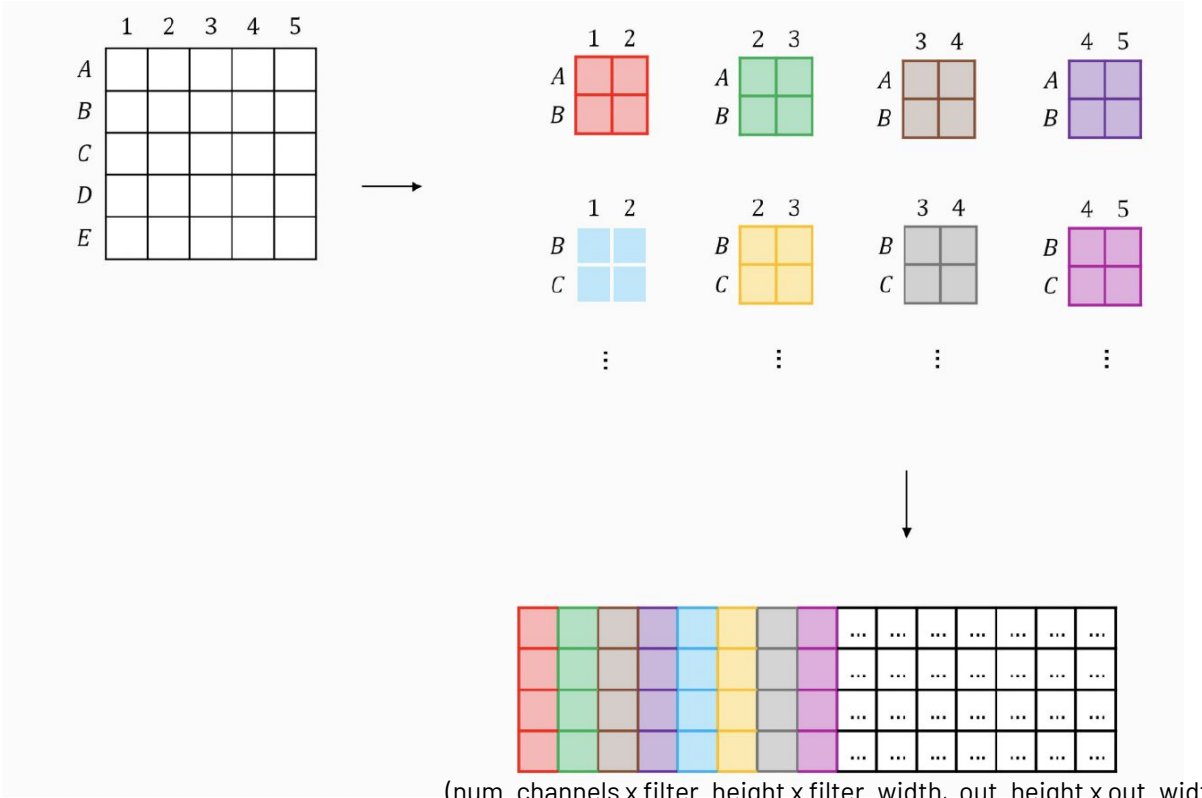
Im2col Simple Example

For this example, we have the following settings:

- Batch_size = 1
- Num_channels = 1
- Filter height and width = 2
- Feature map height and Width = 5
- Padding = 0
- Stride = 1

Key Intuition: Store all pixels of each window of feature map that will be applied to a filter in a single column.

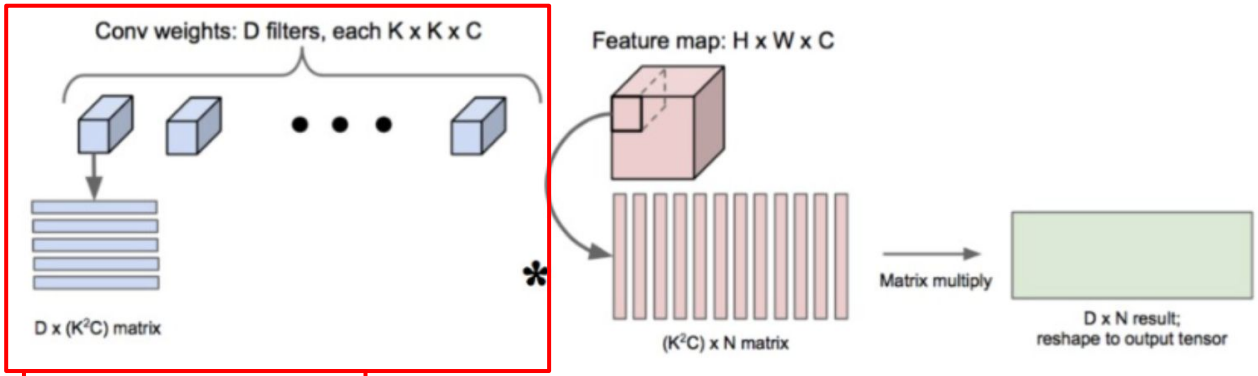
Important: Keep in mind out_height and out_width will be dependent on the padding and stride used!



Forward Pass for Conv Layer

Here are the following steps

1. Transform our input image into a matrix (im2col)
2. Reshape our kernel (flatten)
3. Perform matrix multiplication between reshaped input image and kernel



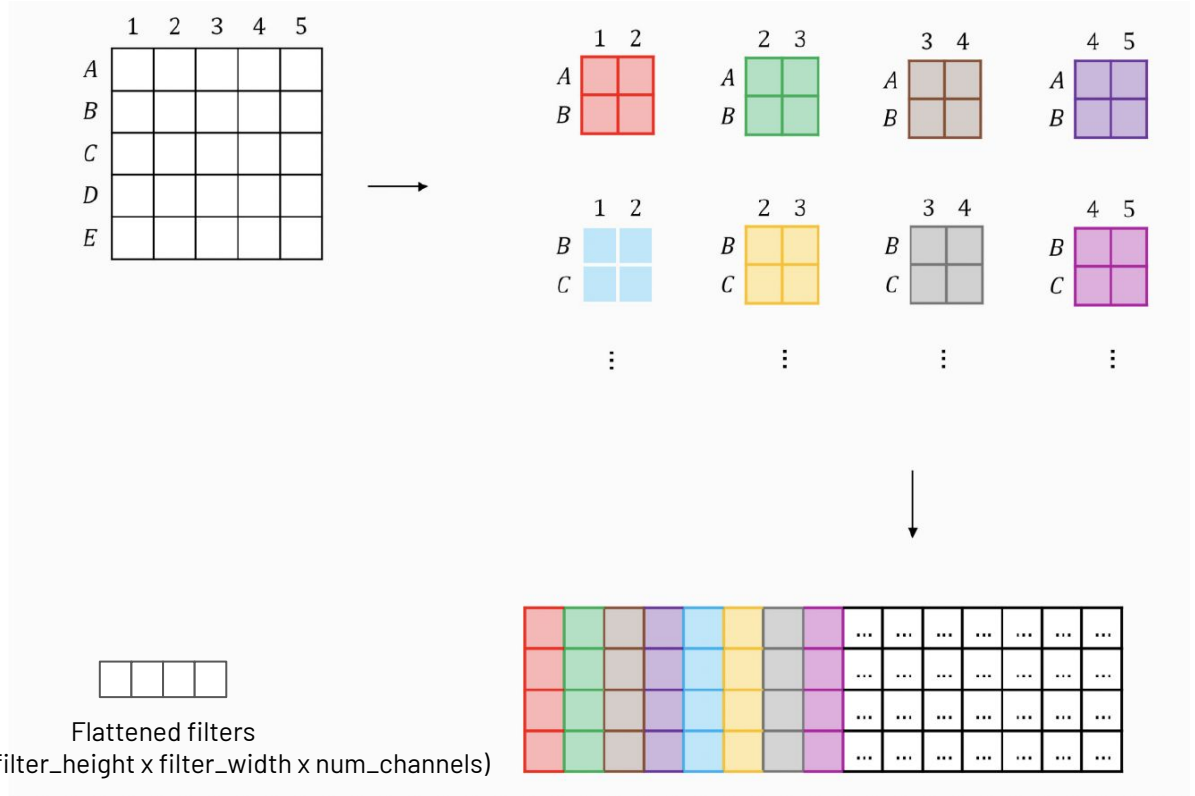
In our case, our filters have size (D, C, K_height, K_width).

Figure 2: Illustration of im2col.

Reshaping our kernels

For this example, we have the following settings:

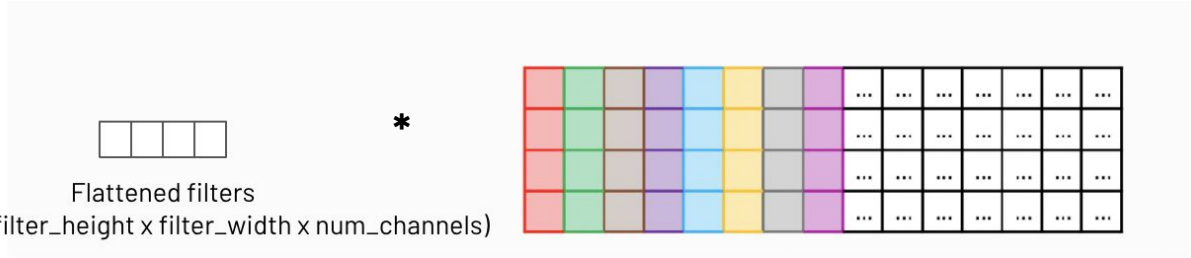
- Batch_size = 1
- Num_channels = 1
- Filter height and width = 2
- Image height and Width = 5
- Padding = 0
- Number of filters = 1



Forward Pass for Conv Layer

Here are the following steps

1. Transform our input image into a matrix (im2col)
2. Reshape our kernel (flatten)
3. Perform matrix multiplication between reshaped input image and kernel



Flattened filters
(num_filters , filter_height x filter_width x num_channels)

Convolution output:
(num_filters, image_height x image-width x batch_size)

Reshape to output tensor of shape (batch_size, num_filters, height, width).

Backward Pass for Conv Layer

Here are the following steps given dLoss

1. Compute gradients w.r.t weights of filters and biases.
    a. Hint: Involves taking matrix product of dLoss and im2col output.
2. Compute gradient w.r.t input image
    a. Run im2col_bw on grad_x_col (a gradient w.r.t im2col output), which is a 2D array of size (num_filters, image_height x image-width x batch_size).

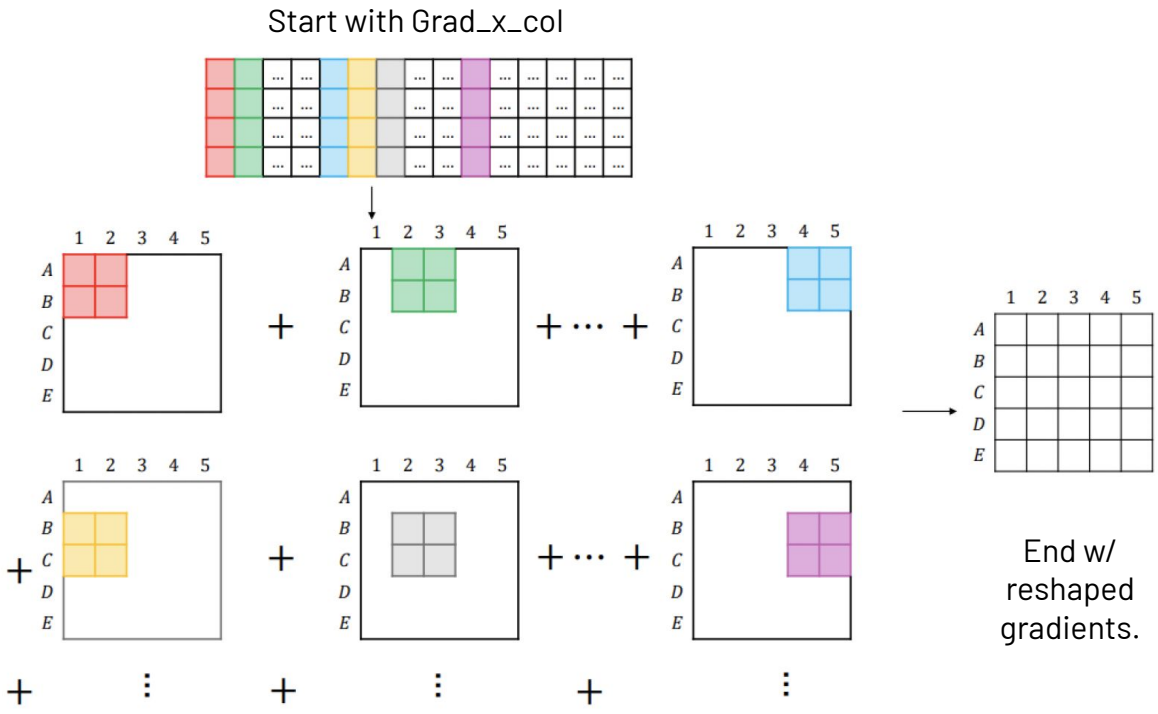
Im2col_bw Simple Example

Im2col_bw takes as an argument grad_X_col (a 2D array).

For this example, we have the following settings:

- Batch_size = 1
- Num_channels = 1
- Filter height and width = 2
- Image height and Width = 5
- Padding = 0

Return output with similar shape to input for im2col (batch_size, num_channels, height, width).



Some suggestions

Please **start early**, this assignment takes time!!!

- Im2col usually takes the most effort, make sure to test it correctly with small hand-derived example.
- Useful functions to keep in mind:
    - Np.transpose can be used to shift axes (e.g., A is an array with shape (1,2,3). np.transpose(A, (1,0,2)) will return an array with shape (2,1,3).
    - Np.reshape is useful for “reshaping” your array. Be careful when using np.reshape(dimension, -1) on 3D/4D data as the order of the axes matter!