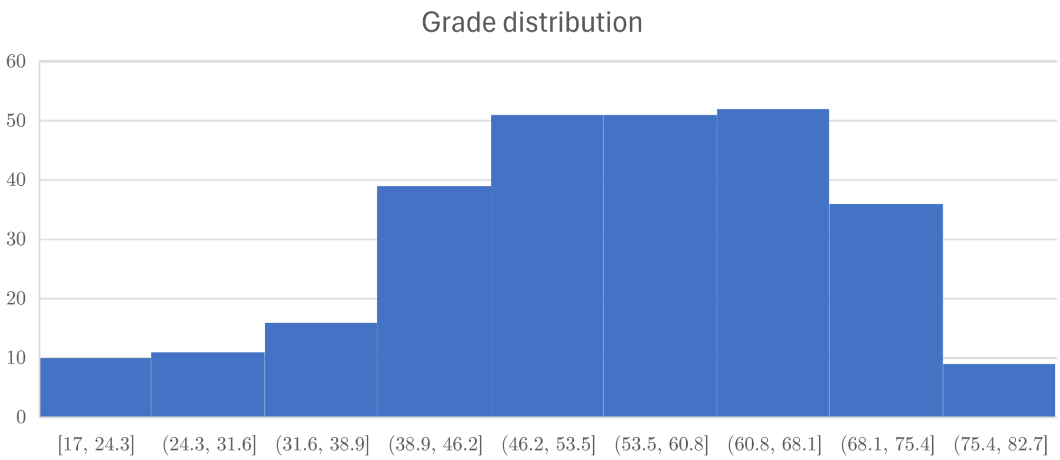
Lecture 15 – Convolutional neural networks and intro to sequence models

Prof. Maggie Makar

Announcements

- Project 2 has been released
- HW2 solutions and grades have been released
- "Quiz" (midterm reflections) is released
- Midterm grades will be released today



Percentile	**Grade**
90	70.7
80	66
70	62.5
60	58.8
50	55
40	52
30	47.5
20	43.1
10	36.4
5	27.2

Midterm announcements

27/85

- If you've scored 27 or less, please sign up for a slot in my office hours on Monday from 10:35-11:45 or 2:30-3:30
  using this link: https://bit.ly/profm-mt-oh

- If you scored higher than that but would like to chat about the MT, show up to my office hours 3:30-4:30 on Monday

A0 B0 C0 D0

Justify.

- All OH are in my office BBB 3769

- Moving forward: notes about the final
  - Check the released solutions ***not*** your own answers

Class outline

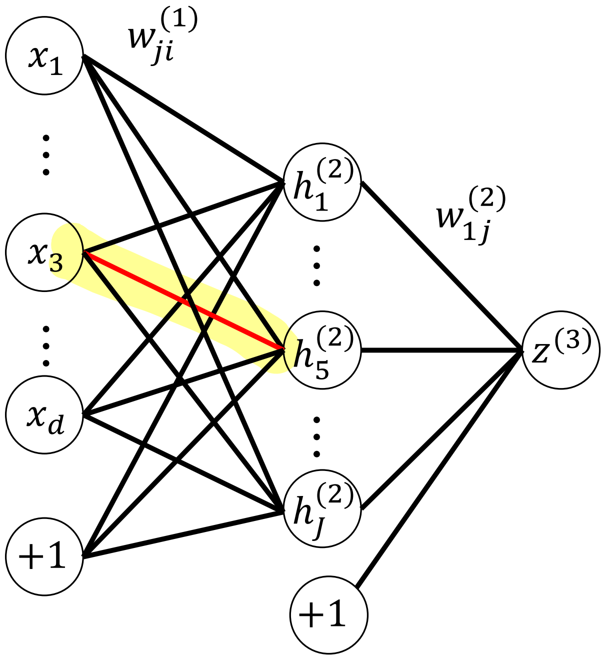
- A note on Backprop
- CNNs:
  - Recap: Motivating CNNs
  - Convolution and padding
  - Max Pooling
  - Final layer
  - Some “famous” CNNs
- RNNs:
  - Motivating RNNs

Why is the forward pass important?

- During the $k + 1$ iteration:
  - Sample some $\bar{x}, y$ from the data
  - Make a prediction $\hat{y} = h(\bar{x}; \bar{\theta}^{(k)})$ ← Forward Pass
  - Measure the loss of the prediction $\hat{y}$ with respect to the true label y

    Call that $\text{Loss}(\mathrm{y}, h(\bar{x}; \bar{\theta}^{(k)}))$
  - Go through each node in reverse order to figure out the contribution of each node to the loss
  - To get $\bar{\theta}^{(k+1)}$, change the values of the parameters to reduce error (SGD step)

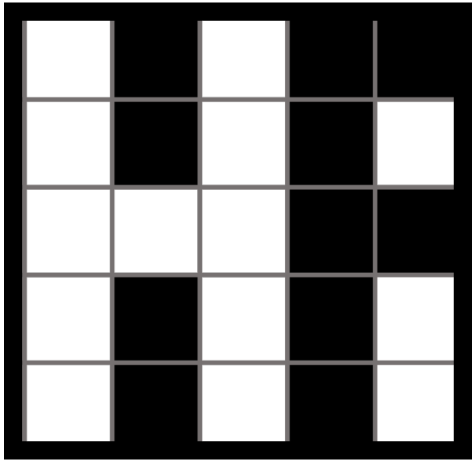
Why is the forward pass important?



$$\frac{\partial \mathrm{Loss}(y, z^{(3)})}{\partial w_{53}^{(1)}} = \frac{\partial \mathrm{Loss}(y, z^{(3)})}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial h_5^{(2)}} \cdot \frac{\partial h_5^{(2)}}{\partial z_5^{(2)}} \cdot \frac{\partial z_5^{(2)}}{\partial w_{53}^{(1)}}$$

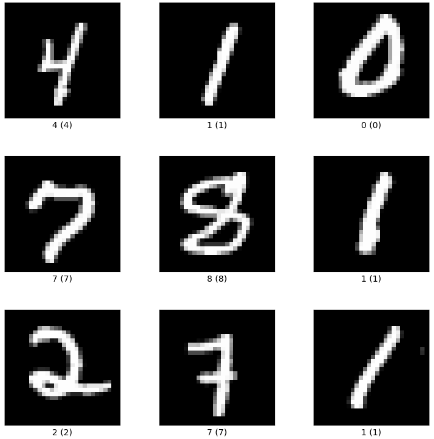
$$= -y[\![1 - yz^{(3)} > 0]\!] \cdot w_{15}^{(2)} \cdot 1[\![z_5^{(2)} > 0]\!] \cdot x_3$$

Images as inputs

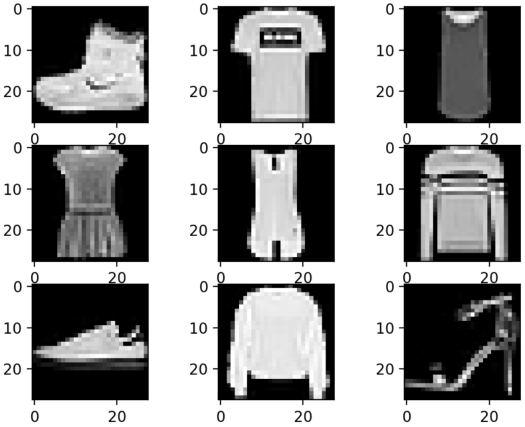


1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

- Start with greyscale images
- Each pixel take a value between 0 and 1
- 0: black, 1: white

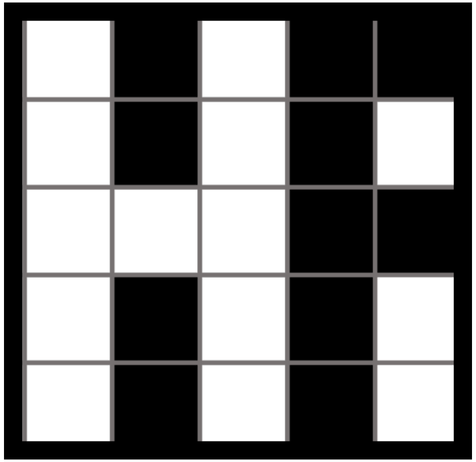


MNIST



Fashion MNIST

Images as inputs



x_1	x_2	x_3	x_4	x_5
x_6	x_7	x_8	x_9	x_{10}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}

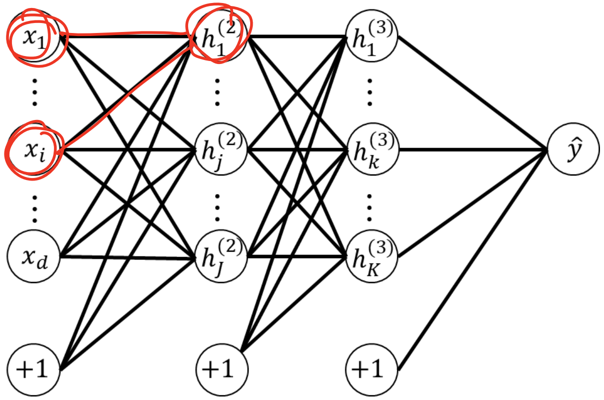
- Start with greyscale images
- Each pixel take a value between 0 and 1
- 0: black, 1: white

$\bar{x} = [x_1, \dots x_d]^T$

$\bar{x} = [x_1, \dots x_5, x_6 \dots x_{10}, \dots x_{25}]^T$

Can we use our fully connected NN here? Yes, but...

- There is exploitable structure in an image
  - Spatial locality

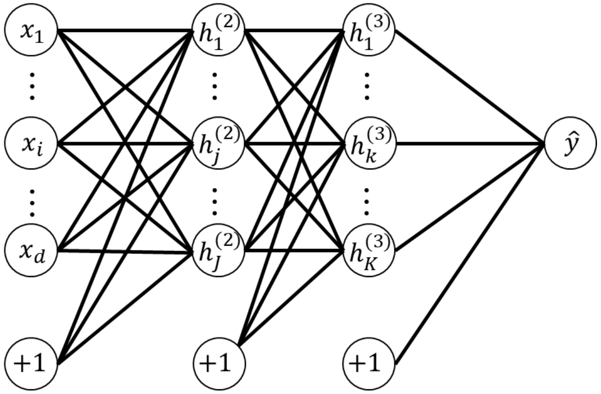


$\bar{x} = [\text{age}, \text{hx}, \text{BMI}]$

$\bar{x}' = [\text{age}, \text{BMI}, \text{hx}]$

Can we use our fully connected NN here? Yes, but...

- There is exploitable structure in an image
  - Spatial locality
  - Translation invariance



- Convolutional neural networks (CNNs)
  - Different type of layers
    - Convolutional layers ←
    - ReLU
    - Max Pooling

Conv Block

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

Convolve the filter with the image: "slide" the filter over the image spatially. Compute dot products as you slide.

2D example: Convolutional layer

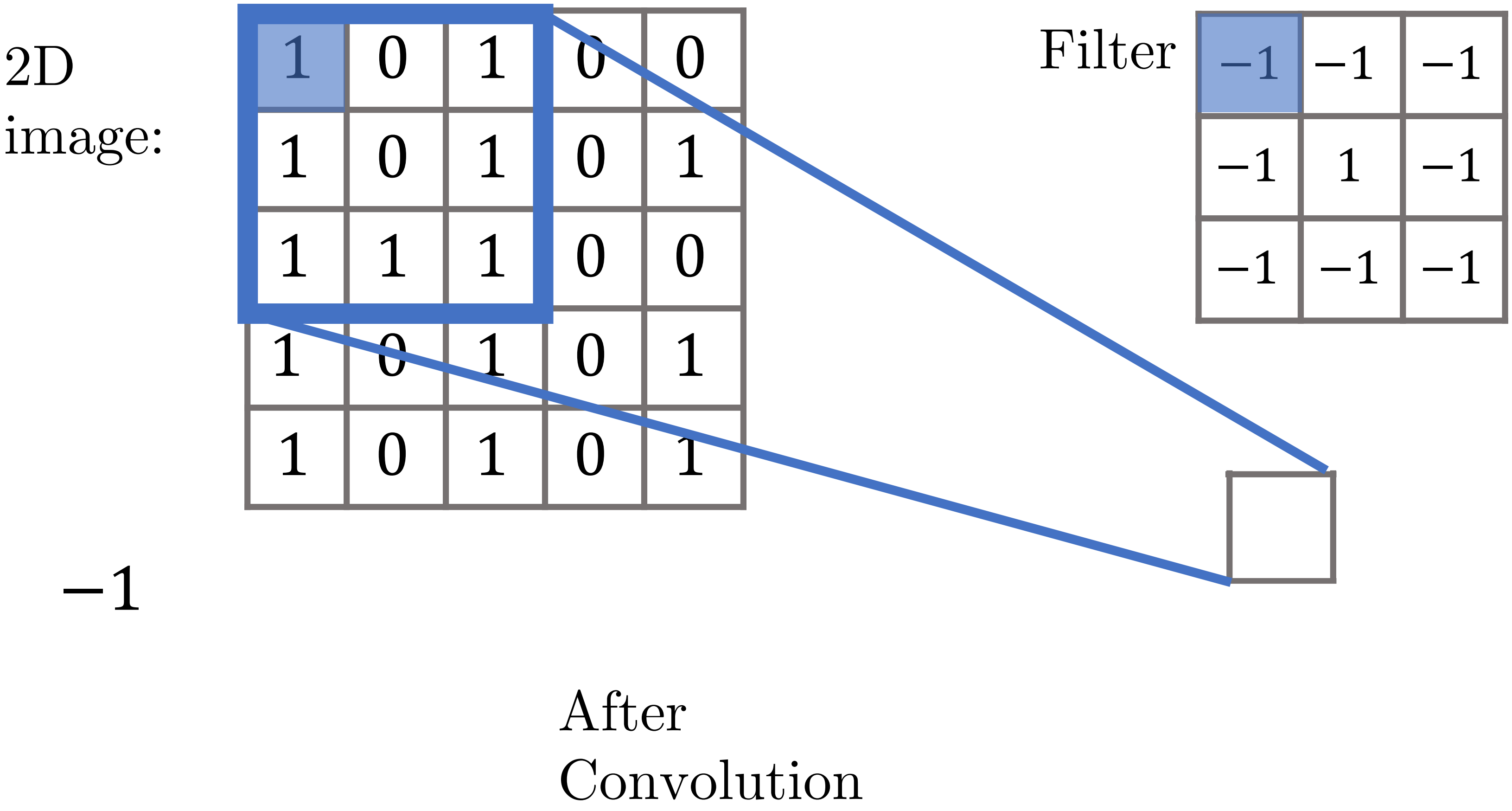
2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

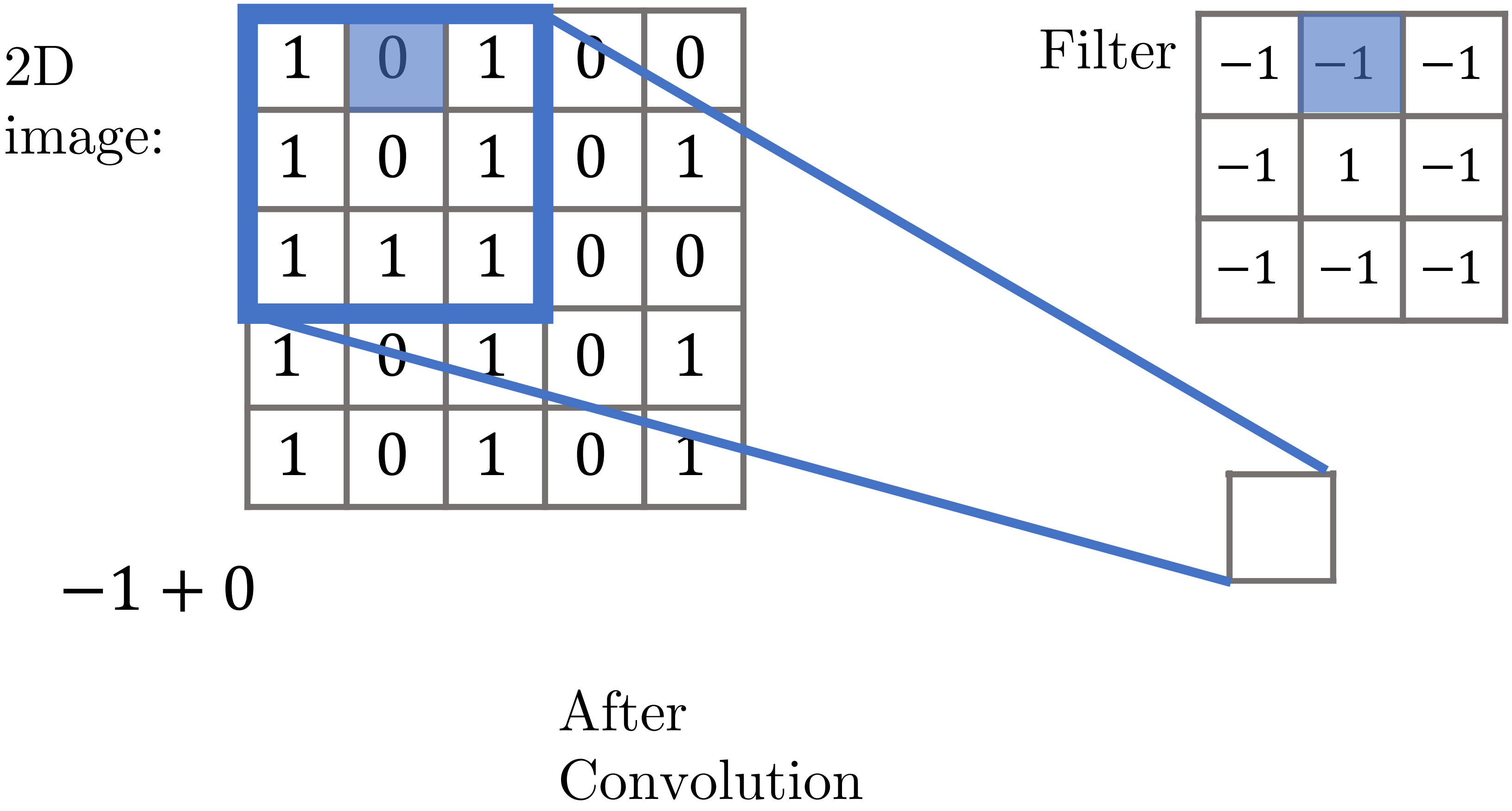
Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

2D example: Convolutional layer



2D example: Convolutional layer



2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

$-1 + 0 \ - 1$

After Convolution

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

$-1 + 0 \; - 1$

-1

After Convolution

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

−1	−1	−1
−1	1	−1
−1	−1	−1

$$-1 + 0 \; - 1$$
$$-1 + 0$$

After Convolution

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

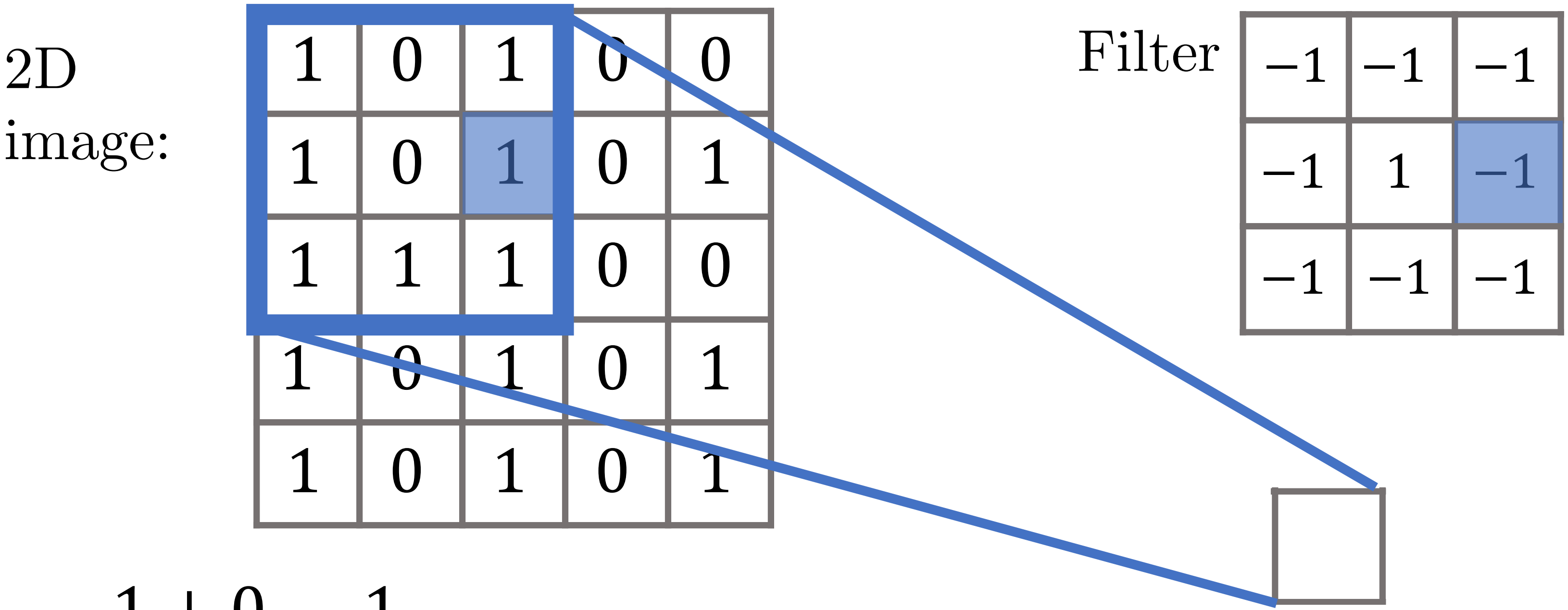
Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

$$-1 + 0 \ - 1$$
$$-1 + 0 - 1$$

After Convolution

2D example: Convolutional layer



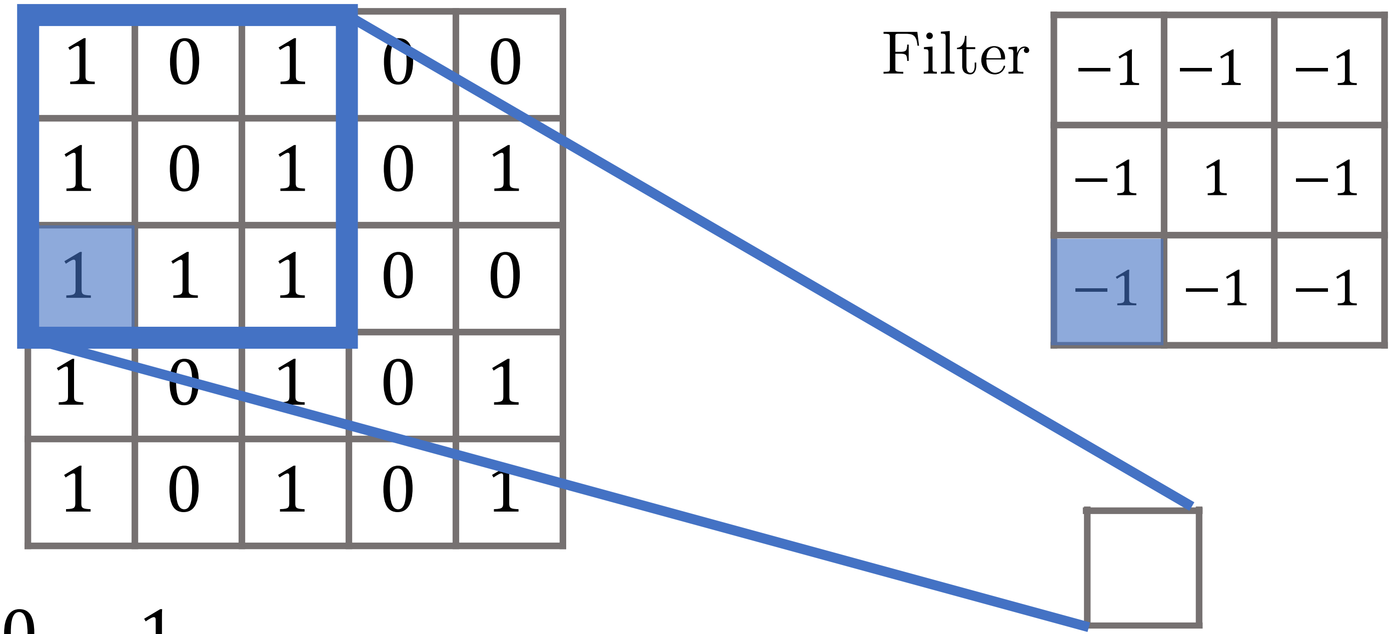
$$-1 + 0 \ - 1$$
$$-1 + 0 - 1$$

After Convolution

2D example: Convolutional layer

2D image:

Filter



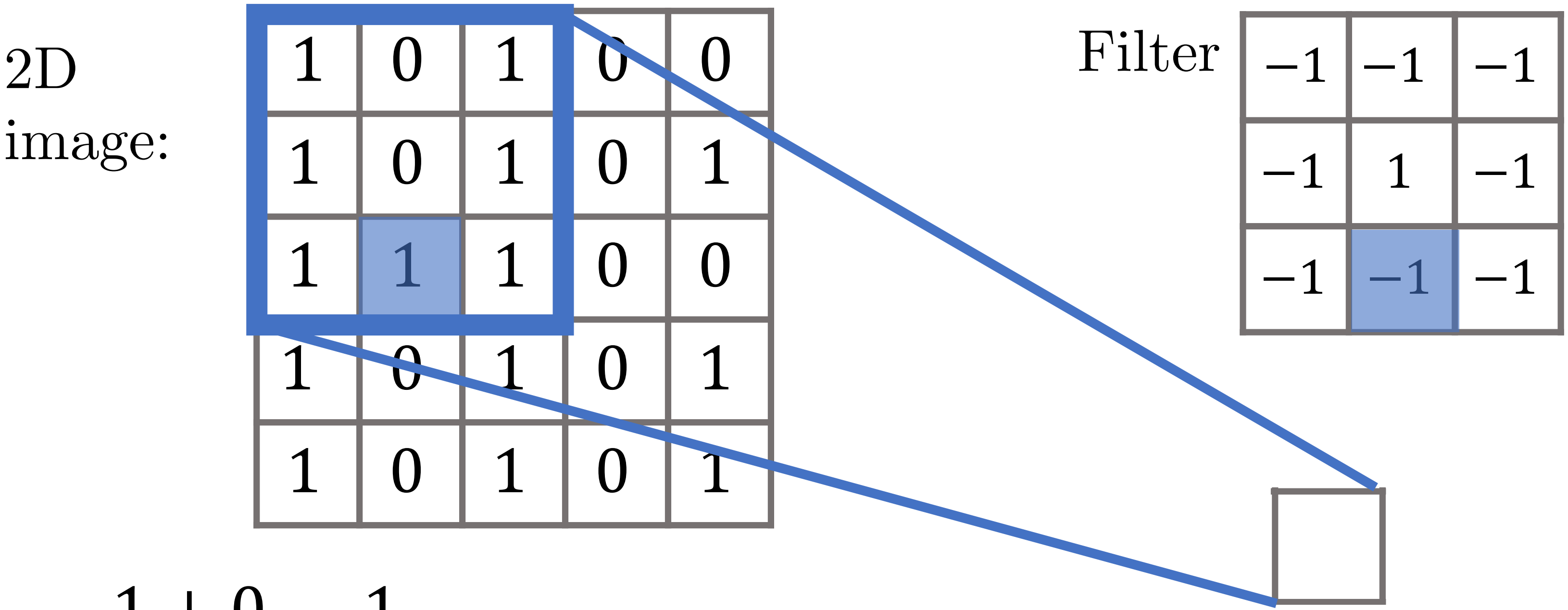
$-1 + 0 \; - 1$

$-1 + 0 - 1$

-1

After Convolution

2D example: Convolutional layer



$-1 + 0 \ - 1$

$-1 + 0 - 1$

$-1 \ - 1$

After Convolution

2D example: Convolutional layer

[1,0,1,10,1,1,1,1]

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

-7

$$-1 + 0 - 1$$
$$-1 + 0 - 1$$
$$-1 - 1 - 1$$

After Convolution

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Stride length $= 1$

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

-7	-2

After Convolution

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

-7	-2	

After Convolution

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

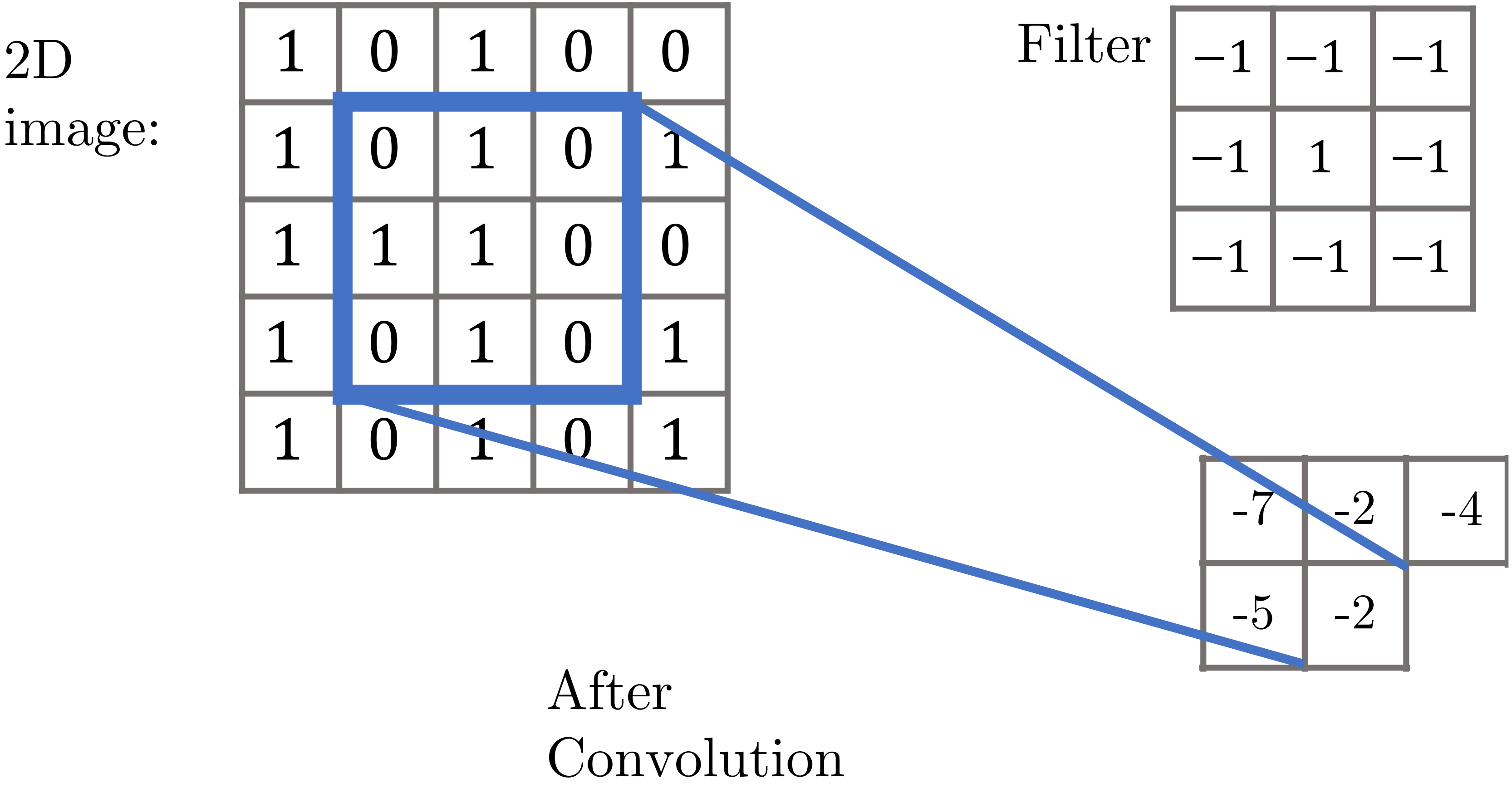
-1	-1	-1
-1	1	-1
-1	-1	-1

Stride length $= 1$

-7	-2	-4
-5		

After Convolution

2D example: Convolutional layer



2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

-7	-2	-4
-5	-2	-5

After Convolution

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

After Convolution

-7	-2	-4
-5	-2	-5
-7		

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

After Convolution

-7	-2	-4
-5	-2	-5
-7	-2	

2D example: Convolutional layer

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

After Convolution

-7	-2	-4
-5	-2	-5
-7	-2	-5

2D example: Convolutional layer with stride length 2

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

After Convolution

-7	-4
-7	-5

2D example: Convolutional layer with stride length 1

2D image:

1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

Shrinking leads to:
- Depth constraint of NN
- Underrepresents info from the edges

After Convolution

-7	-2	-4
-5	-2	-5
-7	-2	-5

2D example: (zero) padding

2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

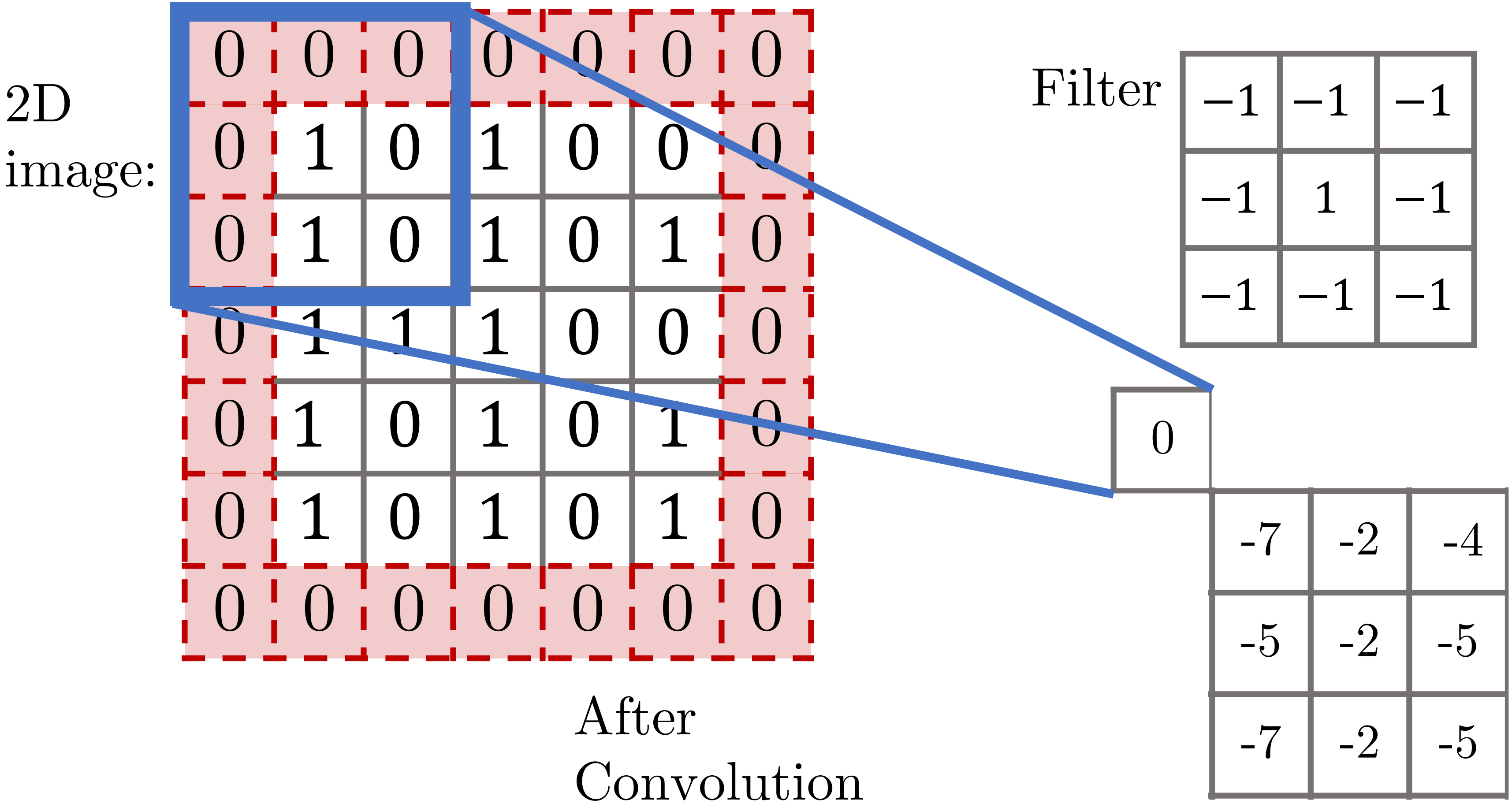
Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

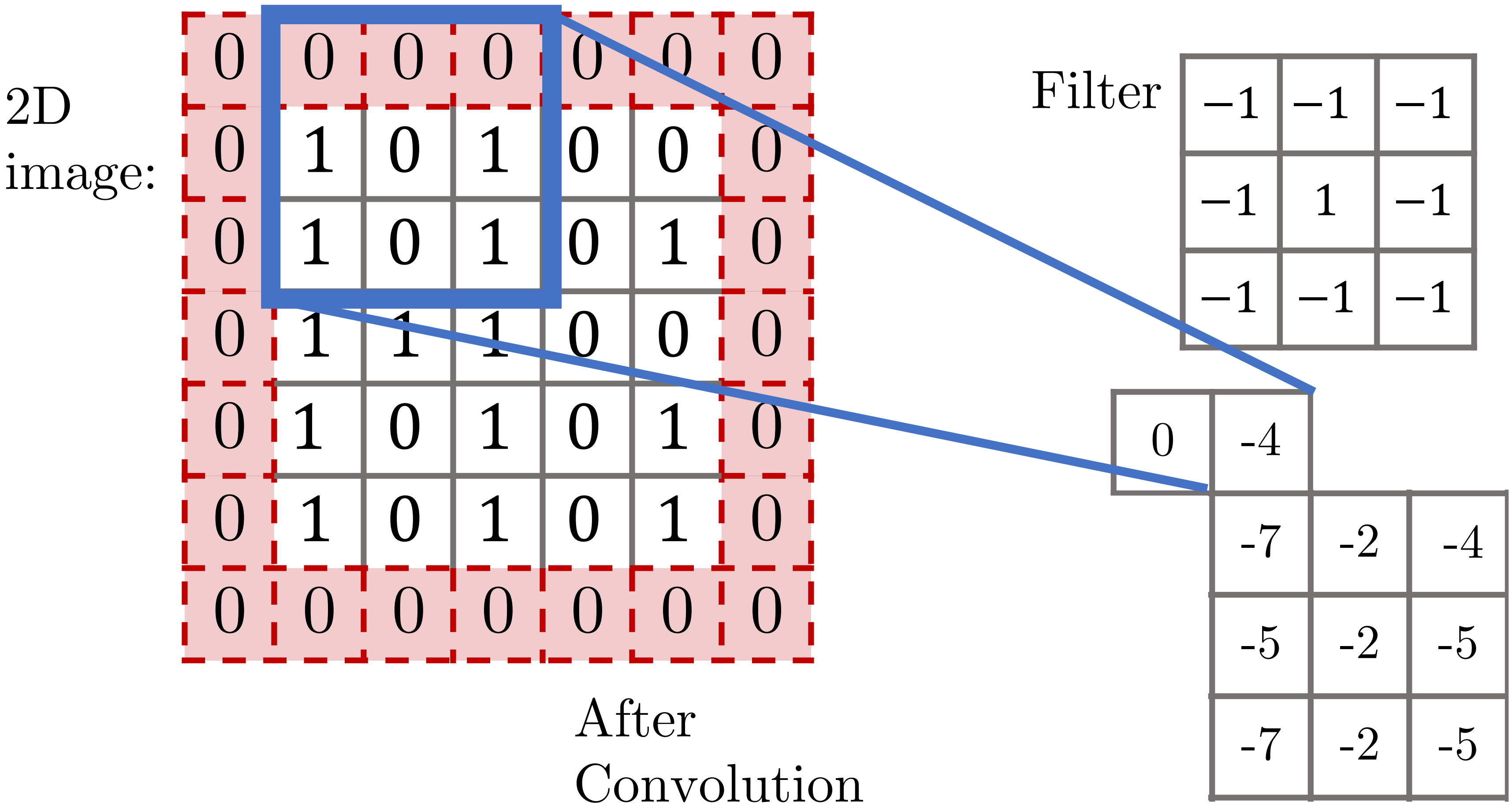
After Convolution

-7	-2	-4
-5	-2	-5
-7	-2	-5

2D example: (zero) padding



2D example: (zero) padding



2D example: (zero) padding

2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

After Convolution

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

2D example: (zero) padding

2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

After Convolution

With bias 3 =

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

$+ 3$

2D example: (zero) padding

2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

After Convolution

With bias 3 =

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

2D example: (zero) padding

2D image:

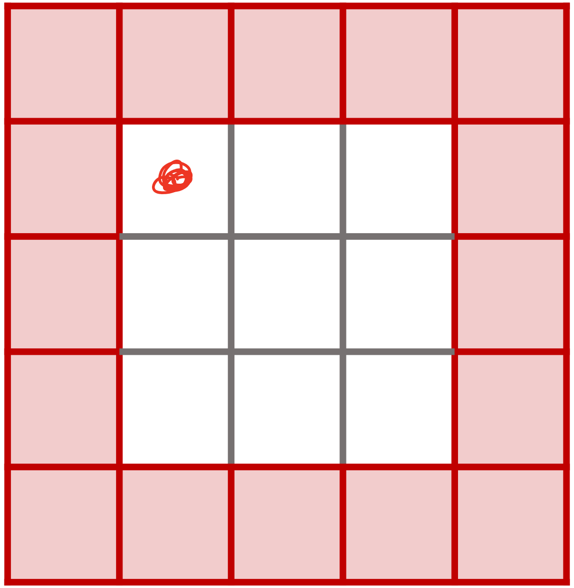
0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

Filter

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

With bias b

After Convolution



$$= w_1 + w_3 + w_4 + w_6 + w_7 + w_8 + w_9 + b$$

2D example: (zero) padding

2D image:

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0

Filter

-1	-1	-1
-1	1	-1
-1	-1	-1

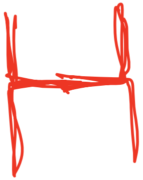
After Convolution

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

2D example: Applying the activation function

Padded
2D image

0	0	0	0	0	0	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	1	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	0	1	0
0	0	0	0	0	0	0



ReLU(0) = max(0,0) = 0

ReLU(−2) = 0

After
Convolution

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

After convolution
and ReLU

0	0	0	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

TL;DPA:

1. Convolutions allow us to learn “local” information in the image

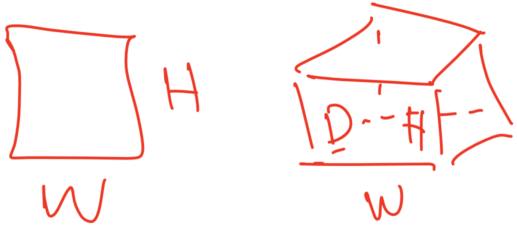
2. Core idea: we apply a filter to the image, which means taking the dot product between the filter & each “window” in the image

3. Stride length determines the size of the resulting convolved features

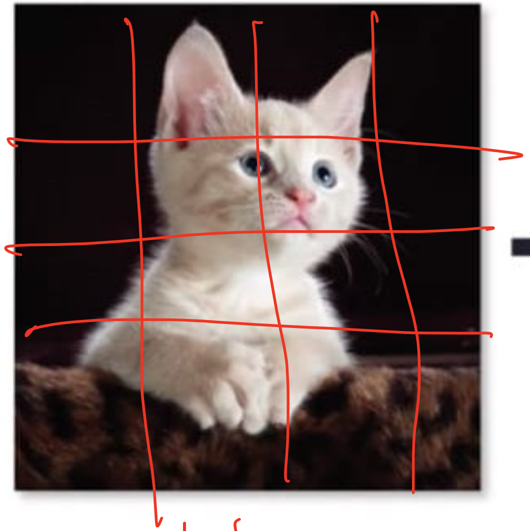
4. Padding ensures that we don’t miss important information in the edges of the image and prevents the convolved features from “shrinking” too much

3D examples: colored images

- Tensor → generalization of a matrix
- D×W×H

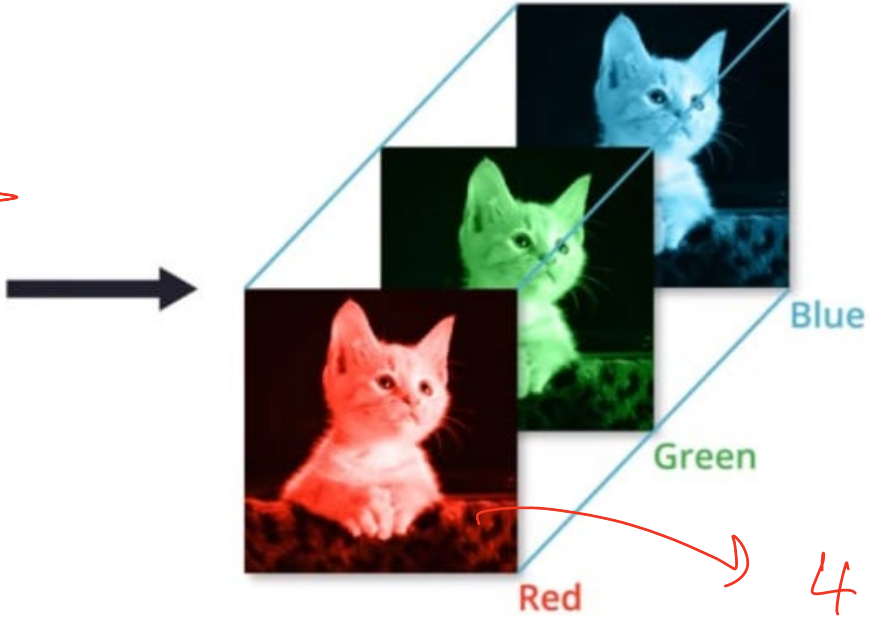


H



W

3×4×4



4×4

Image from https://dev.to/sandeepbalachandran/machine-learning-going-furthur-with-cnn-part-2-41km

3D convolutional layer

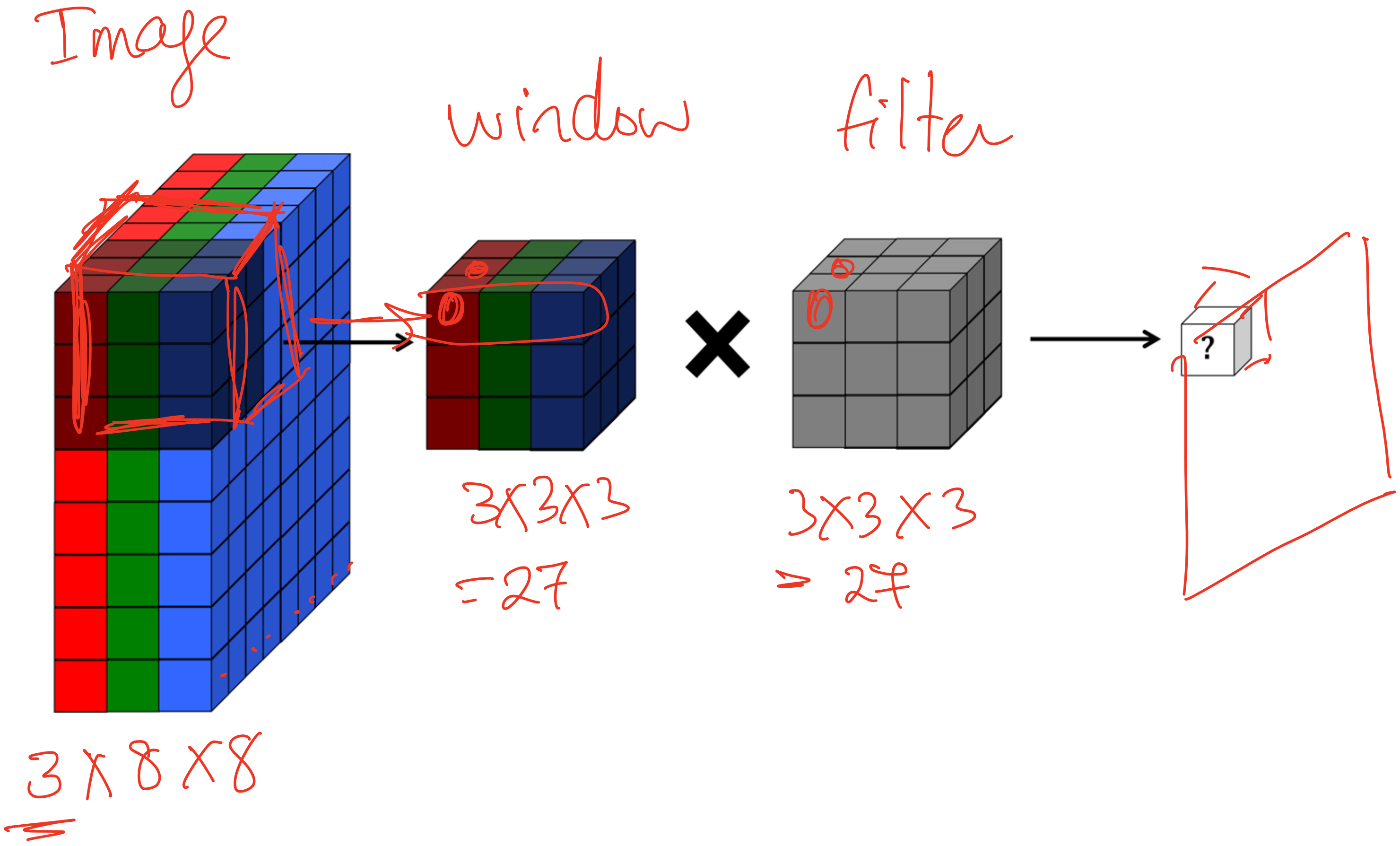


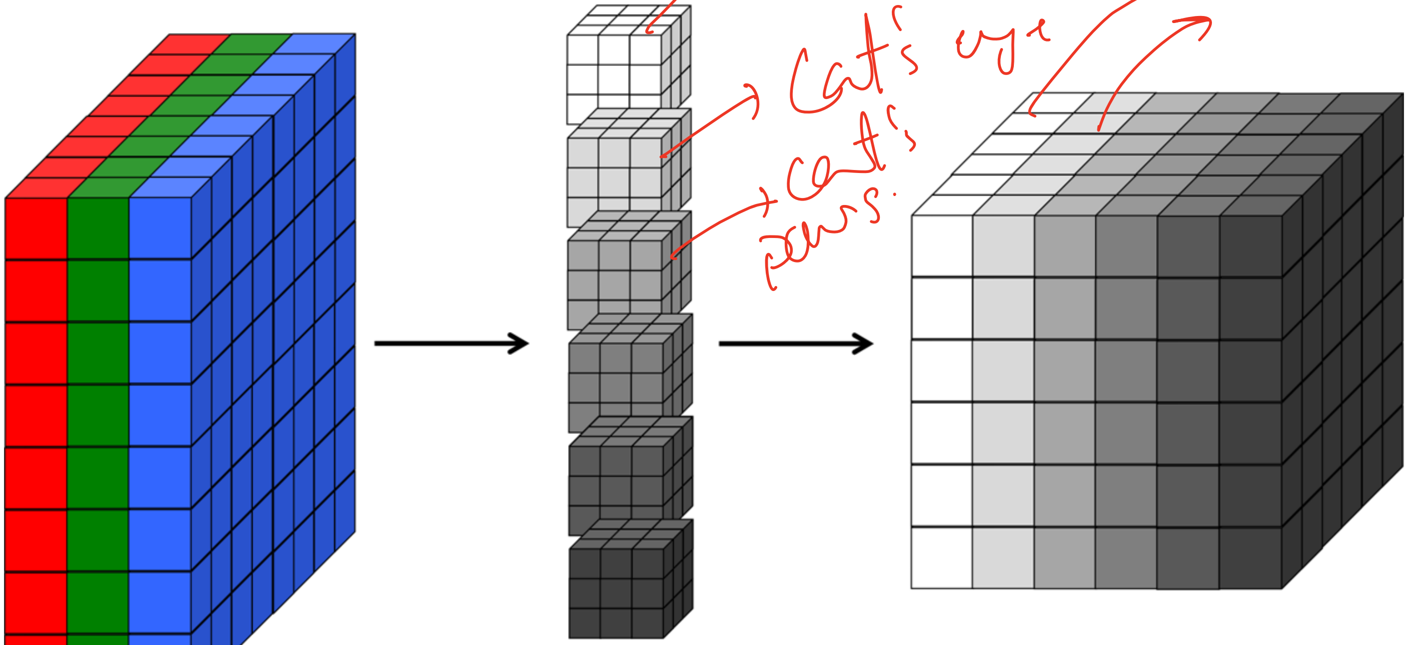
Figure from : Fundamentals of Deep Learning by Nikhil Buduma, Nicholas Locascio

Multiple filters

detector for the cat's ears

cat's eye

cat's paws



6 different filters, each $3 \times 3 \times 3$

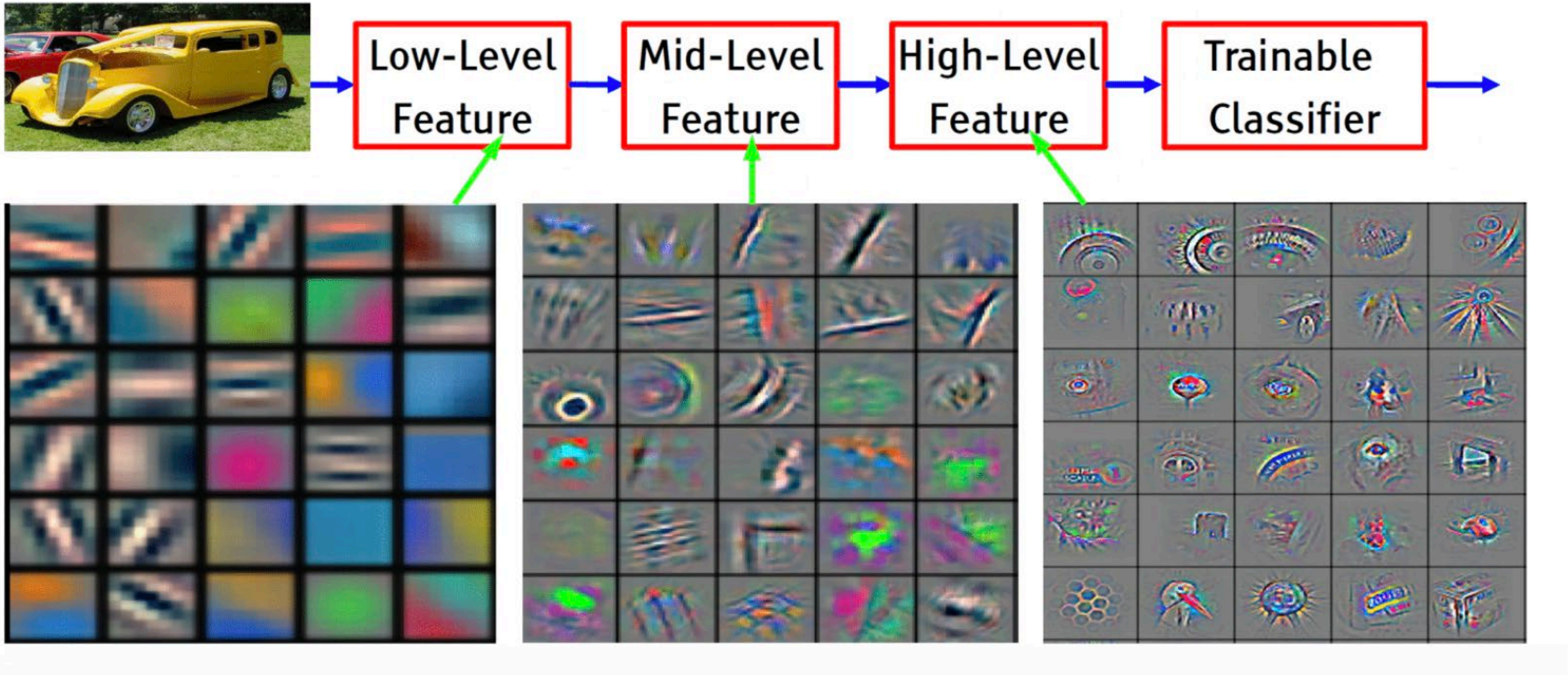
Output of depth 6

$6 \times 3 \times 3$

Figure from : Fundamentals of Deep Learning by Nikhil Buduma, Nicholas Locascio

Filters in a convolutional layer

- Each filter produces a new feature map.
  - Low level features in initial layers, and high level features in later layers
  - e.g., on first layer edge detection and higher layers wheel-like patterns



Filters in a convolutional layer

- Hyperparameters:
  - number of filters
  - stride length
  - filter size
  - zero-padding

- Efficiency concerns
  - parameter sharing (same weights across all depths)

Filters in a convolutional layer

- The size of the resulting convolved features depends on the size of the filter and the stride length
- We can specify different stride lengths in different directions
- Calculating the size of the convolved features (2D example):
  - Input (after padding if applicable): $I_w \times I_h$
  - Filter: $F \times F$
  - Stride: S
  - Output: $O_w \times O_h$ where:

$$O_w = \frac{I_w - F}{S} + 1, \qquad O_h = \frac{I_h - F}{S} + 1$$

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

- Goal: Have I learned any meaningful structure in this local "window" of my image?
- Has the effect of consolidating the information from each window
- Helps give us the desired invariance

2D example: Max pooling

Output from convolutional layer & ReLU:

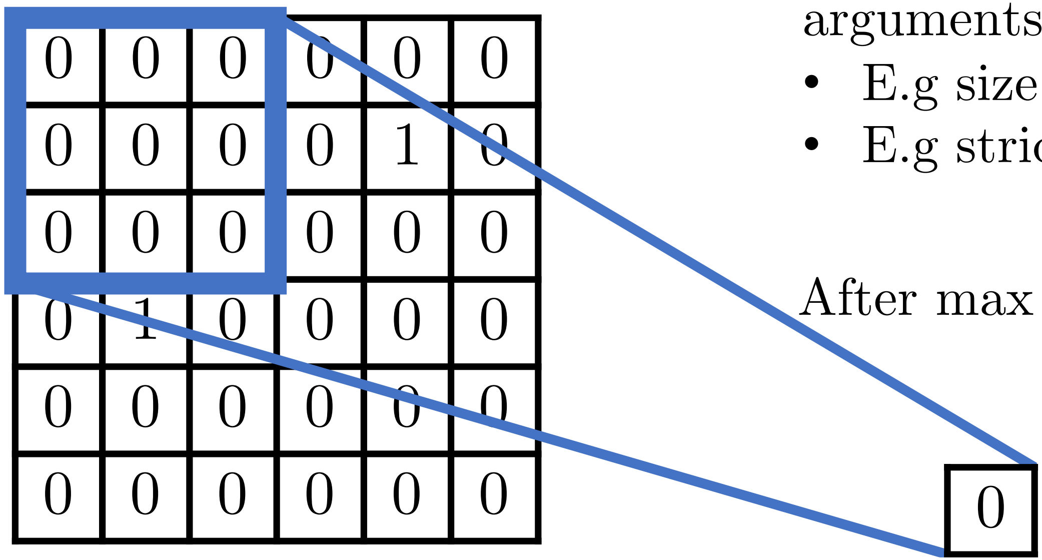
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

2D example: Max pooling

Output from convolutional layer & ReLU:



$\max(0,0,0,0,0,0,0,0,0)$

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

0

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

$\boxed{0}$

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

0	1

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

0	1

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

0	1

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

0	1

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

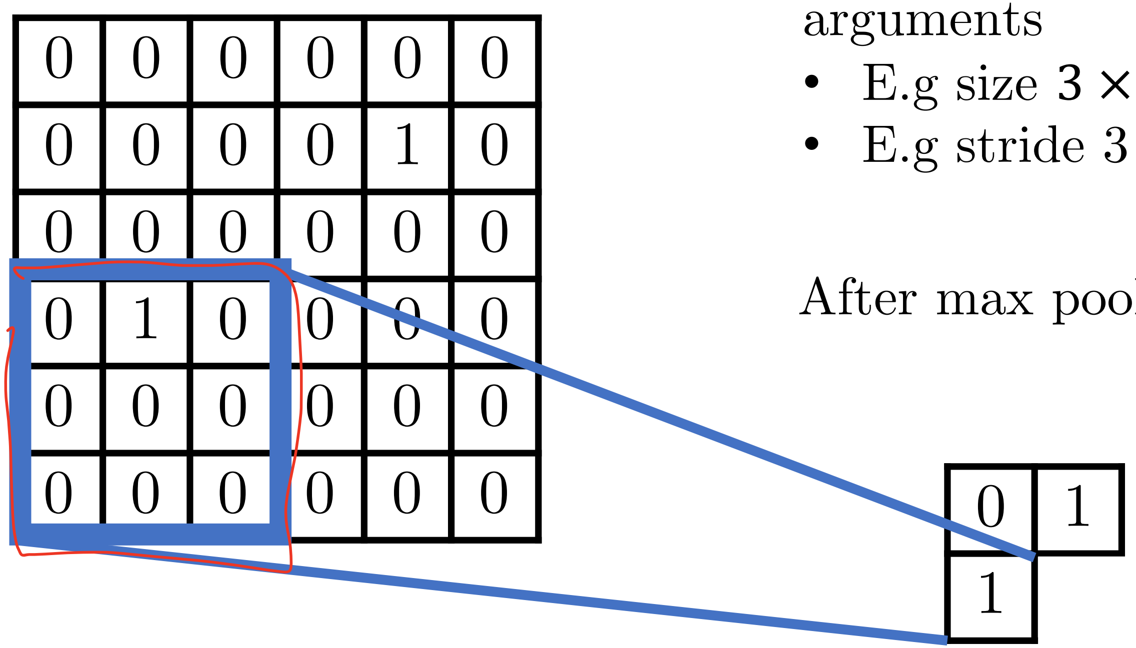
- E.g size 3×3
- E.g stride 3

After max pooling:

0	1

2D example: Max pooling

Output from convolutional layer & ReLU:



Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

2D example: Max pooling

Output from convolutional layer & ReLU:

0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Max pooling: returns max of its arguments

- E.g size 3×3
- E.g stride 3

After max pooling:

0	1
1	0

No learnable weights in this layer

Depth remains the same, but size shrinks