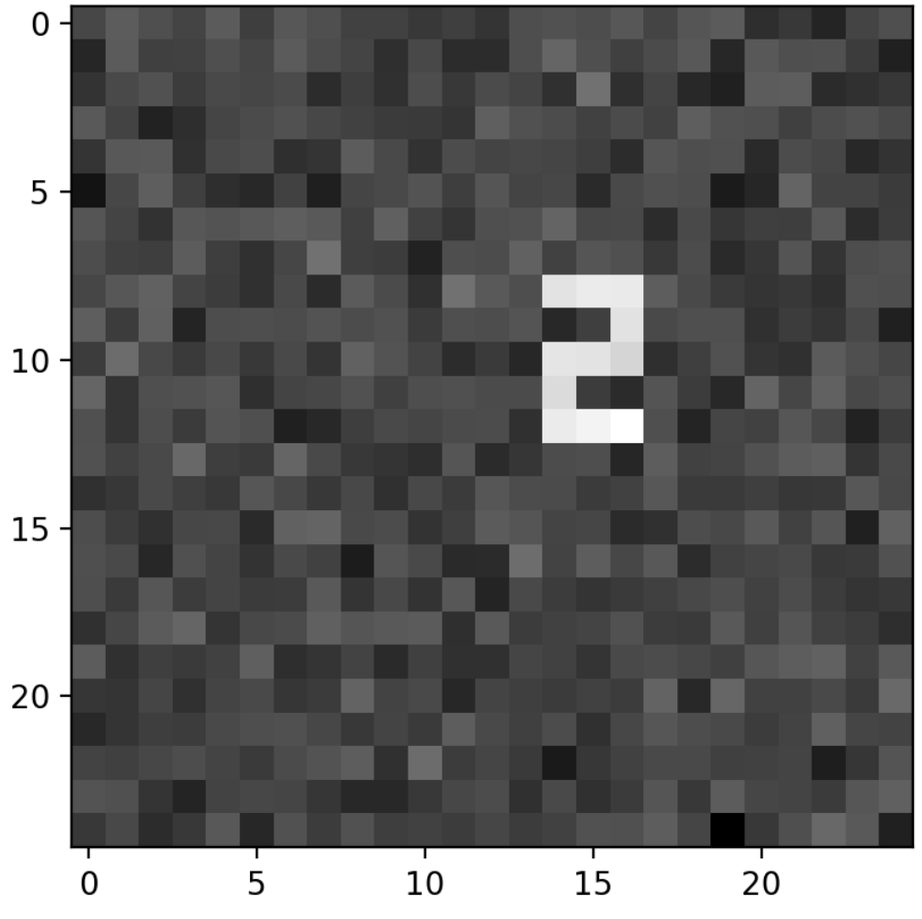
CS 453X: Class 24

Jacob Whitehill

Convolution

Motivation: object detection

- Suppose we wanted to classify an unknown digit of unknown location in an image.



Motivation: object detection

- One simple strategy is based on an old computer vision technique known as template matching:
  - We create a template image that equals one of the objects we want to detect.
  - We “slide” the template across every location in the image and see where the image & template best “match”.

Convolution

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

Image

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

Template

Output

- Without exceeding the image boundaries, how many 2-D locations are there at which the template can be superimposed with the image?

Convolution

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

Image

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

Template

Output

- Without exceeding the image boundaries, how many 2-D locations are there at which the template can be superimposed with the image?

4 rows, 6 columns

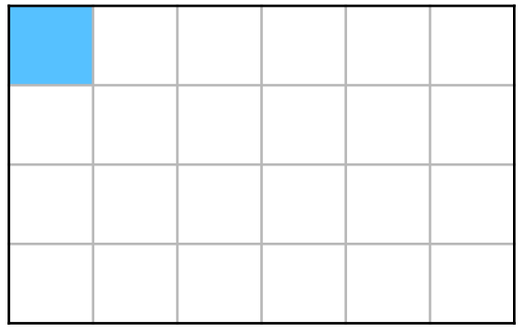
Convolution

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

Image

	-1	0	+1
-2	1	1	1
-1	0	0	1
0	1	1	1
+1	1	0	0
+2	1	1	1

Template



Output

- For simplicity of notation, let's renumber the indices of the elements of the template.

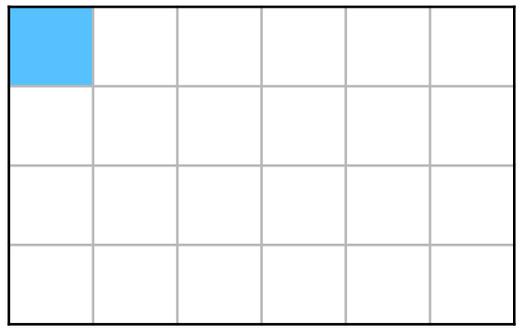
Convolution

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

Image

	-1	0	+1
-2	1	1	1
-1	0	0	1
0	1	1	1
+1	1	0	0
+2	1	1	1

Template



Output

- The output of the “template matching” at (*r*, *c*) is then the sum of products between each template pixel and the corresponding image pixel:

$$\mathrm{TM}(r,c) = \sum_{i=-t_h/2}^{+t_h/2} \sum_{j=-t_w/2}^{+t_w/2} \mathrm{im}[r+i, c+j]\mathrm{t}[i,j]$$

where t_h, t_w are the height and width of the template.

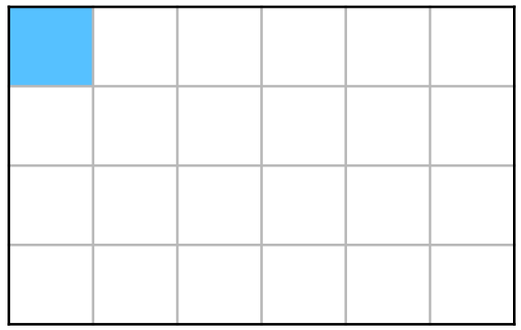
Convolution

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

Image

	-1	0	+1
-2	1	1	1
-1	0	0	1
0	1	1	1
+1	1	0	0
+2	1	1	1

Template



Output

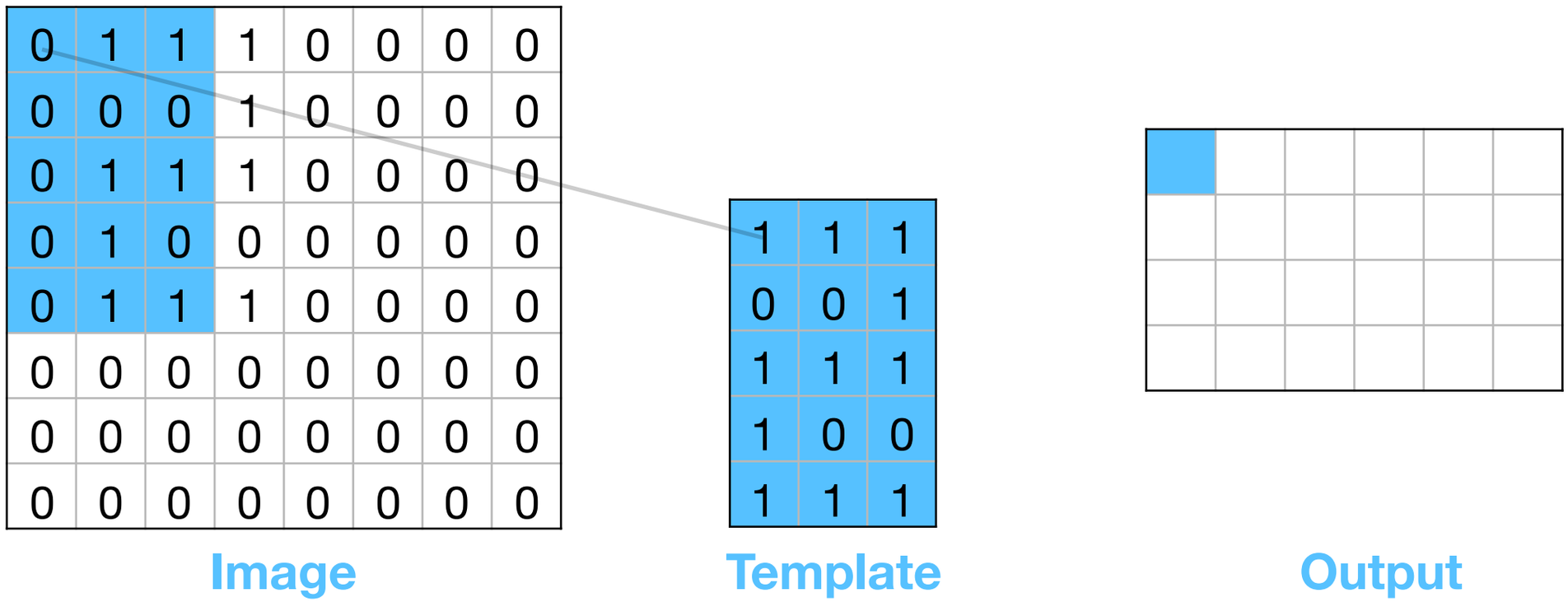
- This can be thought of as a 2-D “dot product” between the image and the template. It is also known as a 2-D **convolution**.*

$$\mathrm{TM}(r,c) = \sum_{i=-t_h/2}^{+t_h/2} \sum_{j=-t_w/2}^{+t_w/2} \mathrm{im}[r+i, c+j]\mathrm{t}[i,j]$$

where t_h, t_w are the height and width of the template.

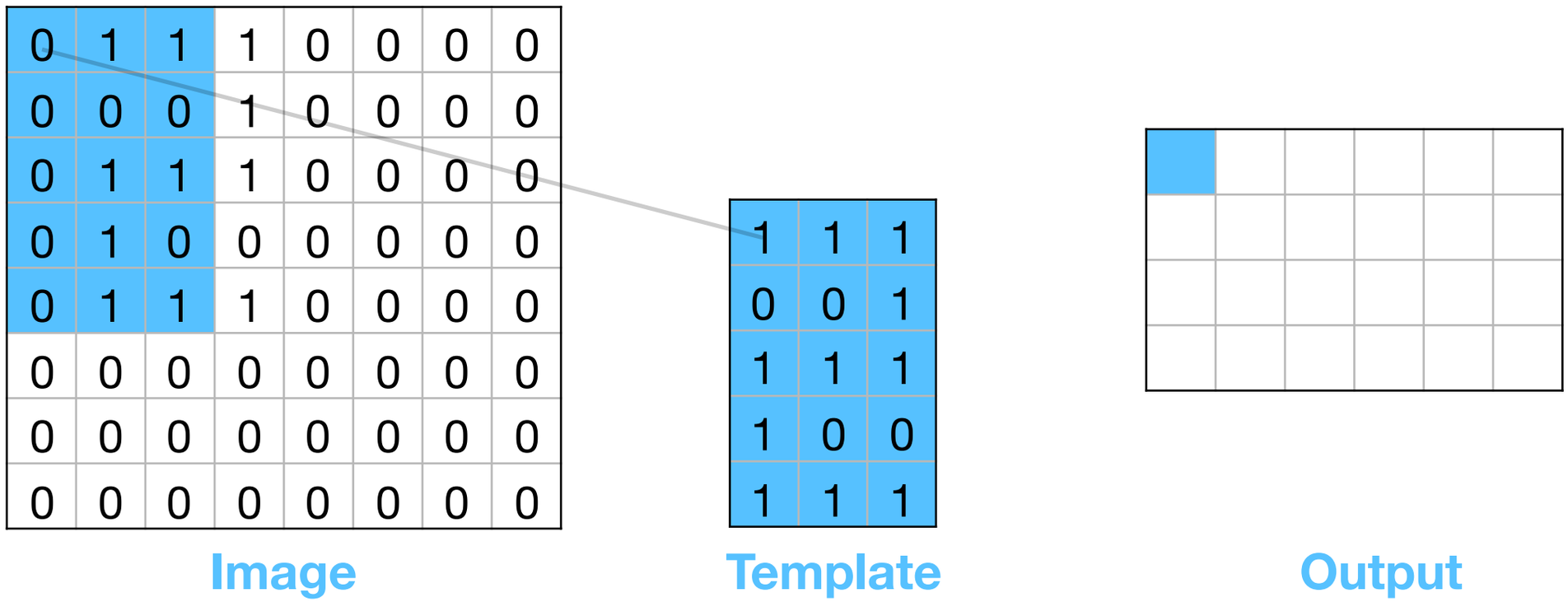
*Technically it is a cross-correlation; convolution would be *r-i*, *c-j*. But with NNs, the difference is not important.

Convolution



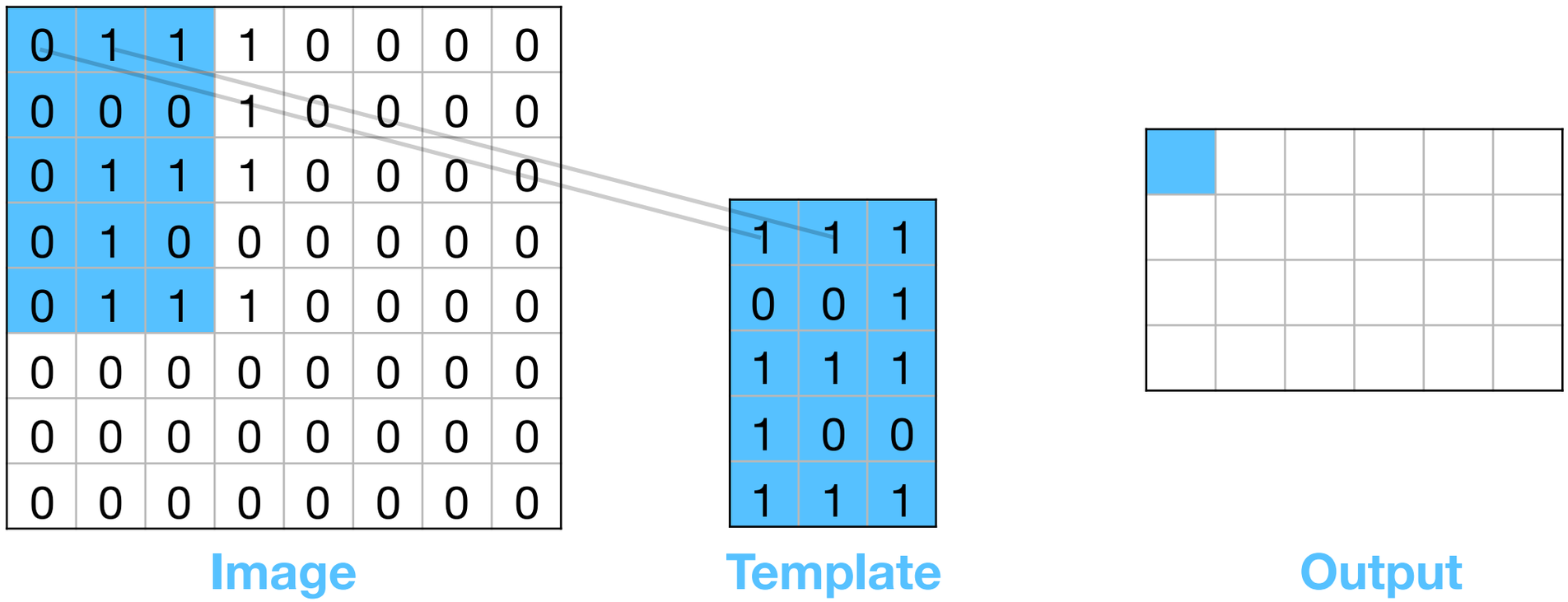
- Here's an illustration of how we construct the entire output image by applying a template to each location of the input image…

Convolution



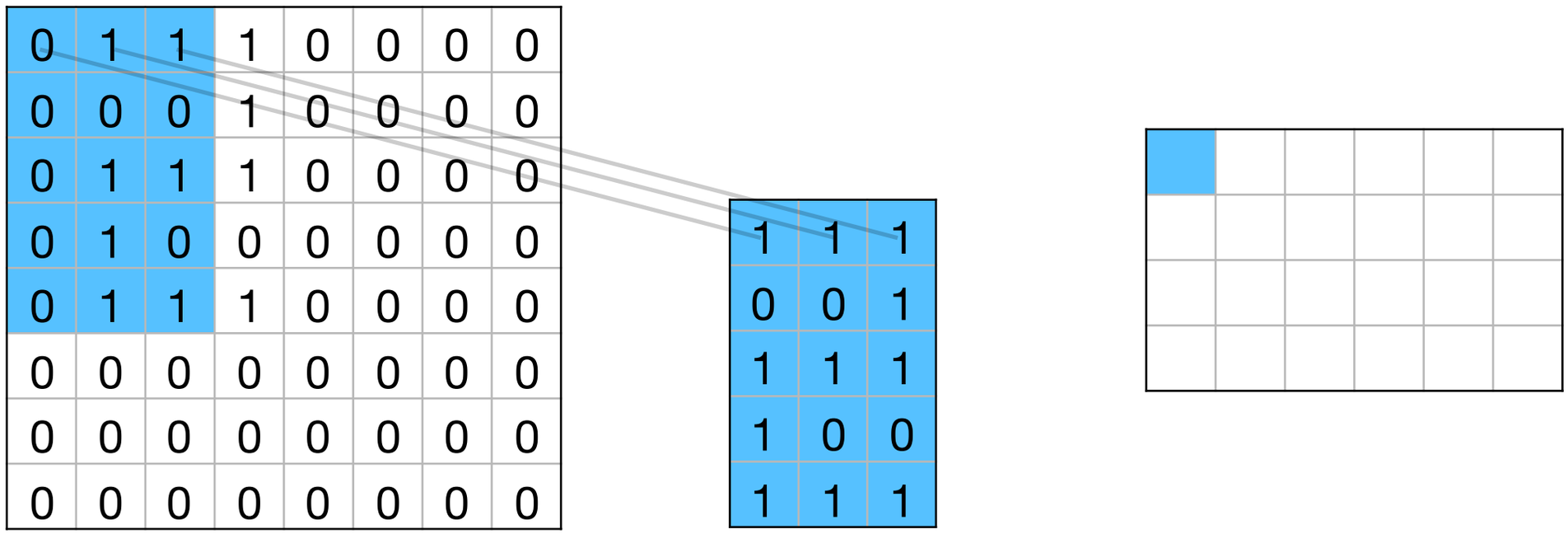
0*1

Convolution



0*1+1*1

Convolution



0*1+1*1+1*1

Convolution

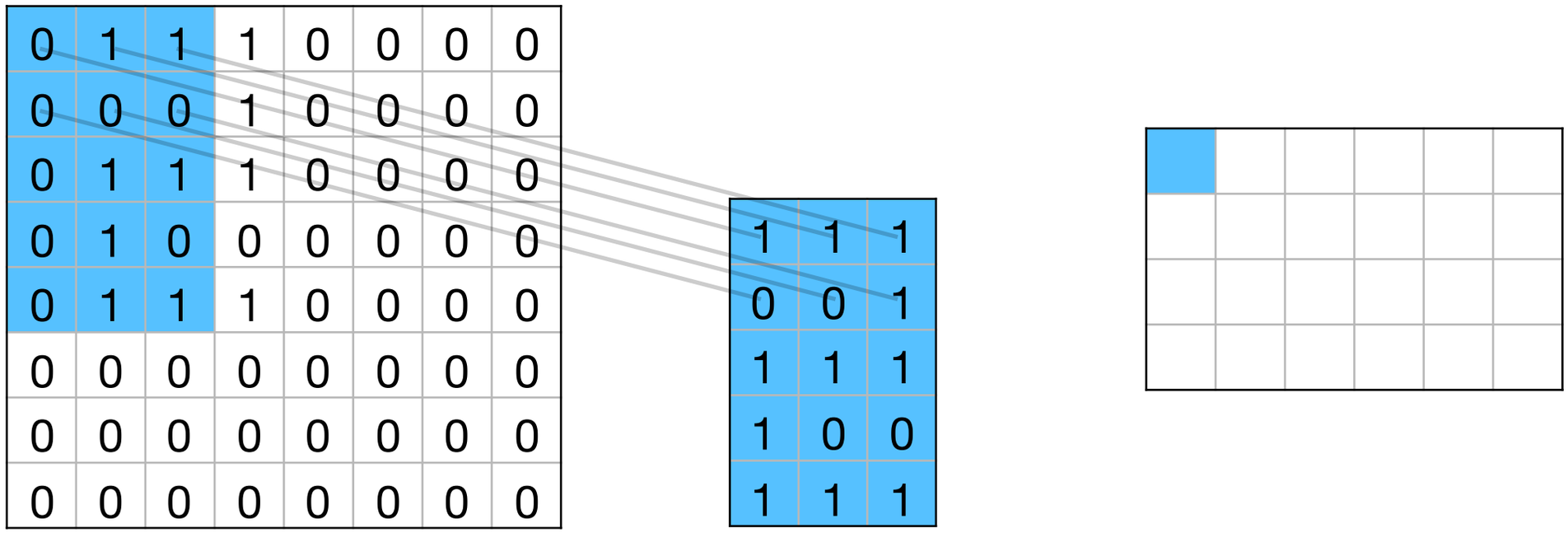


Image Template Output

0*1+1*1+1*1+
0*0+0*0+0*1

Convolution

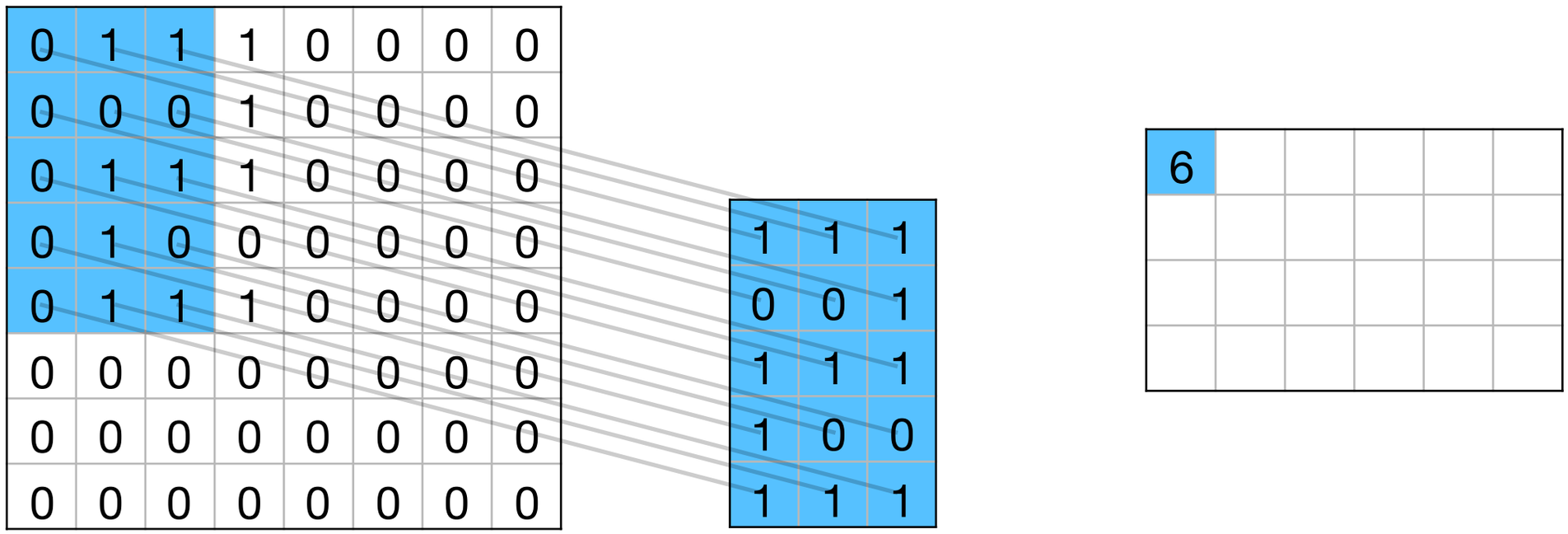


Image Template Output

0*1+1*1+1*1+
0*0+0*0+0*1+
0*1+1*1+1*1+
0*1+1*0+0*0+
0*1+1*1+1*1
=6

Convolution

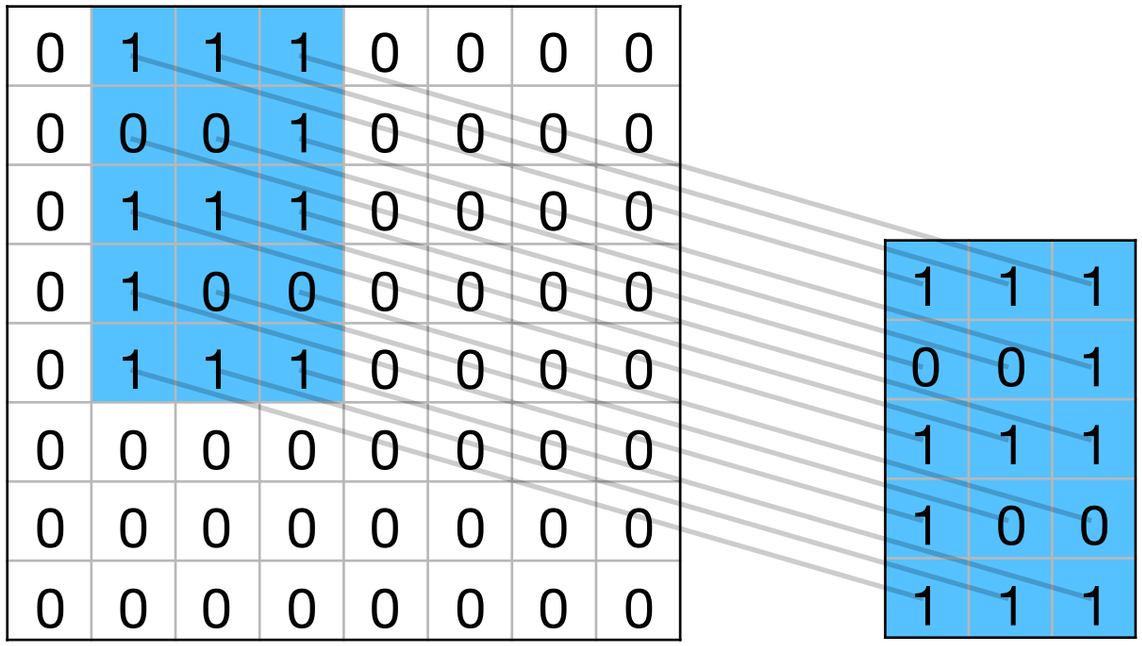


Image Template Output

1*1+1*1+1*1+
0*0+0*0+1*1+
1*1+1*1+1*1+
1*0+1*0+0*0+
1*1+1*1+1*1
=11

Convolution: exercise

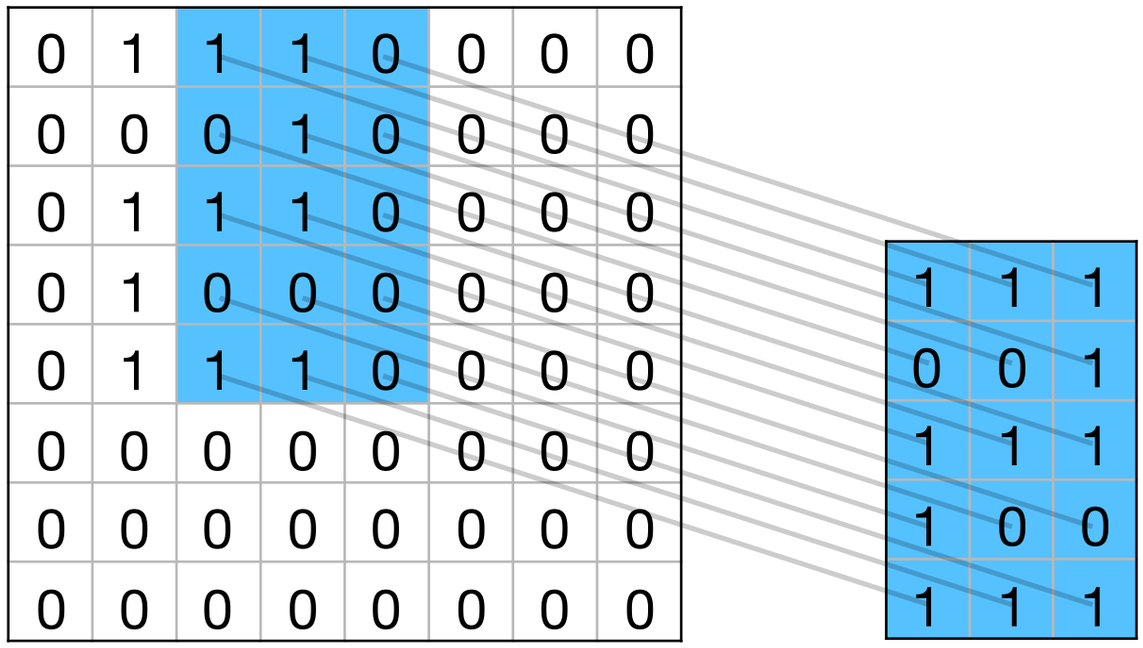


Image Template Output

Convolution: exercise

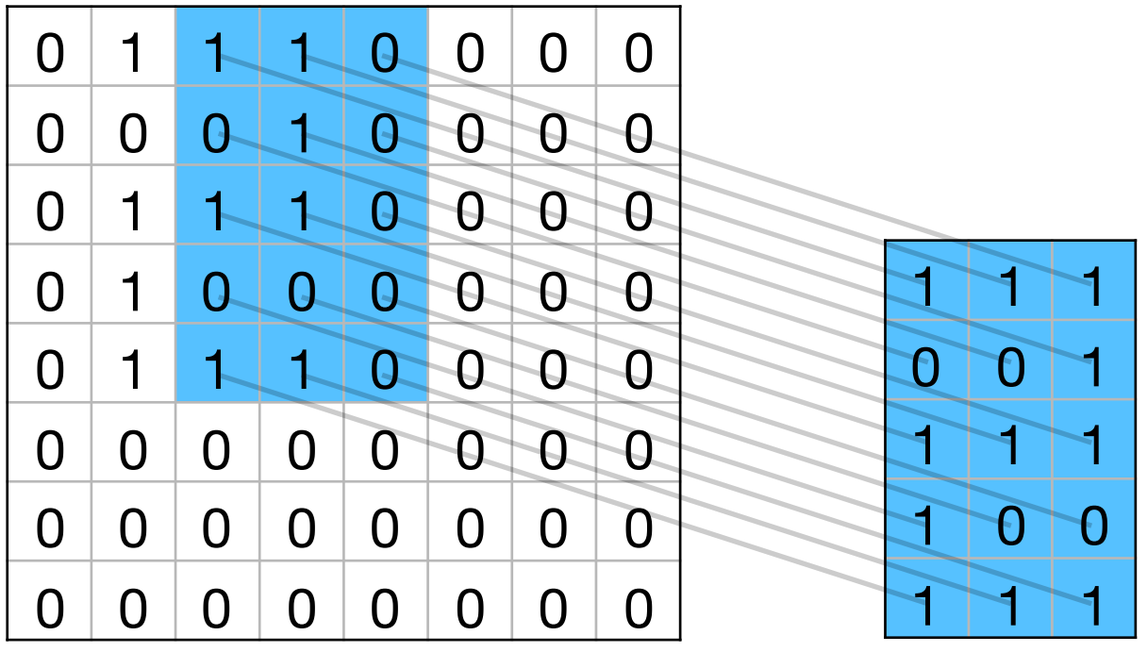


Image Template Output

1*1+1*1+0*1+
0*0+1*0+0*1+
1*1+1*1+0*1+
0*1+0*0+0*0+
1*1+1*1+0*1
=6

Convolution

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

Image

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

Pooling

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

Image

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

- The output image now expresses how much each location in the input image “looks like” the template.

Pooling

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

Image

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

11

Max-pool

- The output image now expresses how much each location in the input image “looks like” the template.
- To classify the object in the image (without caring where it is), we can just compute the *maximum* of the output.
- This is called a **maximum pool** (max-pool).

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Construct a template for each digit class (0-9).

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

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

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

...

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

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:
  - Convolve the image with each template.

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

Image

Class 0

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

Template

5	10	5	…	0	0
…					0
				0	0
			…	0	0

Output

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:
  - Convolve the image with each template.

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

Image

Class 1

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

Template

4	3	4	…	0	0
…					0
				0	0
			…	0	0

Output

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:
  - Convolve the image with each template.

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

Image

Class 2

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:
  - Convolve the image with each template.

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

Image

...

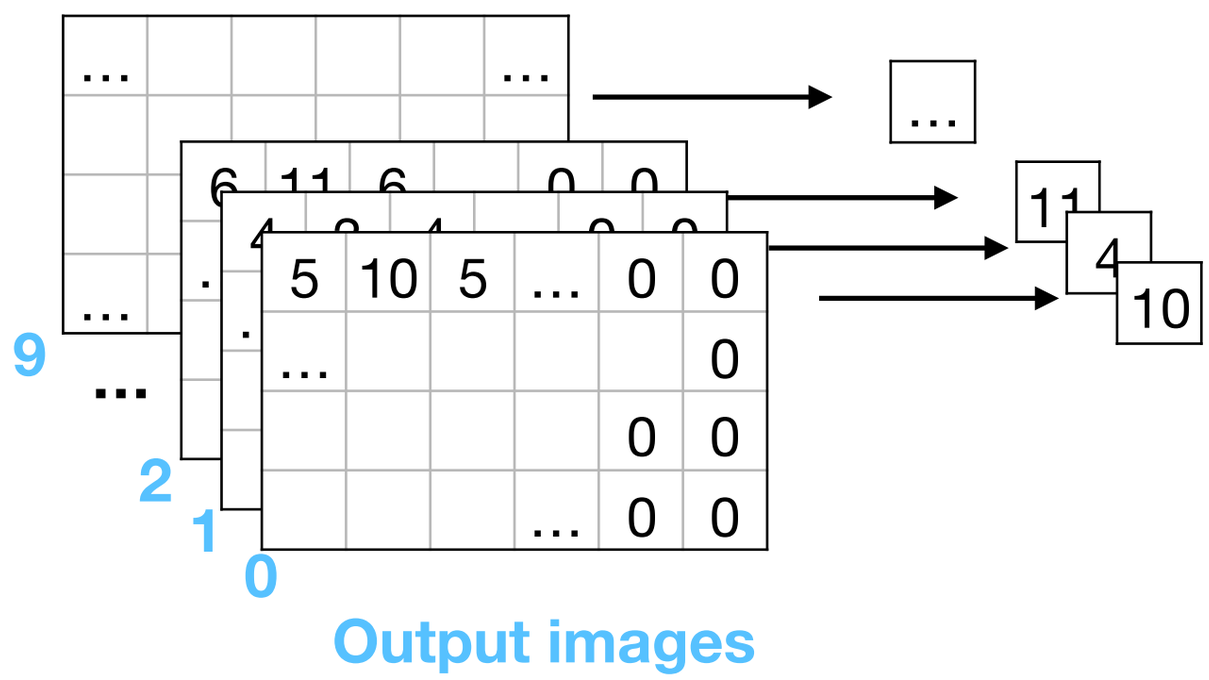
Template

...					...
...					...

Output

Convolution for classification

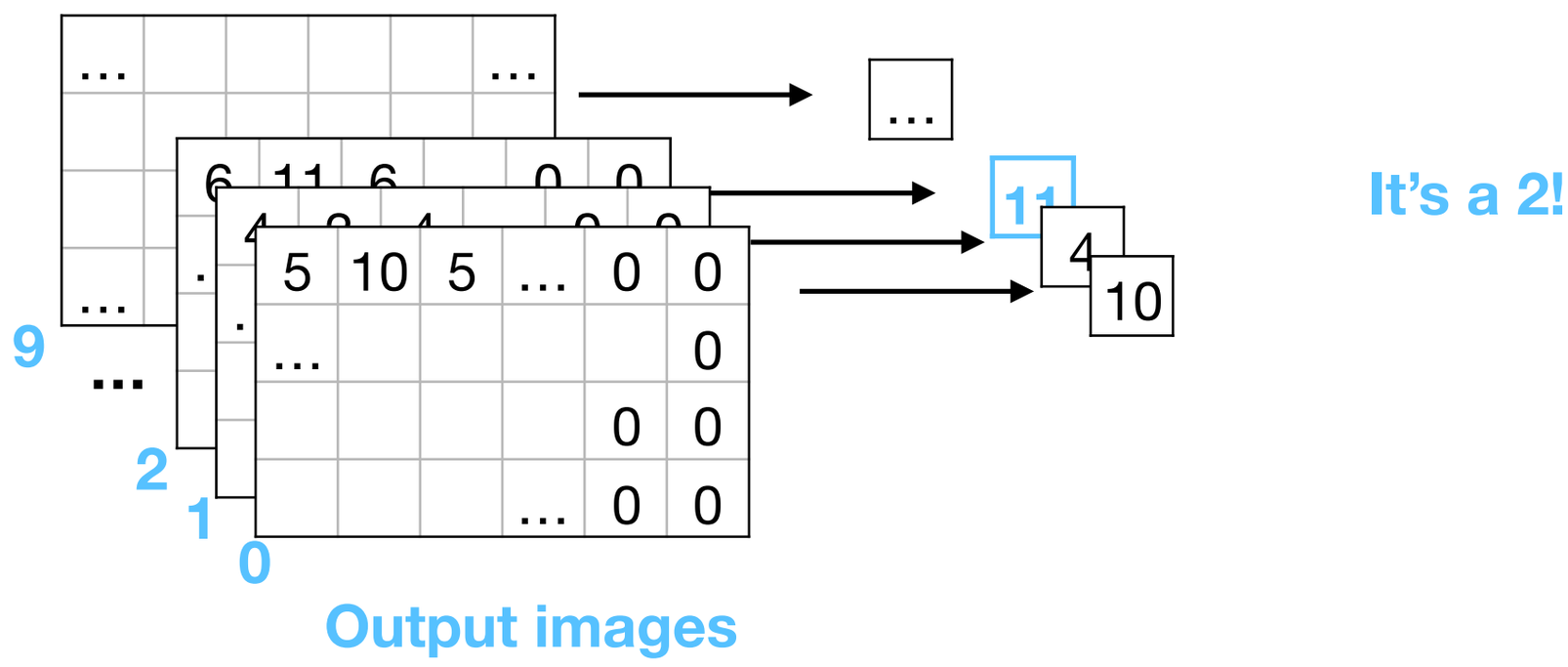
- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:
- Compute the maximum over each output image.



Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Predict the class whose max-pool value was largest.



Convolution: details

- The “templates” are more commonly known as **filters** or **kernels**.

- The output image is more commonly known as a **filter response** or a **feature map**.

8 x 8

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

Image

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

Kernel/ filter

4 x 6

Response/ Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

8 x 8

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

Image

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

Kernel

4 x 6

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- To preserve the image size, we can **pad** the input image:

10 x 10 (with padding)

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

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

Kernel

8 x 8

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

10 x 10 (with padding)

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

Kernel

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

8 x 8

2							

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

10 x 10 (with padding)

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

Kernel

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

8 x 8

2	4						

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

10 x 10 (with padding)

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

Kernel

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

8 x 8

2	4	6					

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

10 x 10 (with padding)

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

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

Kernel

8 x 8

2	4	6					
			…				
							0

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

10 x 10 (with padding)

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

Kernel

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

4 x 4 (with stride *k*=2)

2			

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

10 x 10 (with padding)

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

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

Kernel

4 x 4 (with stride *k*=2)

2	6		

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

10 x 10 (with padding)

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

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

Kernel

4 x 4 (with stride *k*=2)

2	6	3	

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

10 x 10 (with padding)

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

Kernel

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

4 x 4 (with stride *k*=2)

2	6	3	…
3			

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

10 x 10 (with padding)

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

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

Kernel

4 x 4 (with stride *k*=2)

2	6	3	…
3			
			…

Feature map

Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.

Pooling: details

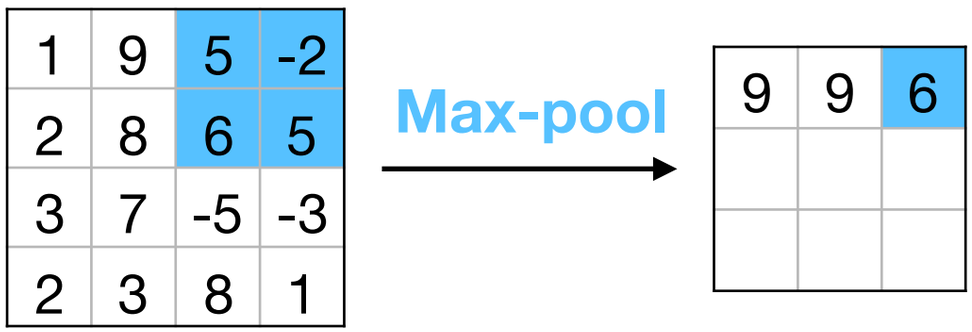
- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.

Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.



Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.

Pooling: details

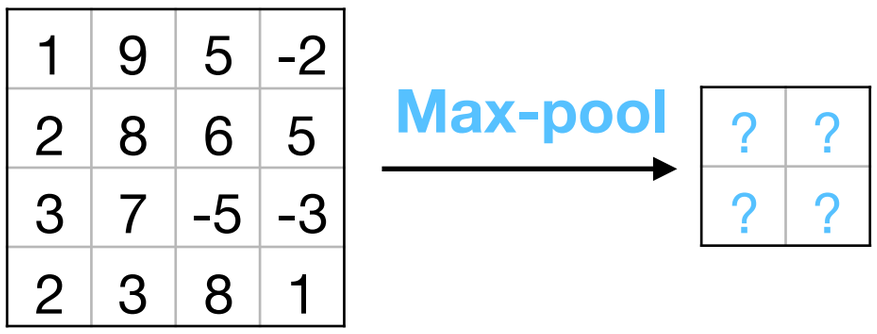
- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.

Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

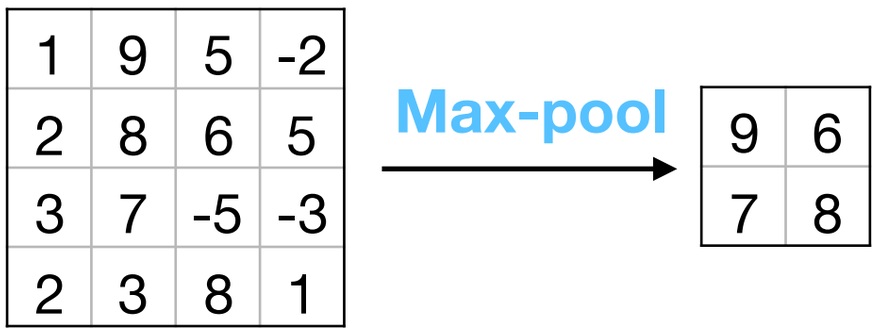
- Example: *w*=2, *k*=2.



Pooling: details

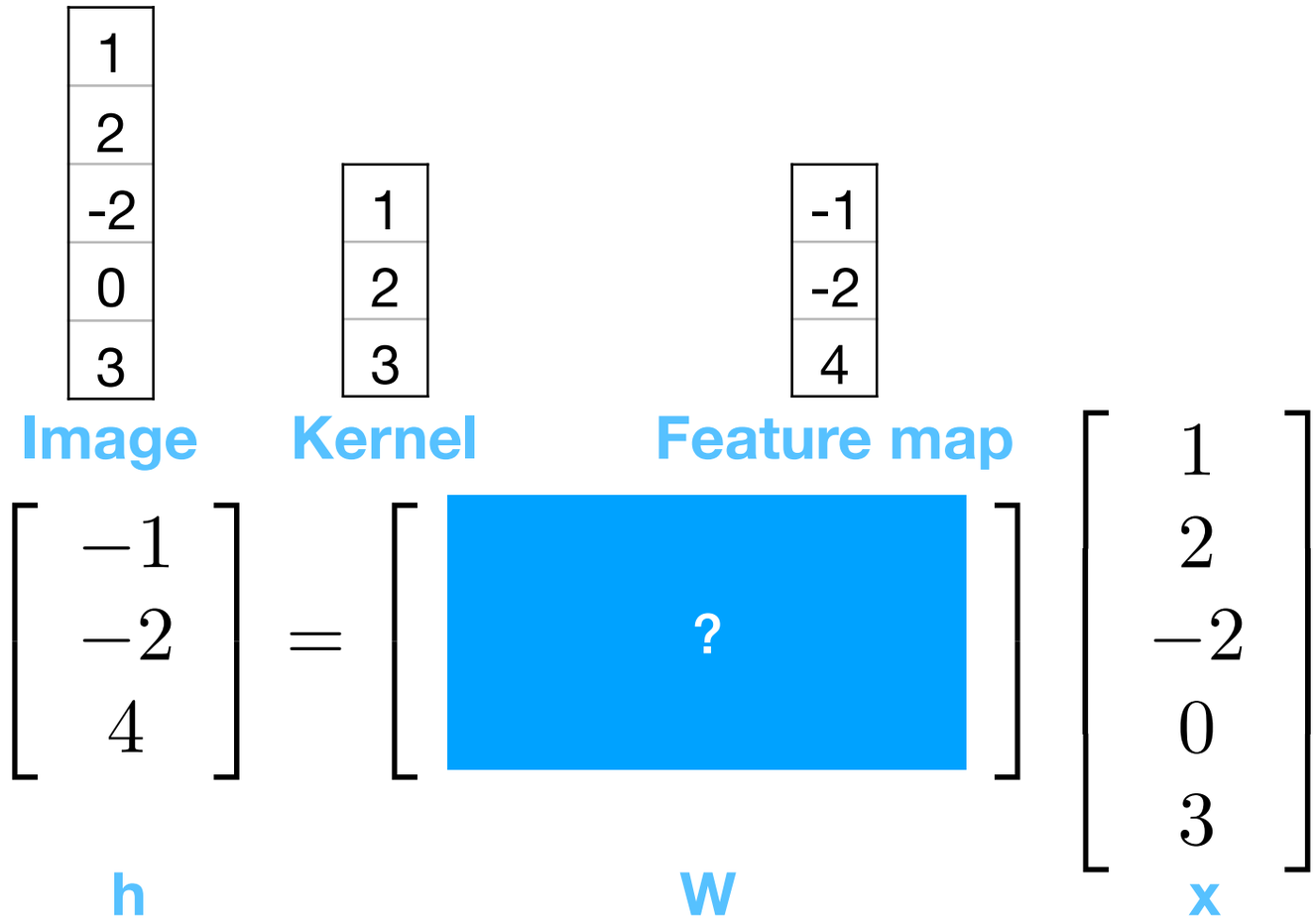
- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=2.



Convolution as a linear function

- It turns out that convolution is a linear function and can therefore be expressed as a matrix multiplication.
- To see how, consider the convolution of a 1-D image with a 1-D template.



Convolution as a linear function

- **W** is a special matrix called a **circulant matrix**, but it is still a matrix.

- This means that convolution is a *special case* of a general feed-forward neural network.

1
2
-2
0
3

Image

1
2
3

Kernel

-1
-2
4

Feature map

$$\begin{bmatrix} -1 \\ -2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -2 \\ 0 \\ 3 \end{bmatrix}$$

h W x

Convolution as a linear function

- Notice that **W** (in this example) has only 3 **free parameters** — all the other elements are equal to the first 3, or equal 0.

- This provides a strong *regularization effect* — if **W** performs a convolution, then it cannot vary as much as a general matrix **W**.

1
2
-2
0
3

Image

1
2
3

Kernel

-1
-2
4

Feature map

$$\begin{bmatrix} -1 \\ -2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -2 \\ 0 \\ 3 \end{bmatrix}$$

h W x

Convolution in 3-D

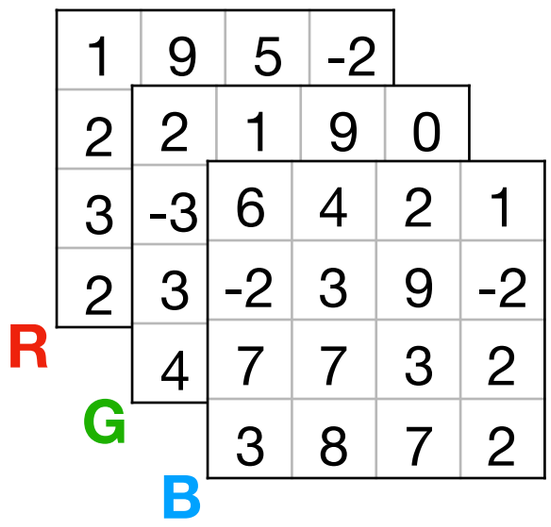
- It is possible to perform convolution in any number of dimensions.
- With neural networks, the usual formula of 3-D convolution is given by:

$$\mathrm{TM}(r,c) = \sum_{c=1}^{t_c} \sum_{i=-t_h/2}^{+t_h/2} \sum_{j=-t_w/2}^{+t_w/2} \mathrm{im}[r+i, c+j, c]\mathrm{t}[i,j,c]$$

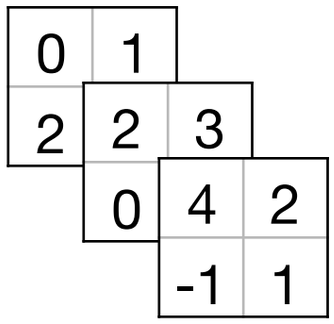
where t_c is the number of **channels** in the convolution kernel (and also the input image).

Convolution in 3-D: example

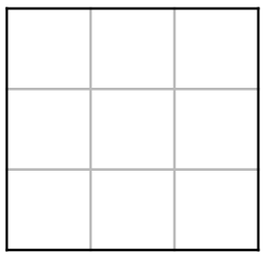
- Suppose our input image is RGB of size 4x4 — i.e., it is 4x4x3 — and we convolve it with a kernel of 2x2x3:



Image



Kernel



Feature map

Convolution in 3-D: example

- To illustrate how this works, let's separate the different channels:

R

1	9	5	-2
2	8	6	5
3	7	-5	-3
2	3	8	1

G

2	1	9	0
-3	0	-7	3
3	7	1	2
4	2	0	1

B

6	4	2	1
-2	3	9	-2
7	7	3	2
3	8	7	2

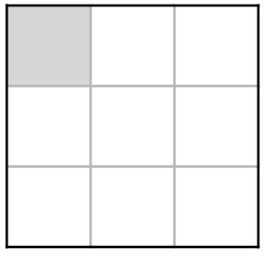
0	1
2	-1

+

2	3
0	2

+

4	2
-1	1



Convolution in 3-D: example

- To illustrate how this works, let's separate the different channels:

R

1	9	5	-2
2	8	6	5
3	7	-5	-3
2	3	8	1

G

2	1	9	0
-3	0	-7	3
3	7	1	2
4	2	0	1

B

6	4	2	1
-2	3	9	-2
7	7	3	2
3	8	7	2

0	1
2	-1

1*0+9*1+2*2 +8*-1=5

+

2	3
0	2

2*2+1*3+-3*0 +0*2=7

+

4	2
-1	1

6*4+4*2+-2*-1 +3*1=37

49		

Convolution in 3-D: example

- To illustrate how this works, let's separate the different channels:

R

1	9	5	-2
2	8	6	5
3	7	-5	-3
2	3	8	1

0	1
2	-1

+

G

2	1	9	0
-3	0	-7	3
3	7	1	2
4	2	0	1

2	3
0	2

+

B

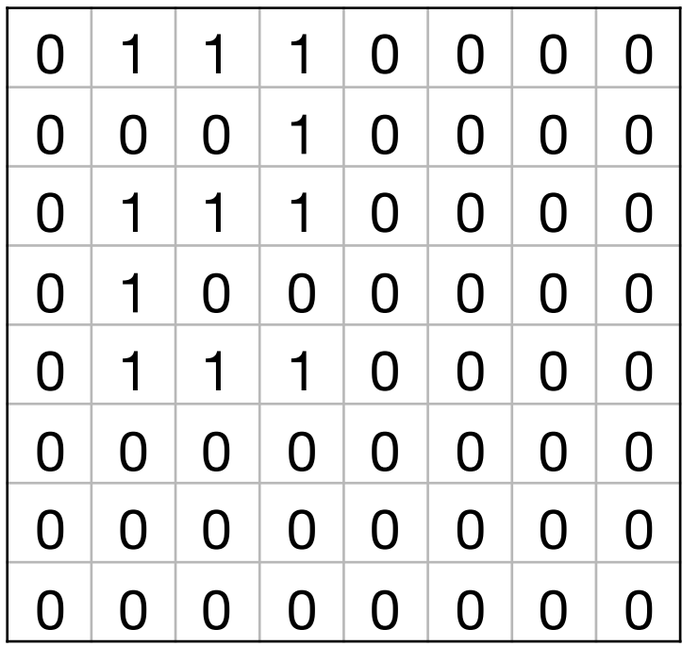
6	4	2	1
-2	3	9	-2
7	7	3	2
3	8	7	2

4	2
-1	1

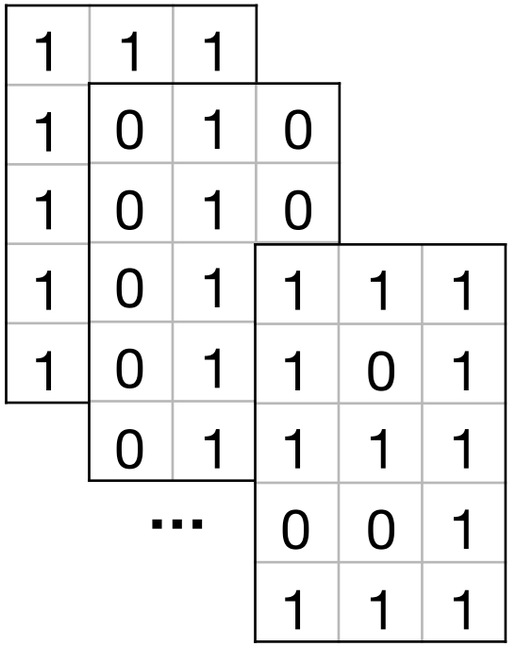
49	…	

Convolution: multiple output channels

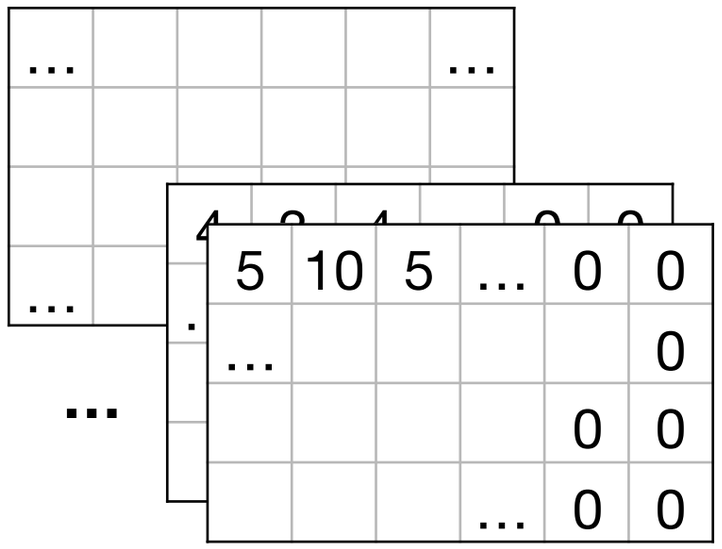
- By applying multiple convolution kernels to each input image, we can produce multiple feature maps (recall the MNIST example):



Image



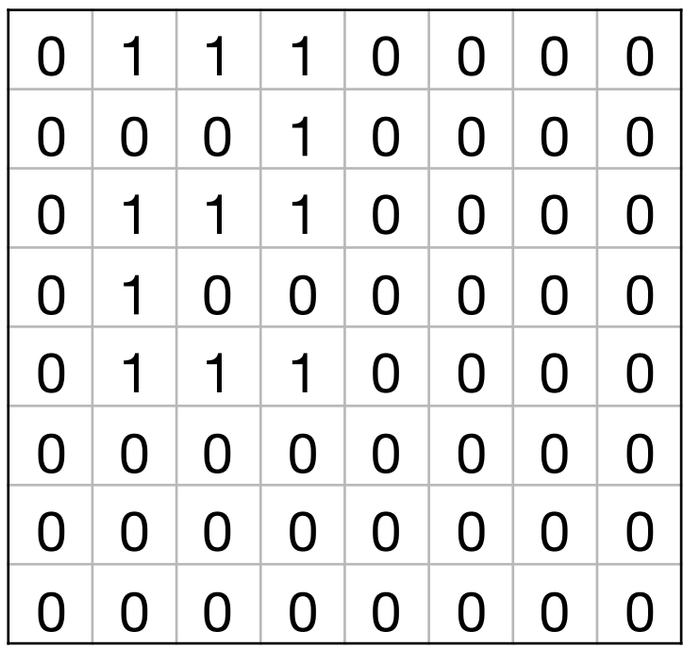
Multiple kernels



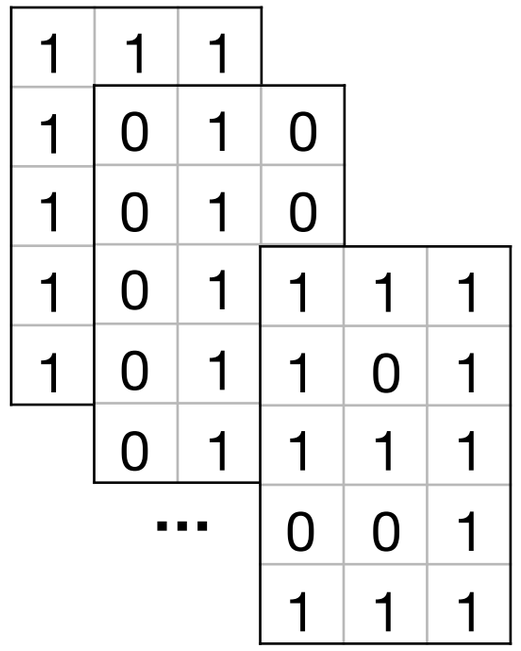
Multiple feature maps

Convolution: multiple output channels

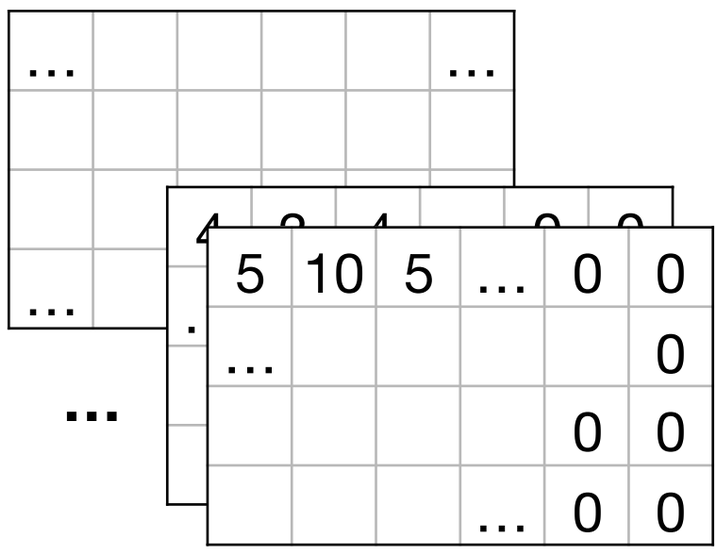
- Hence, convolution can take a multi-channel image as *input*, and also produce a multi-channel image as *output*.



Image



Multiple kernels



Multiple feature maps

Convolutional neural networks

Convolutional neural networks

- Based on this infrastructure, we can construct **convolutional neural networks** (**CNNs**) — networks that consist of 1+ convolutional layers (and usually some non-convolutional layers too).

- CNNs have revolutionized computer vision, especially in object detection, object recognition, semantic segmentation, activity recognition, and other tasks.