Introduction Convolutional Neural Nets (CNNs)

Outline

- Intuition examples
- Convolutional Neural Networks
- Architecture

Convolutional Neural Network
Why Do We Care About CNNs?

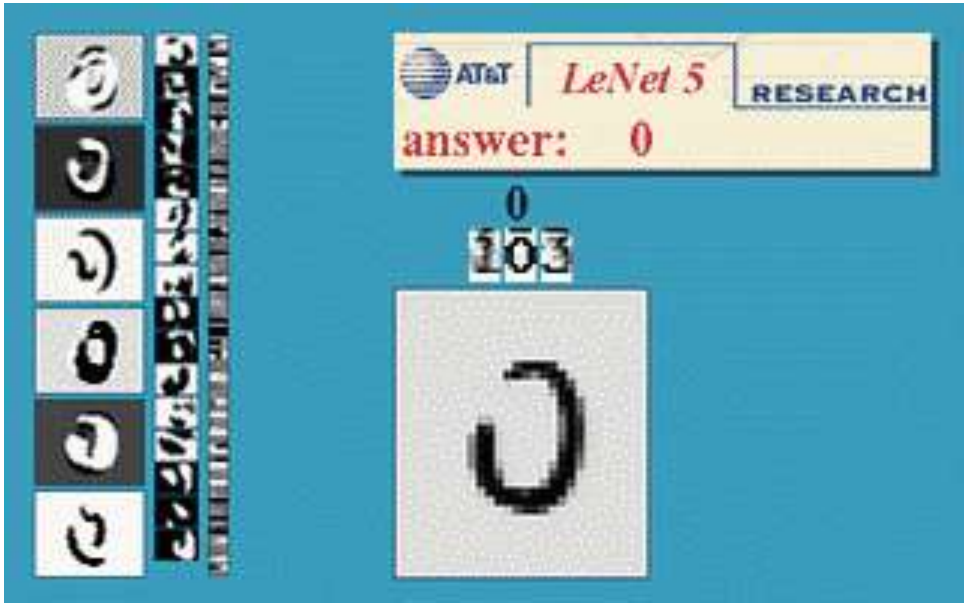
- CNNs extensively used in computer vision
- Powered by the massive data from ImageNet and the modern CPUs and GPUs to train such a model.

- Winning architecture to generate exciting new results in object recognition.

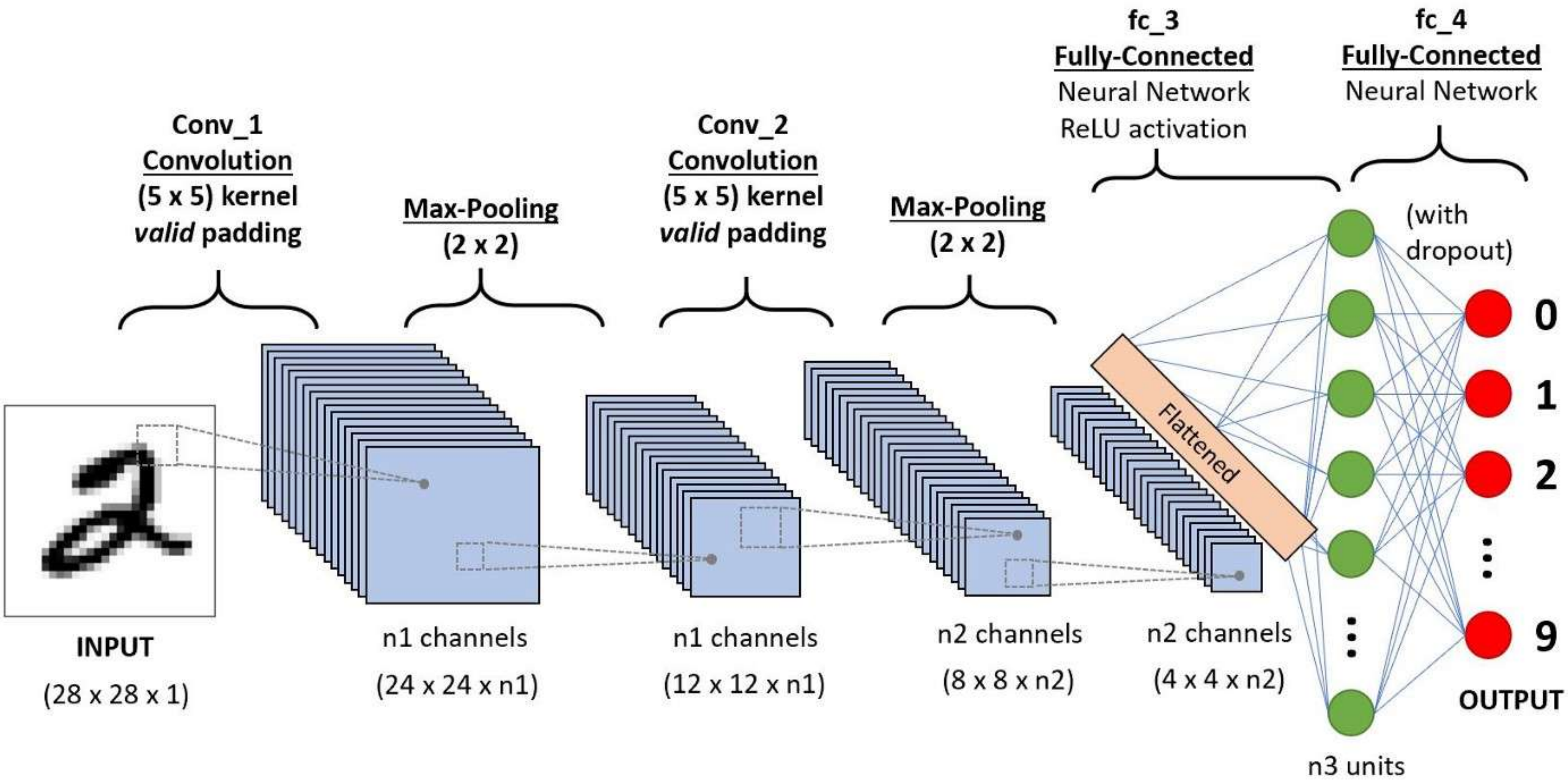
- CNN used also for text and voice

LeNet5

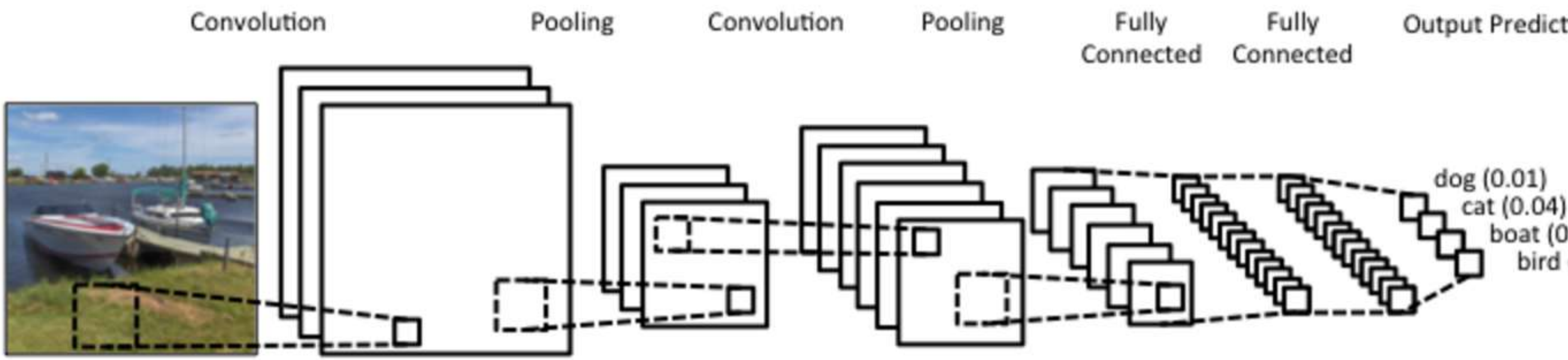
- LeCun 1998



Hand written digit classification (LeNet5) LeCun 1998



Convolutional NN



- Consider local structure and common extraction of features
- Not fully connected (Locality of processing)
- Weight sharing for parameter reduction
- Learn the parameters of multiple convolutional filter banks

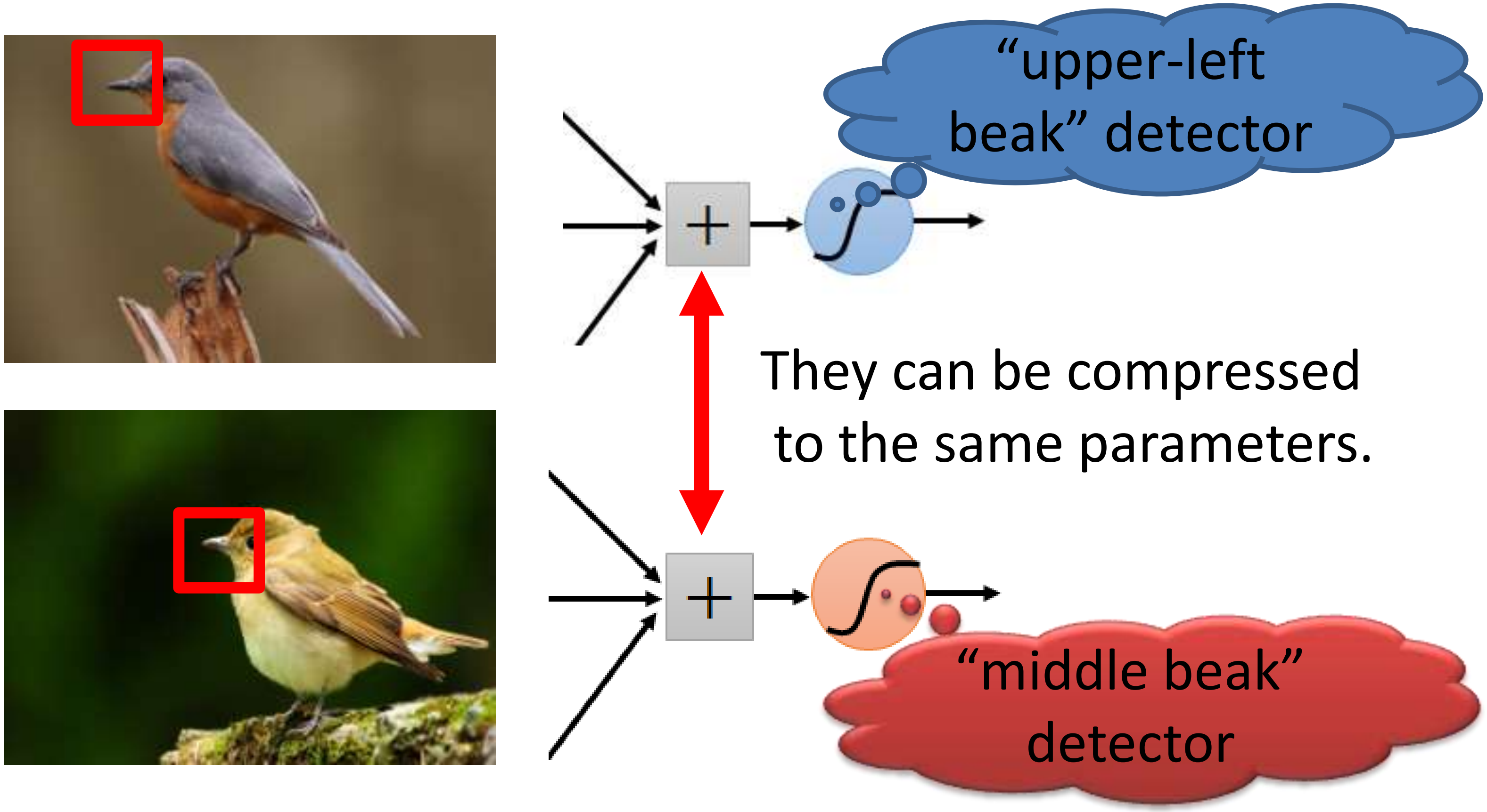
Consider learning an image:

- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters

Same pattern appears in different places

What about training a lot of such “small” detectors and each detector must “move around”.



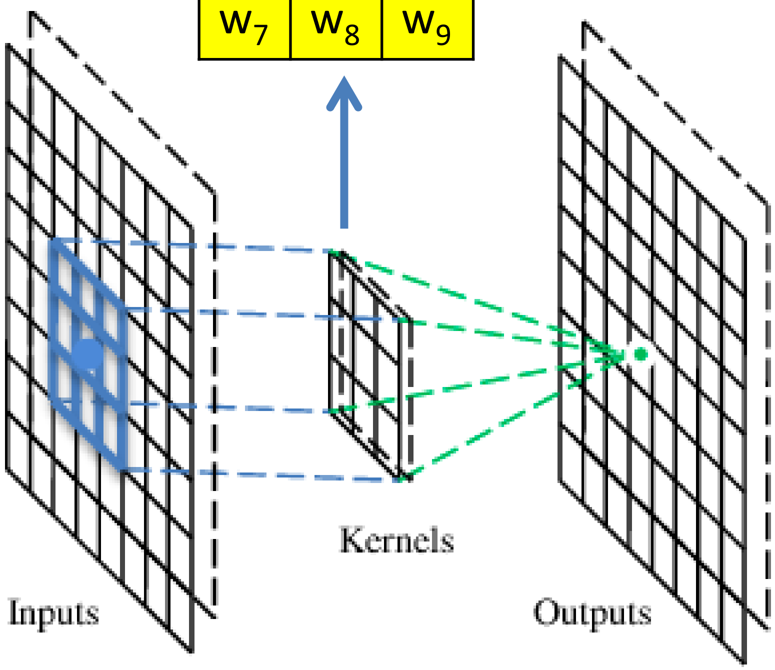
CNN is a neural network with some convolutional layers

- A convolutional layer has a number of filters that does convolutional operation.

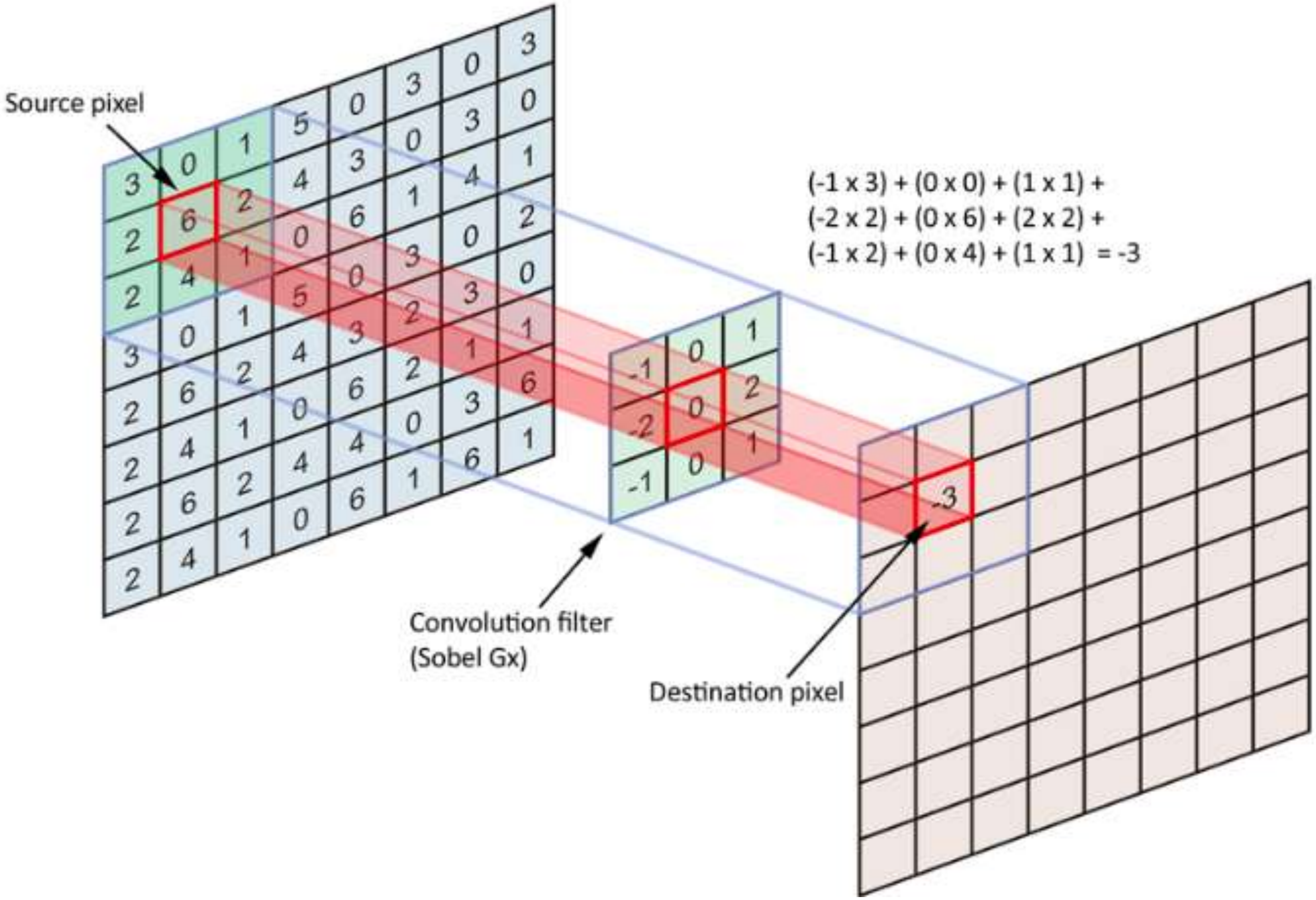
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$$out = w_1.x_1 + w_2.x_2 + w_3.x_3 + \ldots$$

Simple dot product



Convolution

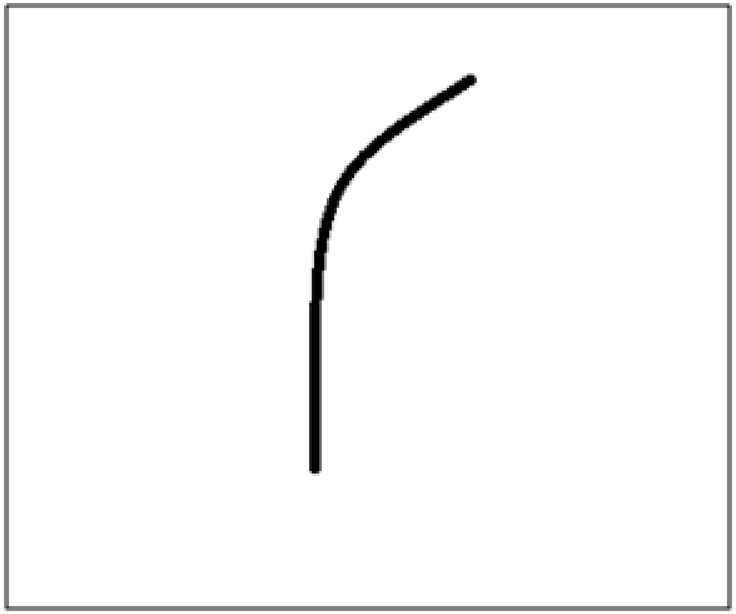


Intuition

- Consider first filter is 7 x 7 and is going to be a curve detector.
- As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area that is a shape of a curve.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

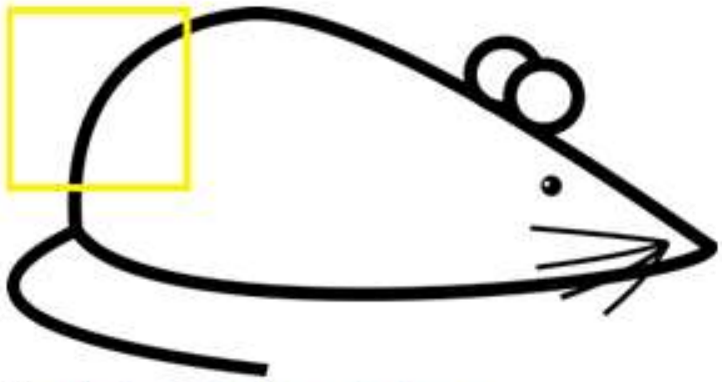


Visualization of a curve detector filter

Example

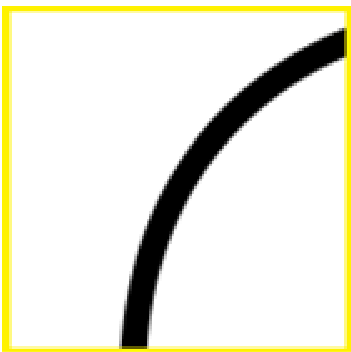
- Let's take an example of an image that we want to classify, and let's put our filter at the top left corner.

Original image



Visualization of the filter on the image

- Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

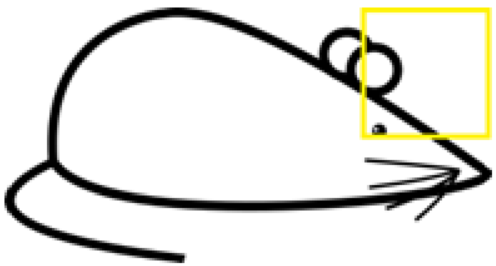
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value!

Let’s see what happens when we move our filter.



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

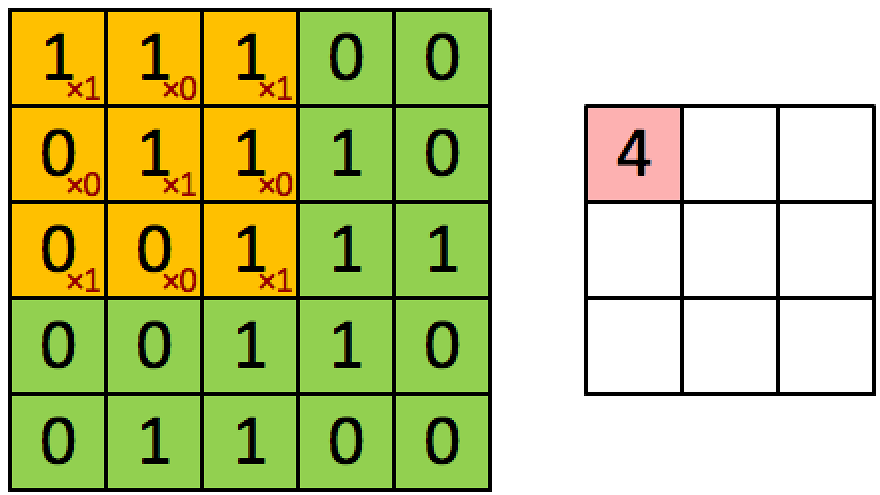
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

- The value is much lower! This is because there wasn’t anything in the image section that responded to the curve detector filter.

Convolved Features



Image

Convolved Feature

Convolution

These are the network parameters to be learned.

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

6 x 6 image

Filter 1

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

Filter 2

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

⋮

Each filter detects a small pattern (3 x 3).

Convolution

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

Filter 1

stride=1

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

6 x 6 image

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

Convolution

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

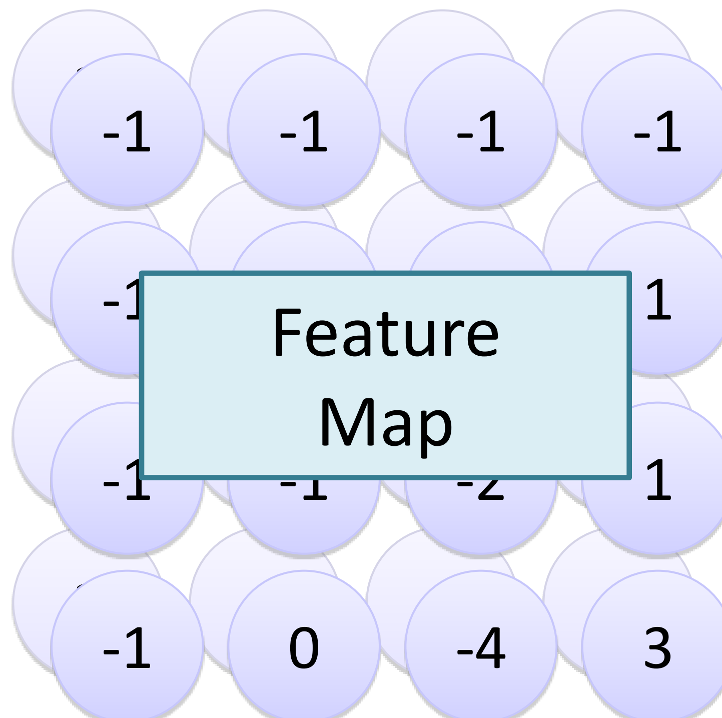
Filter 2

stride=1

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

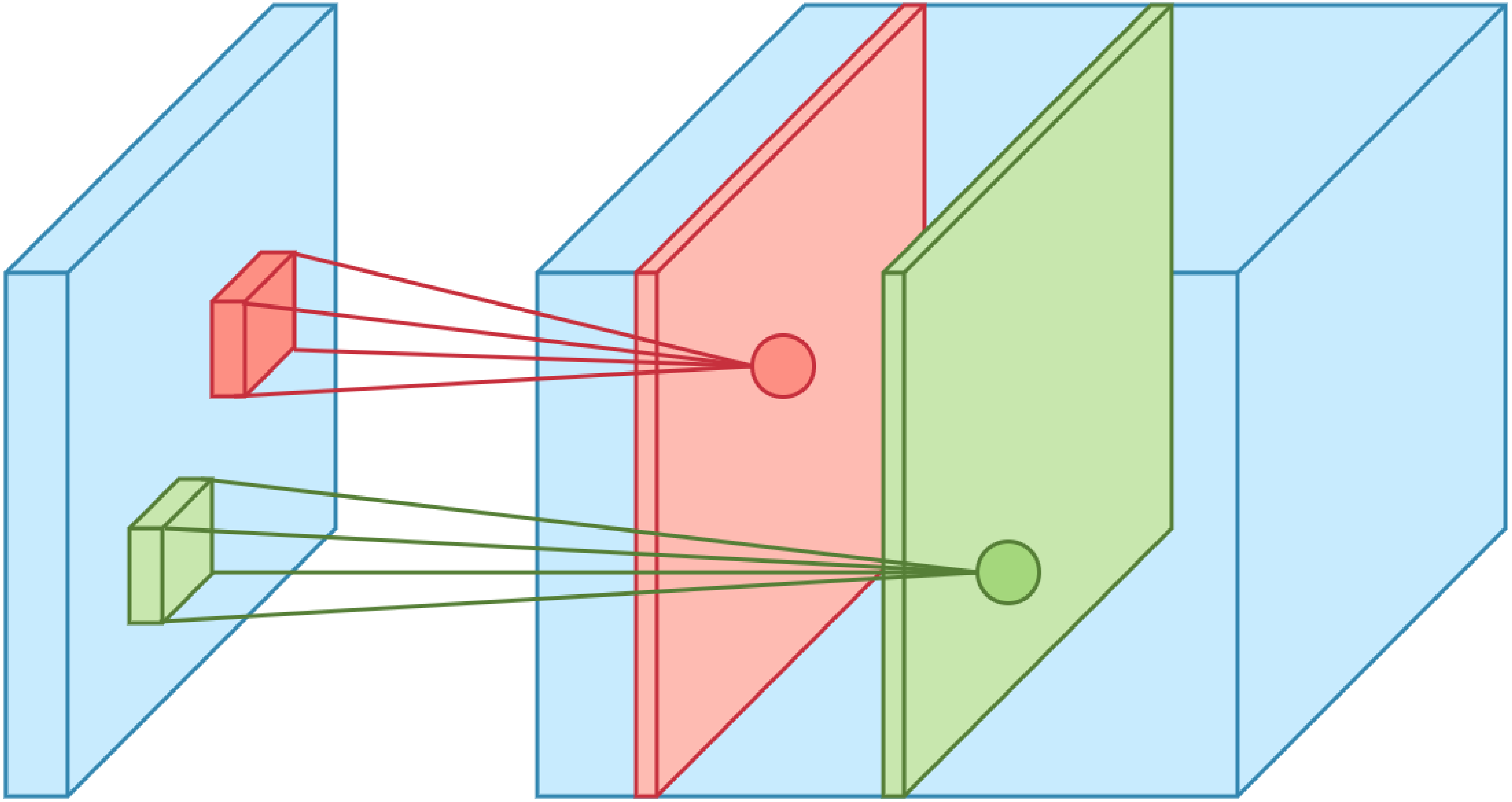
6 x 6 image

Repeat this for each filter

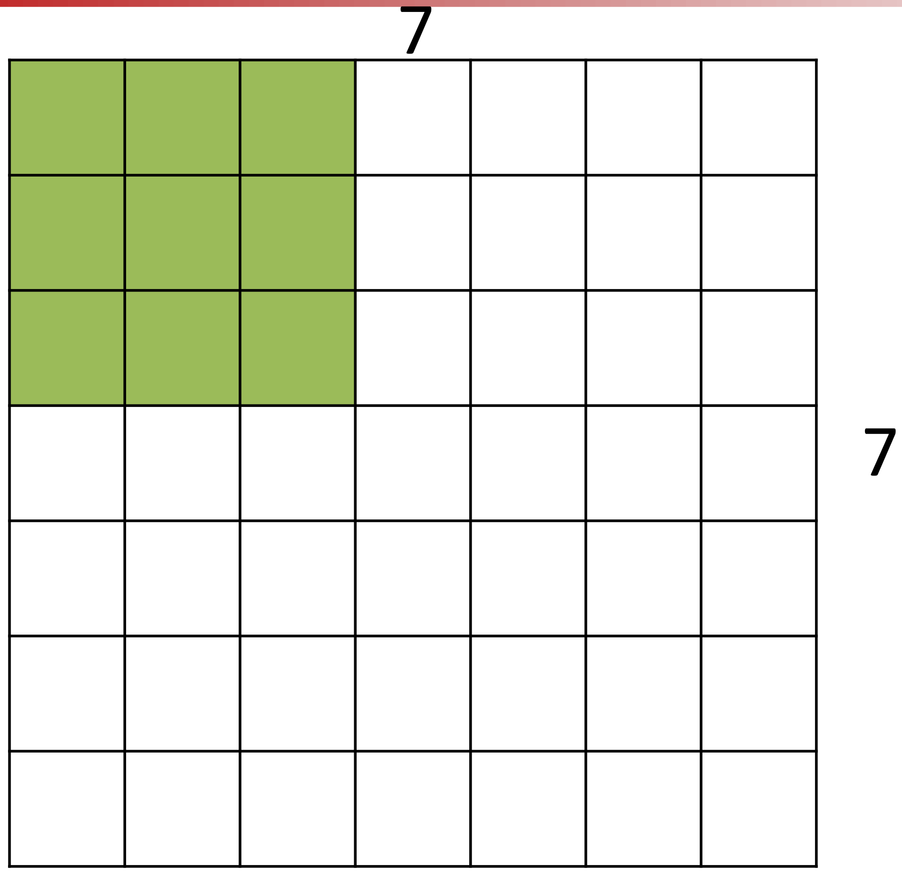


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Filter $\rightarrow$ feature map

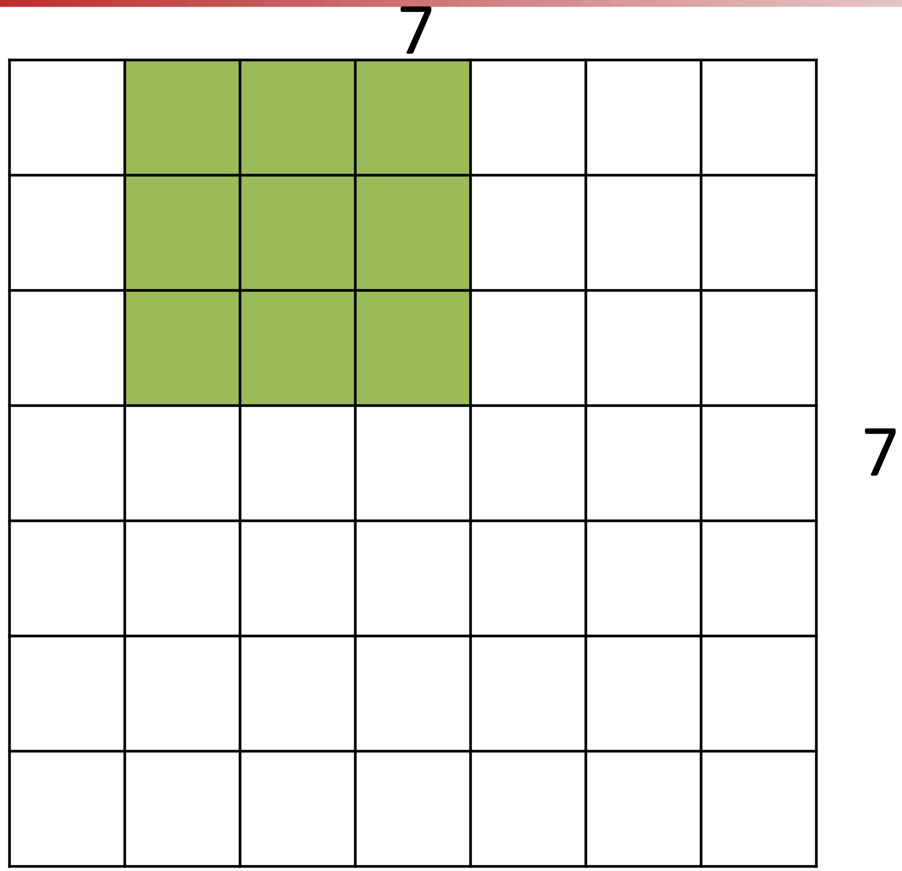


A closer look at spatial dimensions



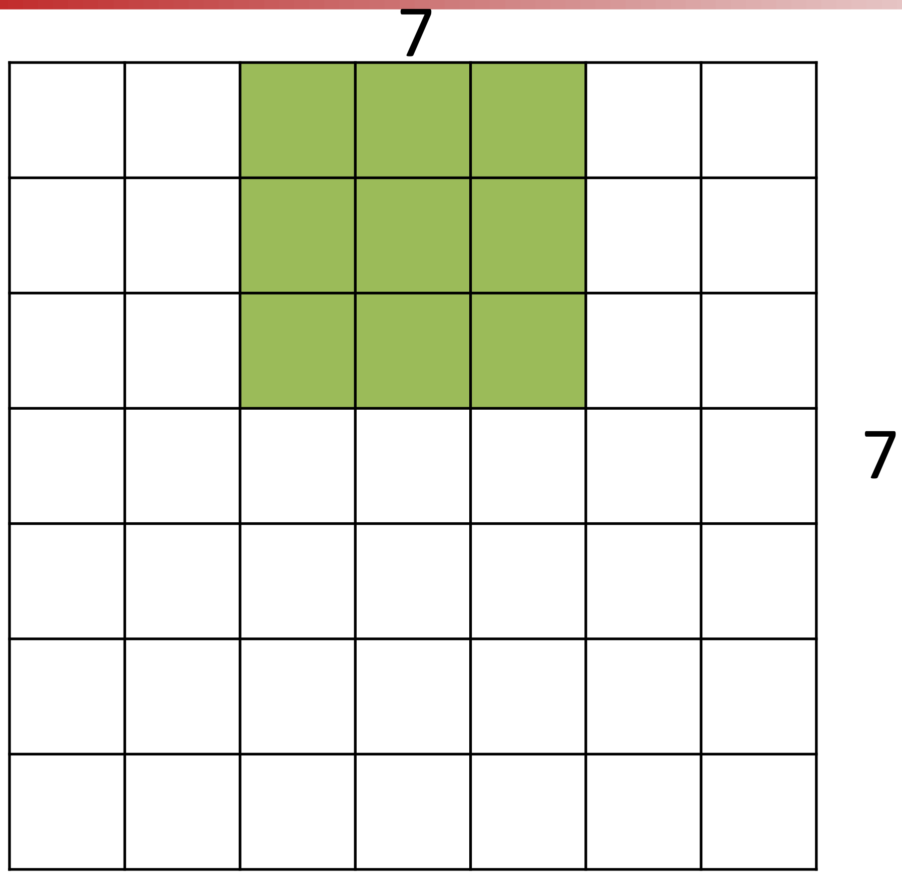
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter

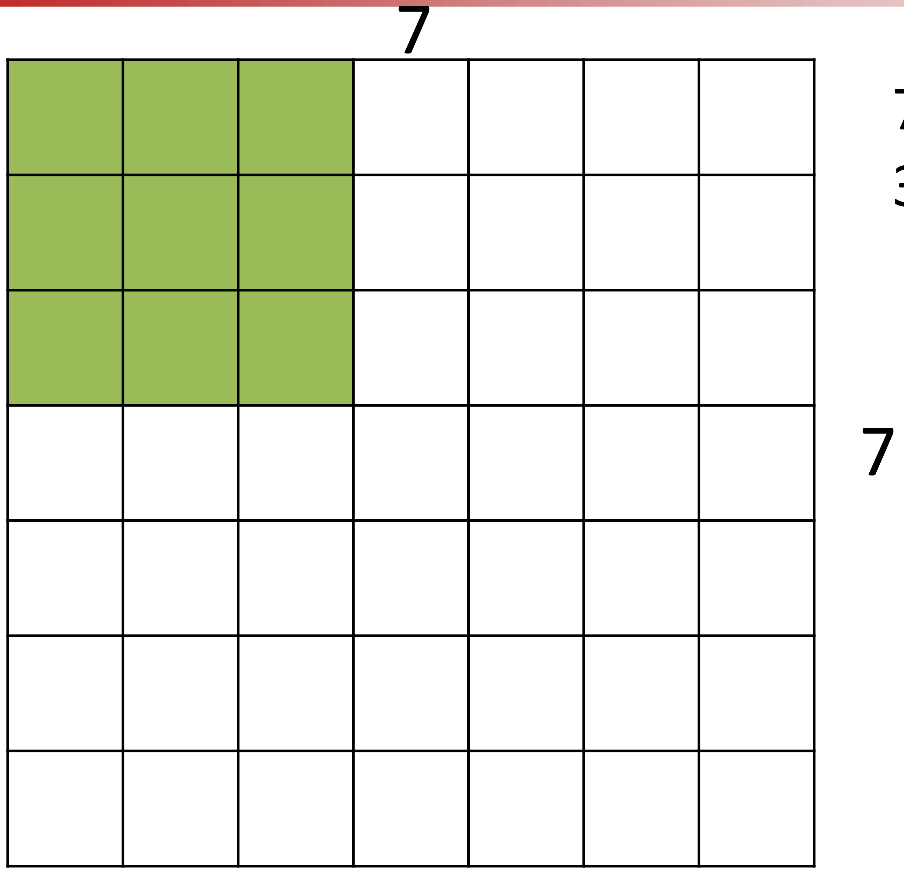
A closer look at spatial dimensions



7x7 input (spatially)
assume 3x3 filter

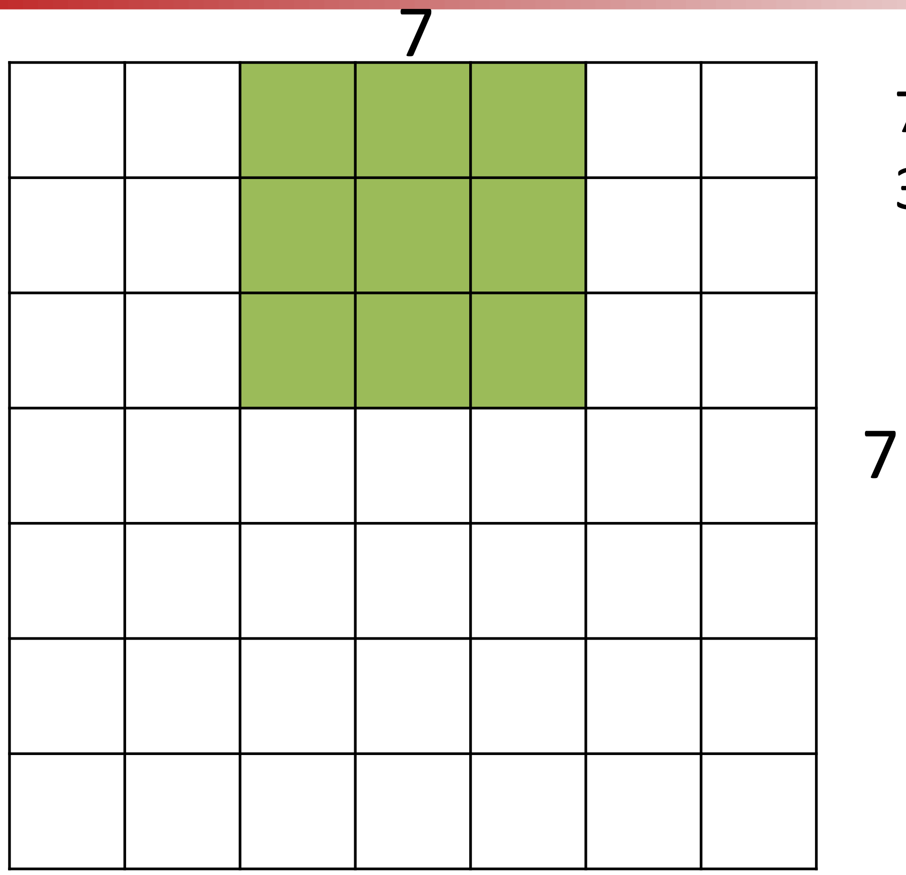
Stride 1 → 5 x 5 output

A closer look at spatial dimensions



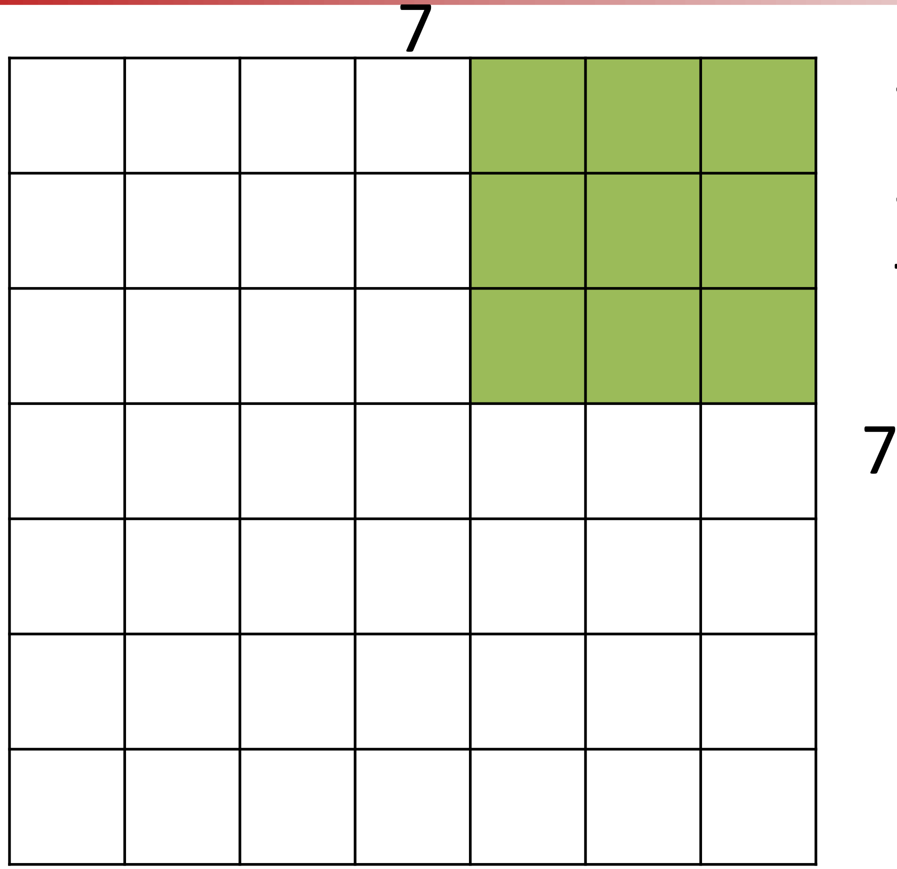
7x7 input (spatially) assume
3x3 filter applied **with stride 2**

A closer look at spatial dimensions

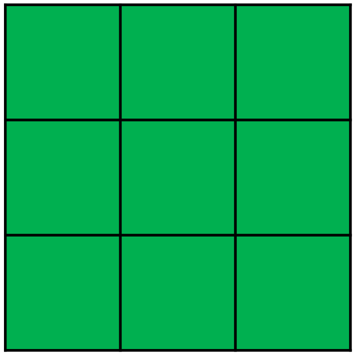


7x7 input (spatially) assume 3x3 filter applied **with stride 2**

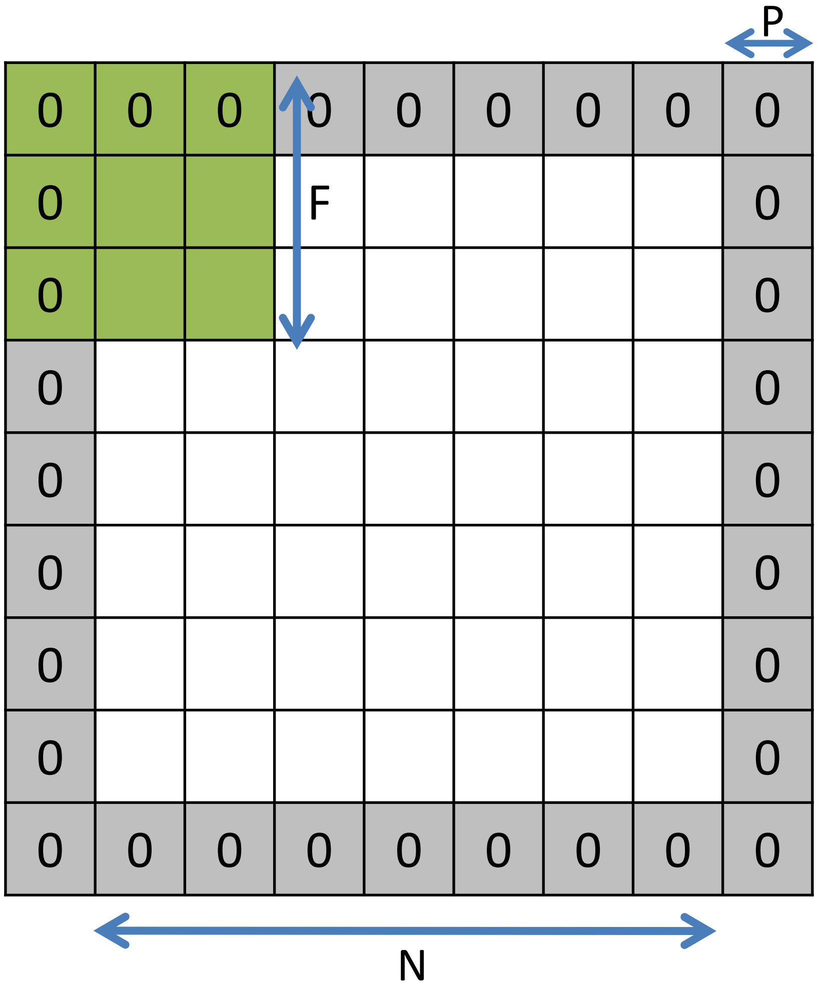
A closer look at spatial dimensions



7x7 input (spatially) assume 3x3 filter applied **with stride 2**
→ **3 x 3 output**



Zero padding to preserve the spacial size



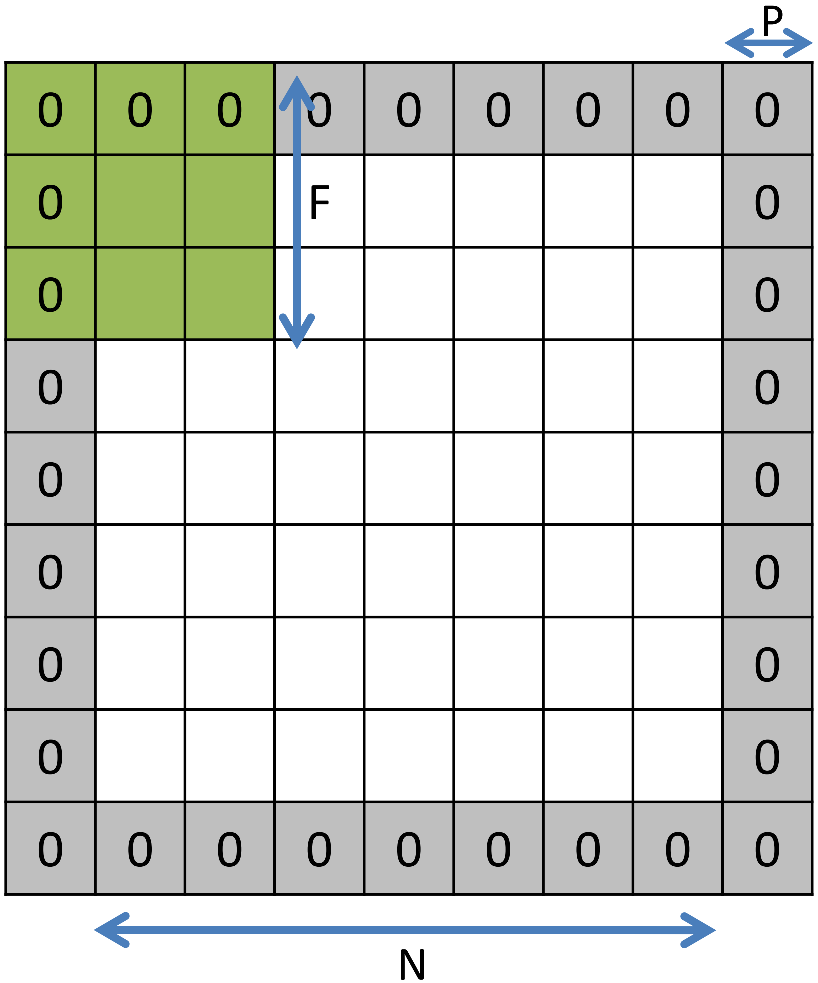
e.g. input N = 7
3x3 filter, applied with stride 1
pad with 1 pixel border → what is the output?

output size: $out = \dfrac{N - F + 2P}{S} + 1$

S: stride
N: size of image
F: filter size
P: amount of padding

7x7 output!

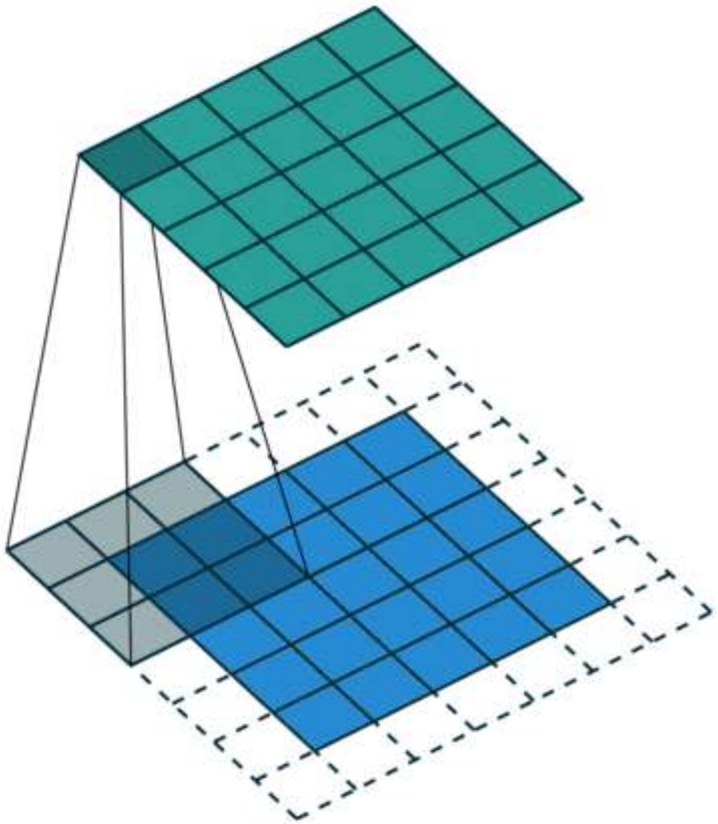
In practice: Common to zero pad the border



In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

SAME padding: 5x5x1 image is padded with 0s to create a 6x6x1 image



SAME padding: 5x5x1 image is padded with 0s to create a 6x6x1 image

Color image: RGB 3 channels

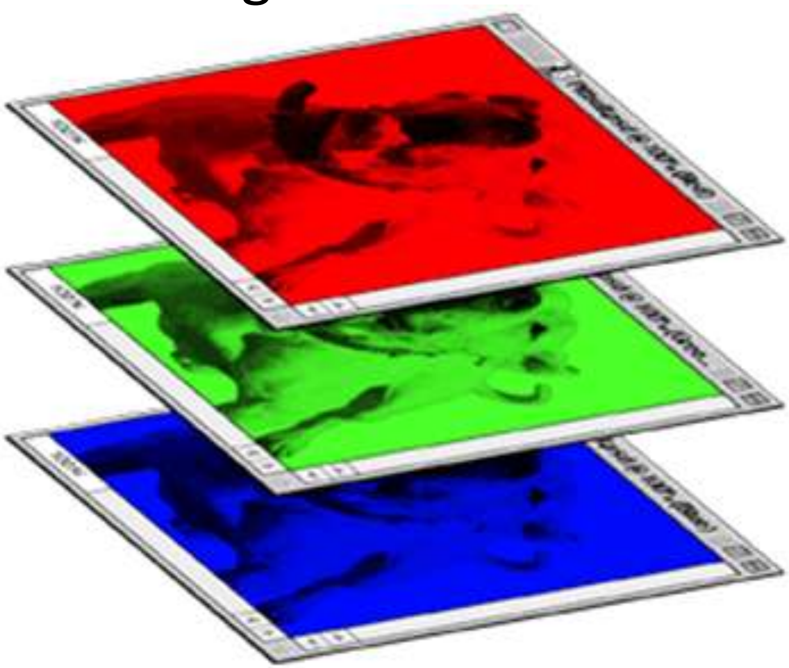
Filter 1

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

Filter 2

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

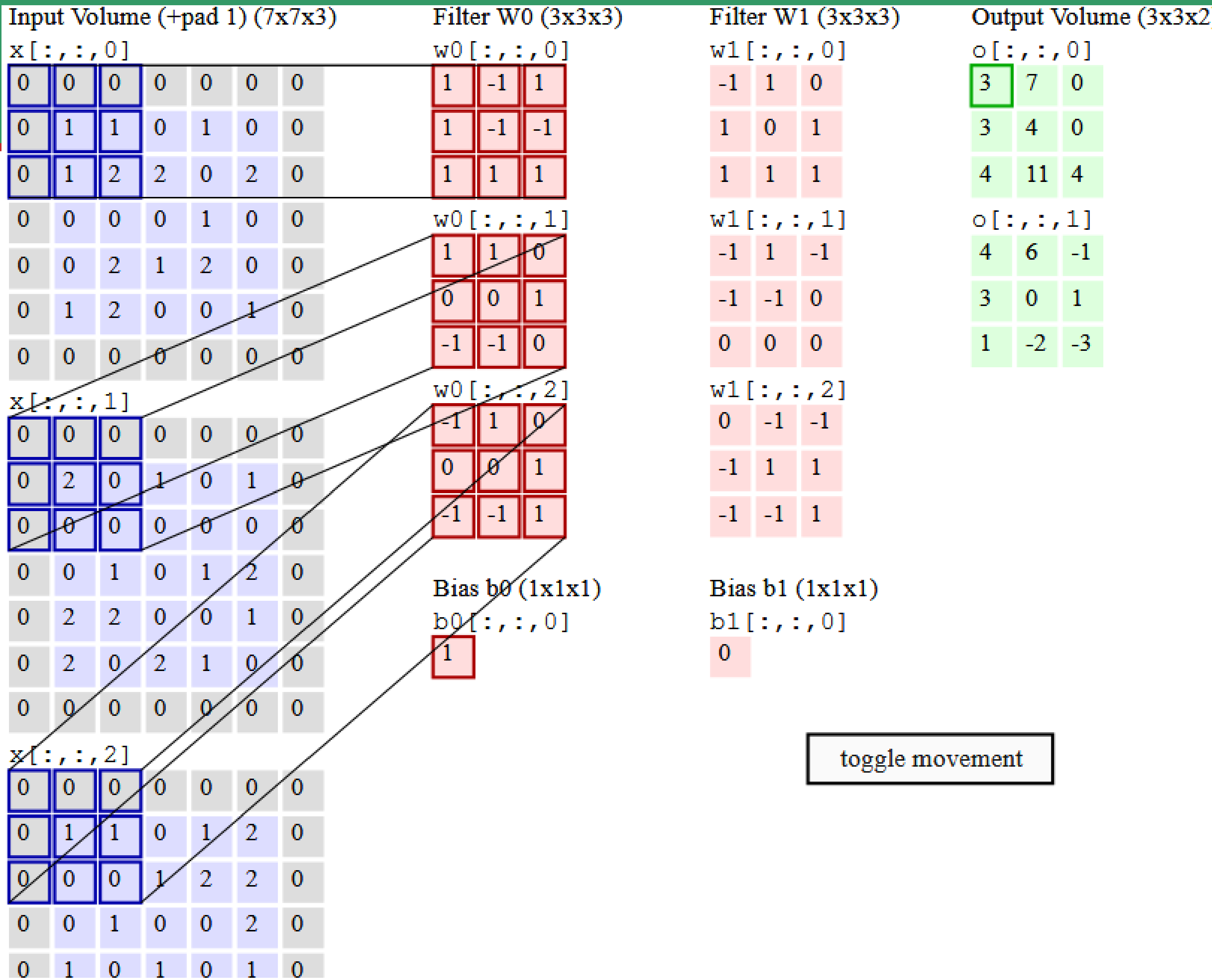
Color image



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

- Demo from:

CS231n Convolutional Neural Networks



Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 1 0 1 0 0
0 1 2 2 0 2 0
0 0 0 0 1 0 0
0 0 2 1 2 0 0
0 1 2 0 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 0 1 0 1 0
0 0 0 0 0 0 0
0 0 1 0 1 2 0
0 2 2 0 0 1 0
0 2 0 2 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 0 1 2 0
0 0 0 1 2 2 0
0 0 1 0 0 2 0
0 1 0 1 0 1 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 1
1 -1 -1
1 1 1
w0[:,:,1]
1 1 0
0 0 1
-1 -1 0
w0[:,:,2]
-1 1 0
0 0 1
-1 -1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 0 1
1 1 1
w1[:,:,1]
-1 1 -1
-1 -1 0
0 0 0
w1[:,:,2]
0 -1 -1
-1 1 1
-1 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
3 7 0
3 4 0
4 11 4
o[:,:,1]
4 6 -1
3 0 1
1 -2 -3
toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	0	1	0	0
0	1	2	2	0	2	0
0	0	0	0	1	0	0
0	0	2	1	2	0	0
0	1	2	0	0	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	0	1	0	1	0
0	0	0	0	0	0	0
0	0	1	0	1	2	0
0	2	2	0	0	1	0
0	2	0	2	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	1	0	1	2	0
0	0	0	1	2	2	0
0	0	1	0	0	2	0
0	1	0	1	0	1	0

Filter W0 (3x3x3)

w0[:,:,0]

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

w0[:,:,1]

1	1	0
0	0	1
-1	-1	0

w0[:,:,2]

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

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	0
1	0	1
1	1	1

w1[:,:,1]

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

w1[:,:,2]

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

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	7	0
3	4	0
4	11	4

o[:,:,1]

4	6	-1
3	0	1
1	-2	-3

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	0	1	0	0
0	1	2	2	0	2	0
0	0	0	0	1	0	0
0	0	2	1	2	0	0
0	1	2	0	0	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	0	1	0	1	0
0	0	0	0	0	0	0
0	0	1	0	1	2	0
0	2	2	0	0	1	0
0	2	0	2	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	1	0	1	2	0
0	0	0	1	2	2	0
0	0	1	0	0	2	0
0	1	0	1	0	1	0

Filter W0 (3x3x3)

w0[:,:,0]

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

w0[:,:,1]

1	1	0
0	0	1
-1	-1	0

w0[:,:,2]

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

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	0
1	0	1
1	1	1

w1[:,:,1]

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

w1[:,:,2]

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

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

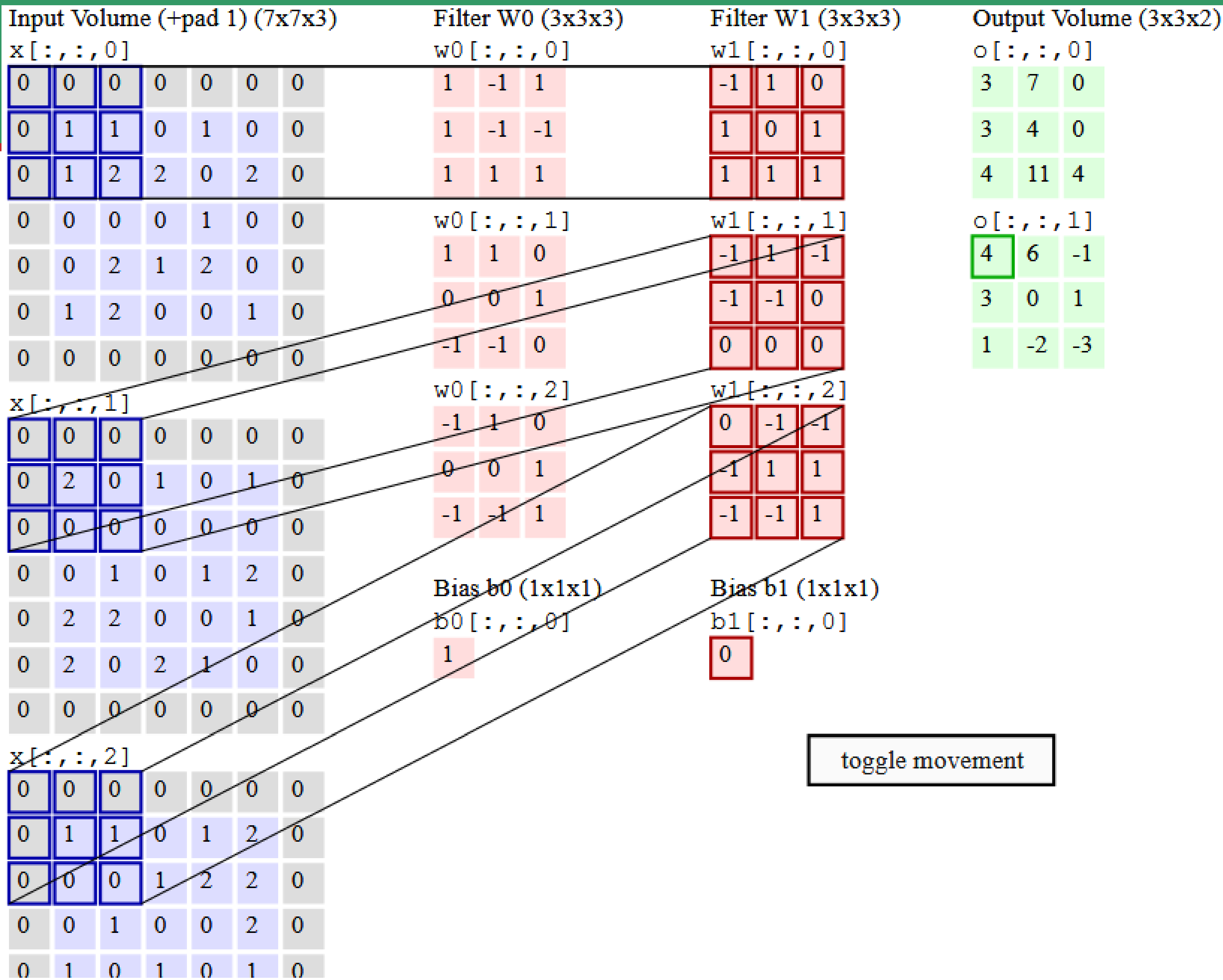
o[:,:,0]

3	7	0
3	4	0
4	11	4

o[:,:,1]

4	6	-1
3	0	1
1	-2	-3

toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 1 0 1 0 0
0 1 2 2 0 2 0
0 0 0 0 1 0 0
0 0 2 1 2 0 0
0 1 2 0 0 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 0 1 0 1 0
0 0 0 0 0 0 0
0 0 1 0 1 2 0
0 2 2 0 0 1 0
0 2 0 2 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 0 1 2 0
0 0 0 1 2 2 0
0 0 1 0 0 2 0
0 1 0 1 0 1 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 1
1 -1 -1
1 1 1
w0[:,:,1]
1 1 0
0 0 1
-1 -1 0
w0[:,:,2]
-1 1 0
0 0 1
-1 -1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 0
1 0 1
1 1 1
w1[:,:,1]
-1 1 -1
-1 -1 0
0 0 0
w1[:,:,2]
0 -1 -1
-1 1 1
-1 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
3 7 0
3 4 0
4 11 4
o[:,:,1]
4 6 -1
3 0 1
1 -2 -3
toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	0	1	0	0
0	1	2	2	0	2	0
0	0	0	0	1	0	0
0	0	2	1	2	0	0
0	1	2	0	0	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	0	1	0	1	0
0	0	0	0	0	0	0
0	0	1	0	1	2	0
0	2	2	0	0	1	0
0	2	0	2	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	1	0	1	2	0
0	0	0	1	2	2	0
0	0	1	0	0	2	0
0	1	0	1	0	1	0

Filter W0 (3x3x3)

w0[:,:,0]

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

w0[:,:,1]

1	1	0
0	0	1
-1	-1	0

w0[:,:,2]

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

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	1	0
1	0	1
1	1	1

w1[:,:,1]

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

w1[:,:,2]

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

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

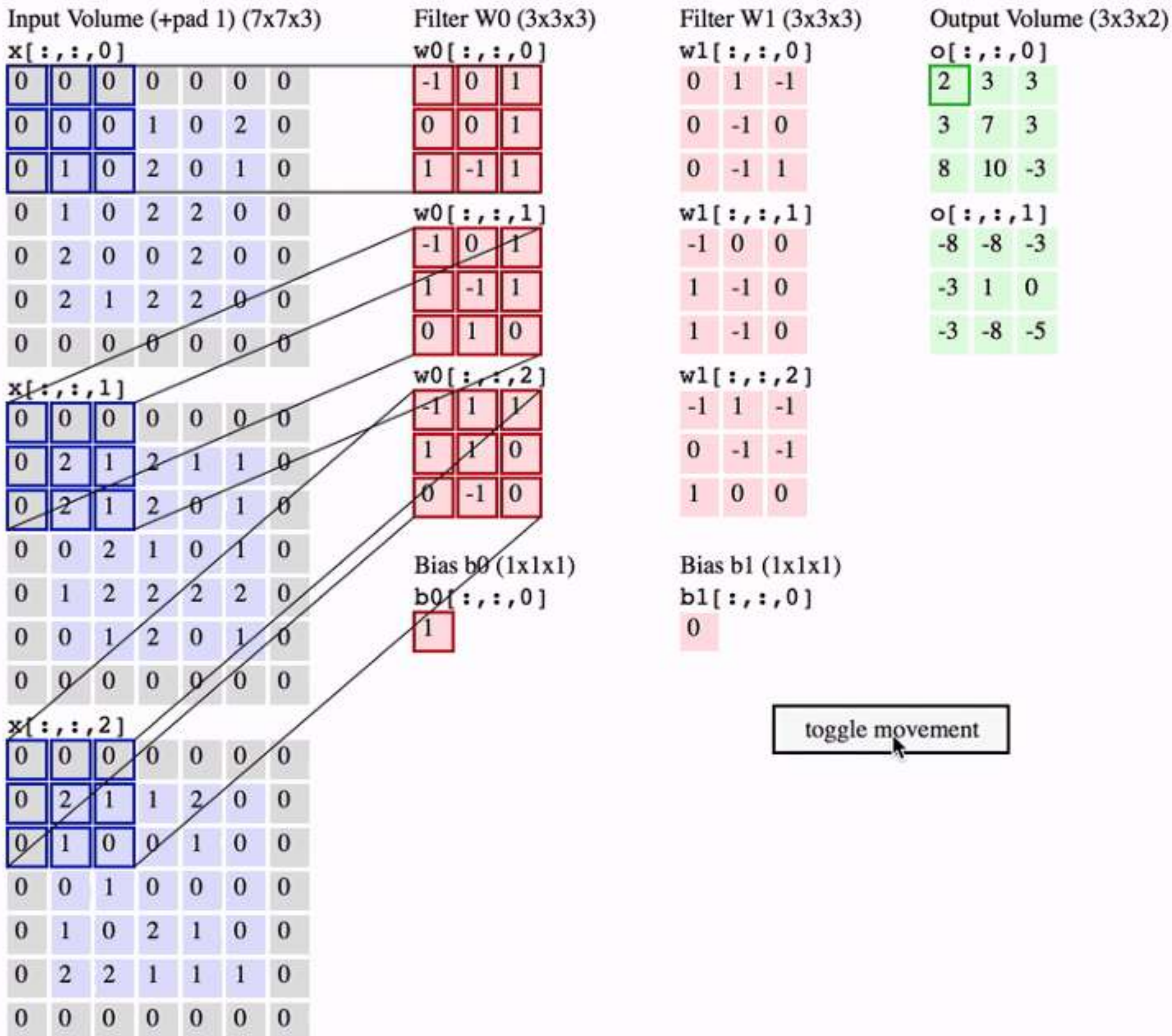
o[:,:,0]

3	7	0
3	4	0
4	11	4

o[:,:,1]

4	6	-1
3	0	1
1	-2	-3

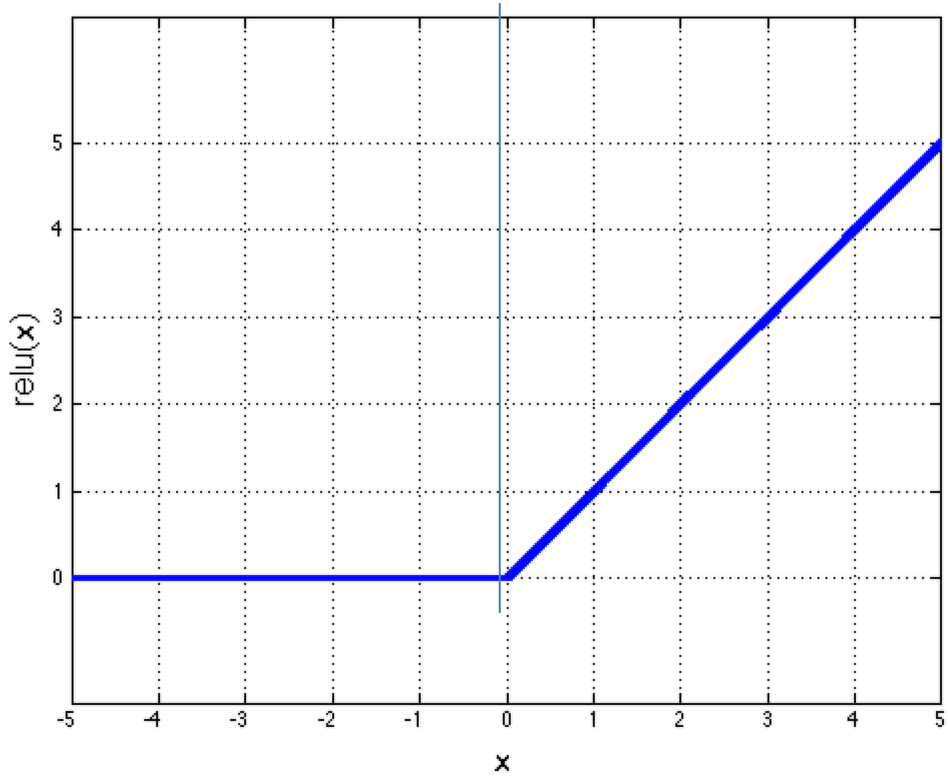
toggle movement


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 0 2 0 1 0
0 1 0 2 2 0 0
0 2 0 0 2 0 0
0 2 1 2 2 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 1 2 1 1 0
0 2 1 2 0 1 0
0 0 2 1 0 1 0
0 1 2 2 2 2 0
0 0 1 2 0 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 1 1 2 0 0
0 1 0 0 1 0 0
0 0 1 0 0 0 0
0 1 0 2 1 0 0
0 2 2 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 0 1
0 0 1
1 -1 1
w0[:,:,1]
-1 0 1
1 -1 1
0 1 0
w0[:,:,2]
-1 1 1
1 1 0
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 -1
0 -1 0
0 -1 1
w1[:,:,1]
-1 0 0
1 -1 0
1 -1 0
w1[:,:,2]
-1 1 -1
0 -1 -1
1 0 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
2 3 3
3 7 3
8 10 -3
o[:,:,1]
-8 -8 -3
-3 1 0
-3 -8 -5
toggle movement

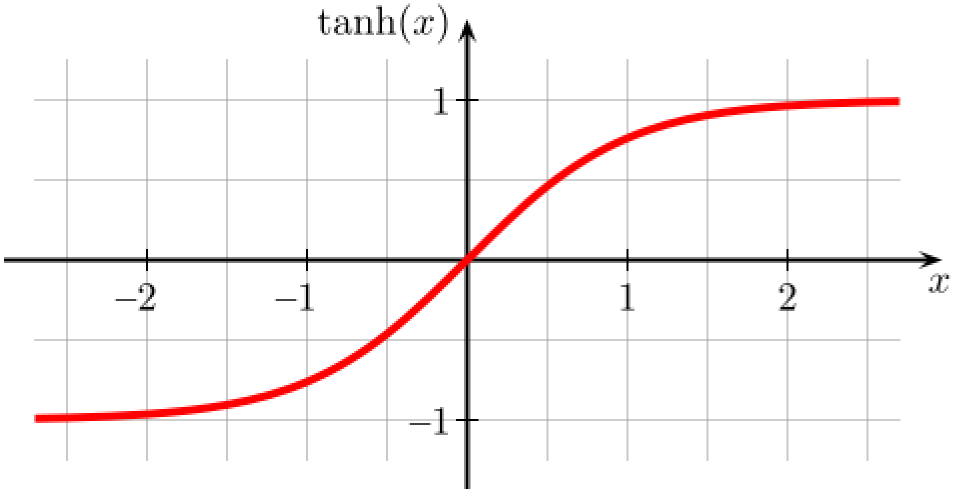
Rectified Linear Unit (ReLU)

ReLU: f(x) = max(0,x)

$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

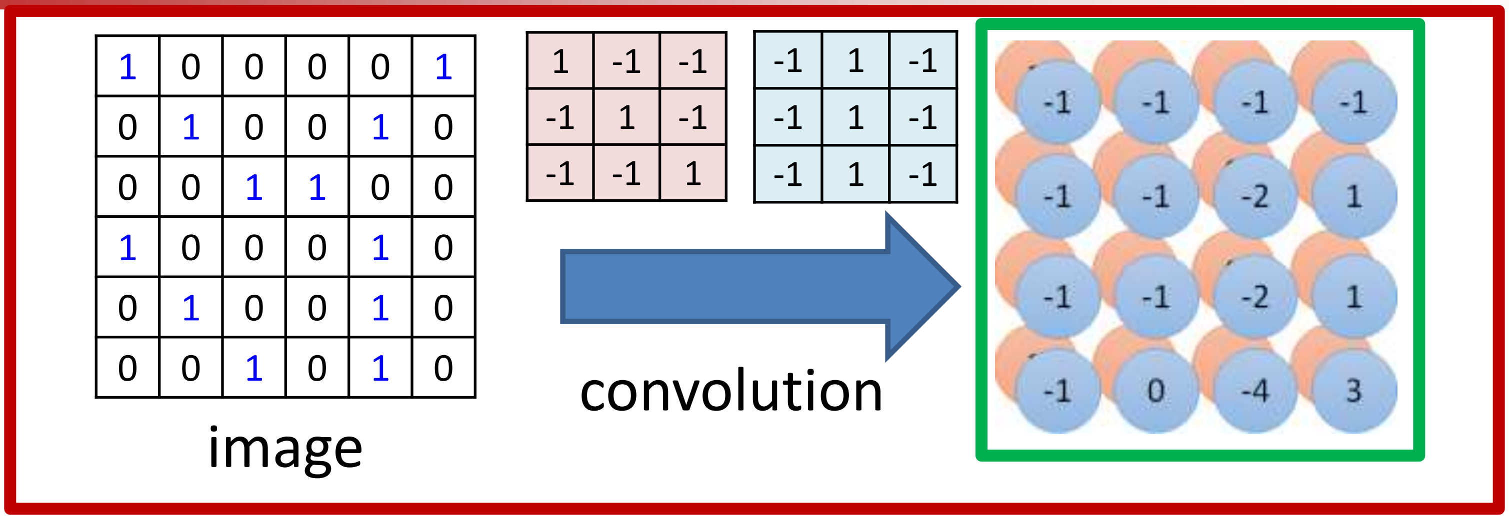


- Tanh(x)

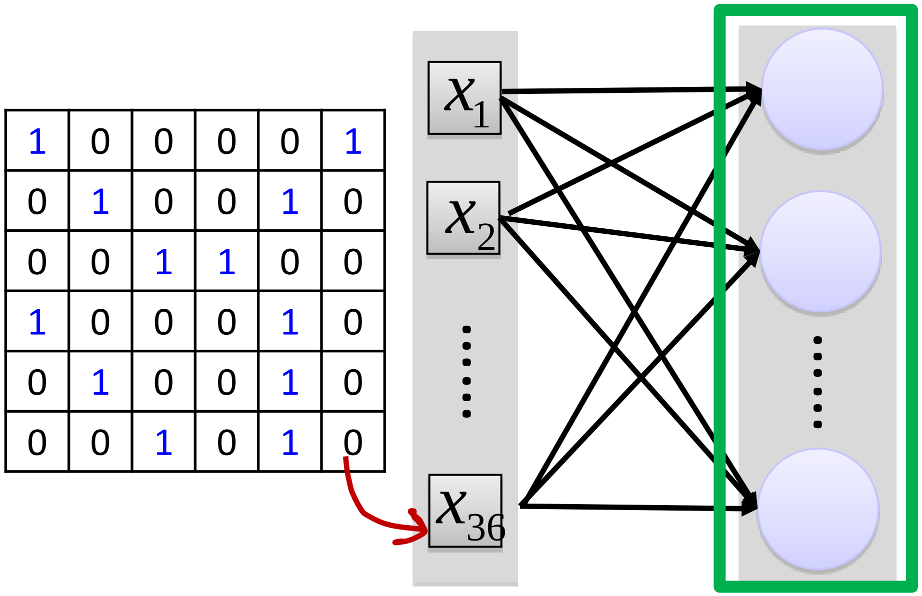


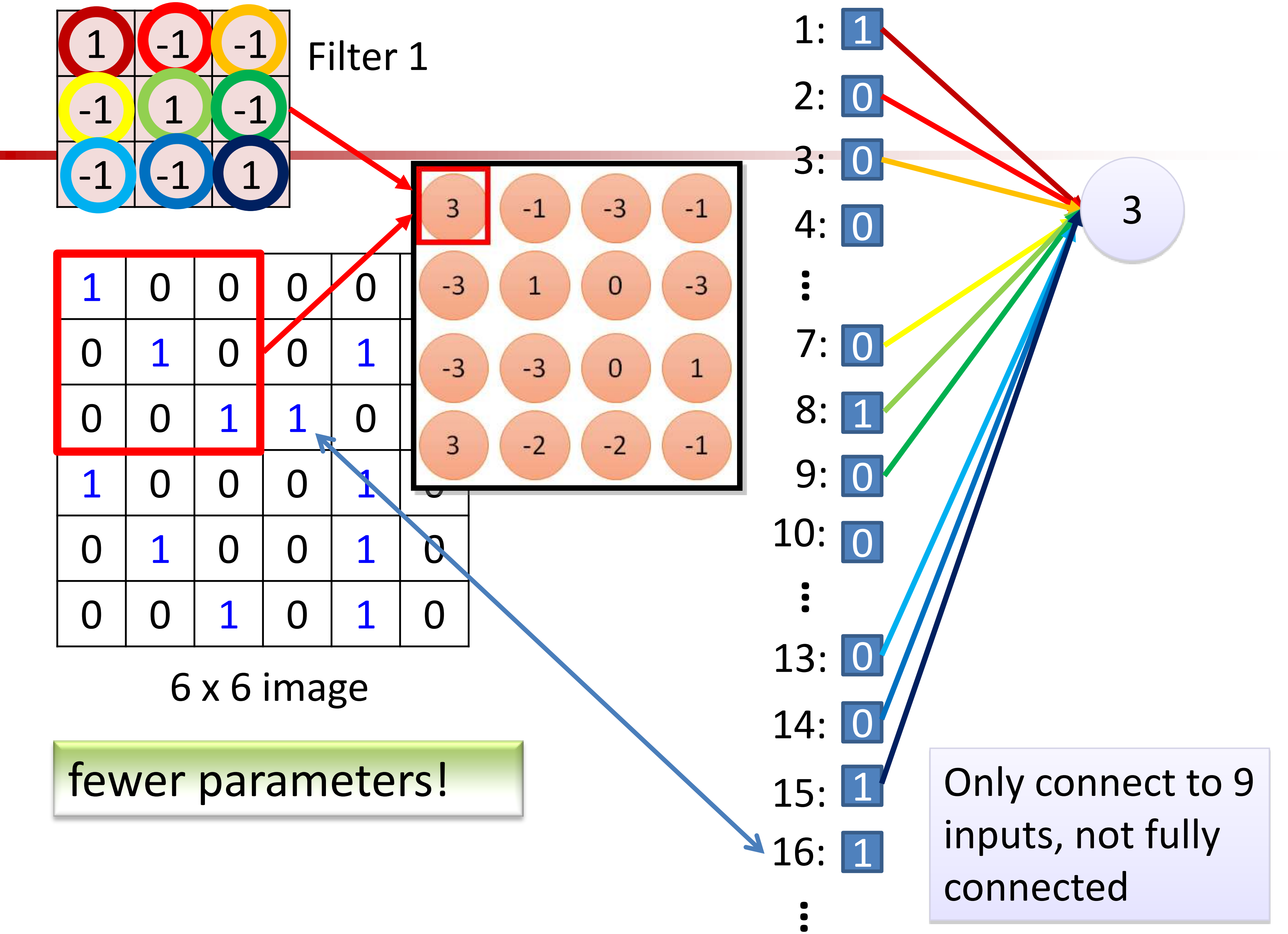
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Convolution v.s. Fully Connected

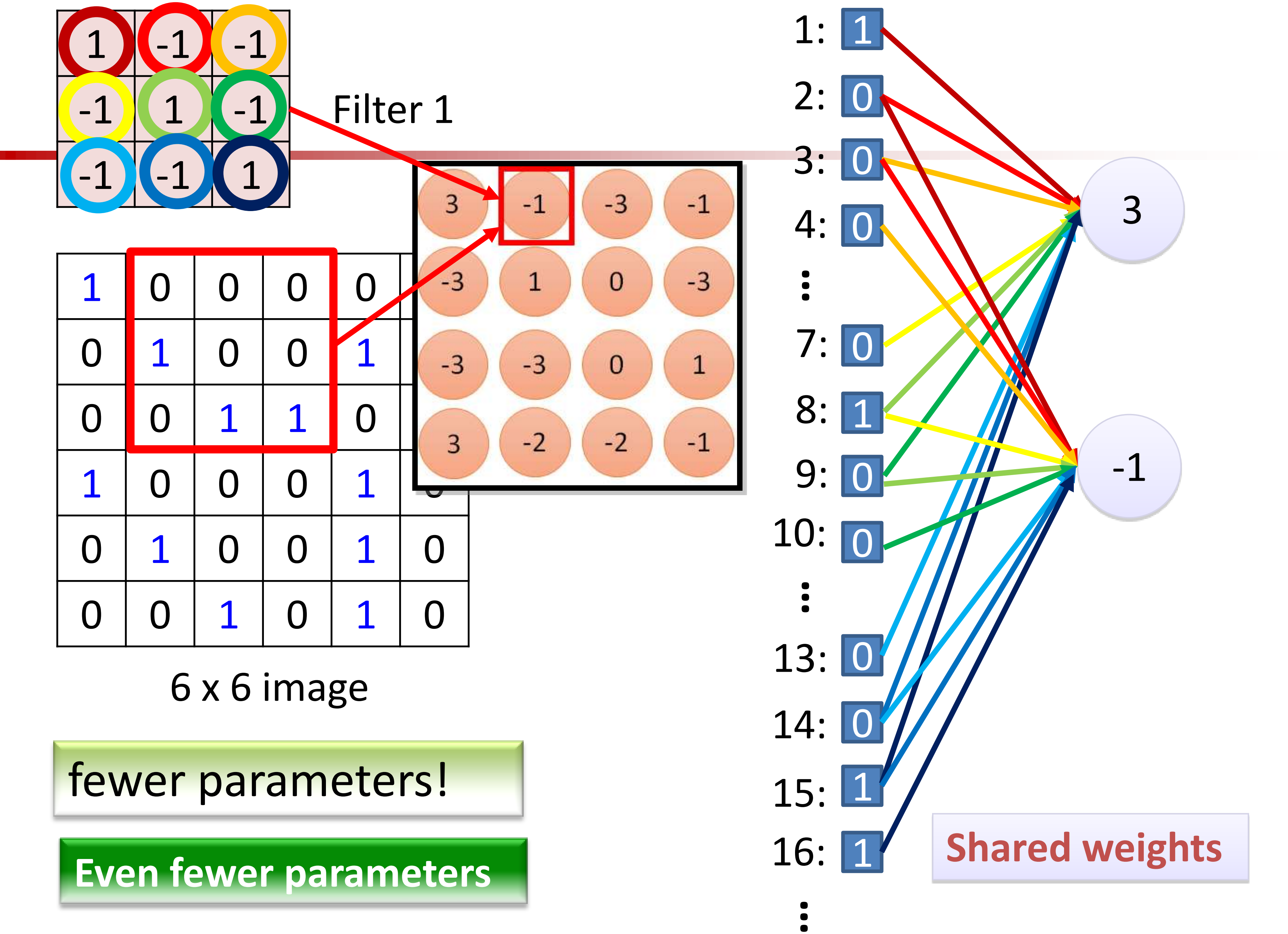


Fully-connected

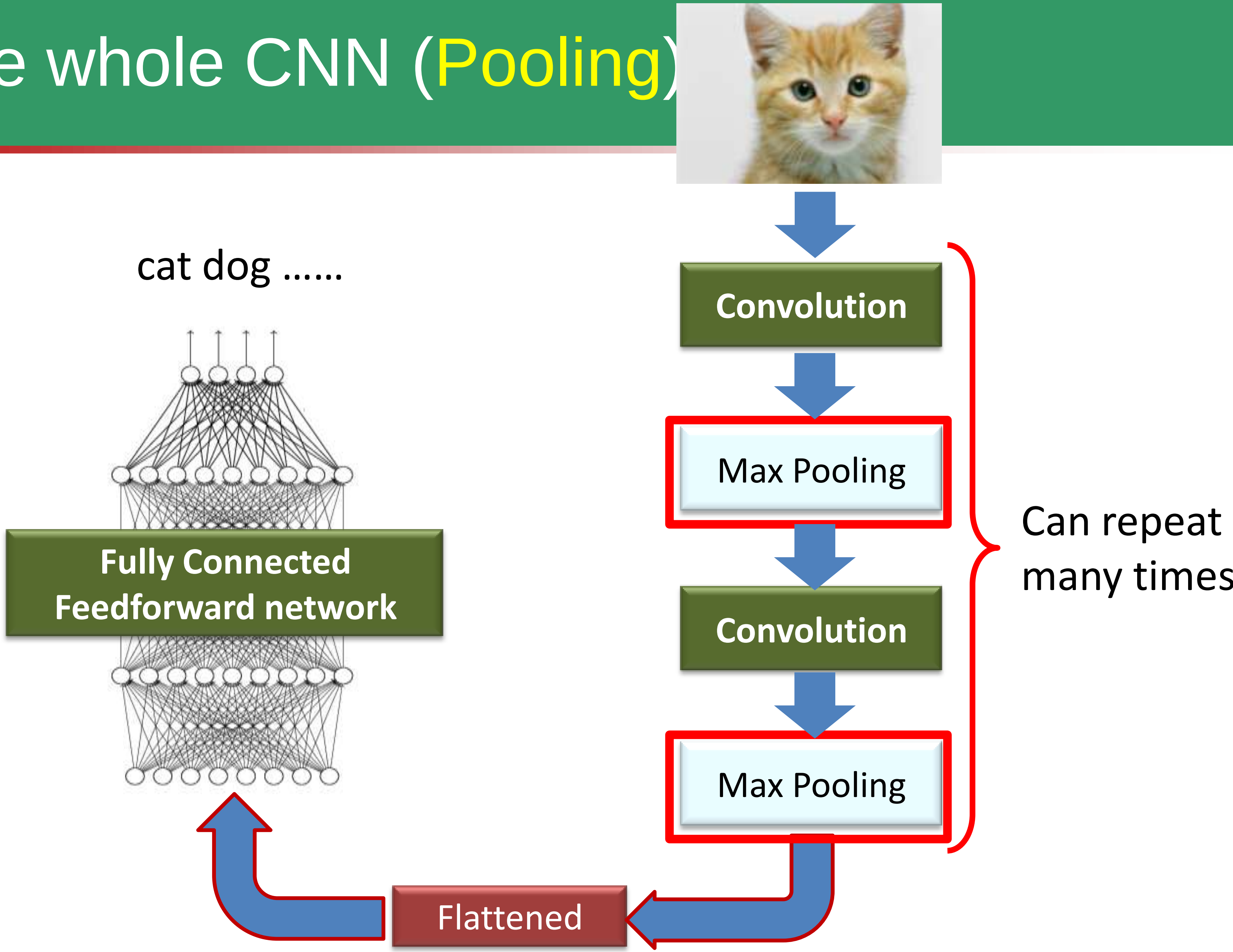




Filter 1
6 x 6 image
fewer parameters!
Only connect to 9 inputs, not fully connected
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3

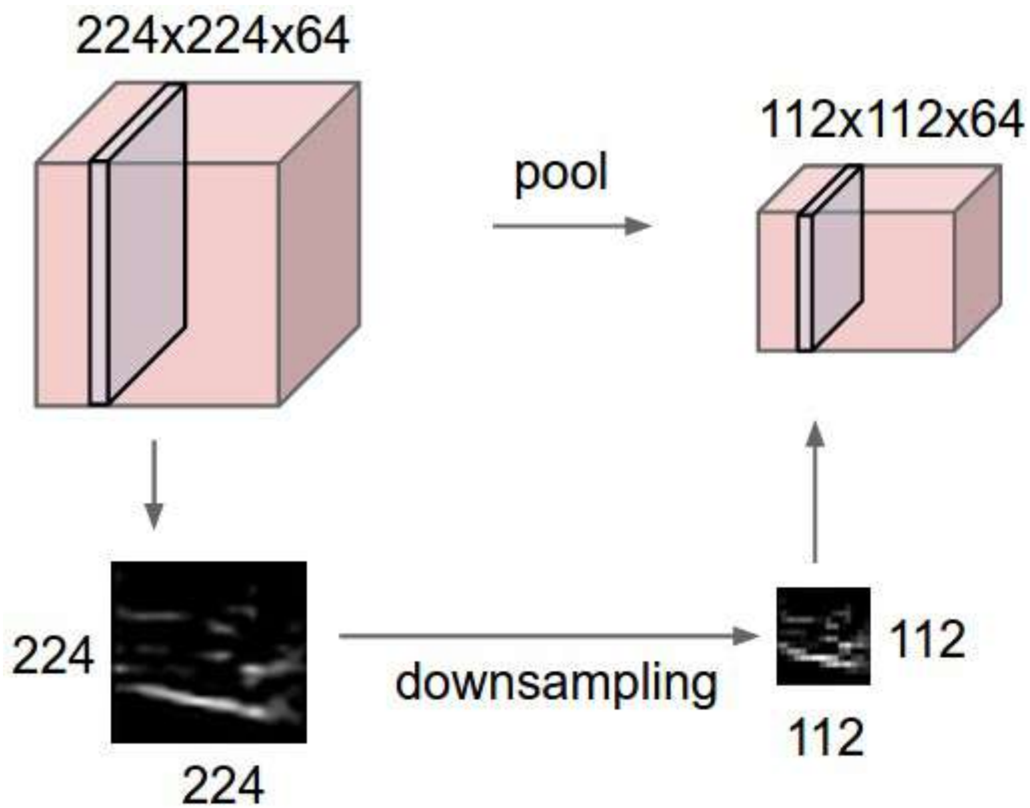

Filter 1
6 x 6 image
fewer parameters!
Even fewer parameters
1:
2:
3:
4:
7:
8:
9:
10:
13:
14:
15:
16:
Shared weights

The whole CNN (Pooling)



Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently



Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

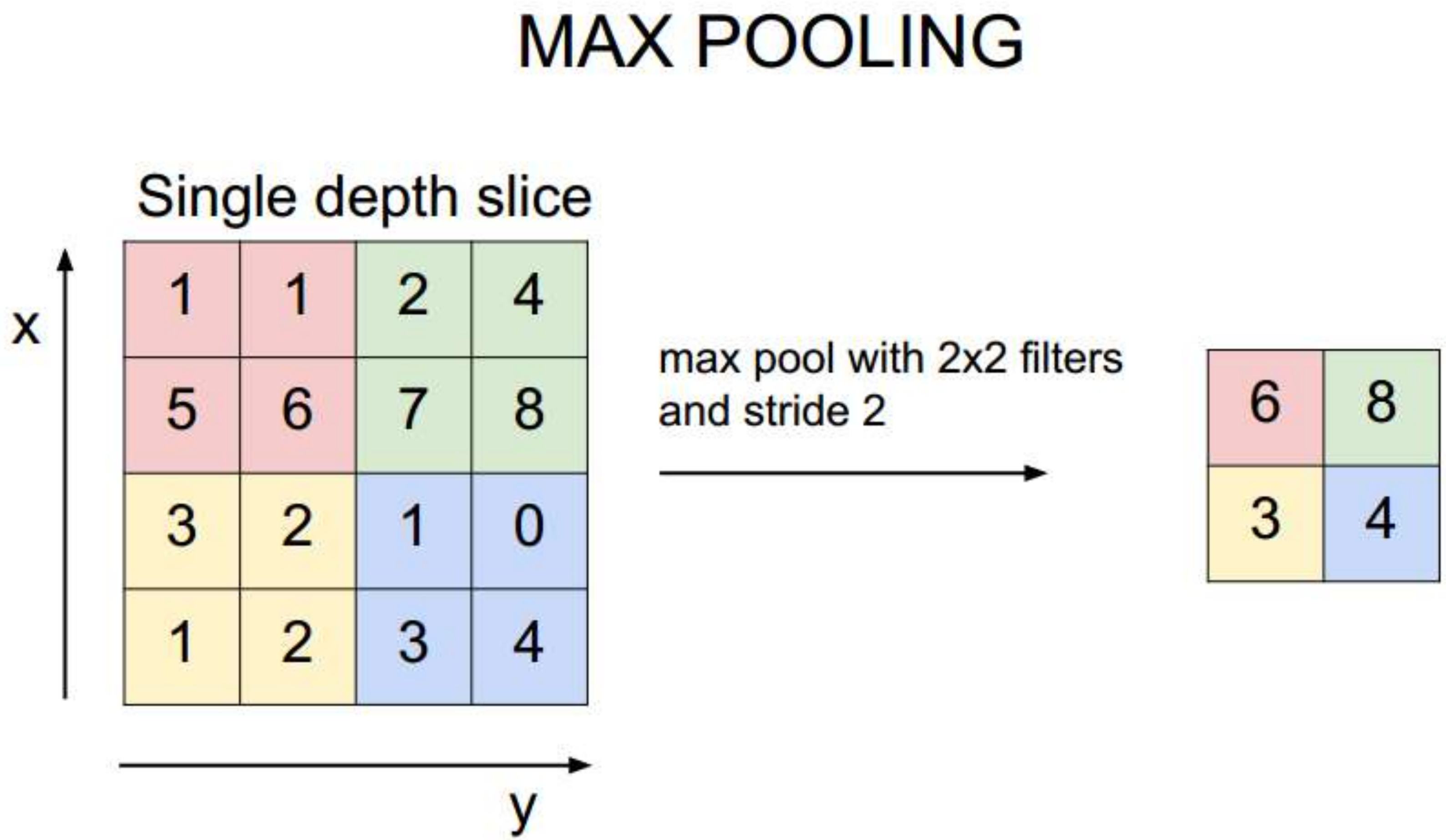
bird

We can subsample the pixels to make image smaller

→ fewer parameters to characterize the image

Max Pooling

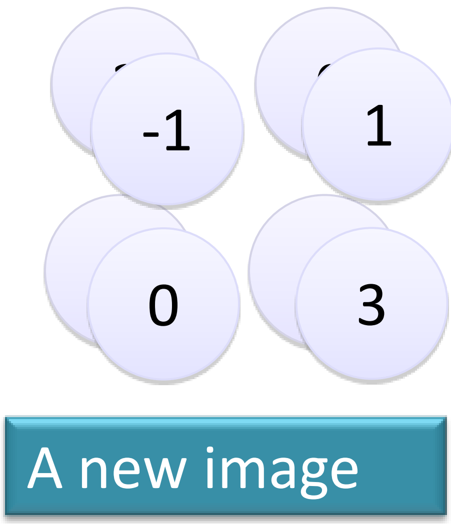
Max-pooling: a pooling unit simply outputs the max activation in the input region

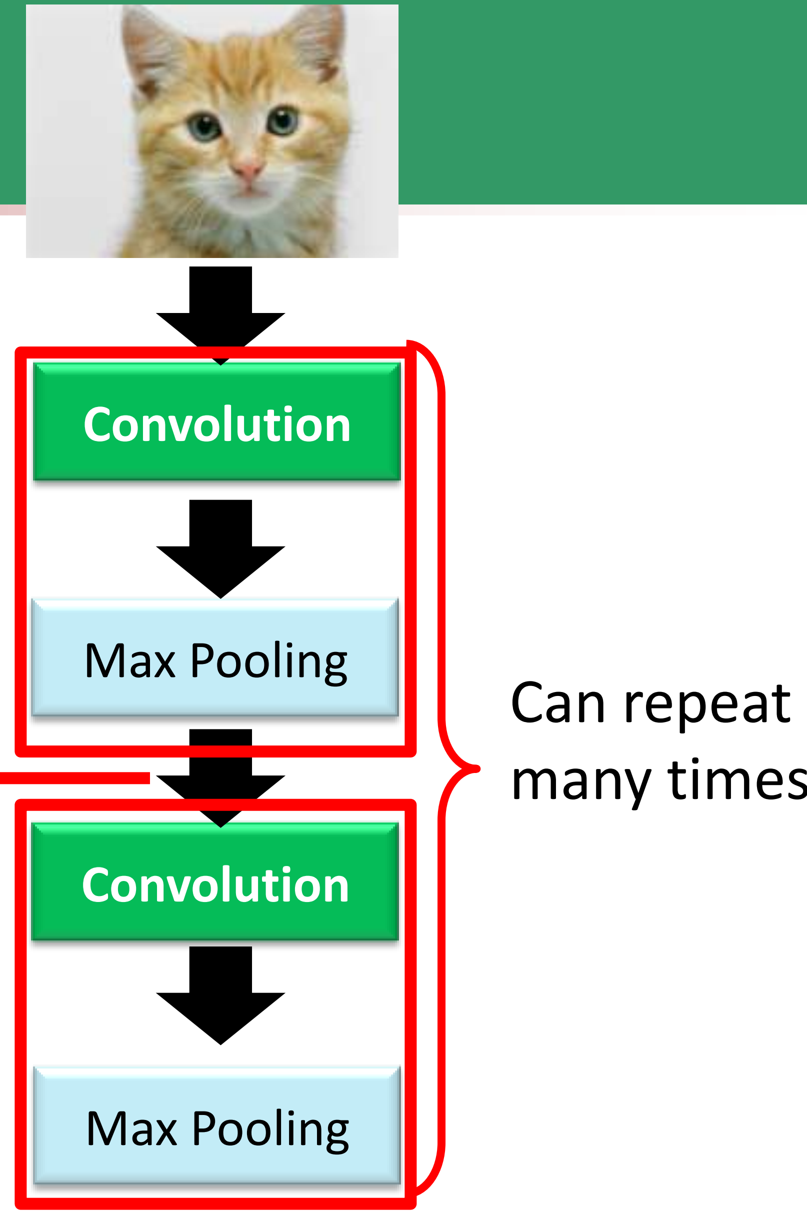


CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights
- Max pooling further reduces the complexity

The whole CNN





Smaller than the original image

The number of channels is the number of filters

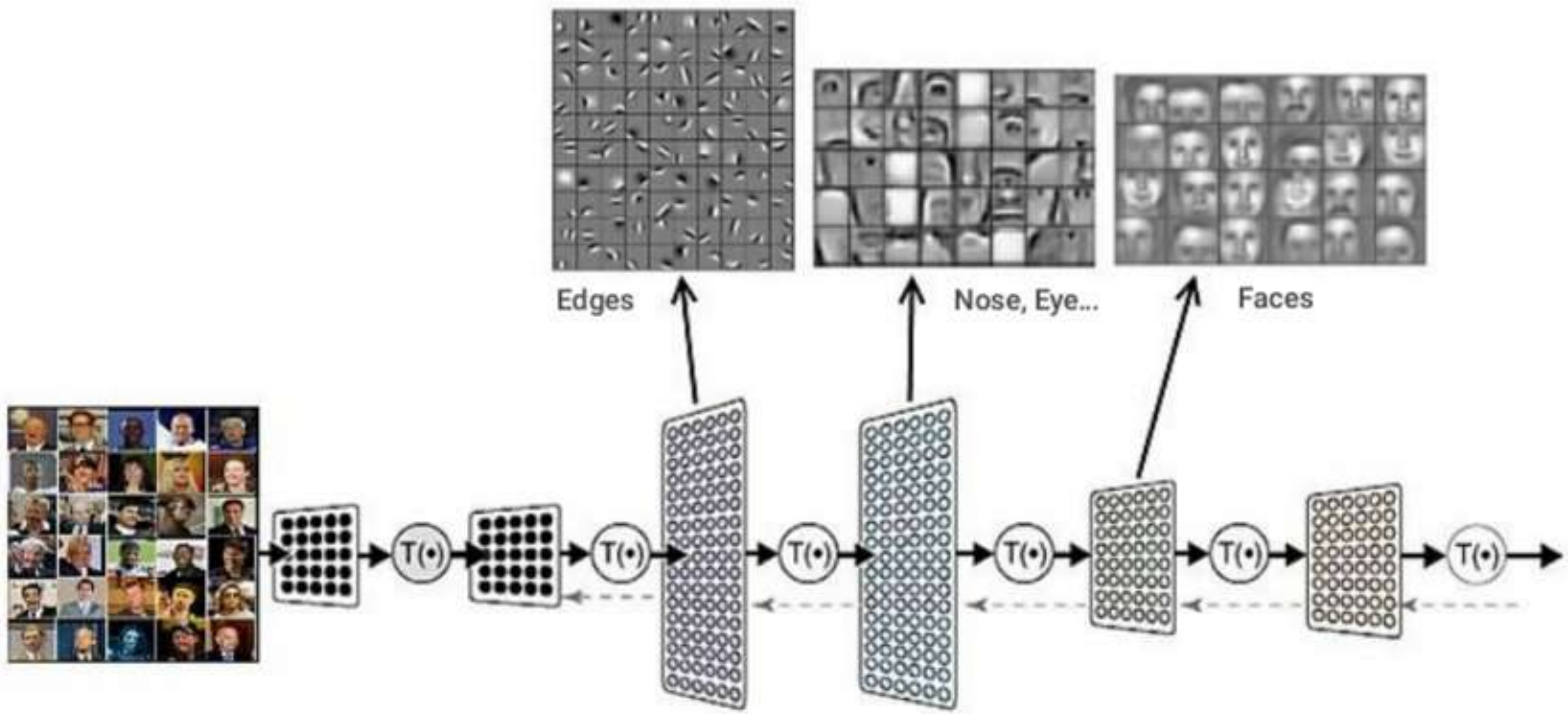
Convolutional Neural Network

- The first layer of a CNN detects high-level patterns like rough edges and curves.

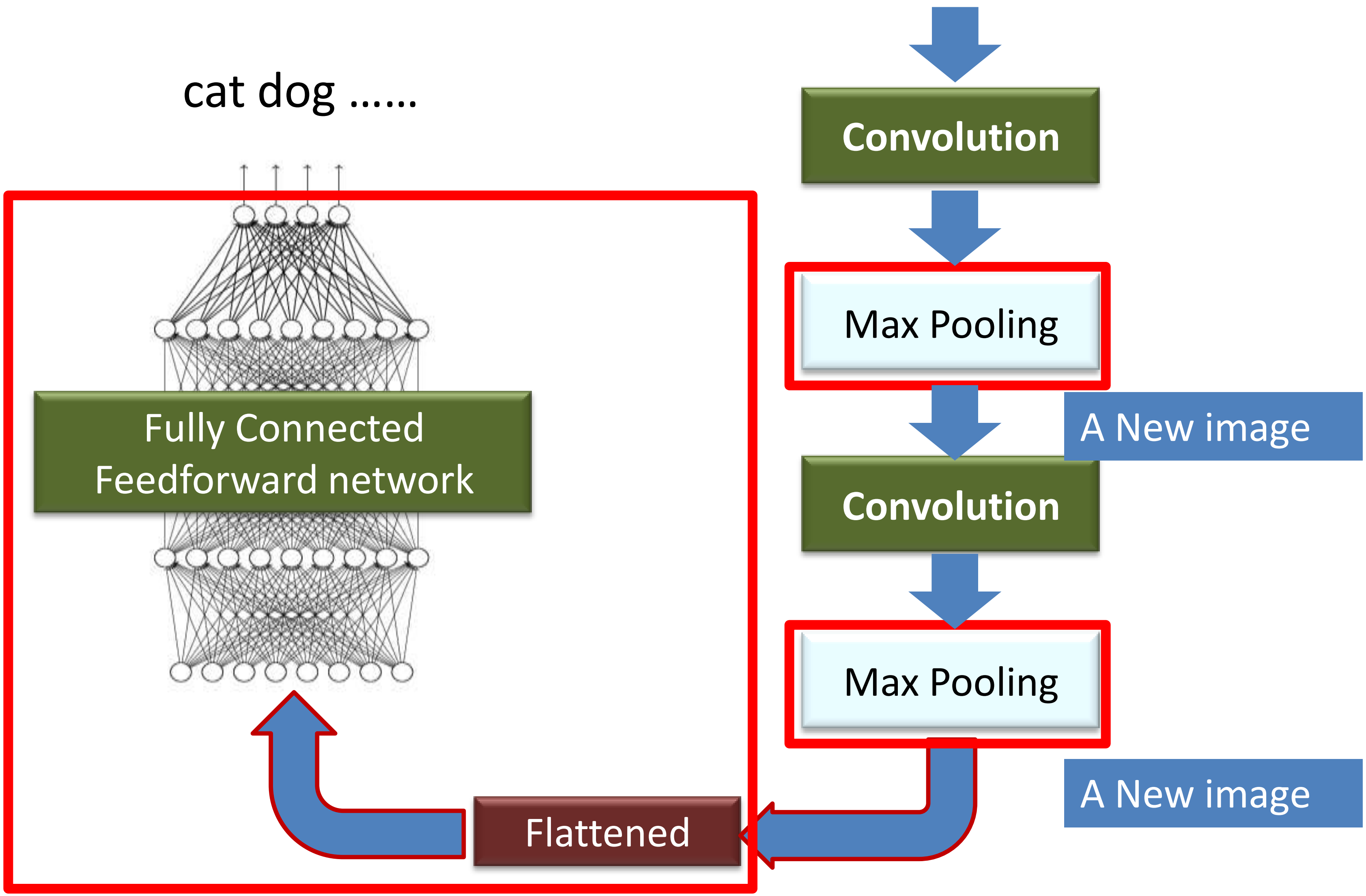
- How does a CNN know what to look for? This is done through a large amount of labeled training set.

- CNN updates its filter values and each iteration performs with slightly more accuracy.

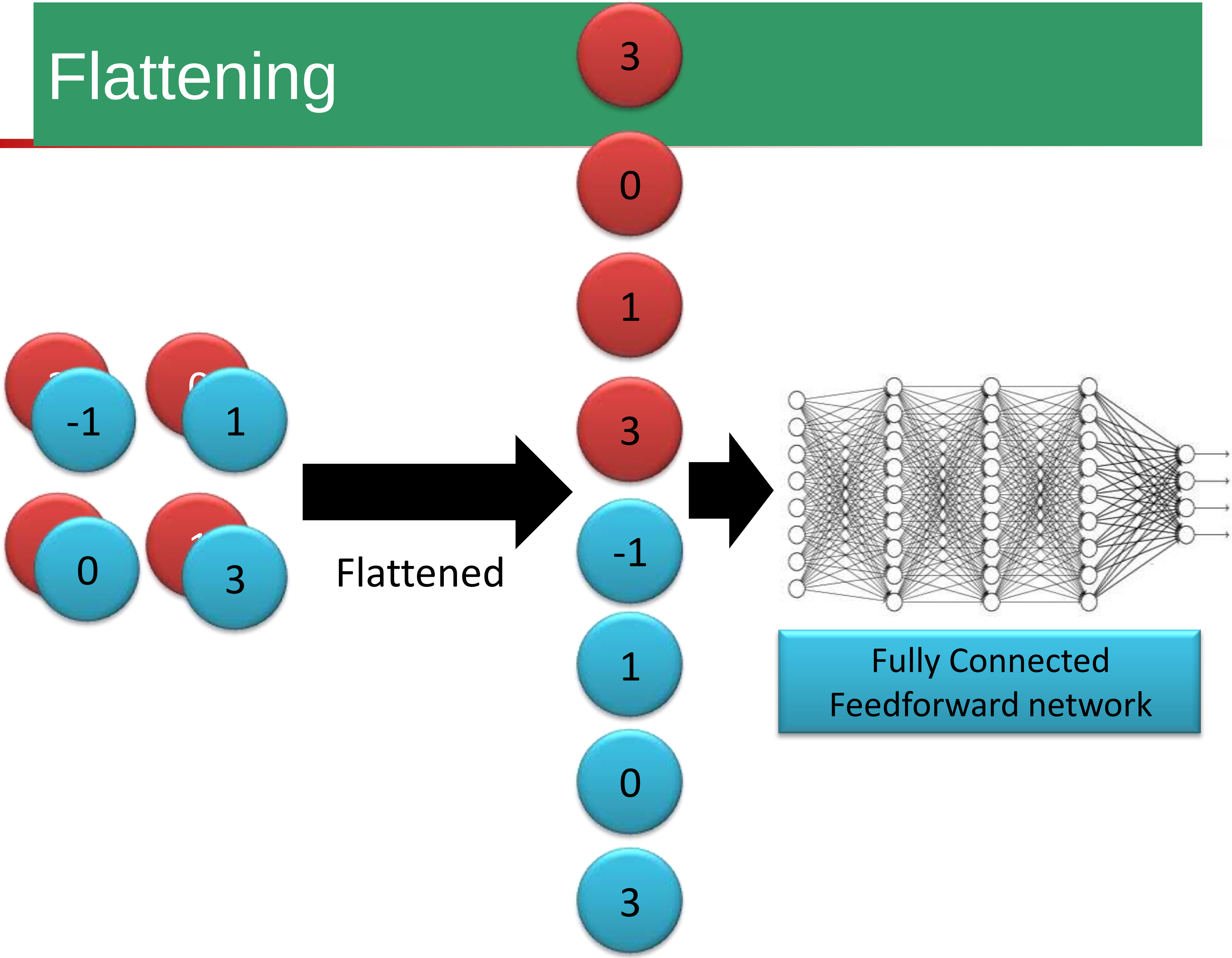
CNN & detected features



The whole CNN



Flattening

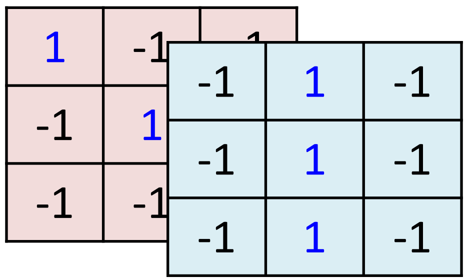


-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected Feedforward network

CNN in Keras

Only modified the **network structure** and **input format (vector -> 3-D tensor)**

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(28,28,1)) )
```



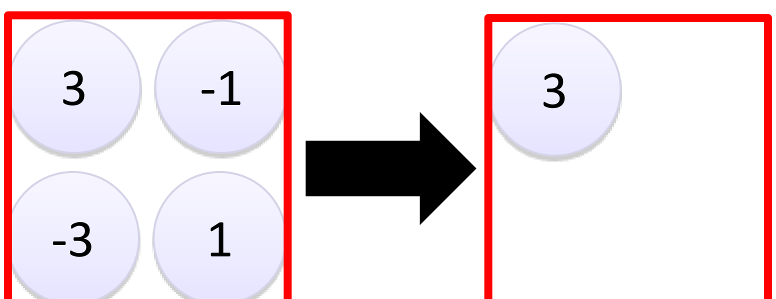
...... There are 25 3x3 filters.

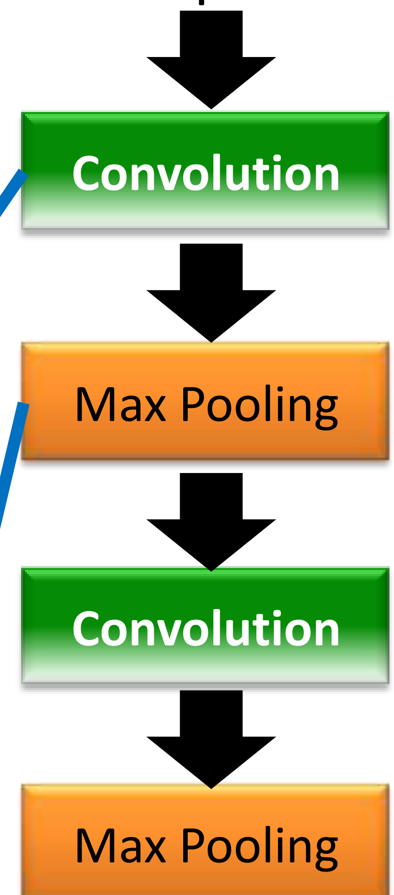
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

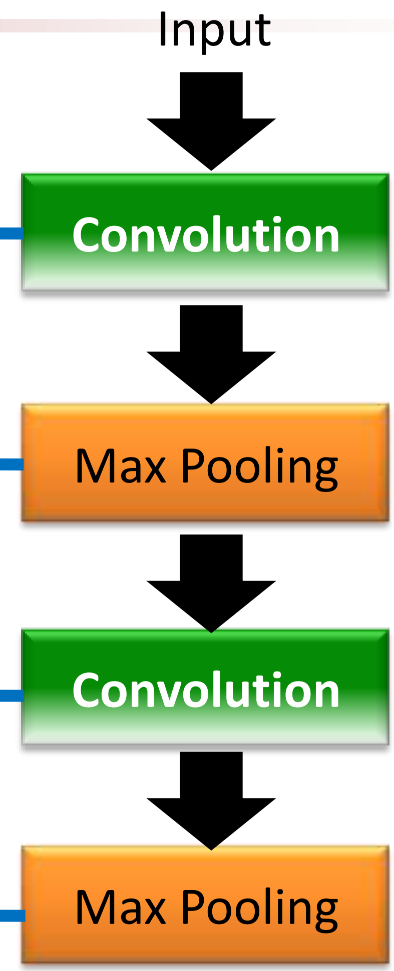
```
model2.add(MaxPooling2D((2,2)))
```





CNN in Keras

Only modified the **network structure** and **input format (vector -> 3-D array)**



1 x 28 x 28

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(28,28,1)) )
```

How many parameters for each filter?

9

25 x 26 x 26

```
model2.add(MaxPooling2D((2,2)))
```

25 x 13 x 13

```
model2.add(Convolution2D(50,3,3))
```

How many parameters for each filter?

225= 25x9

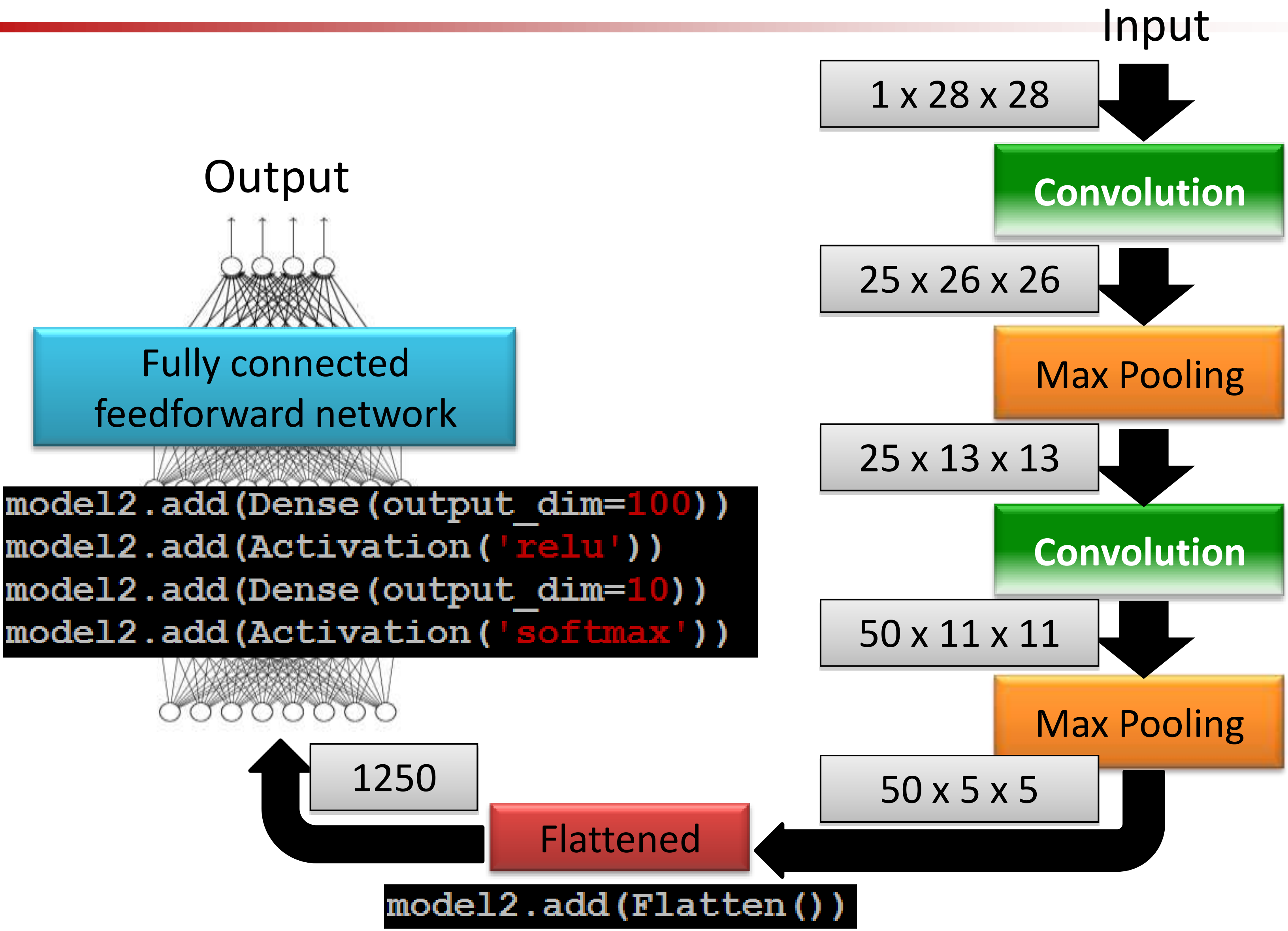
50 x 11 x 11

```
model2.add(MaxPooling2D((2,2)))
```

50 x 5 x 5

CNN in Keras

Only modified the **network structure** and **input format (vector -> 3-D array)**




```
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
```

```
model2.add(Flatten())
```

AlphaGo

19 x 19 matrix

Black: 1

white: -1

none: 0

Neural Network

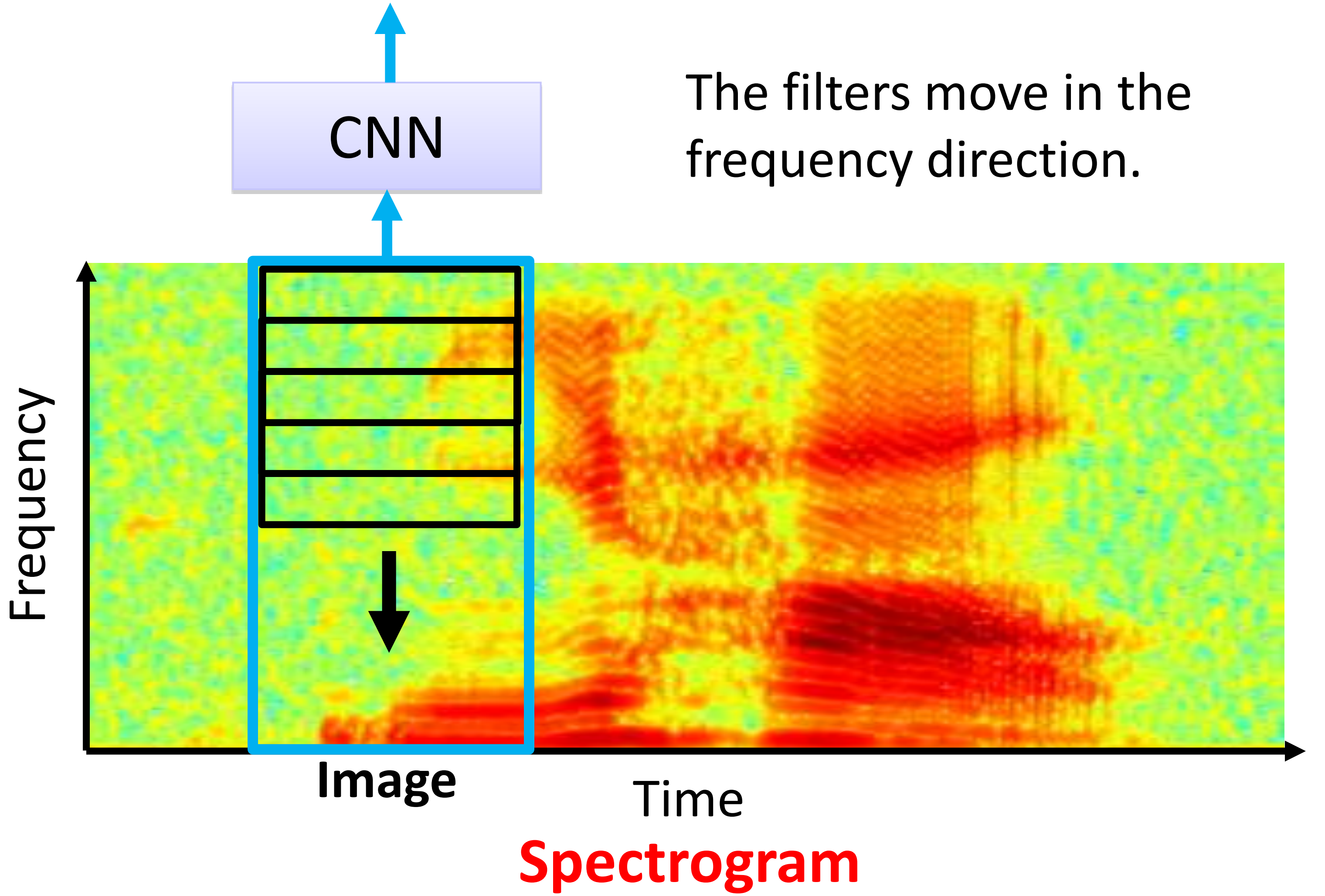
Next move

(19 x 19 positions)

Fully-connected feedforward network can be used

But CNN performs much better

CNN in speech recognition



The filters move in the frequency direction.