Neural Networks and Deep Learning Lecture 5

Wei WANG

cs5242@comp.nus.edu.sg

National University of Singapore
School of Computing

Recap

Training techniques for DNN

- Parameter initialization
  - Randomness to break symmetry for W
  - Gaussian, Uniform for W
    - Gaussian N(0, sqrt(2/(fan_in + fan_out))
    - Gaussian N(sqrt(2/fan_in))
- Batch normalization
  - After affine/linear transformation, before non-linear activation
  - $\widehat{z_k} = \frac{z_k - E[z_k]}{\sqrt{var[z_k]}}$, $\overline{z_k} = \gamma_k \widehat{z_k} + \beta_k$

Training techniques for DNN

- Dropout
  - Randomly set some neurons to be zero with probability p
  - Scale the neurons by 1/(1-p)
- Early stopping
  - To prevent overfitting
- Data (image) augmentation
  - Random crop/rotation/resize during training
  - Fixed crop/rotation/resize during test

Image augmentation (pseudo code)

```
# train augmentation (random)
for i in range(num_iters):
    batch = []
    for b in range(batchsize):
        img = load_image("path")
        img = img.resize((256,256))
        x = math.random.randint(0, 256-224)
        y = math.random.randint(0, 256-224)
        img = img.crop((x, y, 224, 224))
        if math.random.randint(0,1) == 0:
                img = img.flip()
        batch.append(img)
    train(batch)

# test time augmentation
img = load_image("path")
img = img.resize((256,256))
offsets = [(0,0), (0, 32), (32, 0), (32, 32), (16, 16)]
for (x, y) in offsets:
    img = img.crop((x, y, 224, 224))
    batch.append(img)
    batch.append(img.flip())
results = predict(batch)  # a matrix of shape 5xC
print('prediction is %d' % np.argmax(results.average(axis=0)))
```

Convolutional neural network (CNN)

From MLP to CNN

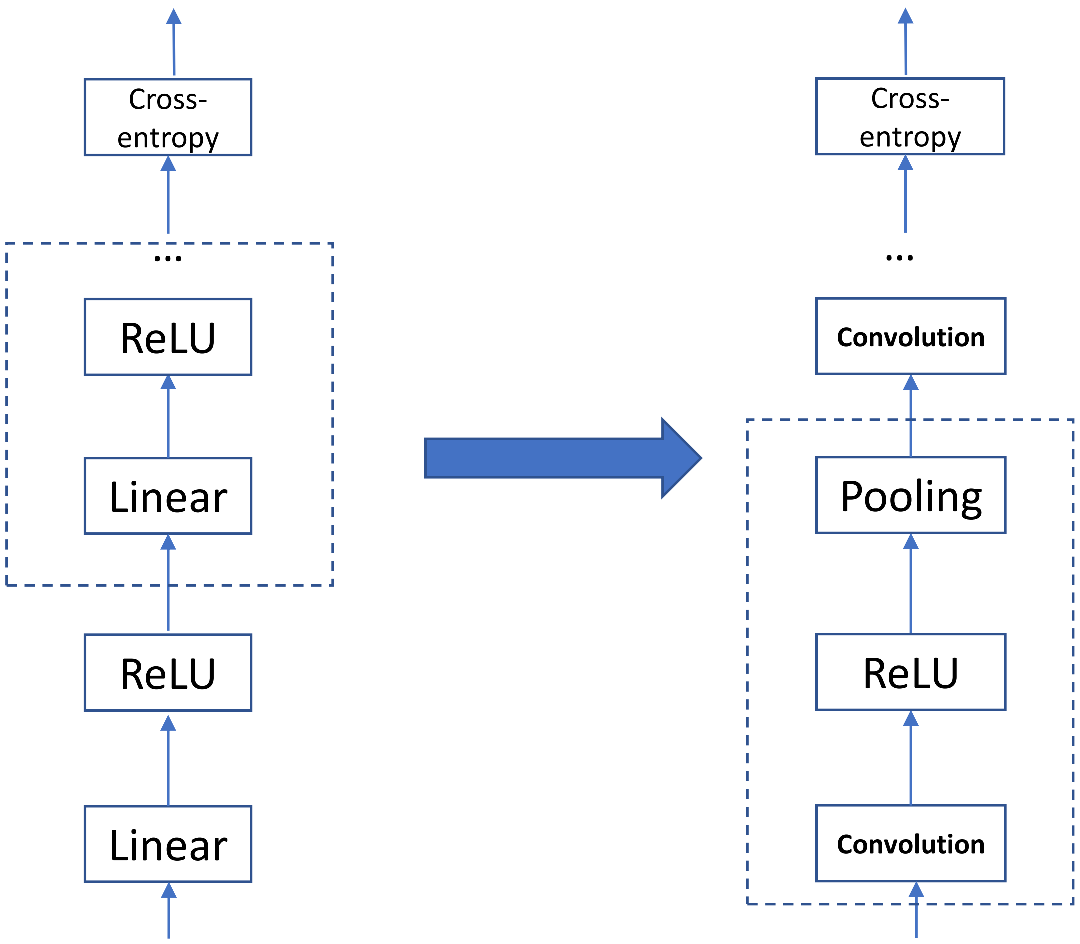
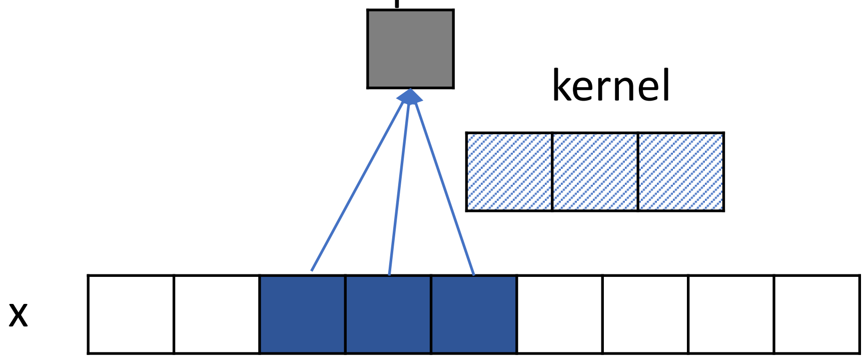


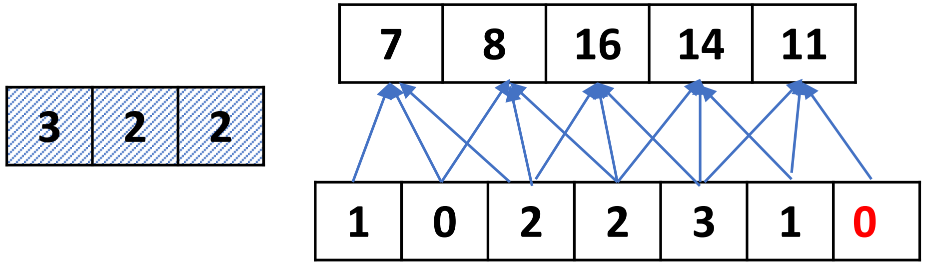
Image source: Stanford cs231n

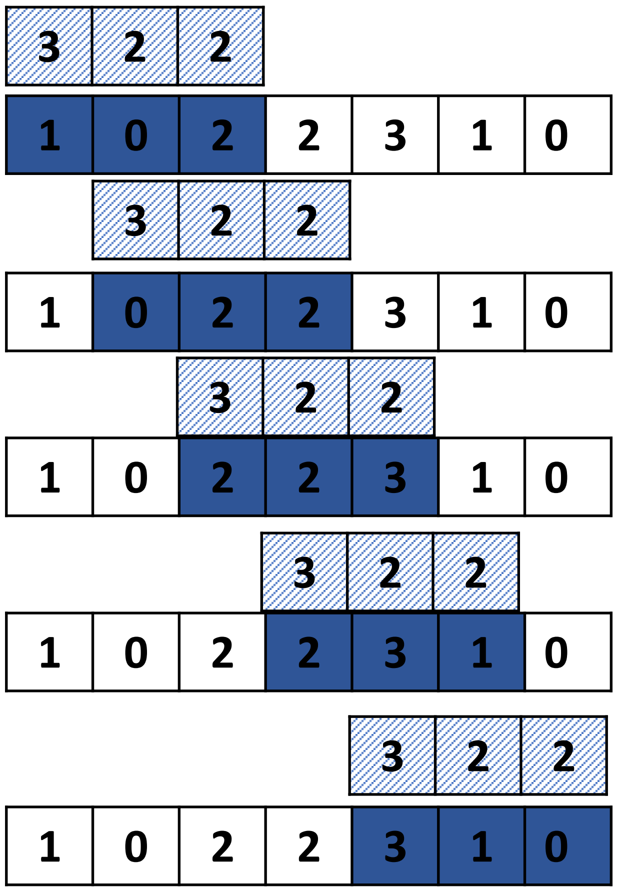
Convolution

- 1D convolution
  - Text processing
- 2D convolution
  - Image processing
- 3D convolution
  - 3D data, e.g. CT.

1D convolution







$3x1+2x0+2x2=7$

$3x0+2x2+2x2=8$

$3x2+2x2+2x3=16$

$3x2+2x2+2x1=14$

$3x3+2x1+2x0=11$

1D convolution

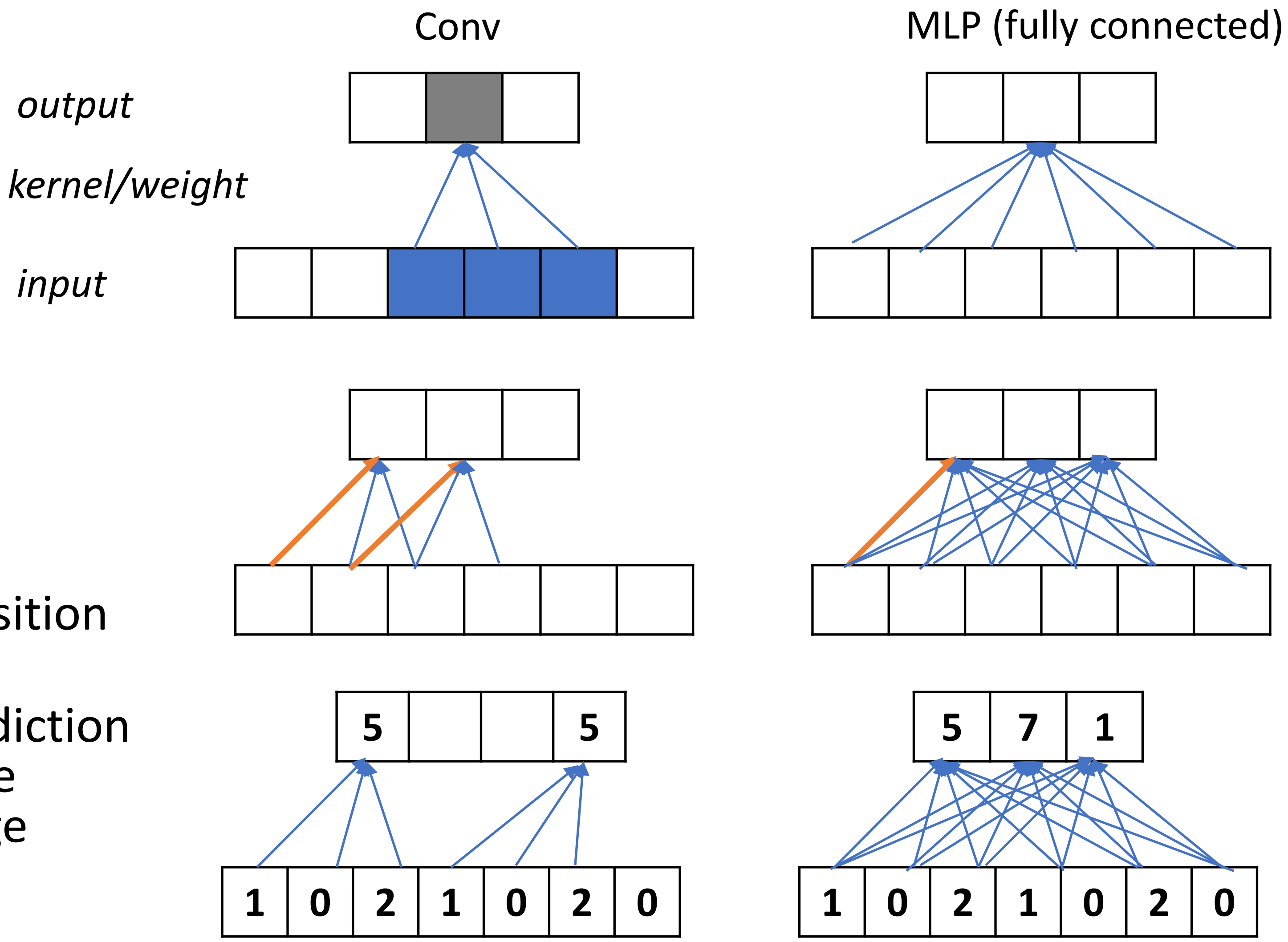
- Cross-correlation (https://en.wikipedia.org/wiki/Cross-correlation)

$$y_t = \sum_{i=0}^{k-1} w_i \times x_{t+i}$$

  - **In CNN, convolution refers to cross-correlation**
  - w is called kernel/filter; the parameters to be trained; length k
  - x is the input; length l
  - the input area, i.e. t-(k-1),…, t-1, t is called the receptive field
    - One receptive field generates one output value
  - y_t is the output feature; length o

Properties (Why Convolution better?)

- Sparse connection
  - Fewer parameters
  - Less overfitting
- Weight sharing
  - Regularization
  - Less overfitting
- Location invariant
  - Robust to object position in the image
  - Make the same prediction no matter where the object is in the image



Perceptron, MLP and Convolution

Perceptron

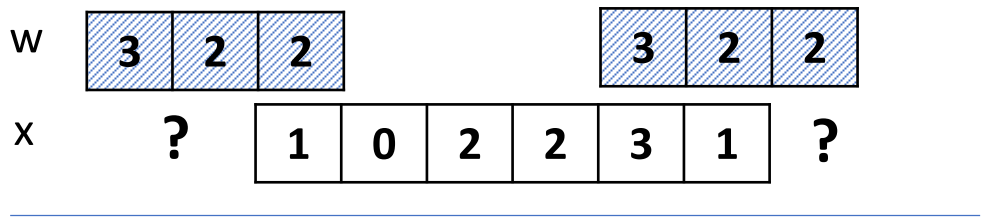
Perceptron is too simple → underfitting→ add more layers → MLP

MLP

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

CNN

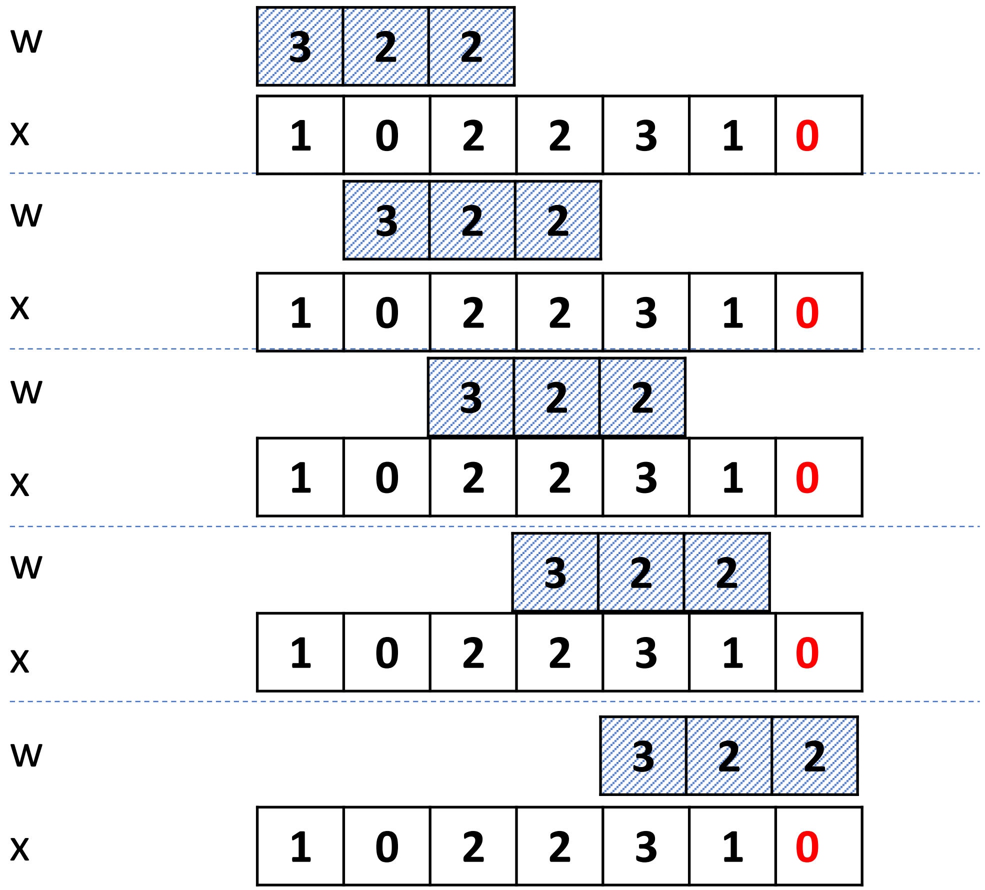
Padding



- $o = l - k + 1 \leq l - 1 \; (k \geq 2)$
  - By stacking multiple convolution layers, the output is shorter and shorter
  - Less information per layer

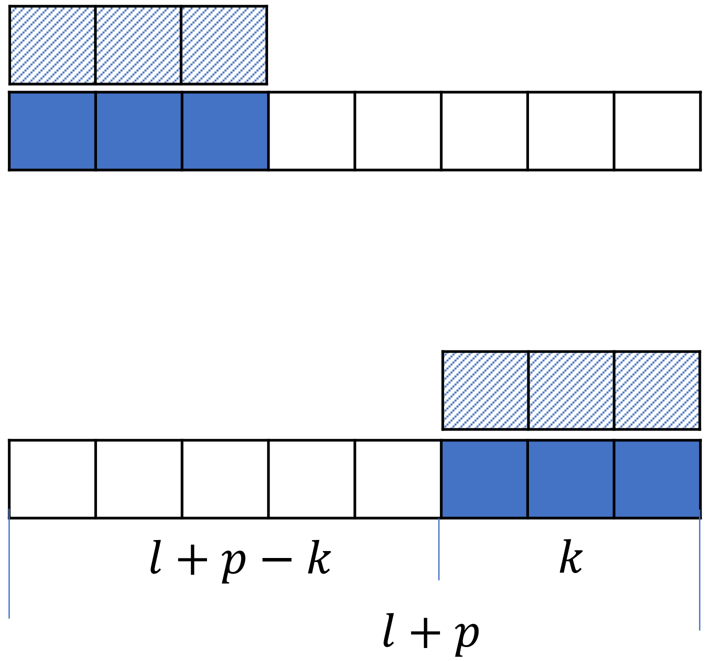
Padding

- 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
  - Torch, PyTorch, Caffe, SINGA
  - https://github.com/apache/incubator-singa



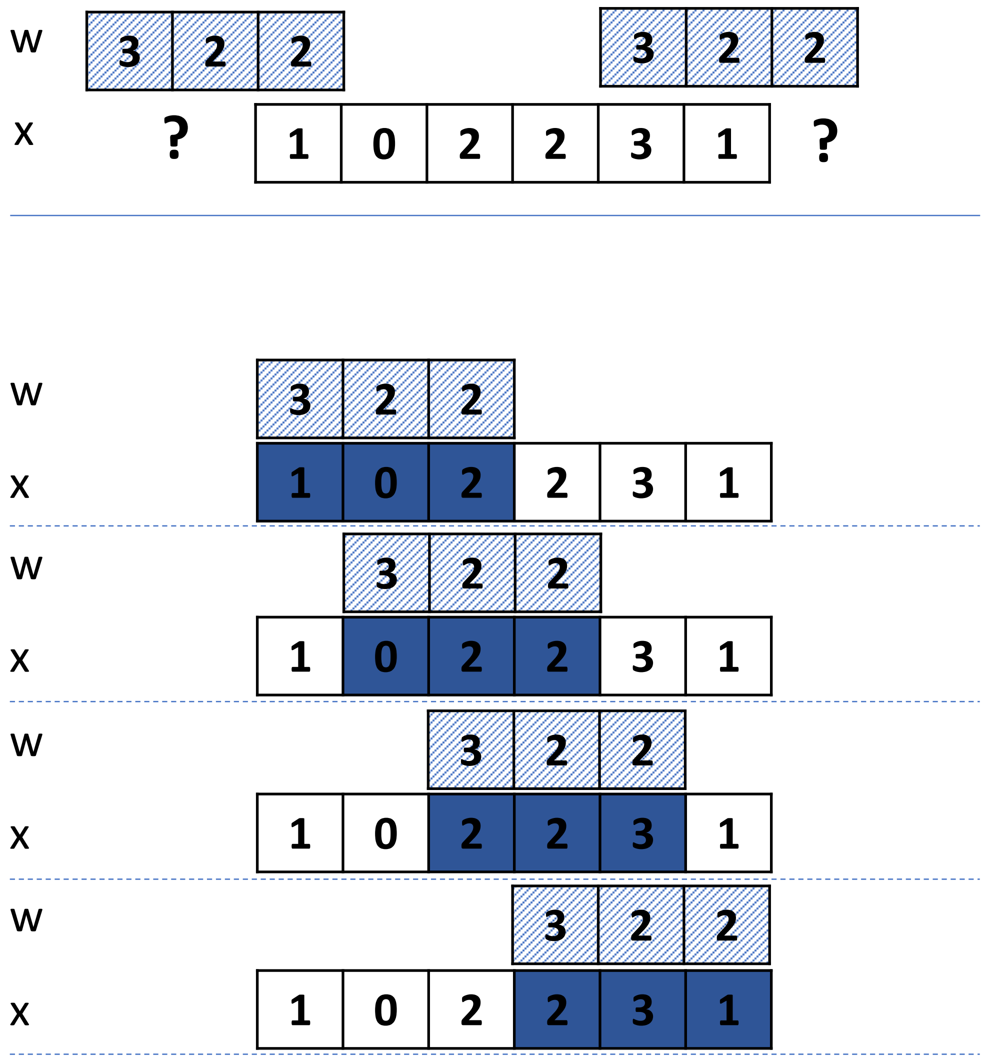
Padding

- Manual padding (p)
  - Kernel size/length: k
  - Input length: l
  - # outputs $o = l + p - k + 1$



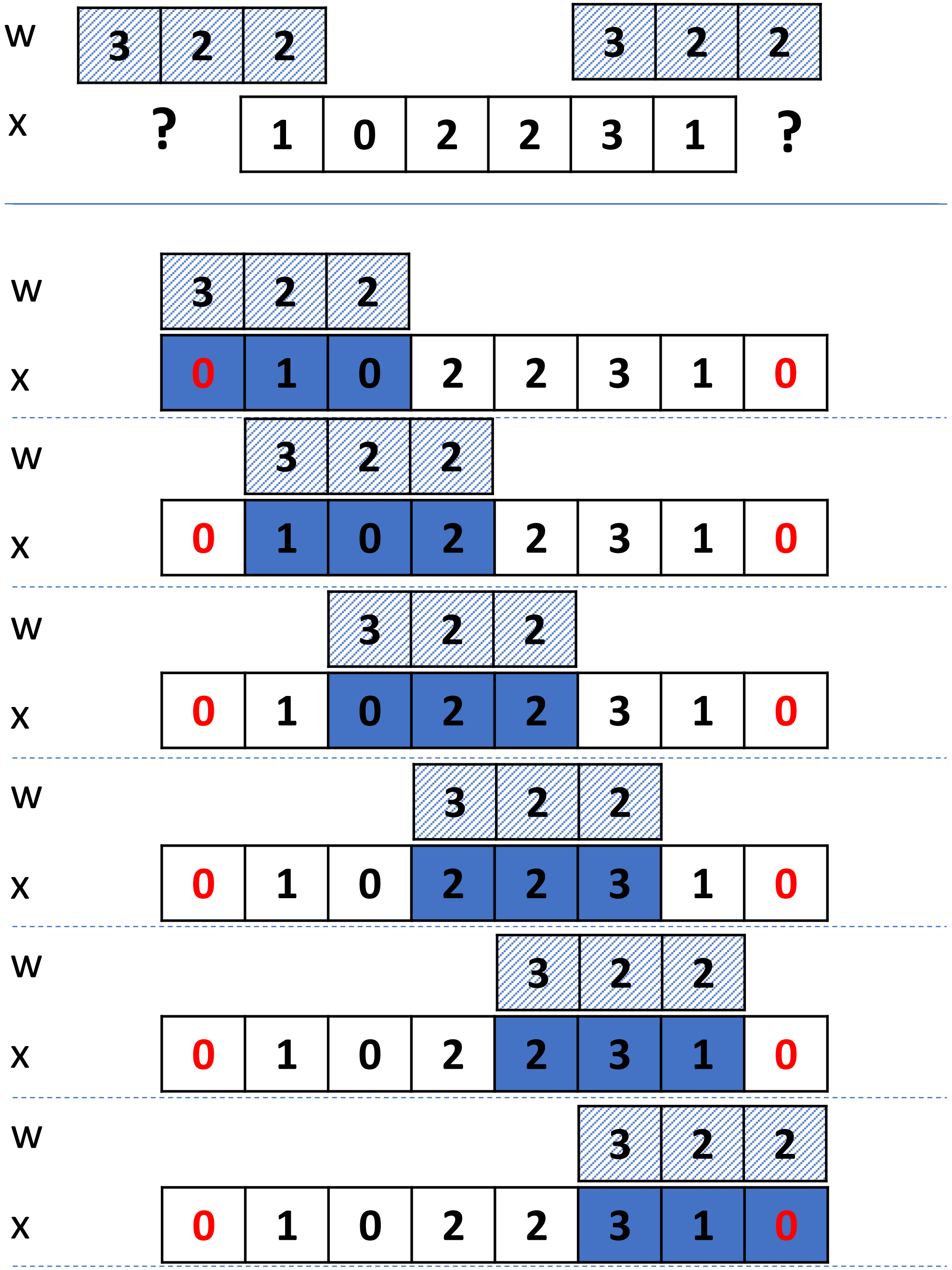
Padding

- Valid/No padding ($p = 0$)
  - # inputs denoted as l
  - # outputs $o = l - k + 1$=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
  - TensorFlow, Keras



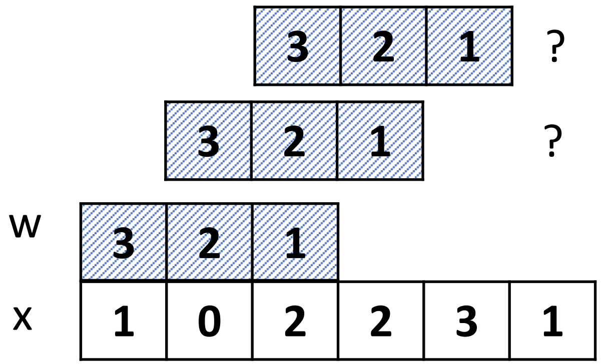
Padding

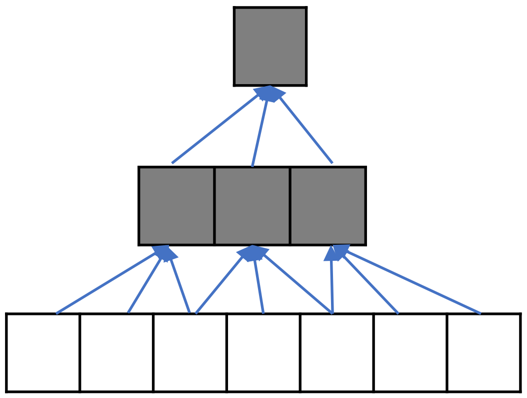
- Same padding (p?)
  - $l + p - k + 1 = l$
  - $p = k - 1$
    - Left padding = $\lfloor p/2 \rfloor$
    - Right padding = $\lceil p/2 \rceil$
  - Output values
    - 3x0+2x1+2x0=2
    - 3x1+2x0+2x2=7
    - 3x0+2x2+2x2=8
    - 3x2+2x2+2x3=16
    - 3x2+2x3+2x1=14
    - 3x3+2x1+2x0=11
  - TensorFlow,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

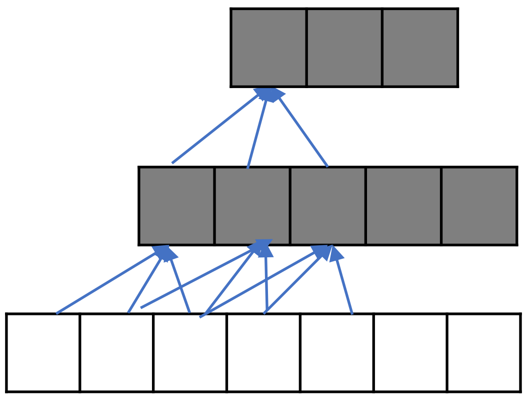


Stride (Why?)

- How many steps to move towards the next receptive field
  - s=1, every receptive field is considered -> many outputs
  - s>1, some receptive fields are skipped.
    - Faster
    - Fewer outputs
    - *Effective receptive field size is increased quickly*

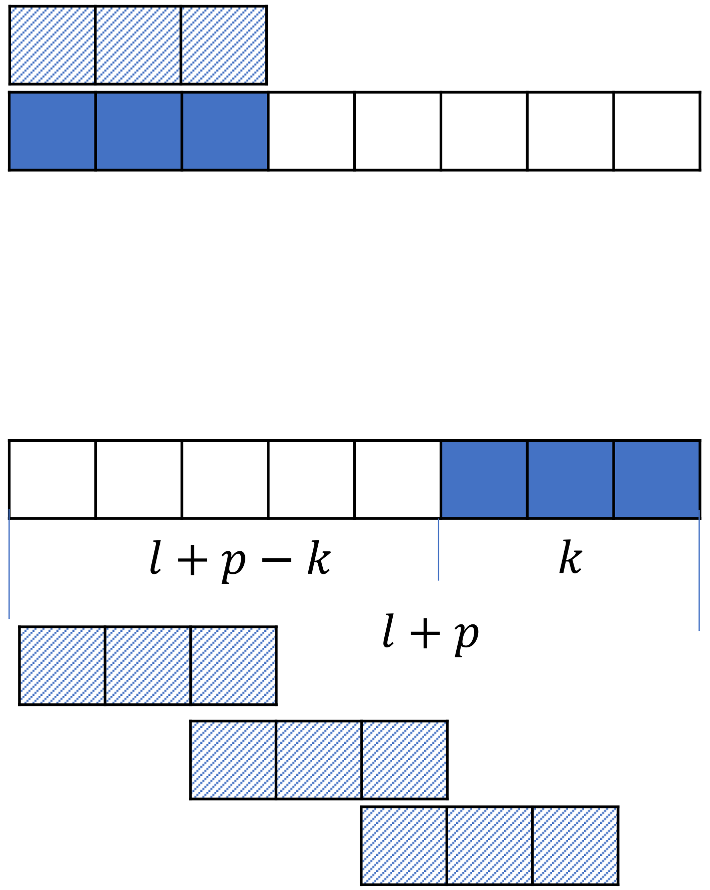






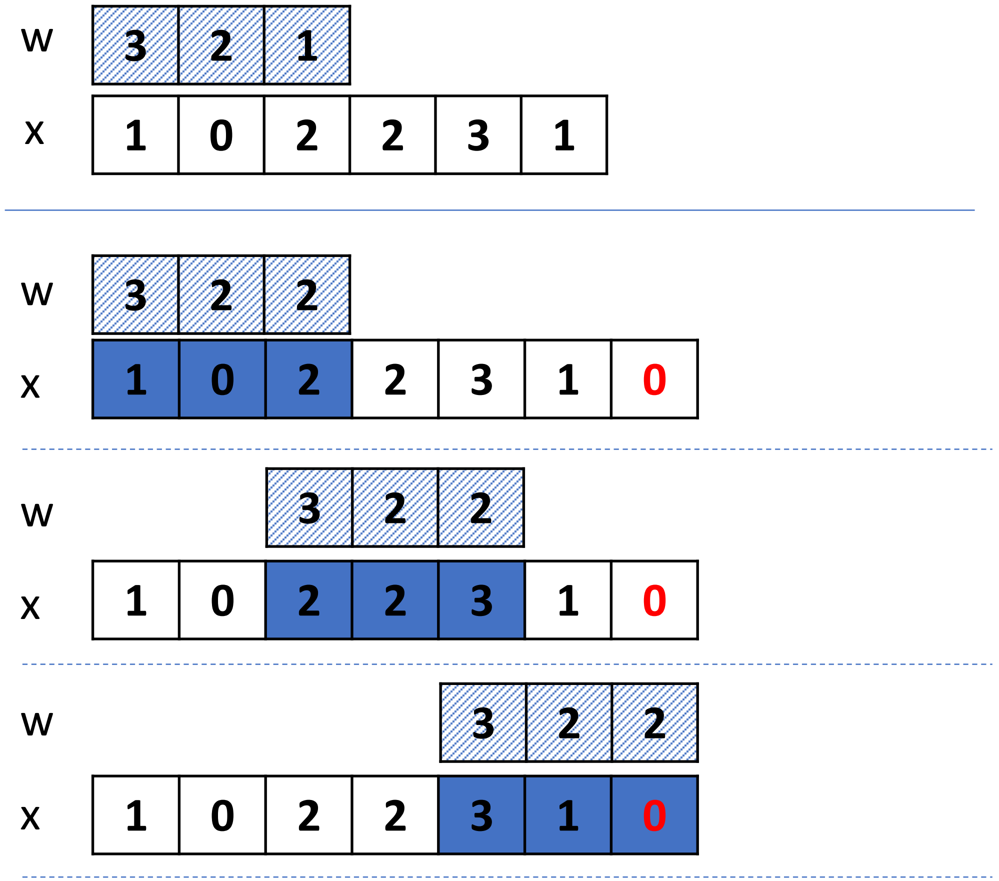
Stride

- $o = \left\lfloor \frac{l+p-k}{s} \right\rfloor + 1$



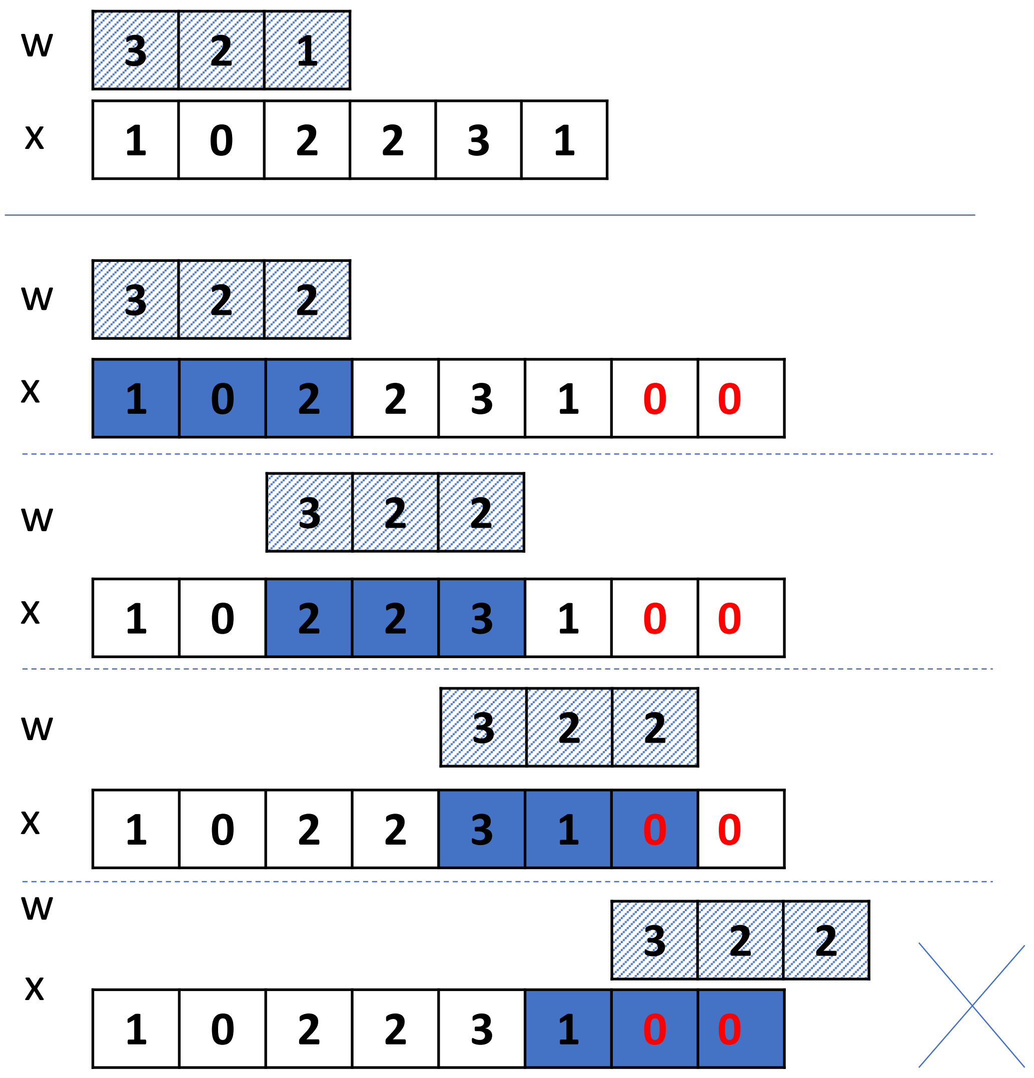
Stride

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



Stride

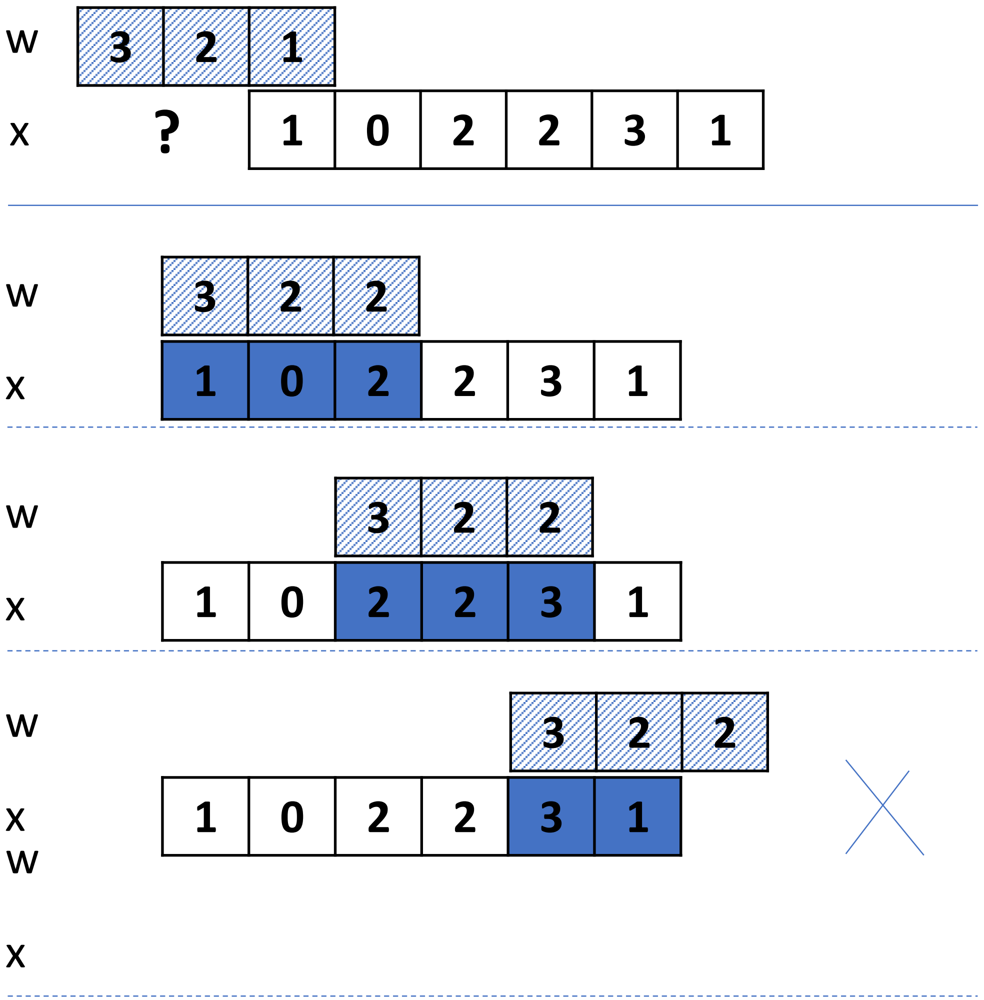
- Not exact matching
  - With padding p (=2)
  - $o = \left\lfloor \frac{l+p-k}{s} \right\rfloor + 1$
  - (6+2-3)/2+1=3



Stride (for Tensorflow)

- Given stride s, what is the padding size if we want Valid padding?

$$p = 0, o = \left\lfloor \frac{l+p-k}{s} \right\rfloor + 1 = 2$$



Stride (for Tensorflow)

- Given stride s, what is the padding size if we want Same padding?

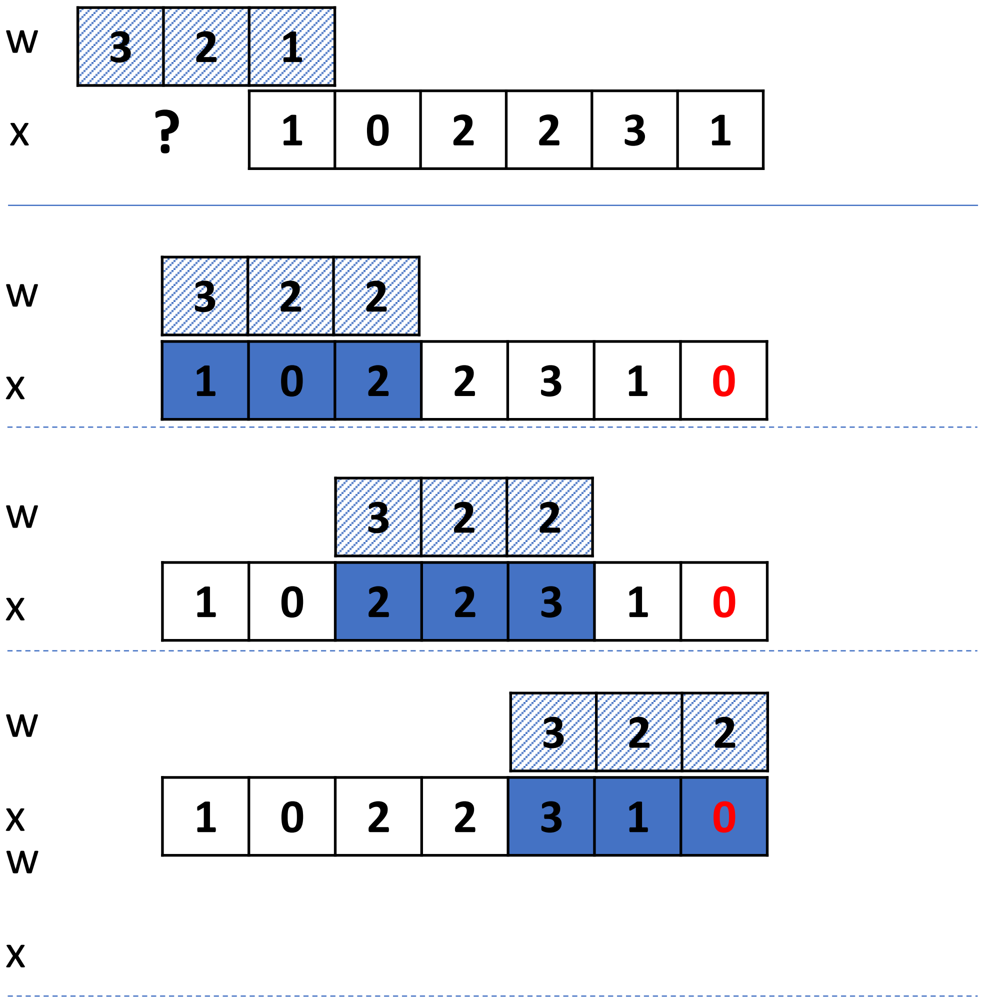
$$o = \left\lceil \frac{l}{s} \right\rceil$$

$$o = \left\lfloor \frac{l+p-k}{s} \right\rfloor + 1$$

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

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

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



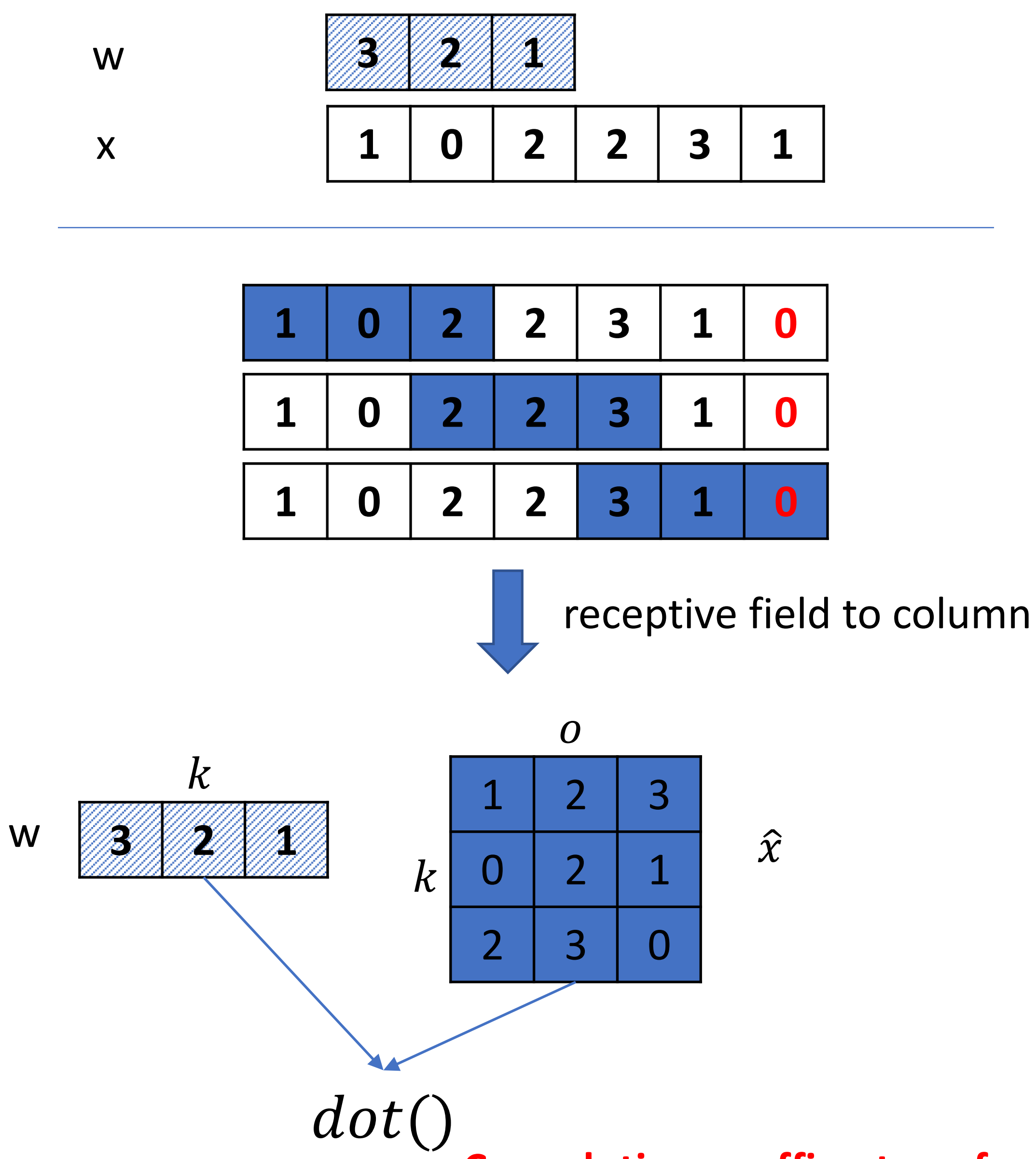
Implementation (fwd)

- For loop

```
for t in range(o):
    for i in range(k):
        y[t] += w[k]*x[t+k]
```

- Receptive field to column

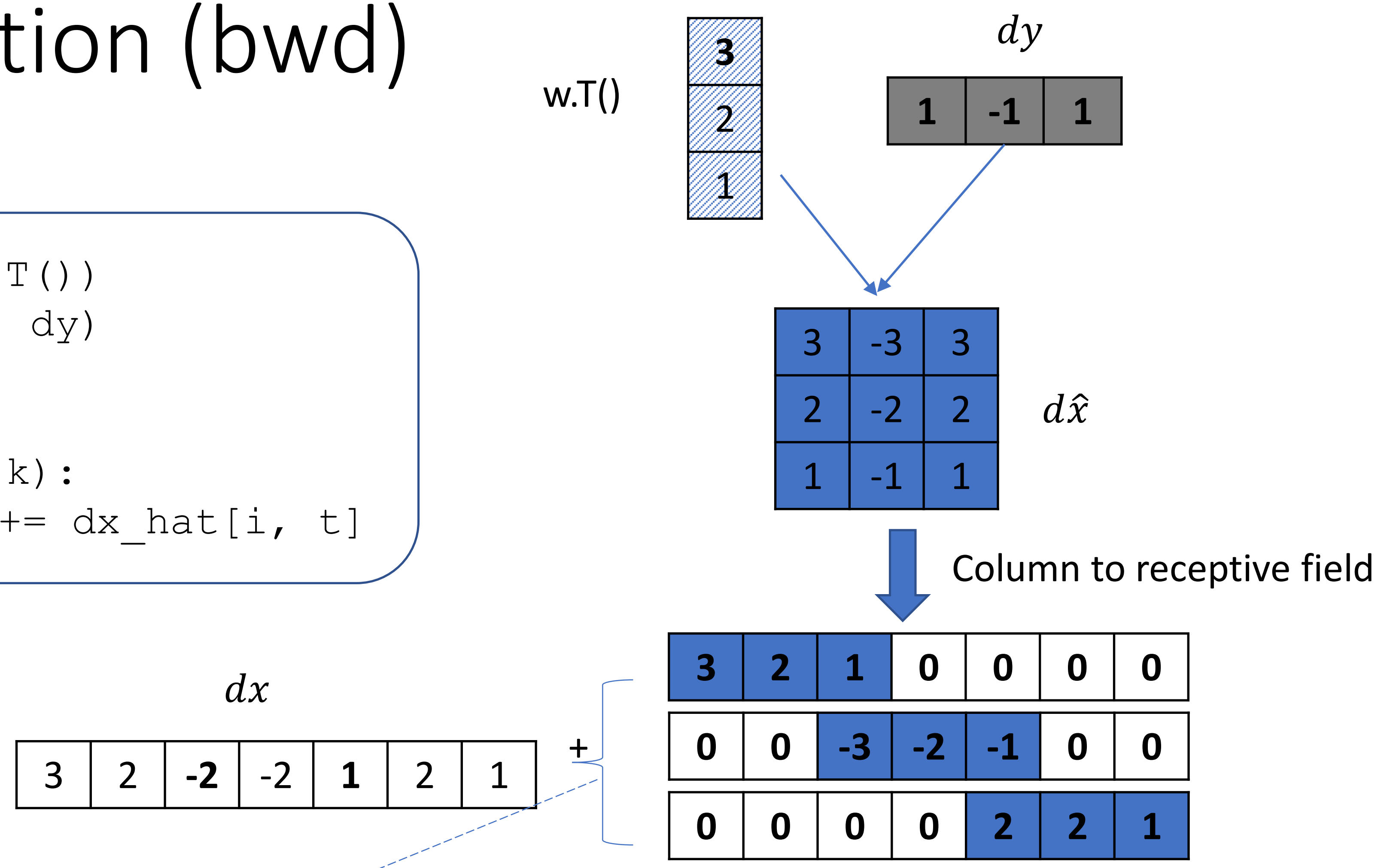
```
for t in range(o):
    for i in range(k):
        x_hat[i, t]= x[t+i]
y = dot(w, x_hat)
```

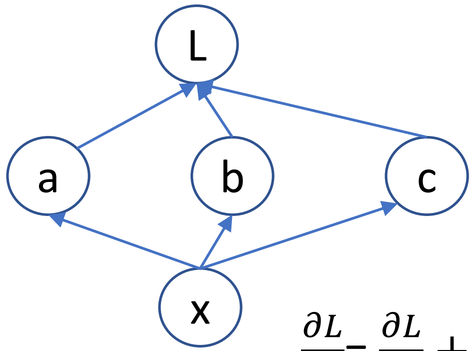


Convolution -> affine transformation

Implementation (bwd)

```
dw = dot(dy, x_hat.T())
dx_hat = dot(w.T(), dy)
set dx to 0s
for t in range(o):
    for i in range(k):
            dx[t+i] += dx_hat[i, t]
```

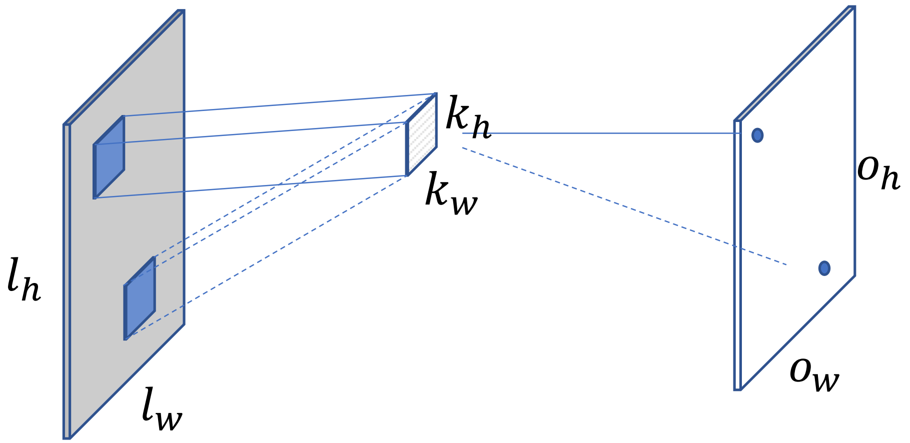




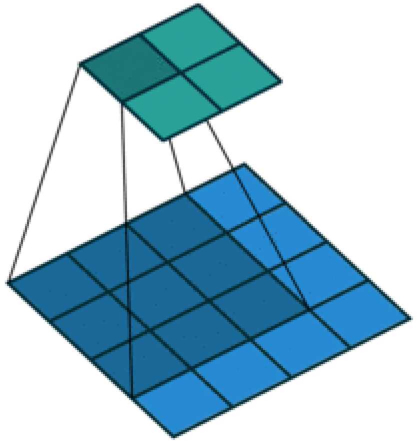
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a} + \frac{\partial L}{\partial b} + \frac{\partial L}{\partial c}$$

Why addition?

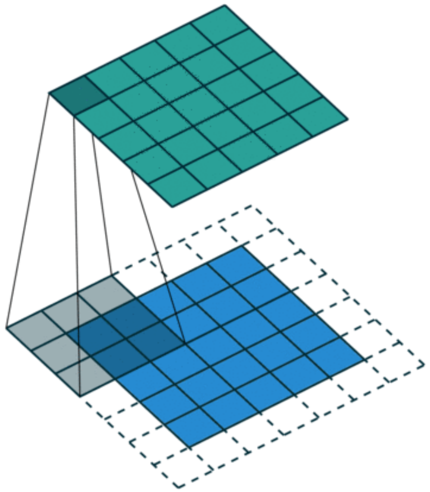
2D Convolution



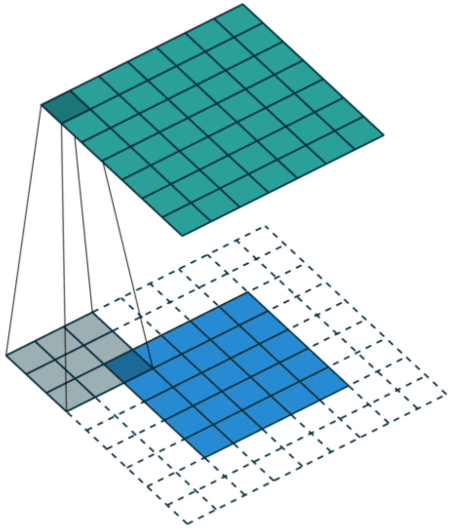
2D Convolution



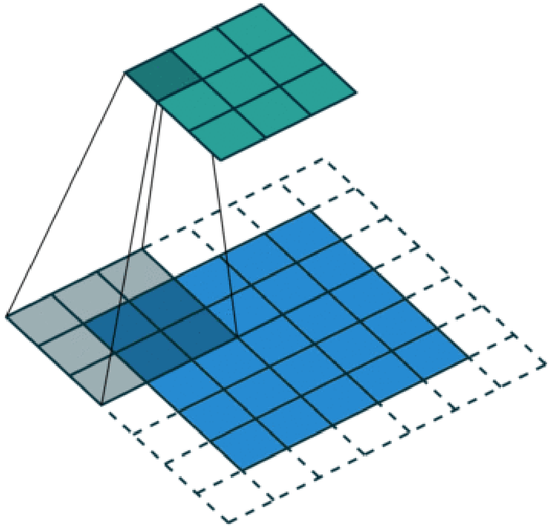
k=3, p=0, s=1 (Valid)



k=3, p=2, s=1 (Same)



k=3, p=4, s=1 (Full)

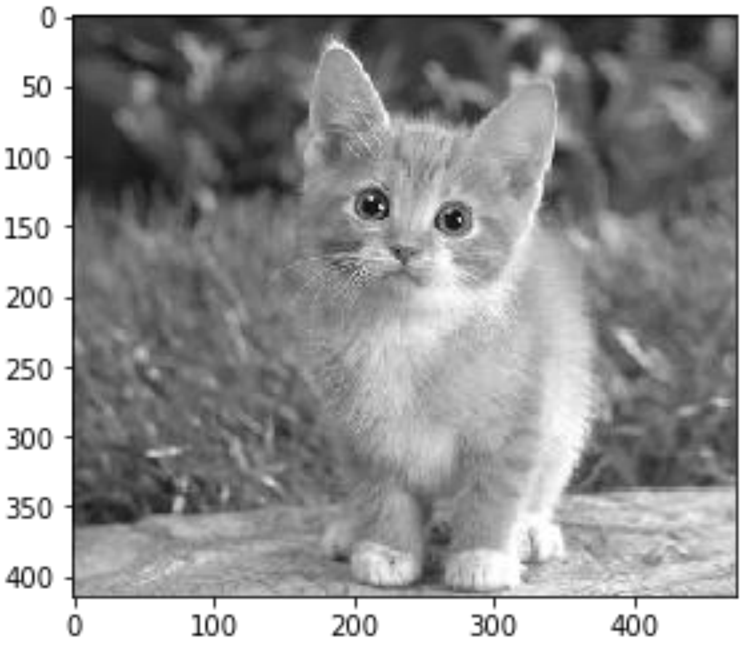


k=3, p=2, s=2

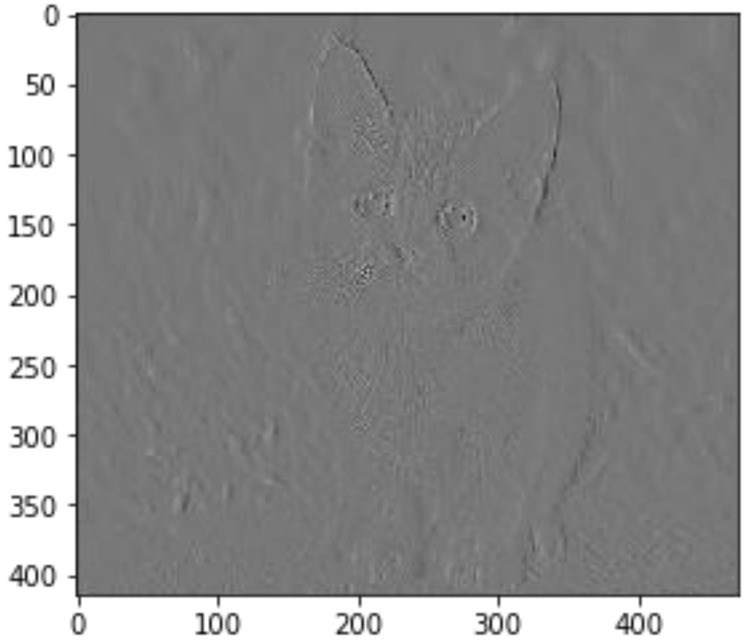
Source: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

Convolution

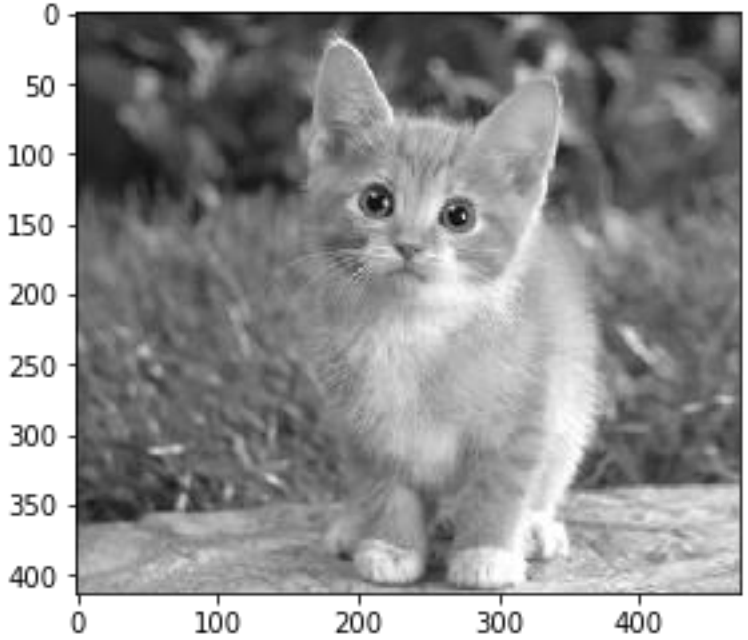
http://setosa.io/ev/image-kernels/



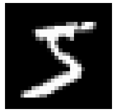
$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

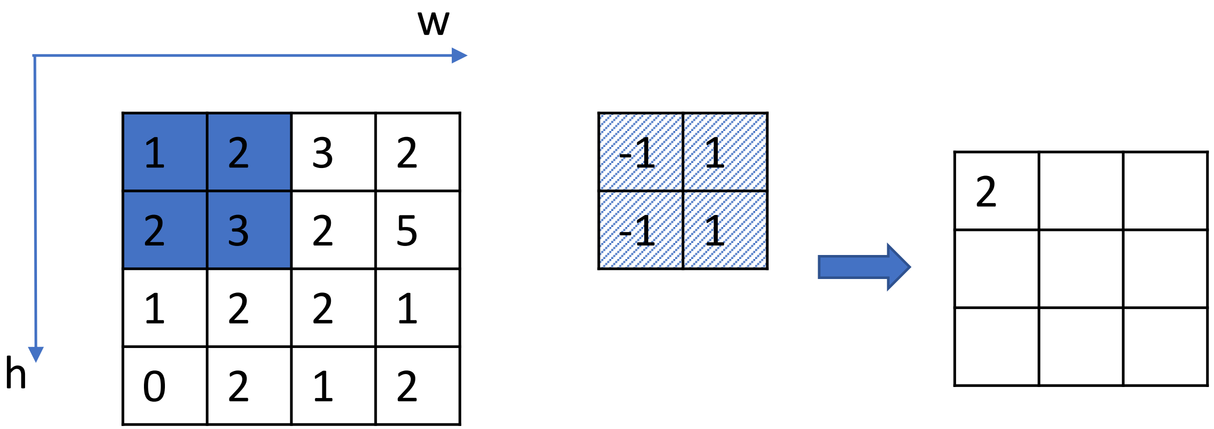


2D convolution



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

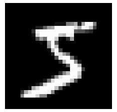
- Input $x \in R^{l_h \times l_w}$
- Kernel $W \in R^{k_h \times k_w}$
- Output y $\in R^{o_h \times o_w}$
- $o_h \leftarrow l_h, k_h, s_h, p_h$
- $o_h \leftarrow l_w, k_w, s_w, p_w$



- $y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{i+a,j+b} \times W_{a,b}$

2D convolution

- Input $x \in R^{l_h \times l_w}$
- Kernel $W \in R^{k_h \times k_w}$
- Output y $\in R^{o_h \times o_w}$
- $o_h \leftarrow l_h, k_h, s_h, p_h$
- $o_h \leftarrow l_w, k_w, s_w, p_w$



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

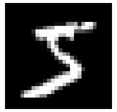
-1	1
-1	1

2	0	

- $y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{i+a,j+b} \times W_{a,b}$

2D convolution

- Input $x \in R^{l_h \times l_w}$
- Kernel $W \in R^{k_h \times k_w}$
- Output $\mathrm{y} \in R^{o_h \times o_w}$
- $o_h \leftarrow l_h, k_h, s_h, p_h$
- $o_h \leftarrow l_w, k_w, s_w, p_w$



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

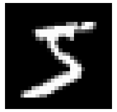
-1	1
-1	1

2	0	2

- $y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{i+a,j+b} \times W_{a,b}$

2D convolution

- Input $x \in R^{l_h \times l_w}$
- Kernel $W \in R^{k_h \times k_w}$
- Output y $\in R^{o_h \times o_w}$
- $o_h \leftarrow l_h, k_h, s_h, p_h$
- $o_h \leftarrow l_w, k_w, s_w, p_w$



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

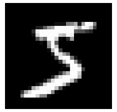
-1	1
-1	1

2	0	2
2		

- $y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{i+a,j+b} \times W_{a,b}$

2D convolution

- Input $x \in R^{l_h \times l_w}$
- Kernel $W \in R^{k_h \times k_w}$
- Output $y \in R^{o_h \times o_w}$
- $o_h \leftarrow l_h, k_h, s_h, p_h$
- $o_h \leftarrow l_w, k_w, s_w, p_w$



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

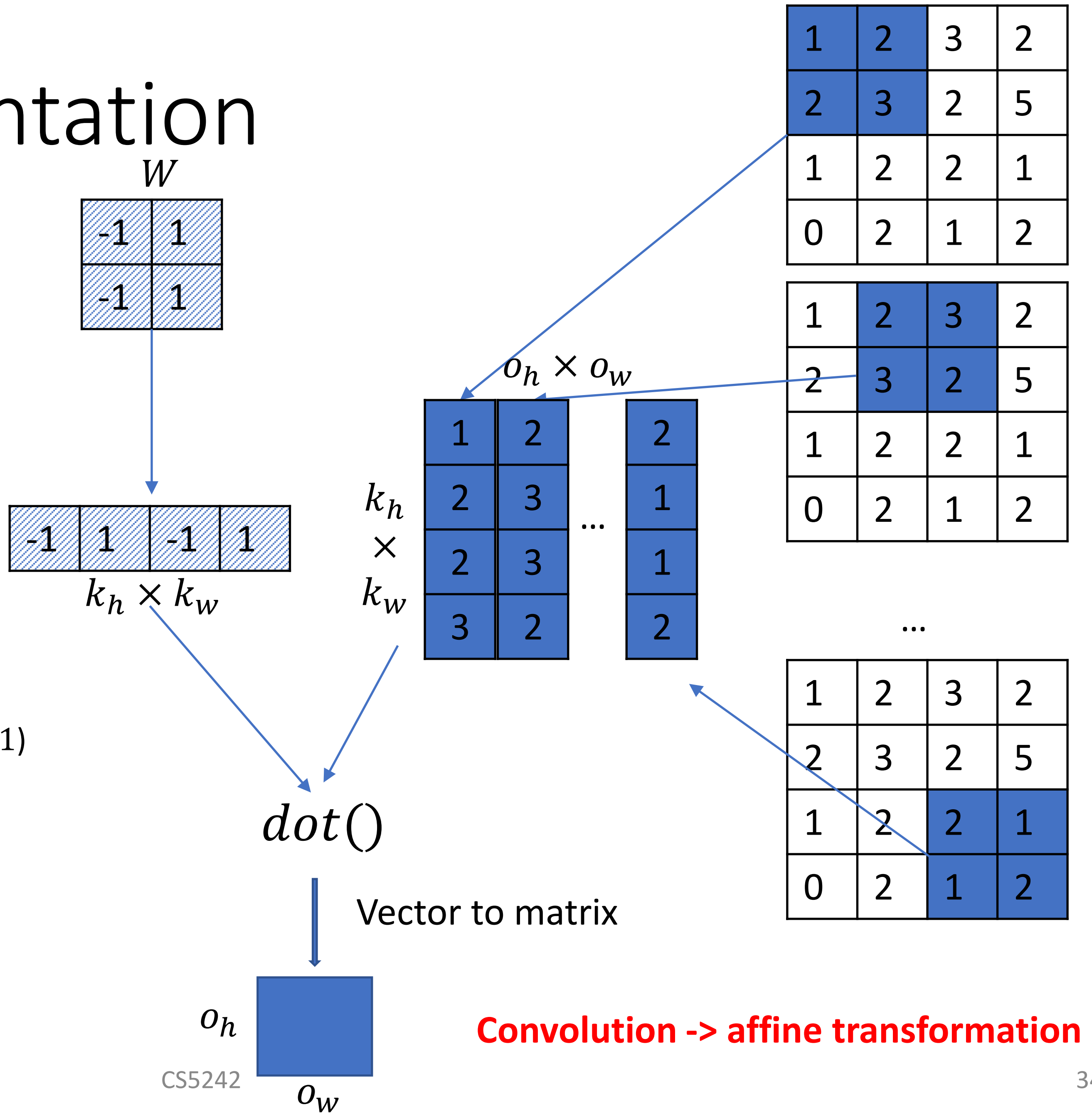
-1	1
-1	1

2	0	2
2	-1	2
3	-1	0

- $y_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{i+a,j+b} \times W_{a,b}$

2D conv implementation

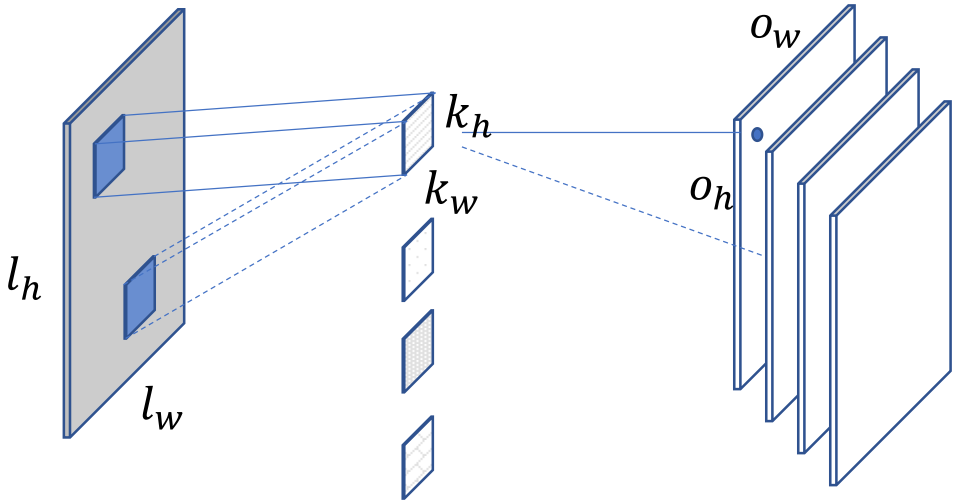
- Img2Col
  - Convert each receptive field into a column
  - All columns construct a matrix
  - Convolution = vector matrix product
- Parameter size
  - $k_h \times k_w$
- Output shape
  - $(o_h, o_w) = (\left\lfloor \frac{l_h + p_h - k_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{l_w + p_w - k_w}{s_w} \right\rfloor + 1)$
- Computation cost
  - $O(\quad k_h \times k_w \quad \times \quad o_h \times o_w)$
  (float multiplication ops, FLOP)



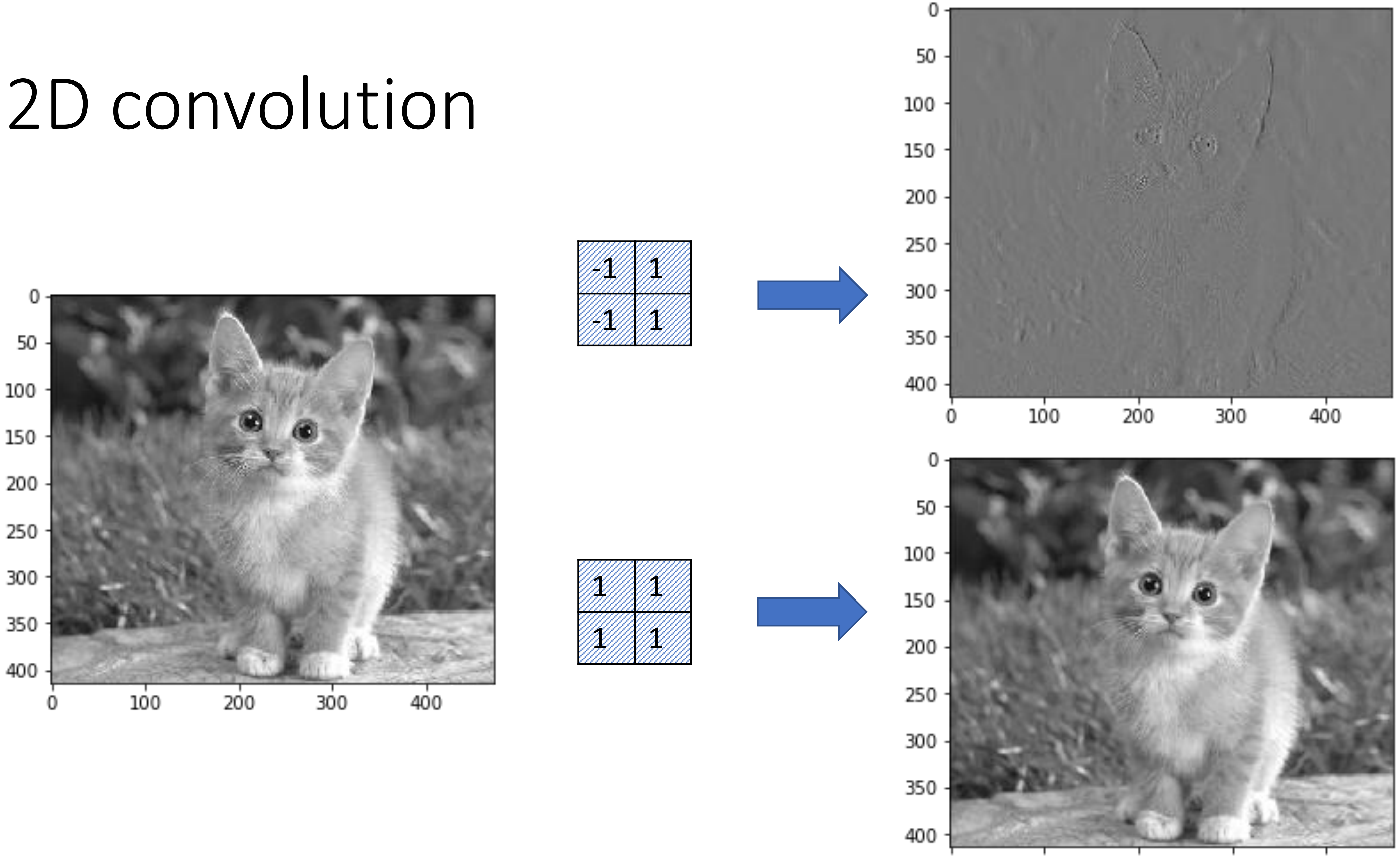
Convolution -> affine transformation

2D Convolution

- Multiple kernels/filters （what is a filter -> feature engineering)

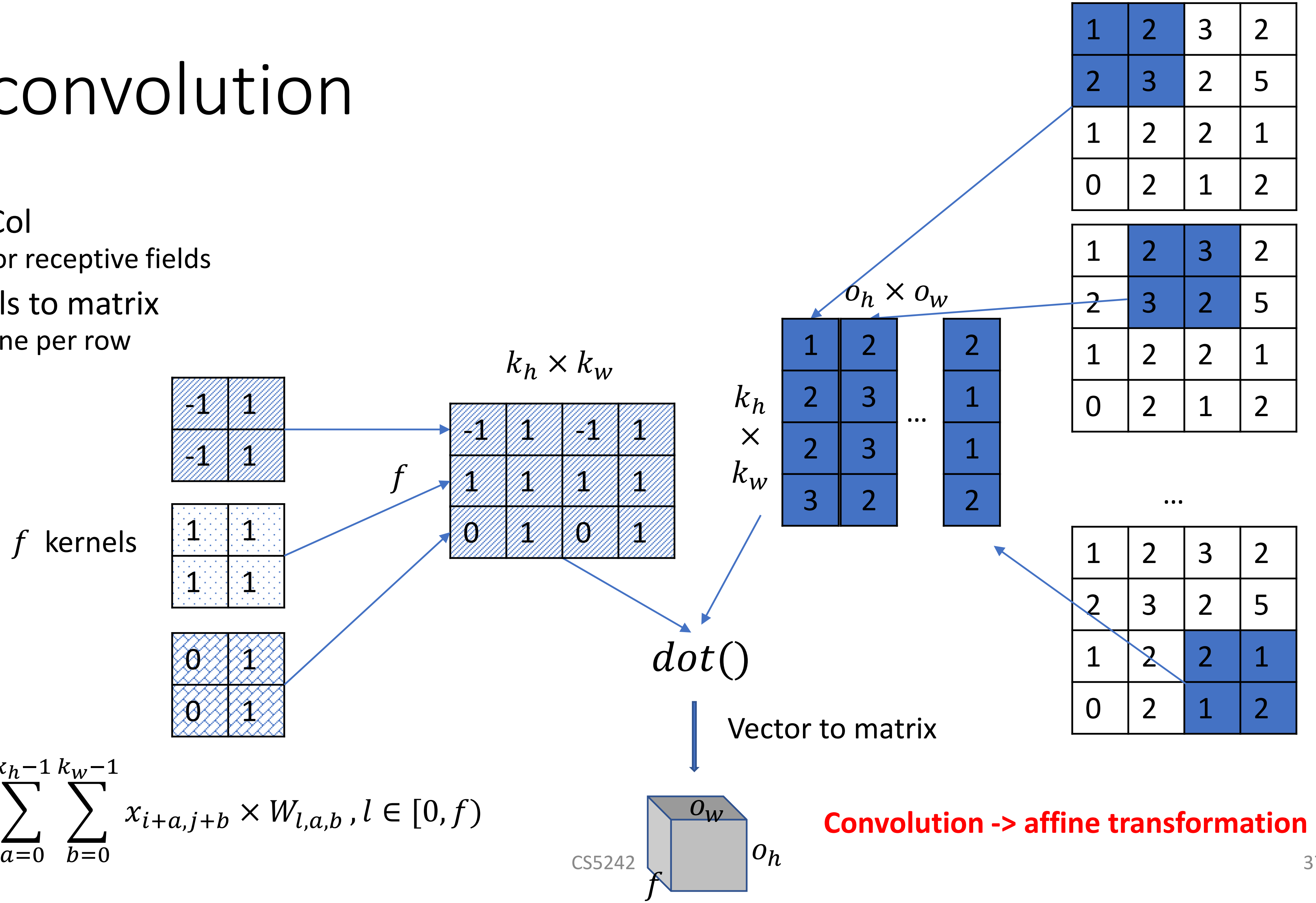


2D convolution



2D convolution

- Img2Col
  - For receptive fields
- Kernels to matrix
  - One per row



$$y_{l,i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{i+a,j+b} \times W_{l,a,b} \, , l \in [0, f)$$

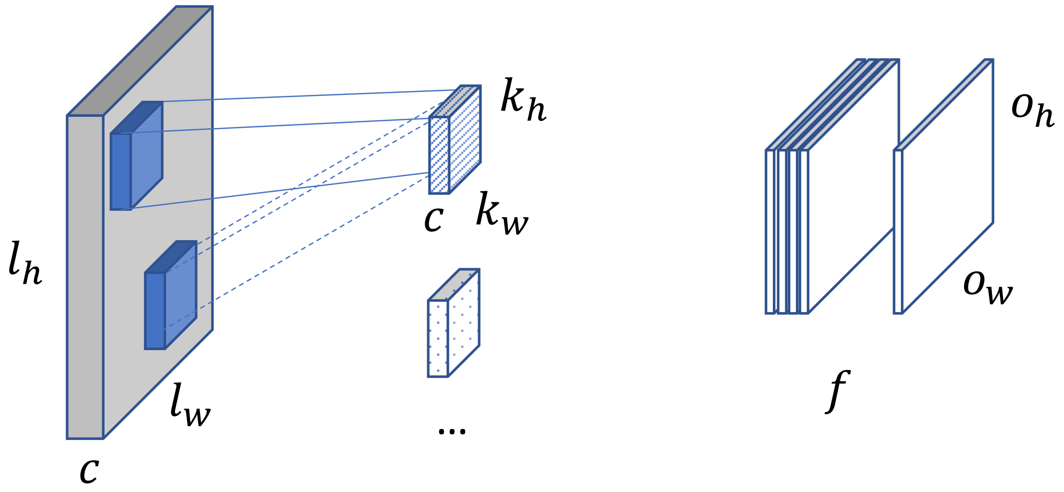
Convolution -> affine transformation

2D convolution

- With multiple kernels (filters)
- Parameter size
  - $f \times k_h \times k_w$
- Output shape
  - $(f, o_h, o_w) = (f, \left\lfloor \frac{l_h + p_h - k_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{l_w + p_w - k_w}{s_w} \right\rfloor + 1)$
- Computation cost
  - $O(\quad f \quad \times k_h \times k_w \quad \times \quad o_h \times o_w)$ (float multiplication ops, FLOP)

2D Convolution

- With multiple input channels and kernels (filters)



f filters/kernels

http://cs231n.github.io/convolutional-networks/
https://www.youtube.com/watch?v=jajksuQW4mc

2D Convolution

- Img2col
  - receptive fields across feature maps are concatenated into to one column

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

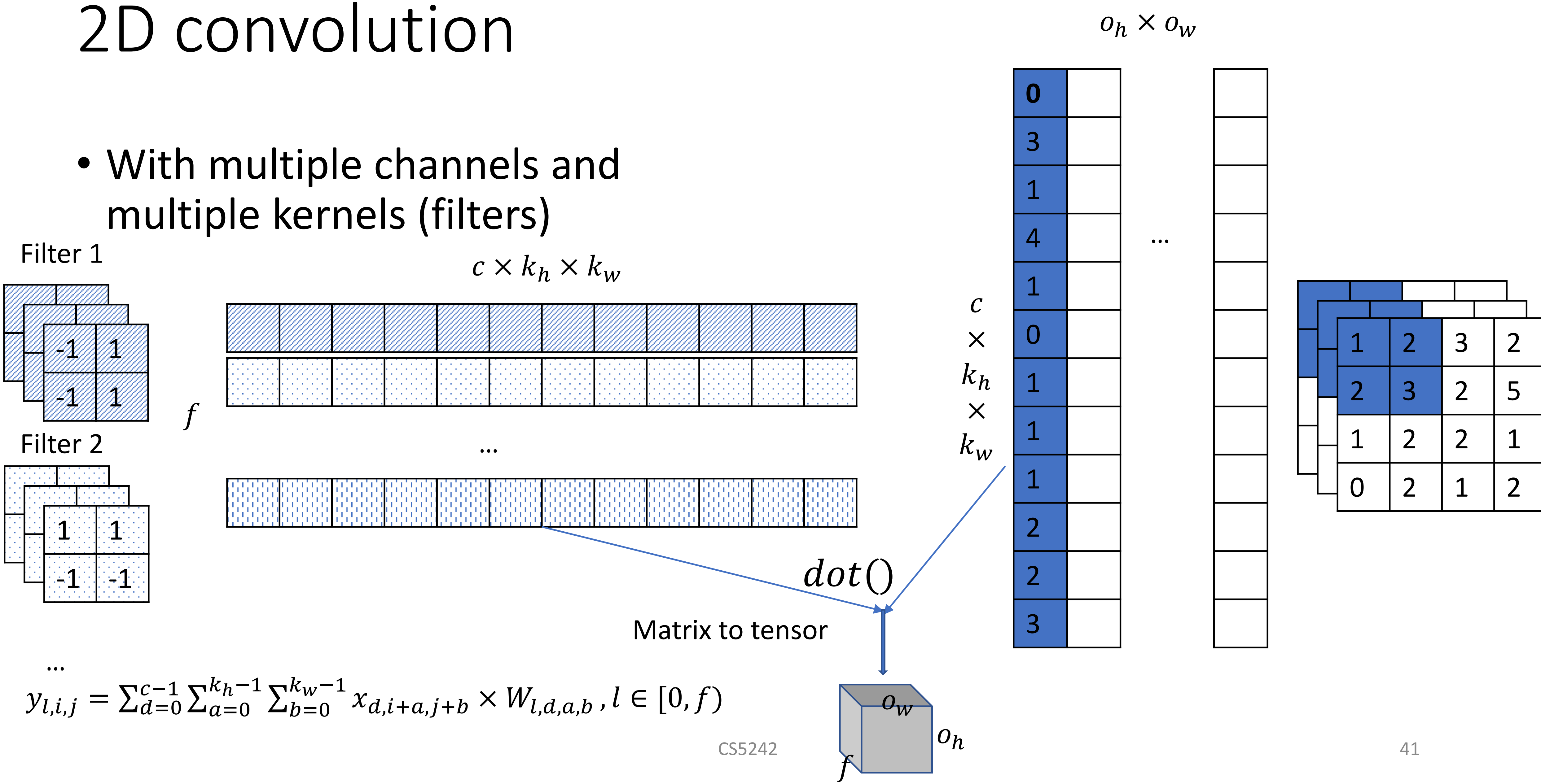
$$c \times k_h \times k_w = 2 \times 2 \times 2$$

$$o_h \times o_w = 2 \times 3$$

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

2D convolution

- With multiple channels and multiple kernels (filters)



$y_{l,i,j} = \sum_{d=0}^{c-1} \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} x_{d,i+a,j+b} \times W_{l,d,a,b}\, , l \in [0, f)$

BP of convolution

- 2D Convolution
  - Create filter matrix
    - W=np.random.rand(f, c*k*k) * 0.01
  - Forward
    - Convert input feature maps X into matrix $\hat{X}$ (img2col)
    - $Y = W\hat{X}$
  - Backward
    - Given $\frac{\partial L}{\partial Y}$
    - Compute $\frac{\partial L}{\partial W}$= $\frac{\partial L}{\partial Y}\hat{X}^T$, $\frac{\partial L}{\partial \hat{X}}$=$W^T\frac{\partial L}{\partial Y}$
    - Column to receptive field
    - Refer to the implementation for 1D convolution

2D convolution

- With multiple channels and kernels (filters)

- Parameter size
  - $f \times (c \times k_h \times k_w)$
- Output shape
  - $(f, o_h, o_w) = (\left\lfloor \frac{l_h + p_h - k_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{l_w + p_w - k_w}{s_w} \right\rfloor + 1)$
- Computation cost
  - $O(f \times \quad c \times k_h \times k_w \quad \times \quad o_h \times o_w)$ (float multiplication ops)

Pooling

- 1D $y_t = OP_{i\in[0,k)}(x_{t+i})$
- 2D $y_{i,j} = OP_{a\in[0,k_w),b\in[0,k_h)} x_{i+a,j+b}$
- Max pooling
  - OP = Max
- Average pooling
  - OP = Average

Pooling

- Reduce 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

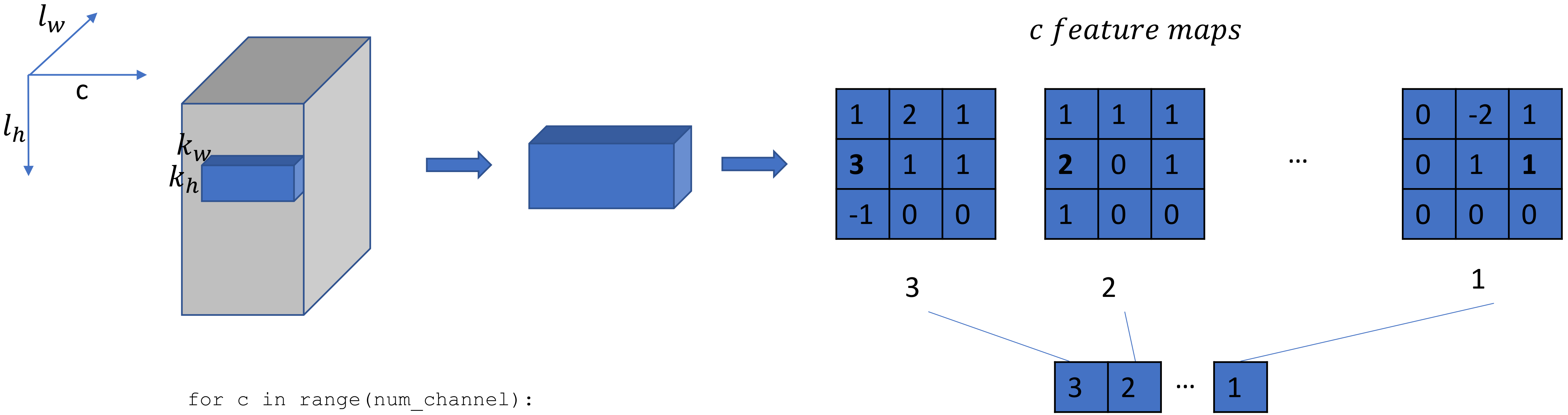
flip

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

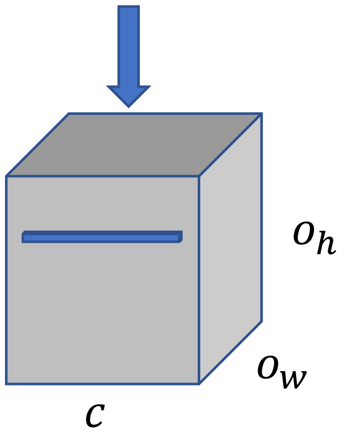
Rotate 90·

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

Max Pooling



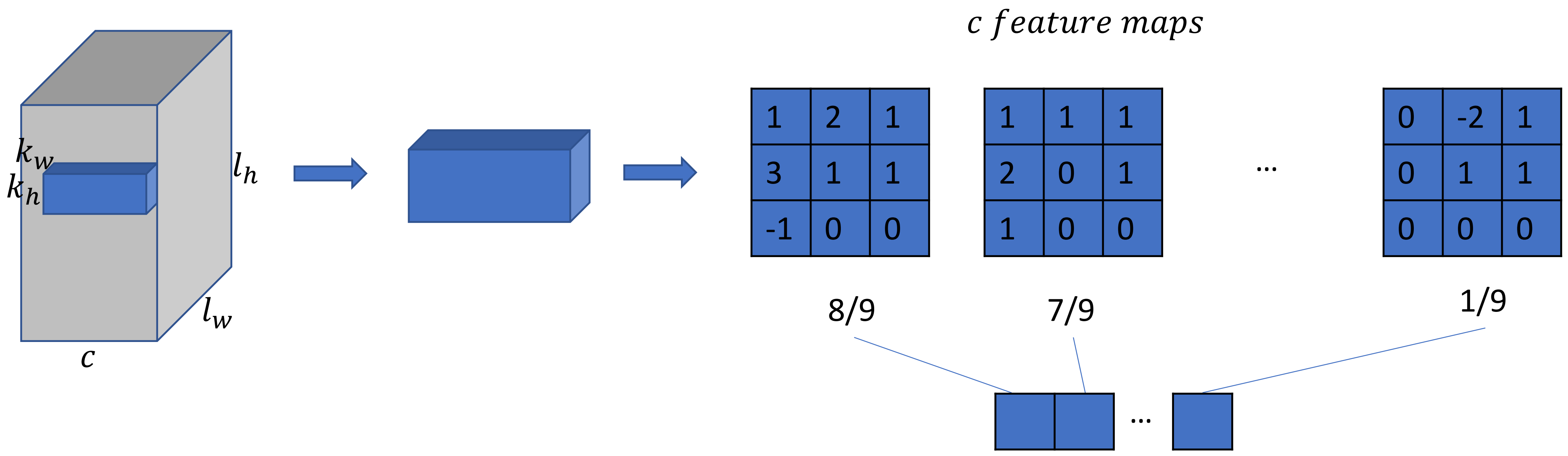

```
for c in range(num_channel):
    for i in range(ox):
        for j in range(oy):
            y[i, j] = -inf
            for p in range(k):
                ii = i + p
                for q in range(k):
                    jj = j + q
                    if y[i, j] < x[ii, jj]
                        y[i, j] = x[ii, jj]
                        loc[i, j] = (ii, jj)
```

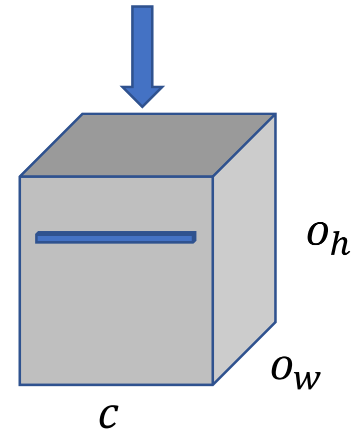


Padding?
Stride?

Backward:
Set 0 for all dx ;
Copy dy[i,j] to dx at position loc[i,j] (accumulation);

Average Pooling





Equivalent to convolution with kernel values = 1/kernel_size