Recap

- Backpropagation
  - A modular way to compute the gradients of the loss w.r.t each parameter
  - Represent the computation using a graph
    - A node for each operation, and an edge for each variable
    - Each operation implements two functions: forward and backward
  - Apply chain rules against the graph
    - Forward the data through every node in topological order
    - Backward the gradient through every node in the reverse order

Slide credit: Wang Wei

Recap

- Mini-batch stochastic gradient descent (SGD)
    - Reduces the chance of local optimal points and saddle points from GD
    - More stable than standard SGD
    - Extensions: Momentum, RMSProp, Adam
        - Exploiting historical updates
        - Adaptive learning rate per parameter
- Training tricks
    - Parameter initialization
    - Data normalization
    - Regularization:
        - Early stopping

Slide credit: Wang Wei

Agenda

- Convolution basics
- 1D and 2D convolution operations
- Pooling operations

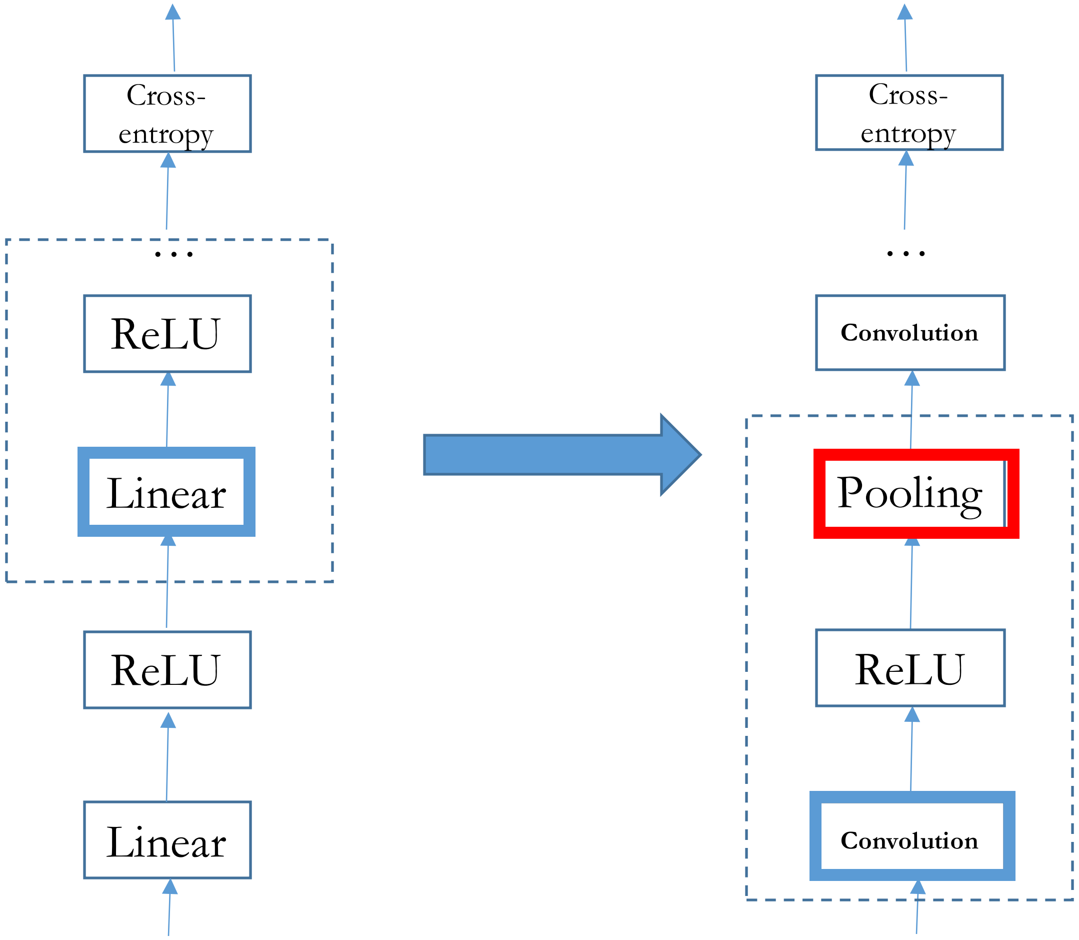
Slide credit: Wang Wei

Convolutional Neural Networks

- Referred to in short form as ConvNets or CNNs
- Most frequently used for image(-related data):
  - Image classification
  - Object detection
  - (Medical) image segmentation
  - Face recognition
  - Image generation
  - Art composition

Slide credit: Wang Wei

From MLP to CNN



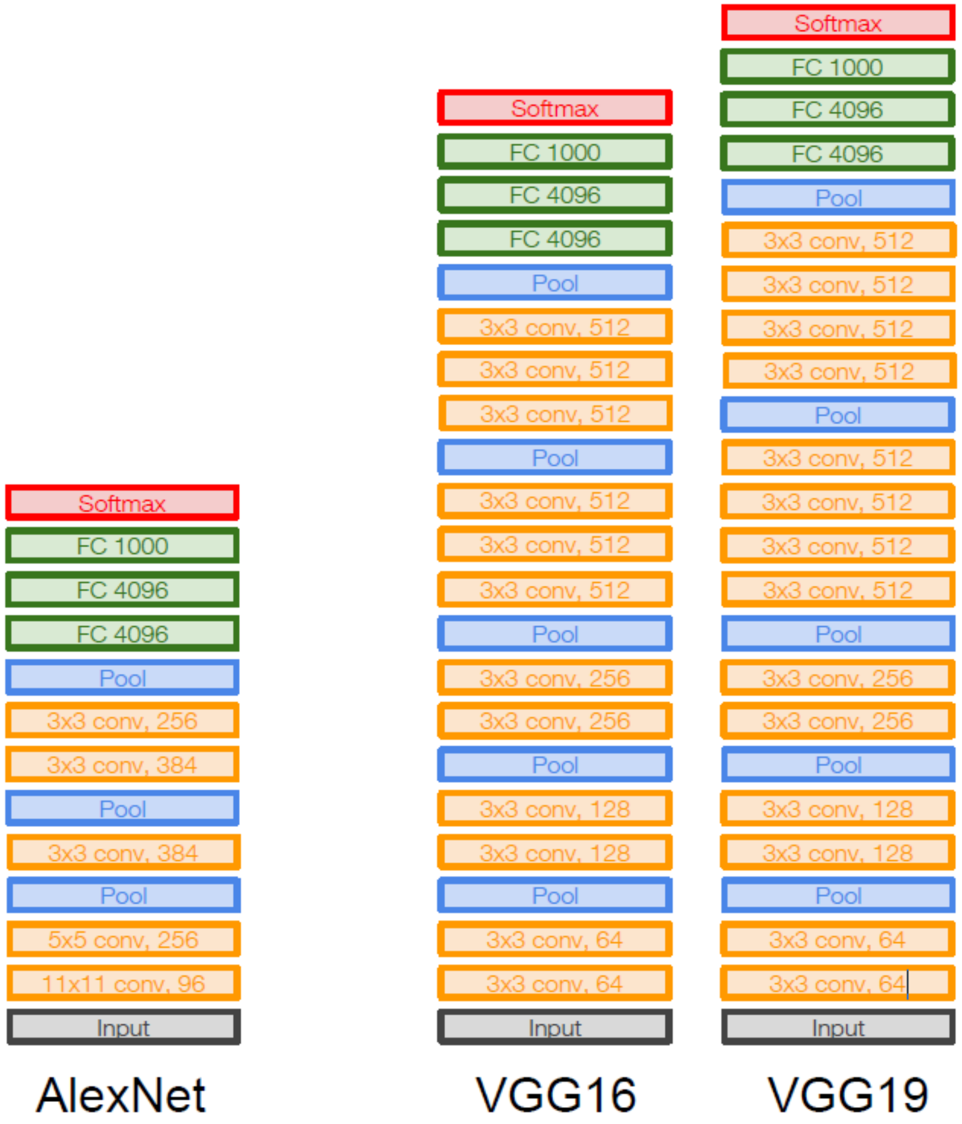


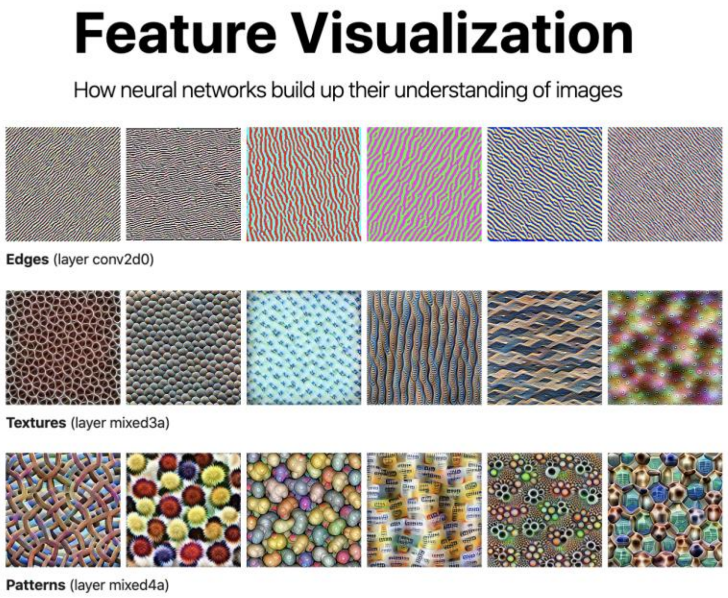
Image source: Stanford cs231n

Slide credit: Wang Wei

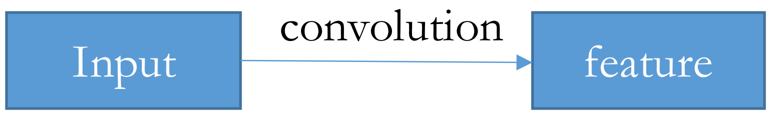
Convolution

How can we get the most valuable information from the input (e.g. image)?

- A linear transformation
- Since output is a "feature" of the input, convolution can be considered feature extraction
- 1D / 2D / 3D
  - 1D: Text processing
  - 2D: Image processing
  - 3D: 3D data, CT, microscopy, etc.
- mD convolution
  - "mD" comes from "m" dimensions of the source data
  - Can apply 1D convolution over 1D or 2D data; More easily, we can convert a matrix to a vector
  - Can apply 2D convolution over 2D or 3D data; More easily, we can convert a tensor to a matrix



Feature visualization



Slide credit: Wang Wei

1D convolution

$$\boldsymbol{y}_t = \sum_{i=0}^{k-1} \boldsymbol{w}_i \times \boldsymbol{x}_{t+i}$$

Cross-correlation operation [link]

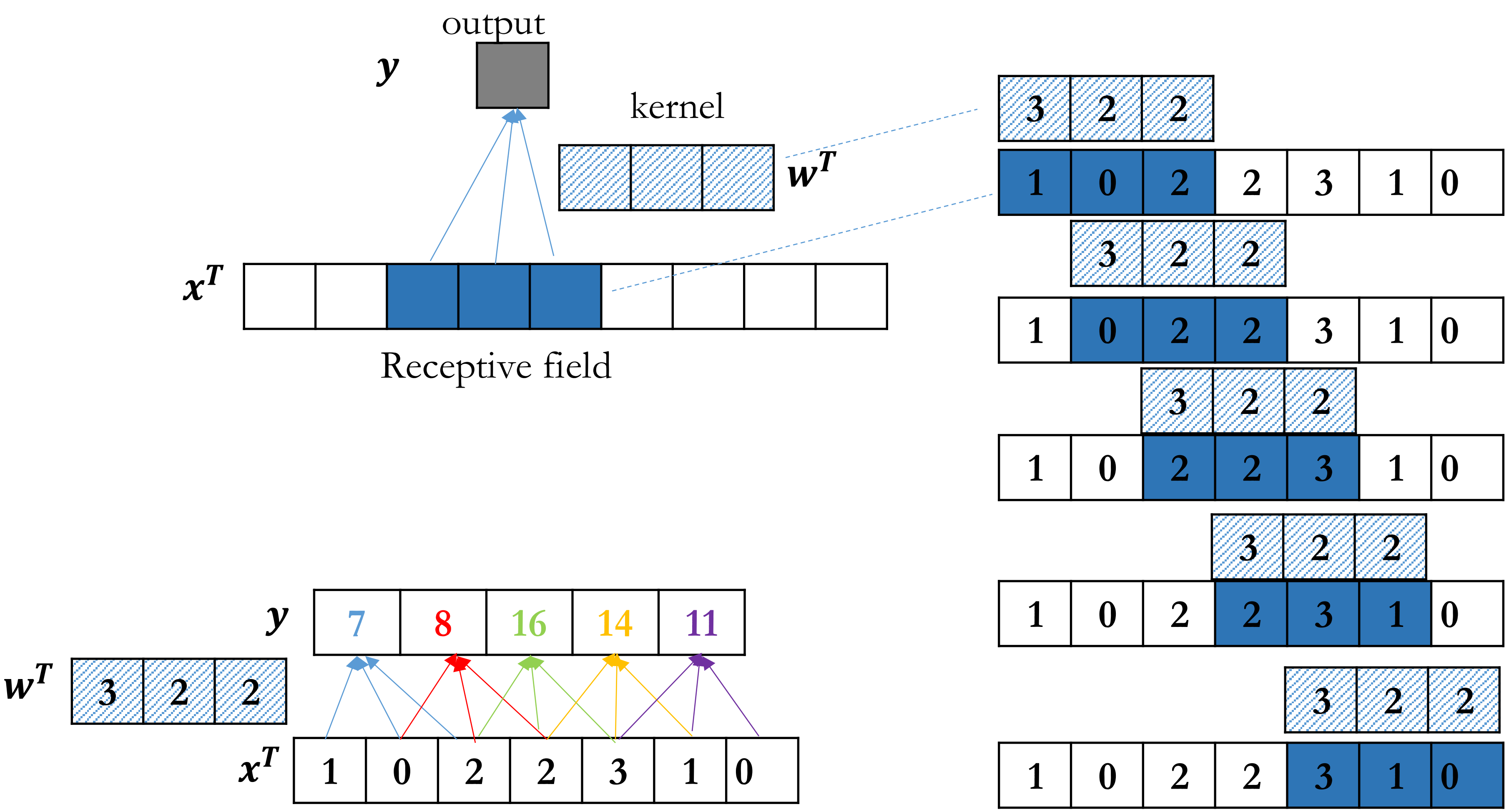
- **In CNNs, convolution refers to cross-correlation**
- $\boldsymbol{W}$ is called the kernel/filter; length k
  - "weights" or parameters to be trained;
- $\boldsymbol{x}$ is the input; length $\boldsymbol{n}$
- the applied or input area, i.e. t, t+1, …, t+k-1 is called the receptive field
  - one receptive field generates one output value
- $\boldsymbol{y}_t$ is the output feature; length $\boldsymbol{o}$

yt should be y, including all the output values

In signal processing, cross-correlation is a measure of similarity of two series as a function of the displacement of one relative to the other.
It is also known as a sliding dot product.

Slide credit: Wang Wei

1D convolution



3x1+2x0+2x2=7

3x0+2x2+2x2=8

3x2+2x2+2x3=16

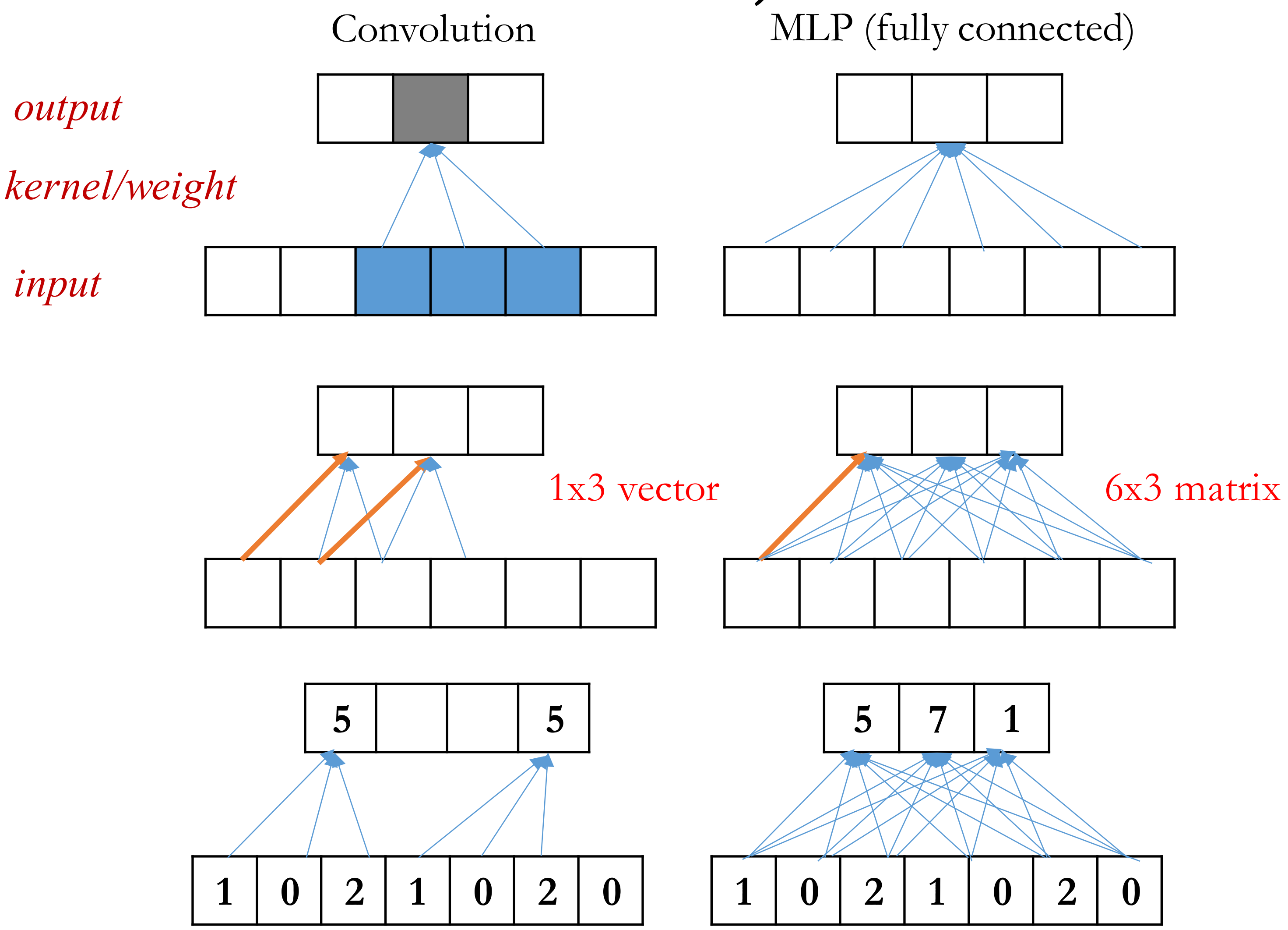
3x2+2x2+2x1=14

3x3+2x1+2x0=11

Slide credit: Wang Wei

Properties (Why is convolution better?)

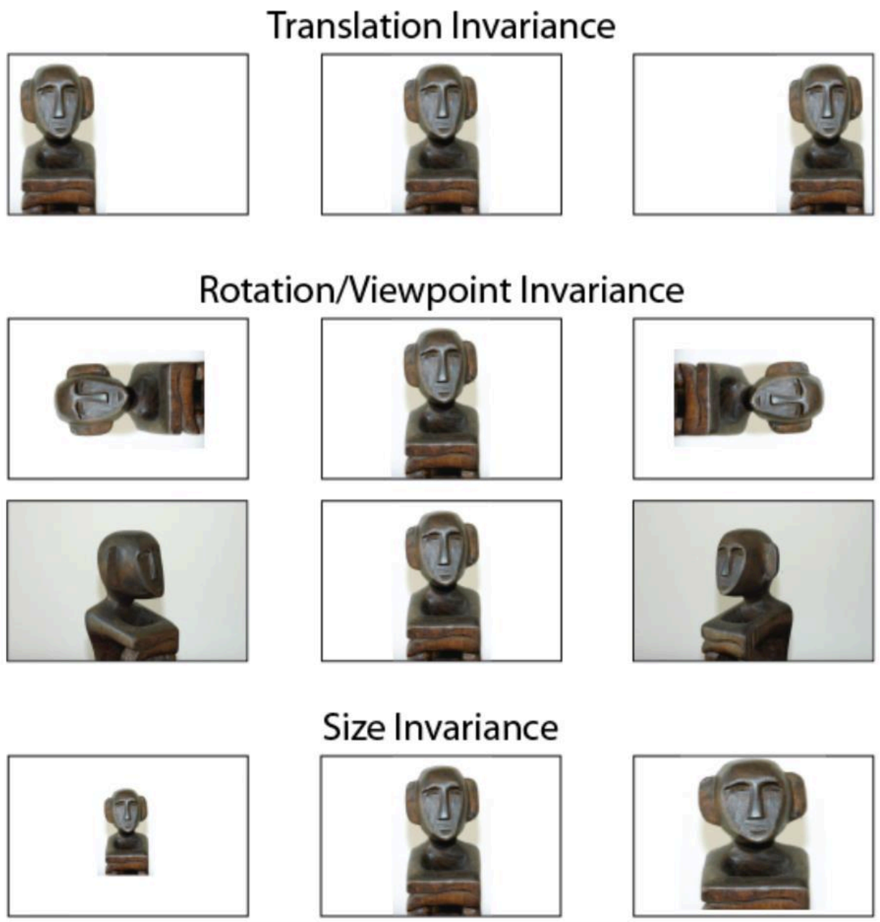
- Sparse connection:
  - each output is connected only to inputs within receptive field vs. all inputs
  - → fewer parameters (each output needs 3 vs. 6 in example)
  - → less overfitting
- Weight sharing vs. unique weights
  - Regularization
  - Less overfitting
- Location or Spatial invariant
  - Function transformations should not depend on the location within the image, i.e.
  - Make the same prediction no matter where the object is in the image



Slide credit: Wang Wei

Location or Spatial Invariant

- You can recognize an object even its appearance varies in some way
- Convolution operator commutes with respect to translation
  - If you convolve f with g, it doesn't matter if you translate the convolved output f*g, or you translate f or g first, then convolve them.
  - https://en.wikipedia.org/wiki/Convolution
- Location or Spatial invariant
  - Function transformations should not depend on the location within the image, i.e.
  - Make the same prediction no matter where the object is in the image



Slide credit: Matt Krause

Perceptron, MLP and Convolution

Perceptron

Perceptron is too simple
→ underfitting
→ add more layers
→ MLP

MLP

MLP has too many parameters
→ High dimension
→ difficult to optimize and overfitting
→ CNN (with more regularization)

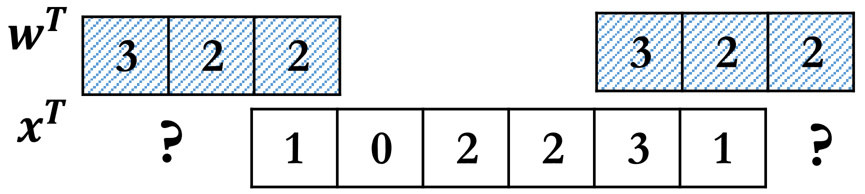
CNN

Slide credit: Wang Wei

Padding

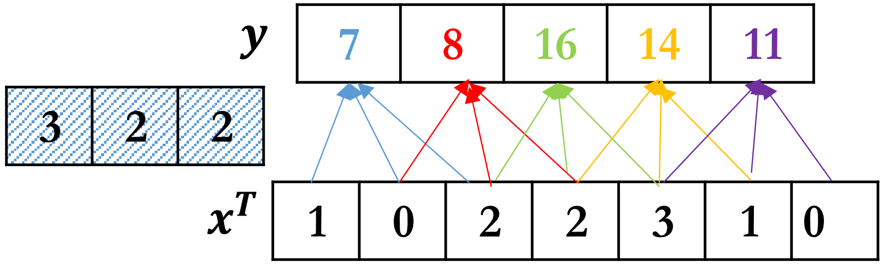
- How to determine the edge values?

  Ignore non-valid regions?



We want the first element of input to be processed by all the parameters in the kernel.

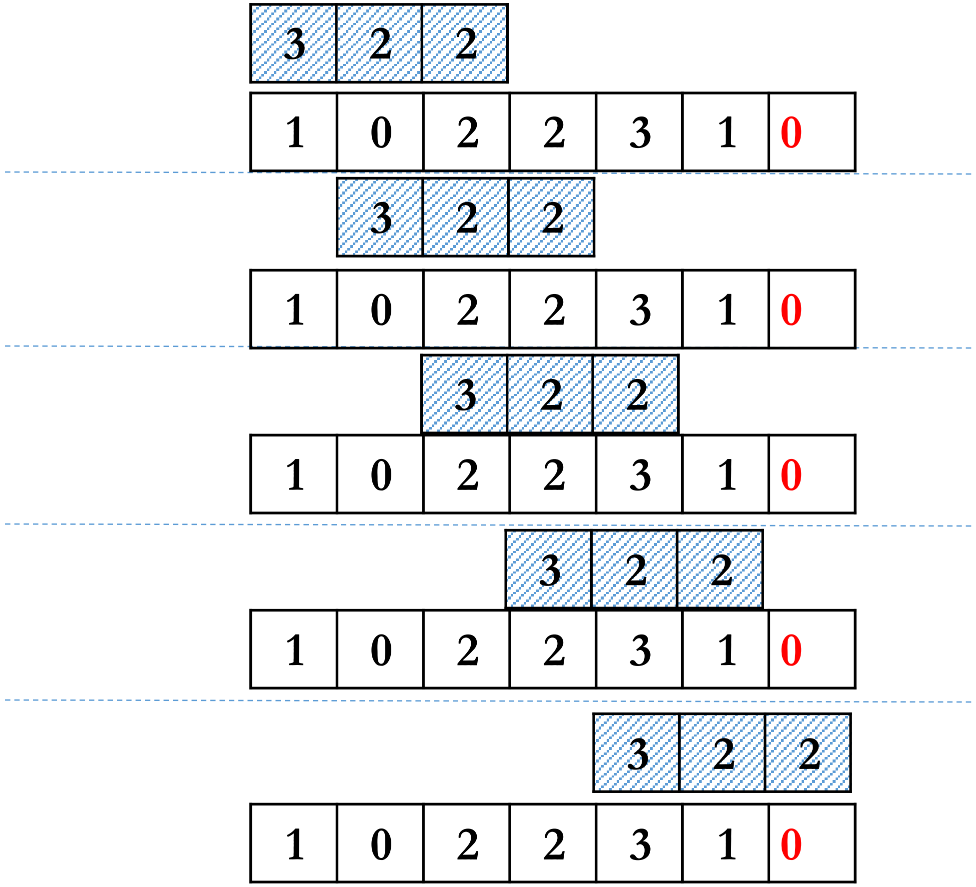
- $o = n - k + 1$
  - n: input length, k: kernel length, o: output length
  - Output is shorter than input
- To retain the resolution/size
  - Pad with extra values (usually 0s)



Slide credit: Wang Wei

Padding

- Manual padding (p)
  - Output feature values for $p = 1$
    - 3x1+2x0+2x2=7
    - 3x0+2x2+2x2=8
    - 3x2+2x2+2x3=16
    - 3x2+2x2+2x1=14
    - 3x3+2x1+2x0=11
- What value to pad with?
  - usu. k << n so value doesn't matter too much; so don't bother tuning
  - 0 picked for convenience
- Operation supported in many deep learning libraries, e.g.
  - Torch, PyTorch, Caffe, SINGA



Slide credit: Wang Wei

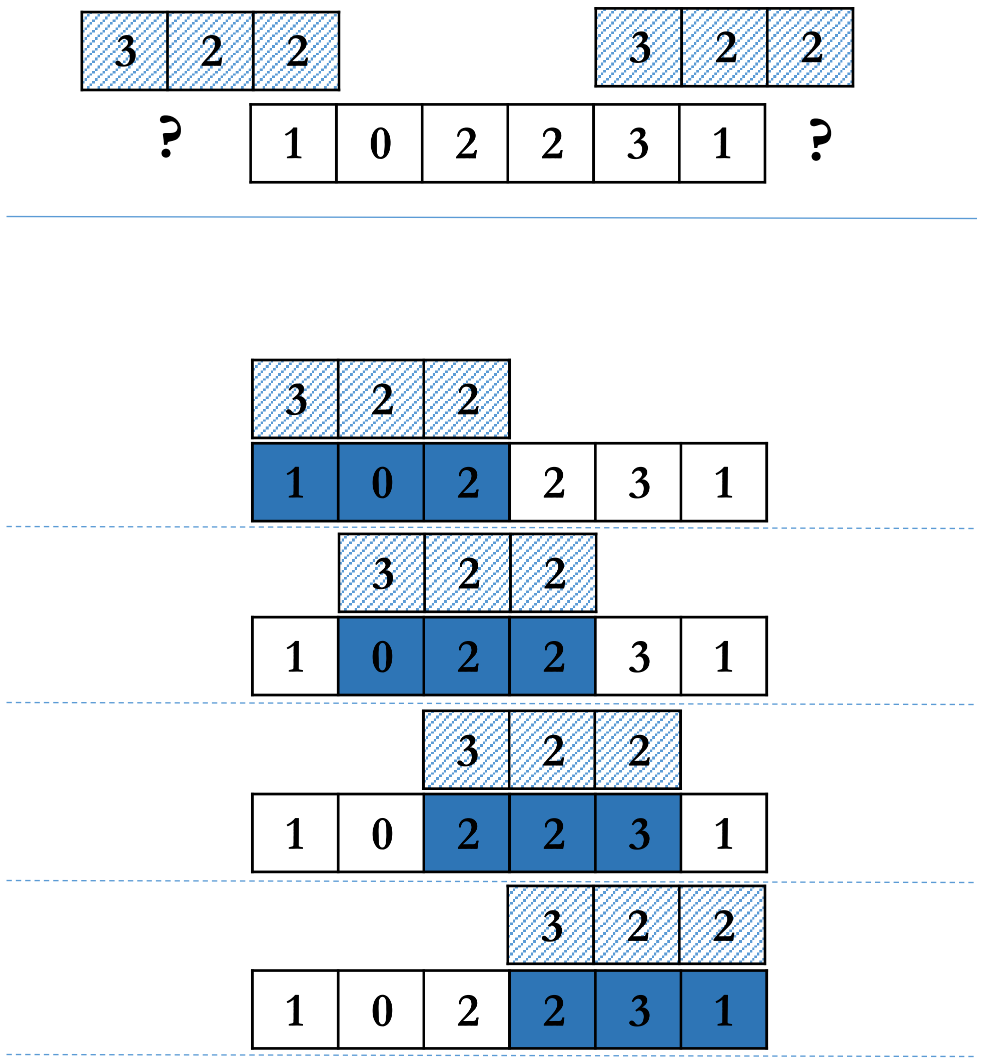
Padding amount (p)

- Given padding (p), what will the output length be?
- Kernel size/length: k, input length: n
- Without padding:
  - # outputs $o = n - k + 1$
- With padding:
  - # outputs $o = (n + p) - k + 1$
- padding length (p) can be set manually or automatically
- 2 special automatic settings:
  - consider only "valid" convolutions → $p = 0$
  - same length output as input → "same"
    - $o = n \rightarrow \quad p = f(k)$ $\quad p = k\text{-}1$

Slide credit: Wang Wei

“Valid” Convolution

- No padding ($p = 0$)
  - # inputs denoted as n
  - # outputs $o = n - k + 1$ = 6-3+1=4
  - Output feature values
    - 3x1+2x0+2x2=7
    - 3x0+2x2+2x2=8
    - 3x2+2x2+2x3=16
    - 3x2+2x2+2x1=14
  - Outputs become shorter
- Is an option to be set in library



Slide credit: Wang Wei

Same Padding

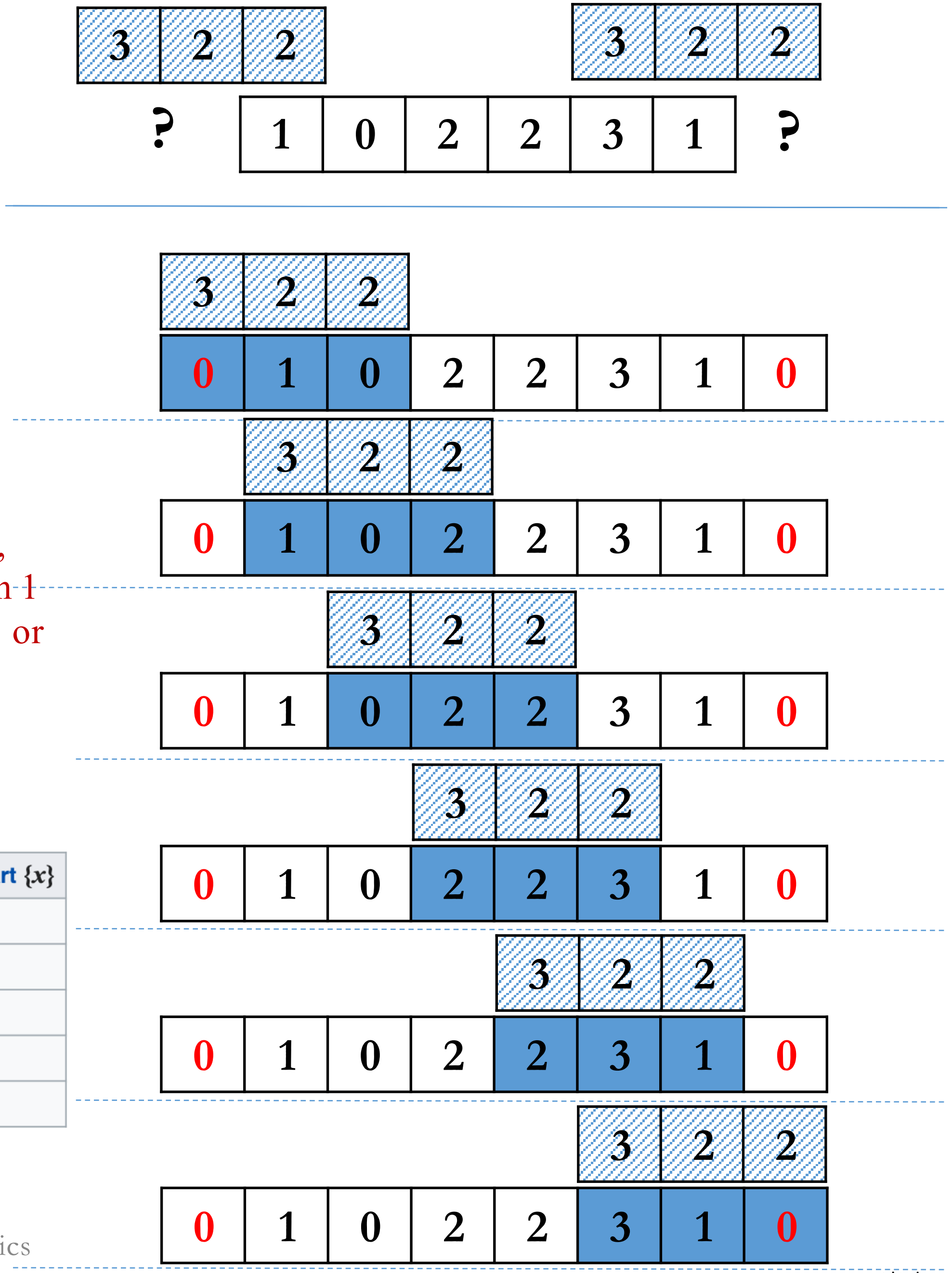
- Same padding (p?)
  - $o = n = n + p - k + 1$
  - $p = k - 1$
    - Left padding = $\lfloor p/2 \rfloor$
    - Right padding = $\lceil p/2 \rceil$
- Output values
  - 3x0+2x1+2x0=2
  - 3x1+2x0+2x2=7
  - 3x0+2x2+2x2=8
  - 3x2+2x2+2x3=16
  - 3x2+2x3+2x1=14
  - 3x3+2x1+2x0=11

If p is an odd number, libraries typically assign 1 less to the left than right or vice versa.

For n an integer, $\lfloor n \rfloor = \lceil n \rceil = [n] = n$.

Examples

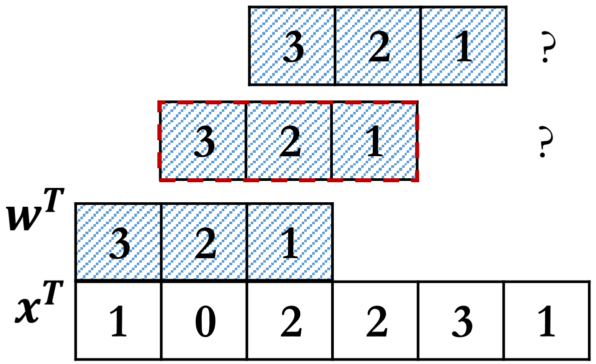
x	Floor $\lfloor x \rfloor$	Ceiling $\lceil x \rceil$	Fractional part $\{x\}$
2	2	2	0
2.4	2	3	0.4
2.9	2	3	0.9
−2.7	−3	−2	0.3
−2	−2	−2	0

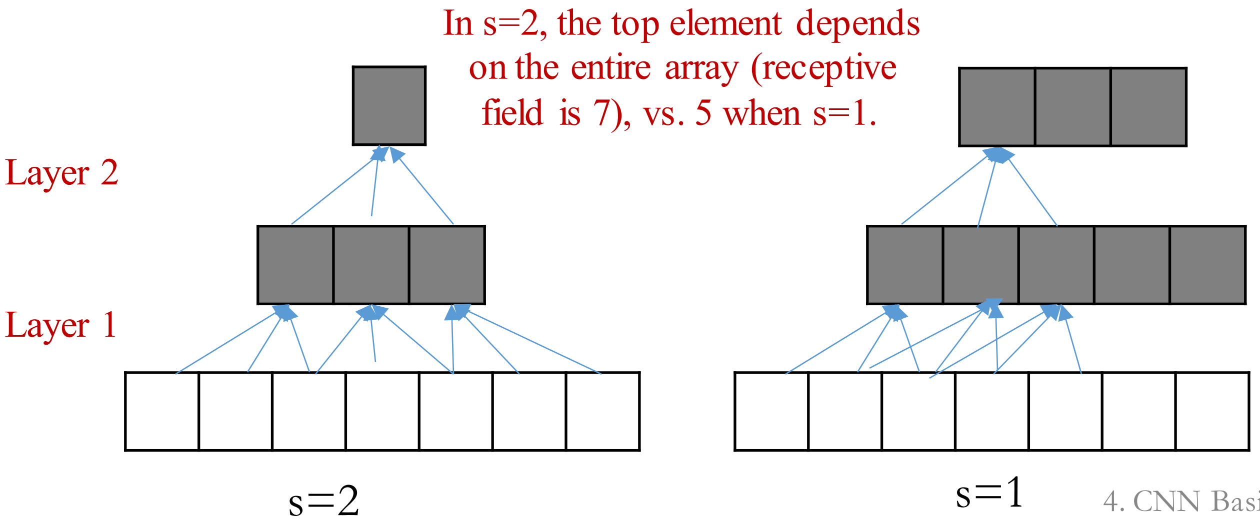


Slide credit: Wang Wei

Stride

- How many steps to move towards the next receptive field
  - so far, we have considered only a stride of 1, where the kernel is applied directly to the next element in the array
  - s>1, skip some elements
    - Faster to compute
    - ***Effective*** *receptive field size increases quickly*

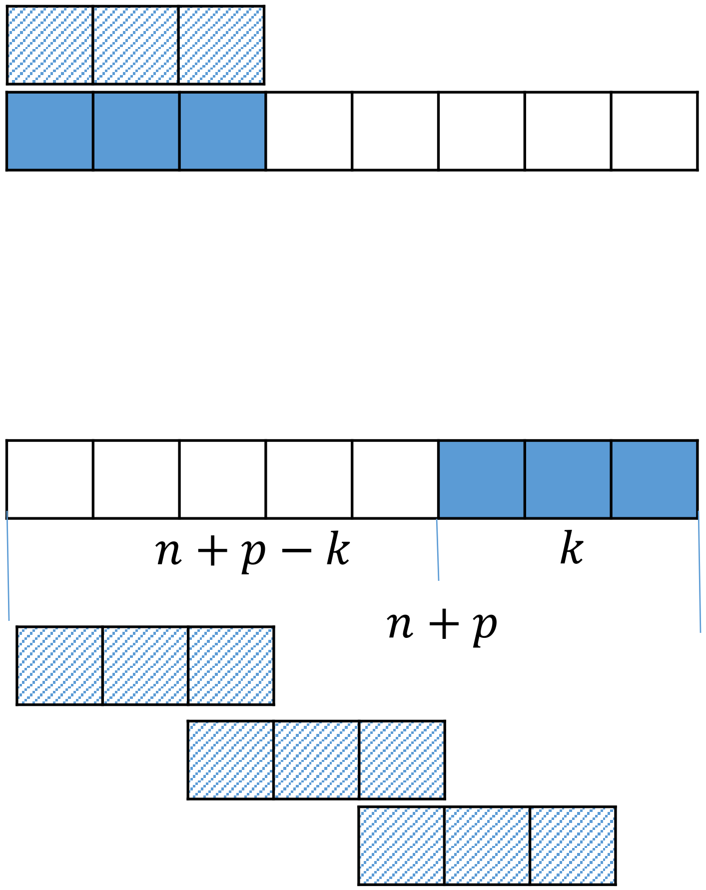




Slide credit: Wang Wei

Stride

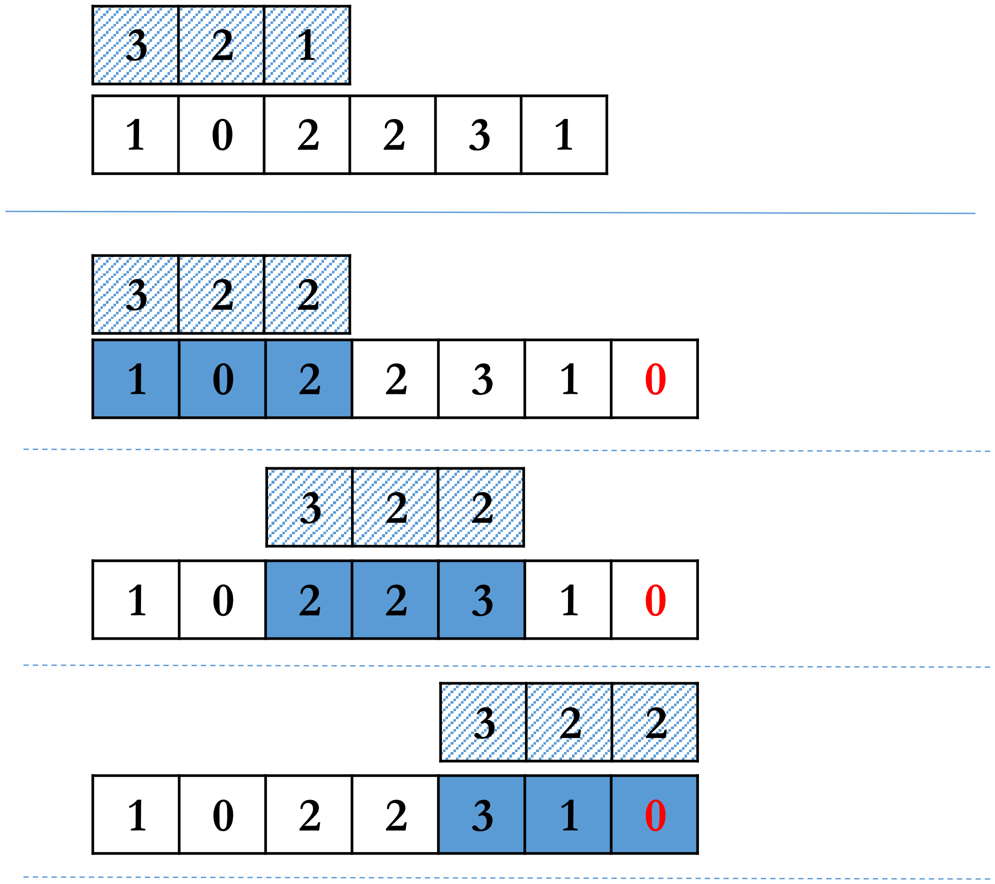
- Increasing the stride is computationally faster, since we compute less convolutions
- Resulting output with higher stride is subsequently shorter
- $o = \left\lfloor \frac{n+p-k}{s} \right\rfloor + 1$
- Tunable hyperparameter, which may also vary depending on the layer



Slide credit: Wang Wei

Stride

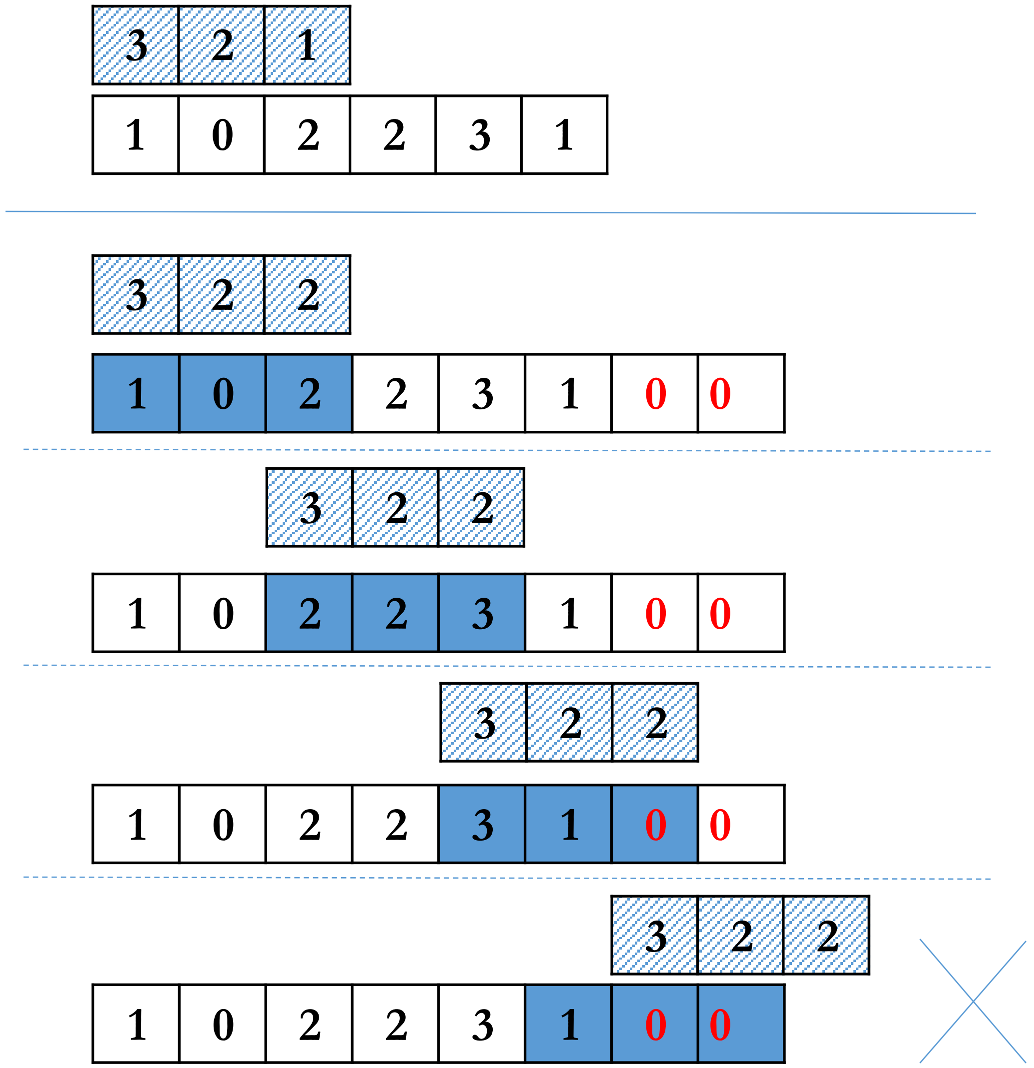
- Exact matching
  - With padding p (=1)
  - $o = \left\lfloor \frac{n+p-k}{s} \right\rfloor + 1$
  - (6+1-3)/2+1=3



Slide credit: Wang Wei

Stride

- Not exact matching
  - With padding p (=2)
  - $o = \left\lfloor \frac{n+p-k}{s} \right\rfloor + 1$
  - (6+2-3)/2+1=3



This last computation is not valid.

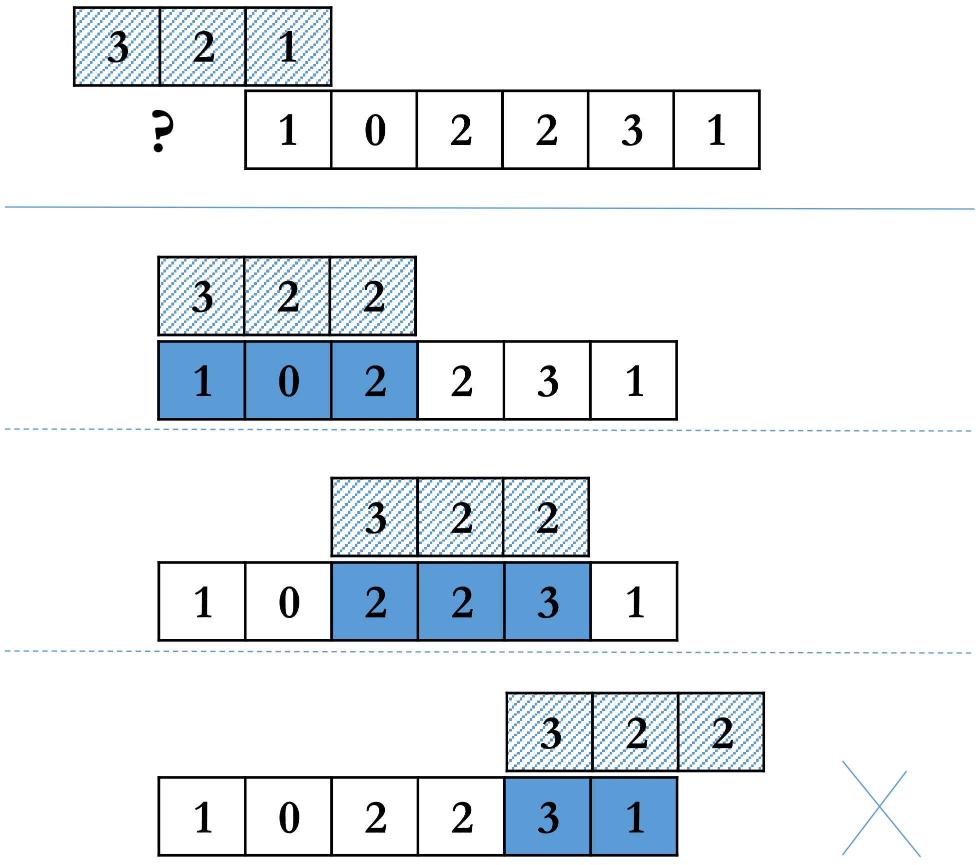
This equation works in both situations.

Slide credit: Wang Wei

Stride (for Tensorflow)

- When stride > 1
  - Valid padding, $p = 0$
  - Same padding, ?
- output length cannot be equal to the input, since a stride greater than 1 will shorten the output
- "same" is defined as:
  - $o = \left\lceil \frac{n}{s} \right\rceil, p = ?$

ceiling operation.



Slide credit: Wang Wei

Stride (for Tensorflow)

- When stride > 1
  - Valid padding, $p = 0$
  - Same padding, $o = \left\lceil \frac{n}{s} \right\rceil$, $p = ?$

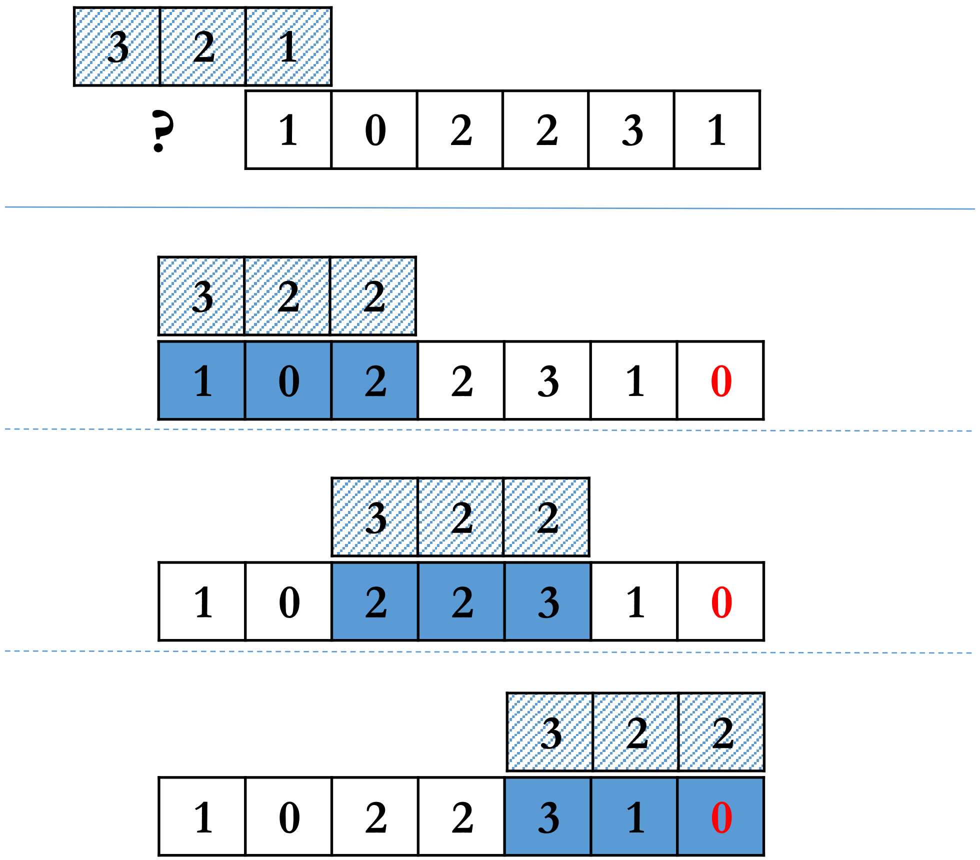
$$o = \left\lfloor \frac{n+p-k}{s} \right\rfloor + 1$$ <= (n+p-k)/s + 1

$$\frac{n+p-k}{s} \geq o - 1$$

$$p \geq s(o-1) + k - n$$

$$p = \max(s(o-1) + k - n, 0)$$

Tensorflow internally computes this p value when we set "same" padding for strides greater than 1.



Slide credit: Wang Wei

Summary by an Example (2-minute Quiz)

- Input Length = 13; Stride = 5; Kernel Length = 6
- "valid" method: no padding, drop the non-valid region

How many elements in input vector will be dropped?

- "same" method: padding at both ends

$p = \max(s(o-1) + k - n, 0)$

What is the padding length? Where?

Slide credit: Wang Wei

Summary by an Example

- Input Length = 13; Stride = 5; Kernel Length = 6

- "valid" method: no padding, drop the non-valid region

```
inputs:         1  2  3  4  5  6  7  8  9  10 11 (12 13)
               |________________|                dropped
                               |_________________|
```

- "same" method: padding at both ends

```
            pad|                                      |pad
inputs:      0 |1  2  3  4  5  6  7  8  9  10 11 12 13|0  0
            |________________|
                           |_________________|
                                          |________________|
```

$$p = \max(s(o-1) + k - n, 0)$$

$$p = \max(s(\text{ceiling}(n/s)-1) + k - n, 0)$$

$$p = \max(5(\text{ceiling}(13/5)-1) + 6 - 13, 0)$$

$$p = \max(5(\text{ceiling}(2.6)-1) + 6 - 13, 0)$$

$$p = \max(5(3-1) + 6 - 13, 0)$$

$$p = \max(10 + 6 - 13, 0) = 3$$

Slide credit: Wang Wei

Computing

- Conv1D
  - Forward(x, w)

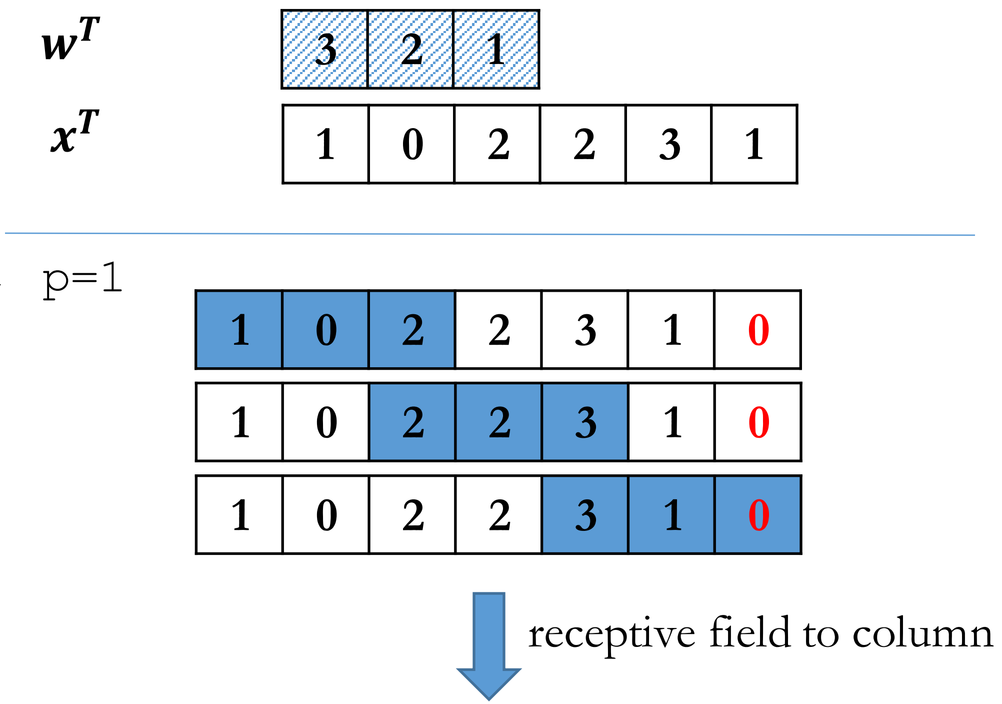
w^T	3	2	1			
x^T	1	0	2	2	3	1

S=2, k=3, p=1

1	0	2	2	3	1	0
1	0	2	2	3	1	0
1	0	2	2	3	1	0

Slide credit: Wang Wei

Forward



S=2, k=3, p=1

Convolution via dot products in a for loop.

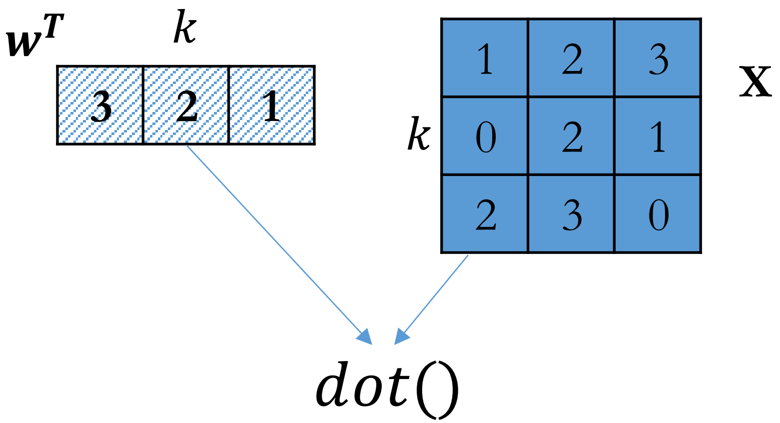
```
x_pad ← x
for t in range(o):
   y[t] = dot(w, x_pad[t*s:t*s+k])
```

Vectorization (img2col)

More efficient vectorized implementation; convert each receptive field as a single column.

```
for t in range(o):
   X[:, t] = x_pad[t*s:t*s+k]
y = dot(w, X)
```

Try out in Google colab!



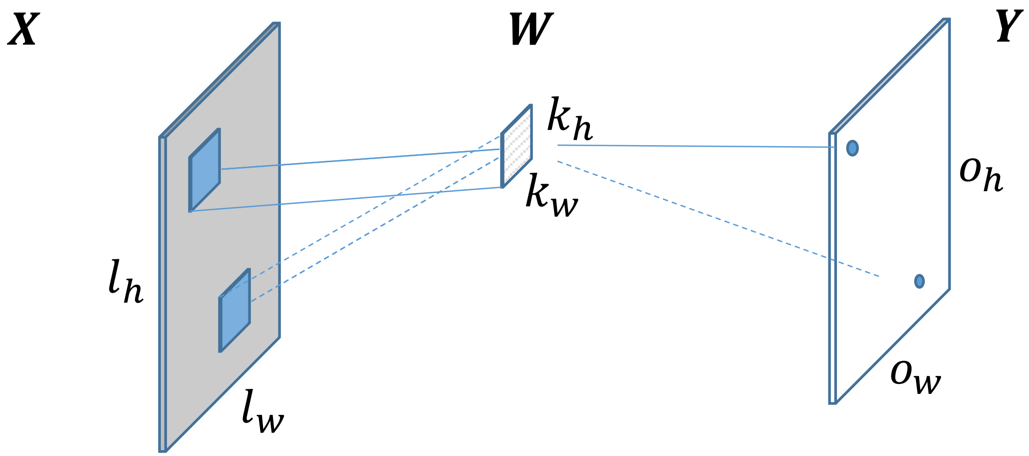
The result y is a row vector; if we compute y as $X^T w$, then the result y is a column vector

Converting many dot products to a vector-matrix multiplication

Slide credit: Wang Wei

2D Convolution

- 2D convolution follows the same principles, but the inputs, kernels and outputs are generalized into 2D matrices instead of 1D vectors



$$\boldsymbol{Y}_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} \boldsymbol{X}_{i*s_h+a,\, j*s_w+b} \times \boldsymbol{W}_{a,b}$$

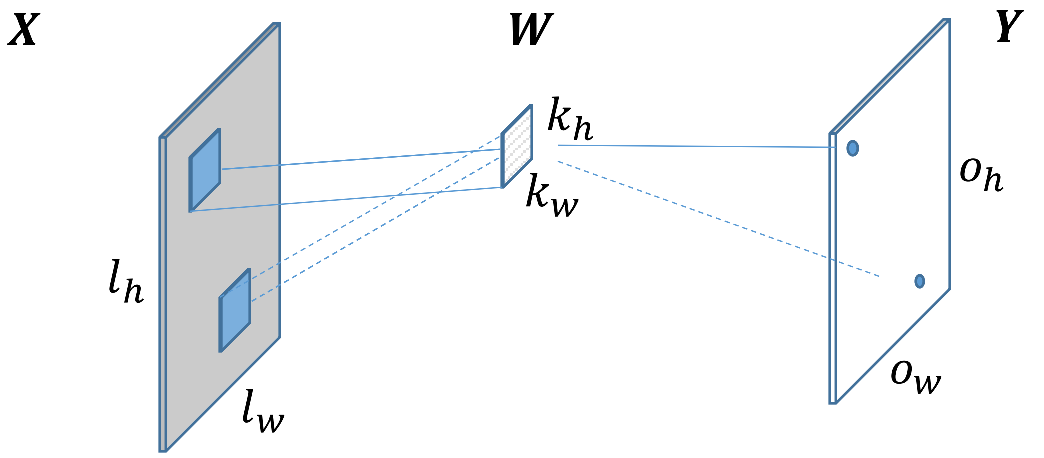
The operation between the receptive field and the kernel.

Is this a matrix-matrix multiplication?

Slide credit: Wang Wei

2D Convolution

- 2D convolution follows the same principles, but the inputs, kernels and outputs are generalized into 2D matrices instead of 1D vectors



No. It is not a matrix-matrix multiply.

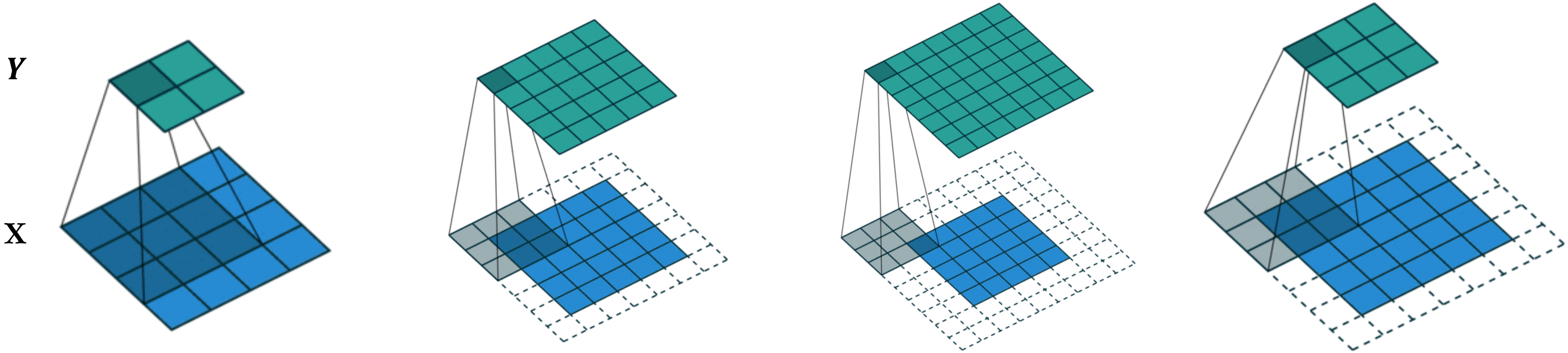
$$Y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} X_{i*s_h+a,\, j*s_w+b} \times W_{a,b}$$

Is this a matrix-matrix multiplication?

Element-wise operation: O(n^2) not O(n^3)

Slide credit: Wang Wei

2D Convolution



k=3, p=0, s=1 (Valid) k=3, p=2, s=1 (Same) k=3, p=4, s=1 (Full) k=3, p=2, s=2

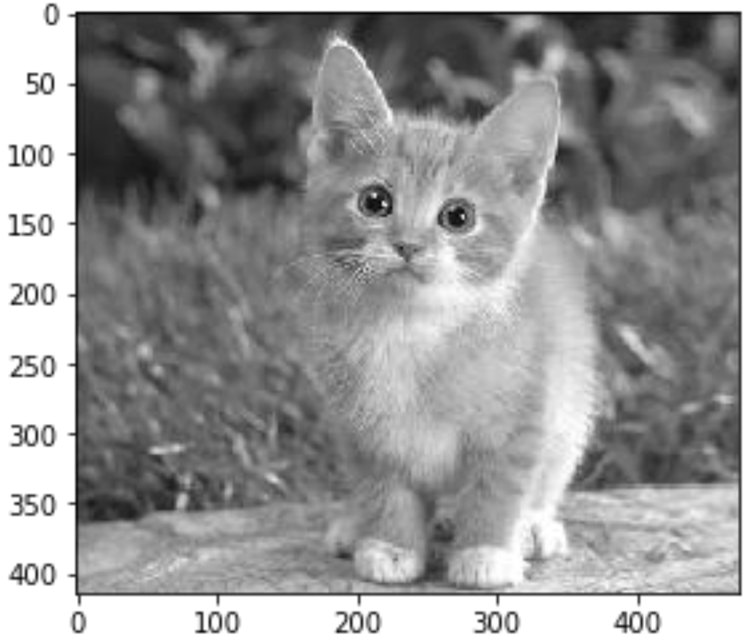
Source: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

Slide credit: Wang Wei

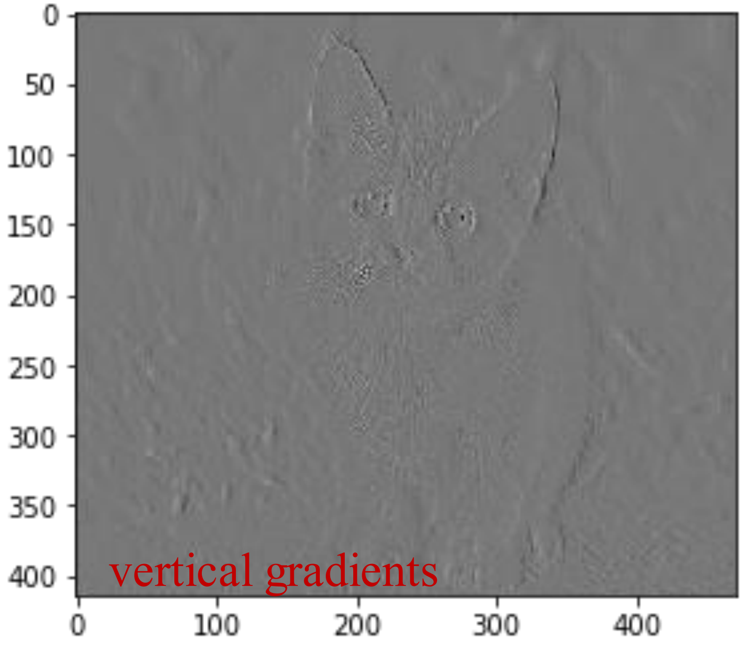
2D Convolution

In deep learning, we learn the kernels or weights which gives us good predictions e.g. classification, regression etc.

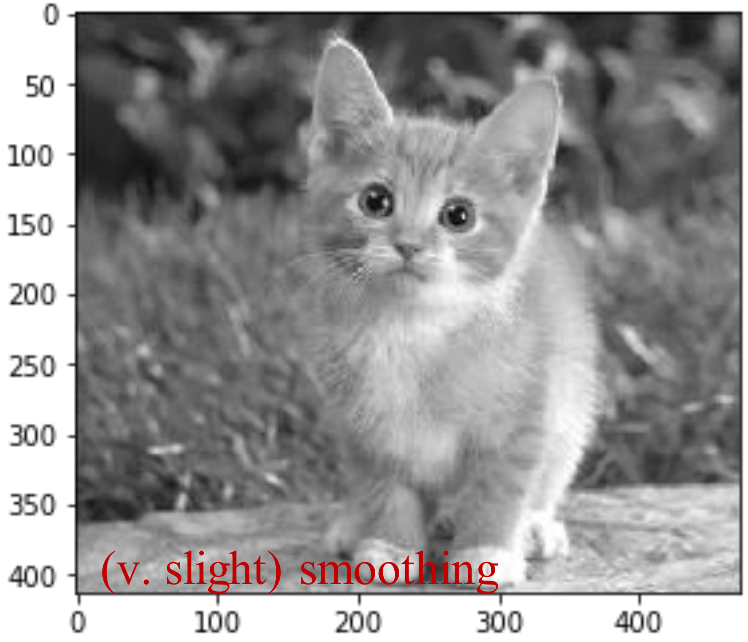
http://setosa.io/ev/image-kernels/



-1	1
-1	1



1	1
1	1



We get different feature maps depending on the kernel used.

Slide credit: Wang Wei

2D Convolution

- $\boldsymbol{Y}_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} \boldsymbol{X}_{i*s_h+a,\, j*s_w+b} \times \boldsymbol{W}_{a,b}$

1	2	3	2
2	3	2	5
1	2	2	1
0	2	1	2

1	2	3	2
2	3	2	5
1	2	2	1
0	2	1	2

-1	1
-1	1

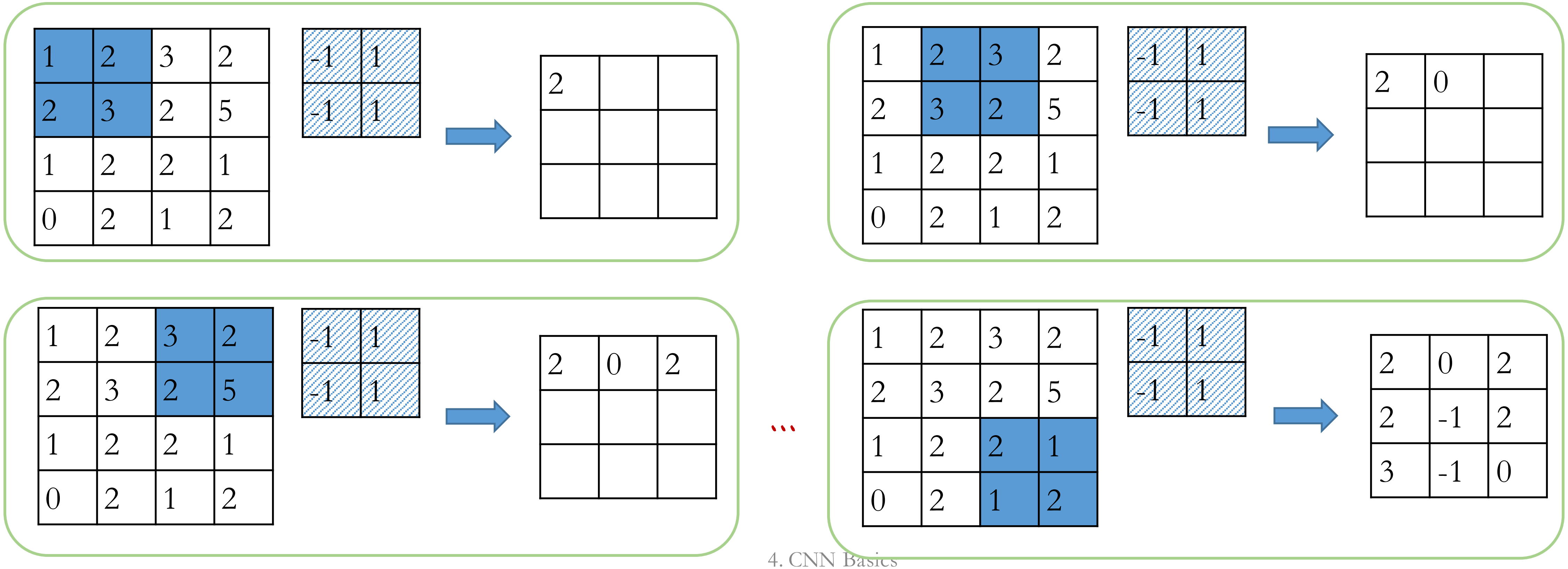
2		

Slide credit: Wang Wei

2D Convolution

1	2	3	2
2	3	2	5
1	2	2	1
0	2	1	2

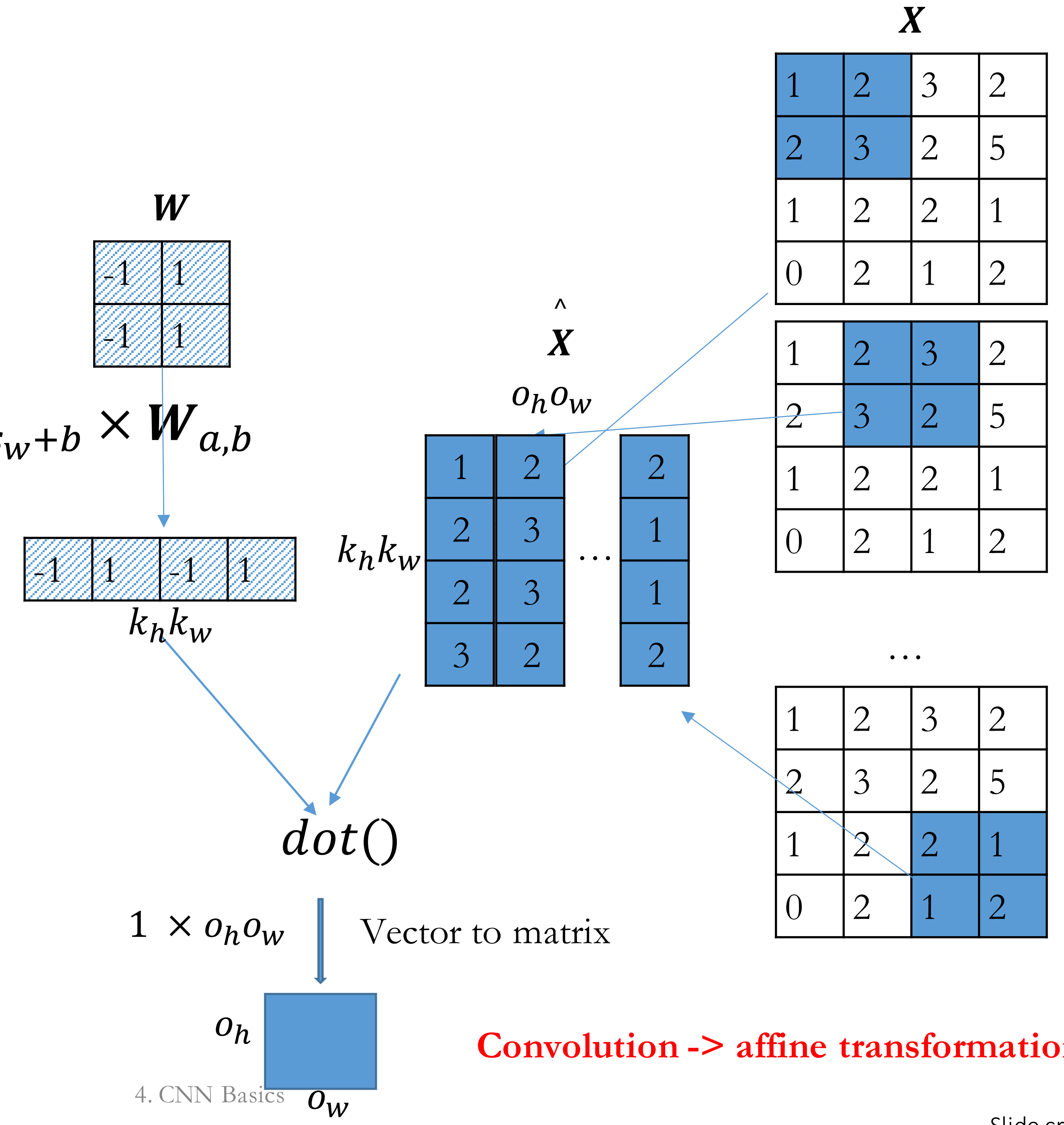
- $\boldsymbol{Y}_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} \boldsymbol{X}_{i*s_h+a,j*s_w+b} \times \boldsymbol{W}_{a,b}$



Slide credit: Wang Wei

Implementation

- Img2Col
  - Convert each receptive field into a column
- $Y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} X_{i*s_h+a,j*s_w+b} \times W_{a,b}$



Convolution -> affine transformation

Slide credit: Wang Wei

Statistics

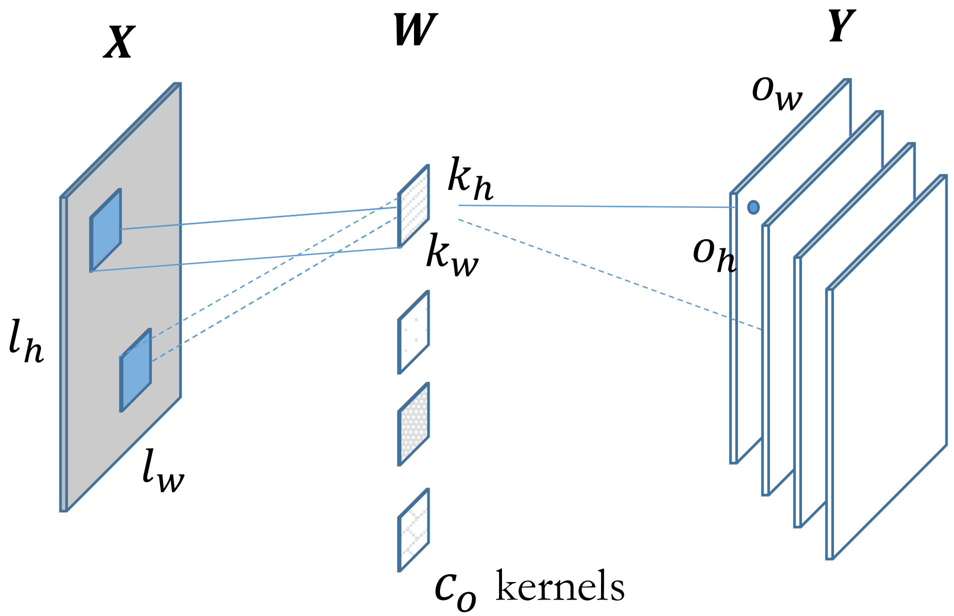
- Shape:
  - Input $\boldsymbol{X} \in R^{n_h \times n_w}$
  - Kernel $\boldsymbol{W} \in R^{k_h \times k_w}$
  - Output $\mathbf{Y} \in R^{o_h \times o_w}$
- Parameter size
  - $k_h \times k_w$
- Output shape
  - $(o_h, o_w) = (\left\lfloor \frac{n_h + p_h - k_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{n_w + p_w - k_w}{s_w} \right\rfloor + 1)$
- Computation cost
  - O($k_h \times k_w \quad \times \quad o_h \times o_w$) (float multiplication ops, FLOP)

Operations for each output or each Kernel computation is K_h x K_w

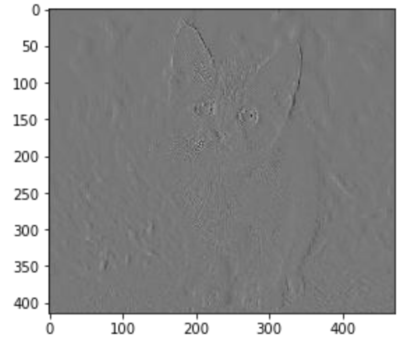
Slide credit: Wang Wei

2D Convolution

- Multiple kernels/filters



-1	1
-1	1



1	1
1	1

c_o kernels

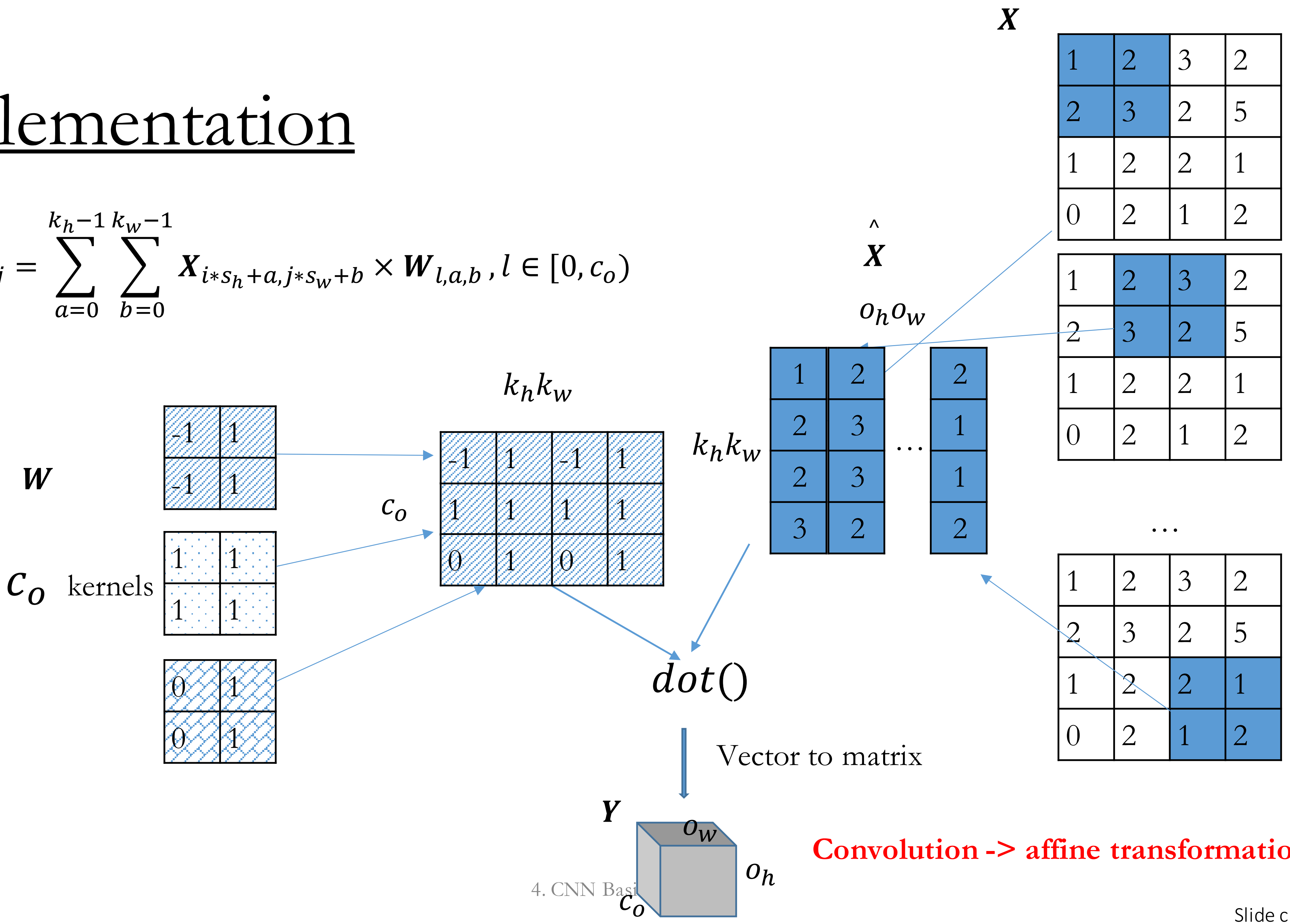
Please note, c_o is not c_0

$$\boldsymbol{Y}_{l,i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} \boldsymbol{X}_{i*s_h+a,\, j*s_w+b} \times \boldsymbol{W}_{l,a,b}\,,\, l \in [0, c_o)$$

Slide credit: Wang Wei

Implementation

$$Y_{l,i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} X_{i*s_h+a,\, j*s_w+b} \times W_{l,a,b}\, ,\, l \in [0, c_o)$$



Convolution -> affine transformation

Slide credit: Wang Wei

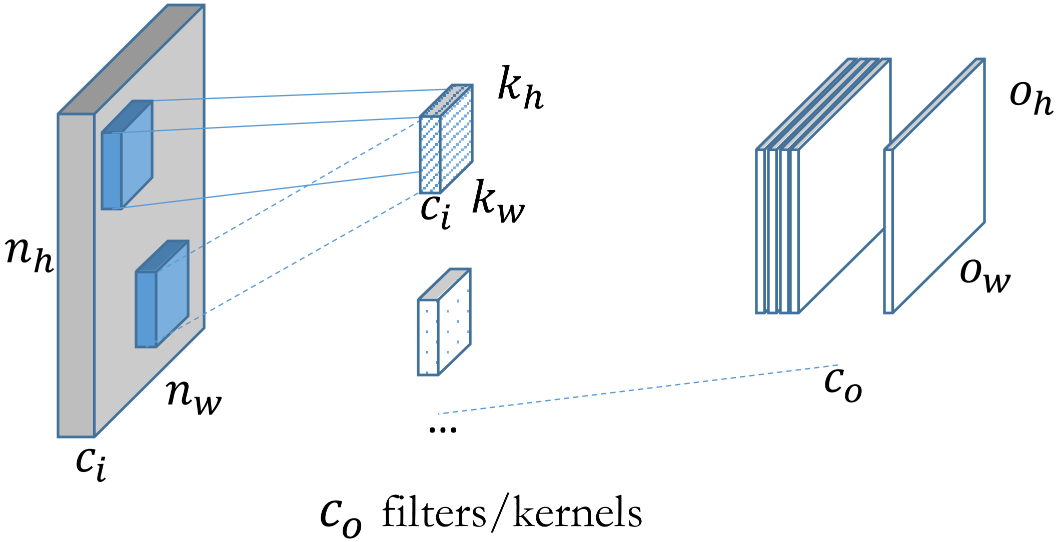
Statistics

- applying multiple kernels (filters) c_o, all of the same stride and padding
- Parameter size
  - $c_o \times k_h \times k_w$
- Output shape
  - $(c_o, o_h, o_w) = (c_o, \left\lfloor\frac{n_h+p_h-k_h}{s_h}\right\rfloor + 1, \left\lfloor\frac{n_w+p_w-k_w}{s_w}\right\rfloor + 1)$
- Computation cost
  - O($(c_o \times k_h \times k_w) \times (o_h \times o_w)$) (float multiplication ops, FLOP)

Slide credit: Wang Wei

2D Convolution

With multiple (c_i) input channels and kernels (filters)



** The convolution results across all input channels are summed.

$$\boldsymbol{Y}_{l,i,j} = \sum_{d=0}^{c_i-1} \underbrace{\sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1}} \boldsymbol{X}_{d,i+a,j+b} \times \boldsymbol{W}_{l,d,a,b} + \boldsymbol{b}_l, l \in [0, c_o)$$

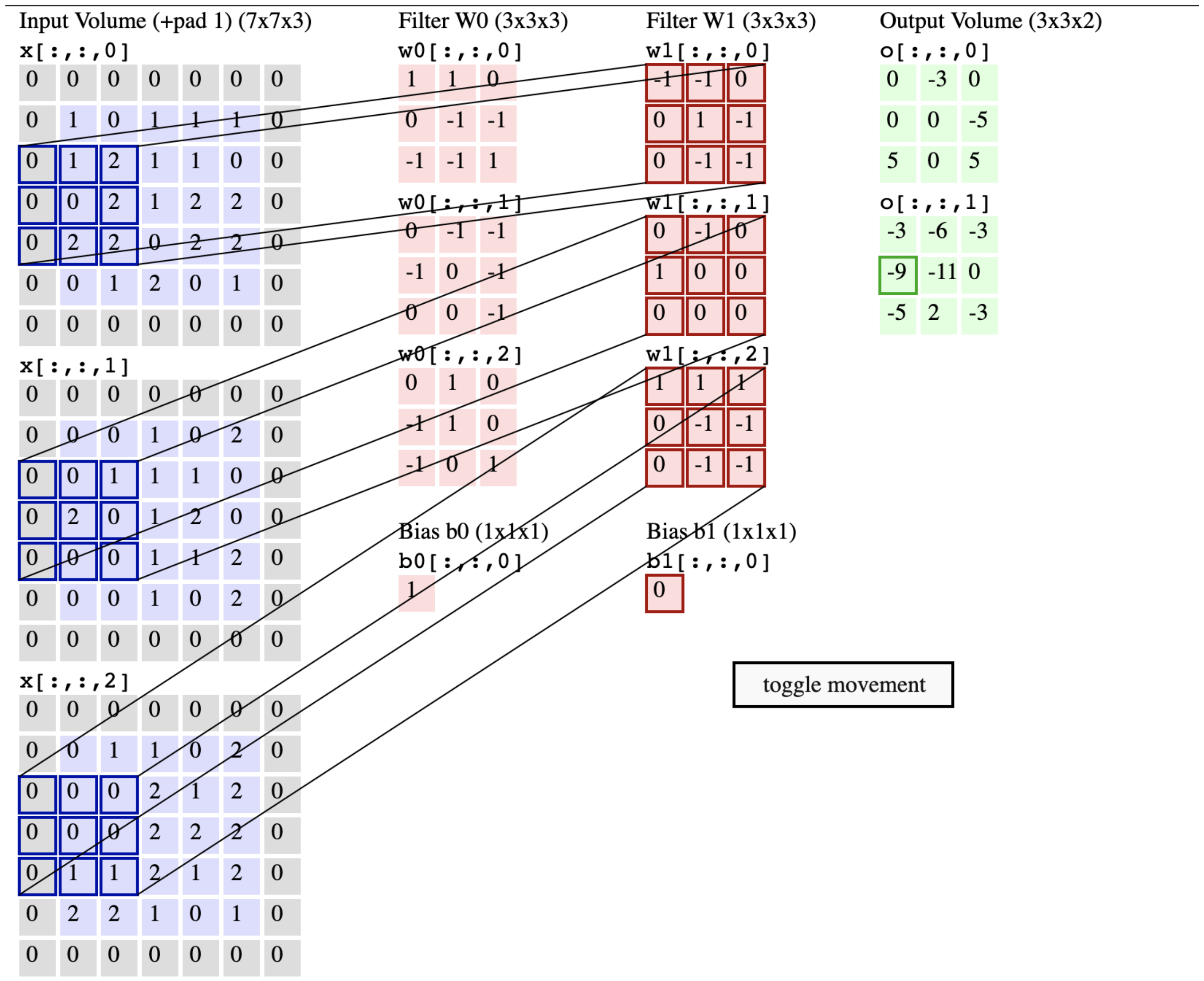
$\boldsymbol{b}$ is a bias vector

l,i,j have noting to do with C_i

** this is still a 2D convolution because the kernel is moved only across the horizontal and vertical dimensions (as indexed by a, b).

Slide credit: Wang Wei

2D Convolution (w/ 3D kernels and data)



Demo [link]
Visualization [link]

Slide credit: Wang Wei

Implementation

- Forward
  - Convert input feature maps $\boldsymbol{X}$ into matrix $\hat{\boldsymbol{X}}$ (img2col) of size $(c_i k_w k_h \times o_h o_w)$
  - Reshape the filters $\boldsymbol{W}$ to $(c_o \times c_i k_h k_w)$
  - $\boldsymbol{Y} = \boldsymbol{W}\hat{\boldsymbol{X}} + \boldsymbol{b}$
  - Computational cost, $O(c_o \times c_i \times k_w \times k_h \times o_h \times o_w)$
- Backward
  - Given $\frac{\partial L}{\partial \boldsymbol{Y}}$
  - Compute $\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{Y}} \hat{\boldsymbol{X}}^T$, $\frac{\partial L}{\partial \hat{\boldsymbol{X}}} = \boldsymbol{W}^T \frac{\partial L}{\partial \boldsymbol{Y}}$
  - Column to receptive field transformation to get gradient wrt original X

Slide credit: Wang Wei

Img2Col

- Slide the window from left to right, top to bottom
- Copy the values from the receptive field into a column of $\hat{\boldsymbol{X}}$
  - Receptive fields across feature maps are concatenated into to one column
- Reshape $\hat{\boldsymbol{X}}$ into $(c_i k_h k_w \times o_h o_w)$

Feature map 0

1	2	1	2	3	4
2	2	3	1	2	0
1	1	2	1	0	1
2	1	2	1	1	3

Feature map 1

0	2	3	2	0	1
1	0	1	1	2	0
2	0	1	1	1	1
1	1	2	1	1	2

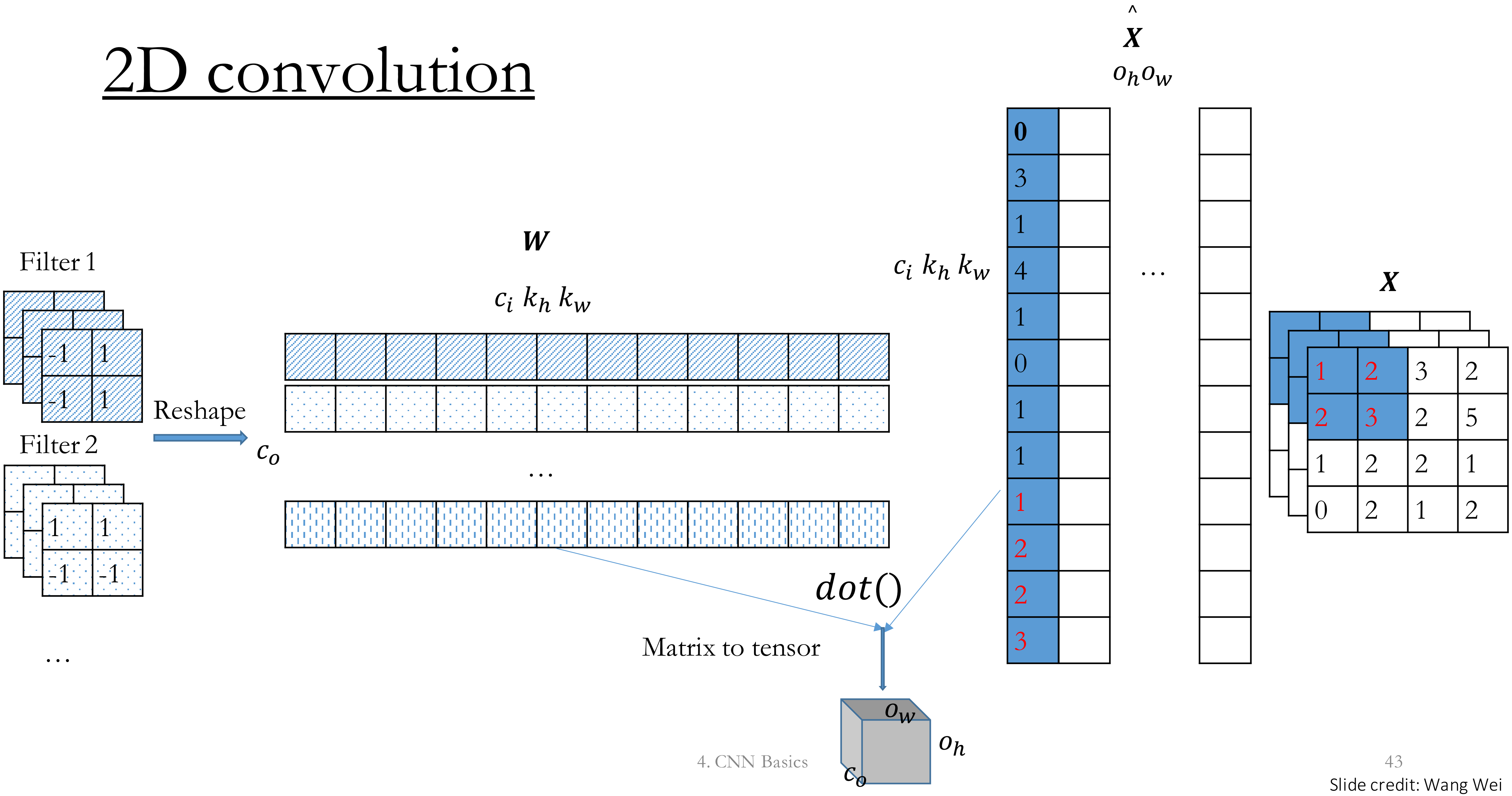
$o_h o_w = 2 \cdot 3$

$c_i k_h k_w = 2 \cdot 2 \cdot 2$

1	1	3	1	2	0
2	2	4	1	1	1
2	3	2	2	2	1
2	1	0	1	1	3
0	3	0	2	1	1
2	2	1	0	1	1
1	1	2	1	2	1
0	1	0	1	1	2

Slide credit: Wang Wei

2D convolution



Slide credit: Wang Wei

Statistics

- Parameter size
  - Weights: $c_o \times (c_i\, k_h\, k_w)$
  - Bias: c_o
- Output shape
  - $(c_o, o_h, o_w) = (c_o, \left\lfloor \frac{n_h + p_h - k_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{n_w + p_w - k_w}{s_w} \right\rfloor + 1)$
- Computation cost
  - $O(c_o \times c_i\, k_h\, k_w \times o_h\, o_w)$

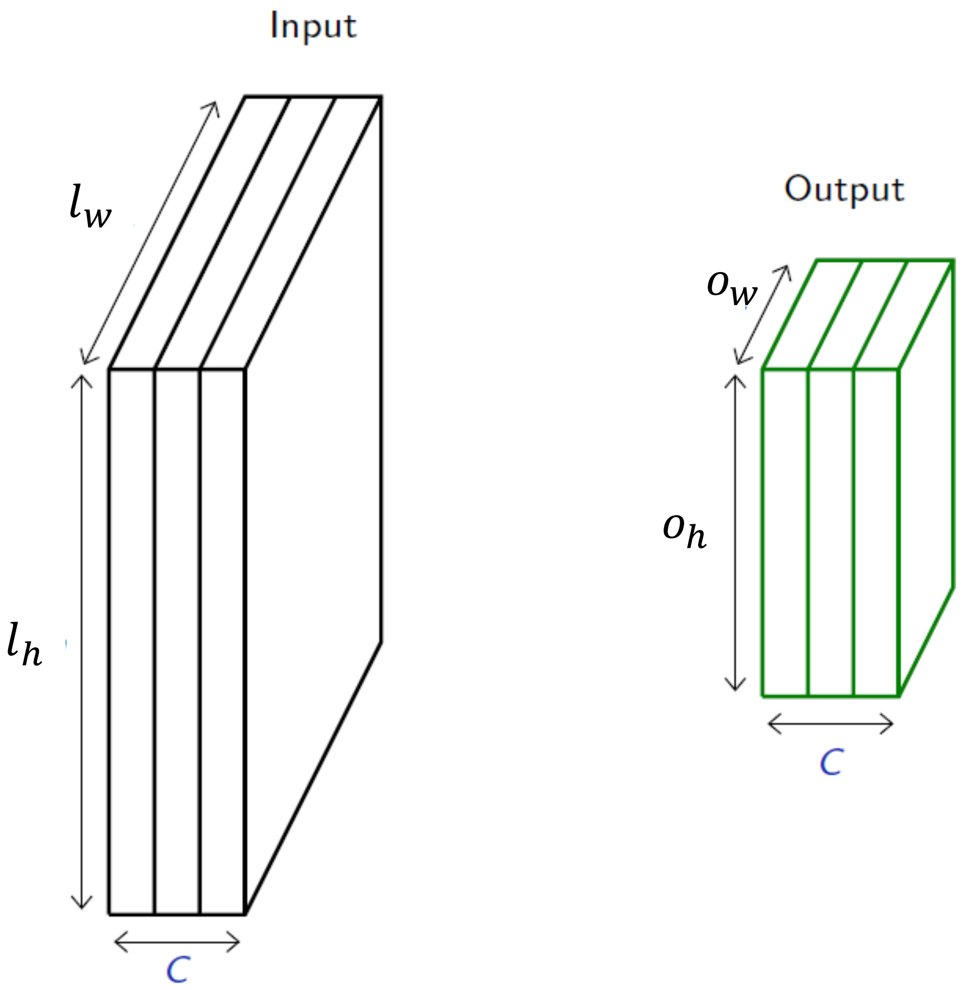
Slide credit: Wang Wei

Pooling

- Aggregate information from each receptive field
  - Max
  - Average
- No parameters
- Applied for each channel respectively
  - #input channels = # output channels, i.e., $c_i = c_o = c$
- Padding and stride can be applied

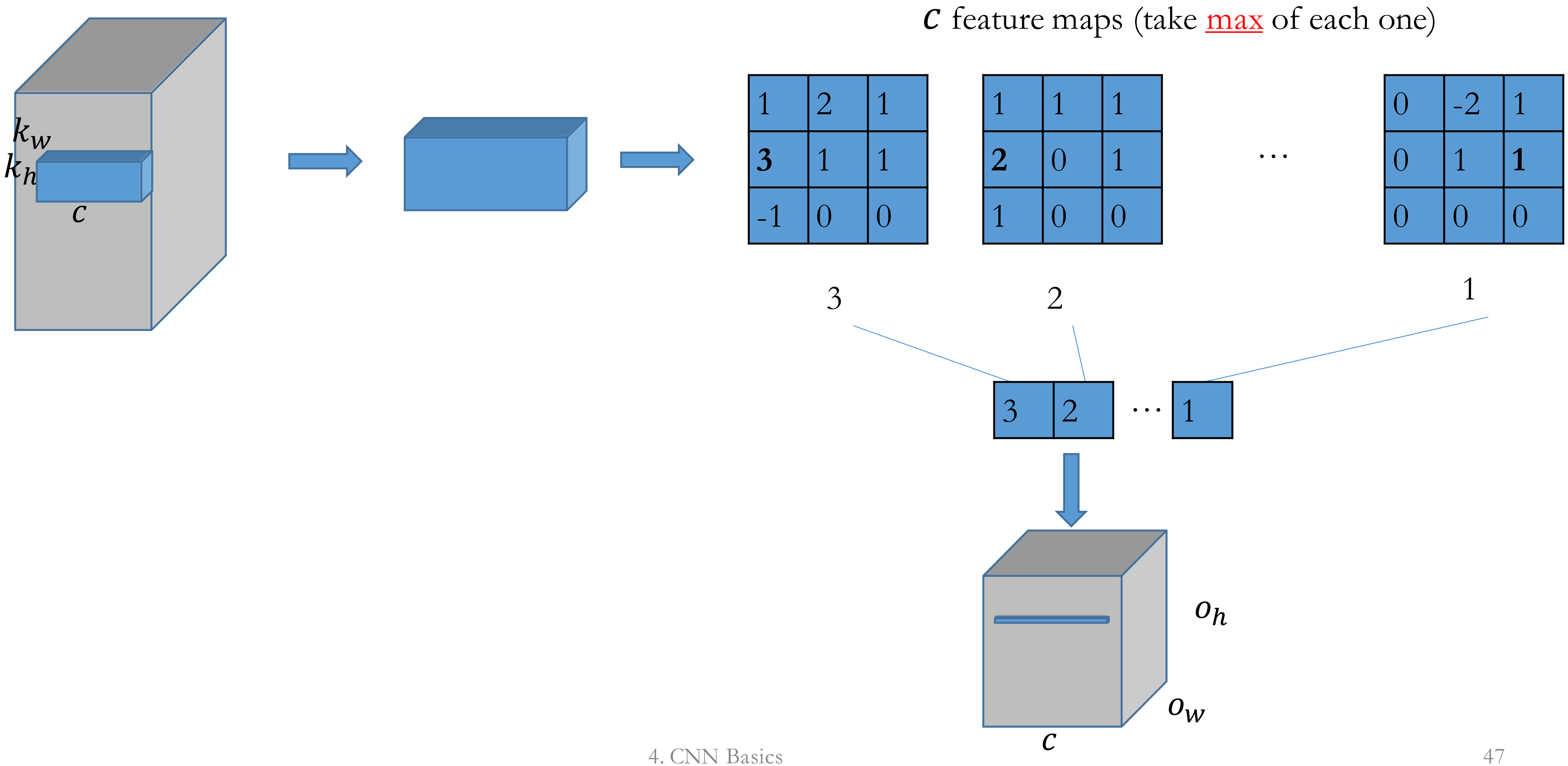
Slide credit: Wang Wei

Pooling Visualization



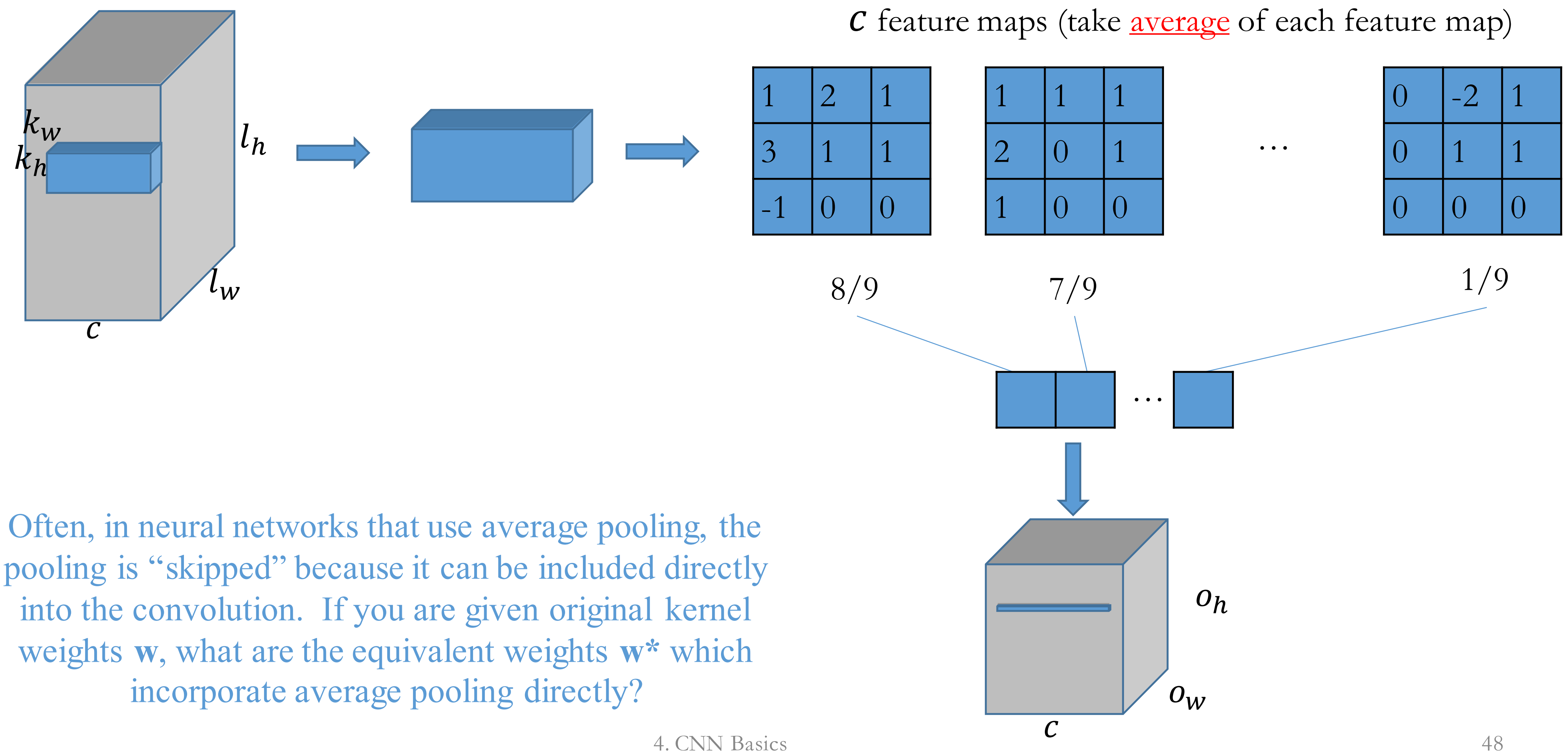
Slide credit: Wang Wei

Max Pooling



Slide credit: Wang Wei

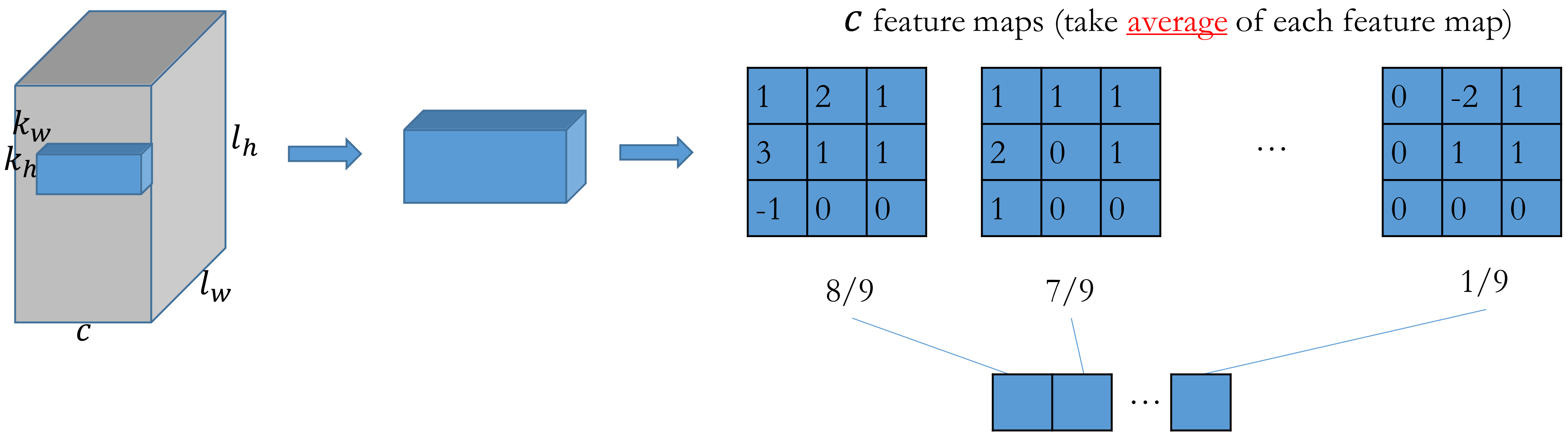
Average Pooling



Often, in neural networks that use average pooling, the pooling is "skipped" because it can be included directly into the convolution. If you are given original kernel weights **w**, what are the equivalent weights **w*** which incorporate average pooling directly?

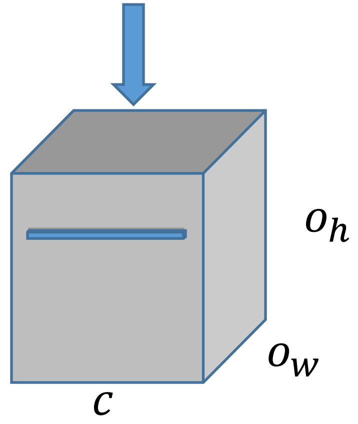
Slide credit: Wang Wei

Average Pooling



Equivalent to convolution with kernel values = 1/kernel_size

Often, in neural networks that use average pooling, the pooling is ”skipped” because it can be directly into the convolution. If you are given original kernel weights **w**, what are the equivalent weights **w*** which incorporate average pooling directly?



Slide credit: Wang Wei

Effect of Pooling

- Reduces the feature size and model size
- Information aggregation
  - Max pooling: invariant to rotation of the **input** image
  - Average pooling: can be replaced by convolution; much cheaper (no weights)

0	1	2	3
4	**5**	**6**	7
8	**9**	**1**	2
3	4	5	6

flip

3	2	1	0
7	**6**	**5**	4
2	**1**	**9**	8
6	5	4	3

Rotate 90·

	1	**6**	
	9	**5**	

Slide credit: Wang Wei

Multiple input channels, multiple filters

Configuration?

2-minute quiz:

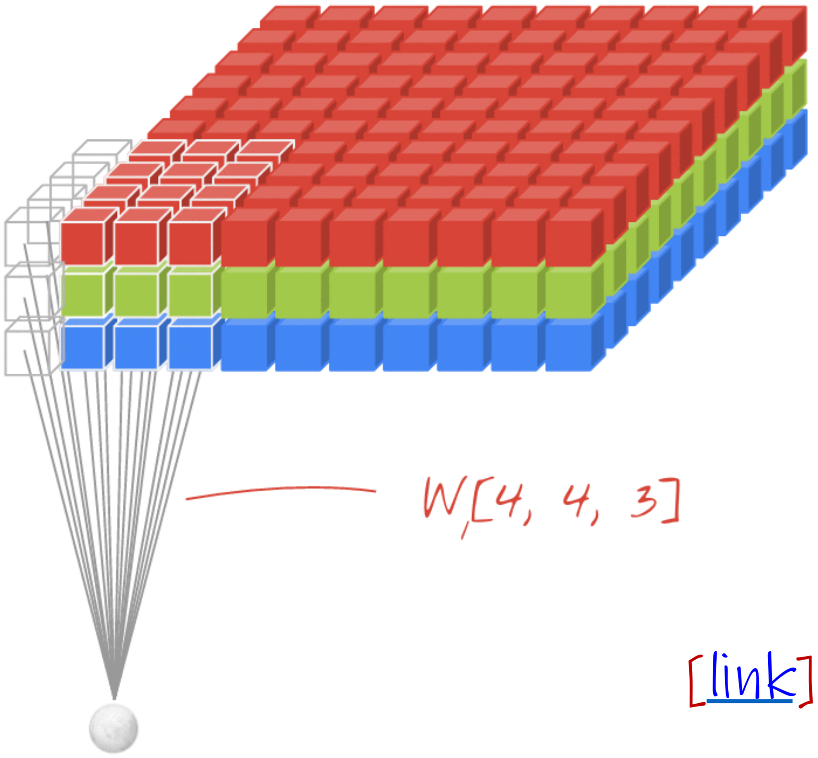
How many input channels?

How many kernels are there?

What is the kernel size?

What is the padding?

What is the stride?



W[4, 4, 3]

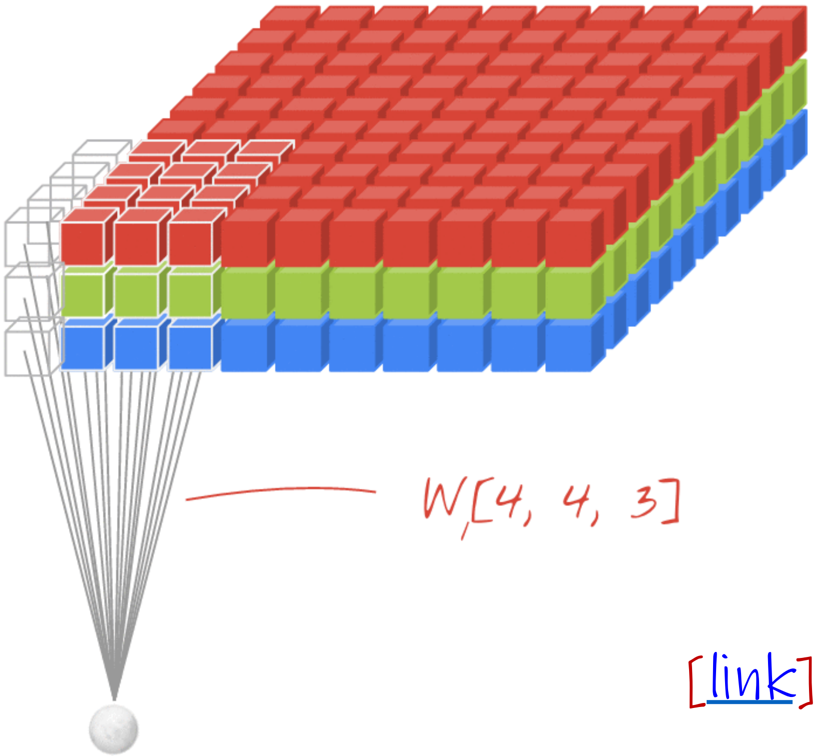
[link]

Slide credit: Wang Wei

Multiple input channels, multiple filters

Configuration?

Kernel size: 3 x 4 x 4
3 input channels
2 filters/kernels
Padding = 3
Stride =1



Slide credit: Wang Wei

Summary

- Convolution
    - Cross-correlation
    - Kernel, receptive field, padding, stride
    - VS MLP
- 2D convolution
    - Single channel, single kernel
    - Single channel, multiple kernels
    - Multiple channels, multiple kernels
- Pooling
    - Max and Average pooling

Slide credit: Wang Wei