CS/DS 541: Class 10

Jacob Whitehill

Universal function approximation

Universal function approximation theorem

- For any closed, bounded, continuous function *f* and any ϵ, we can train a feed-forward 3-layer NN $\hat{f}$ with sigmoidal activation functions and sufficiently many neurons in the hidden layer such that:

$$|f(x) - \hat{f}(x)| < \epsilon \quad \forall x$$

Universal function approximation theorem

- The theorem also generalizes to f with multidimensional inputs and outputs.

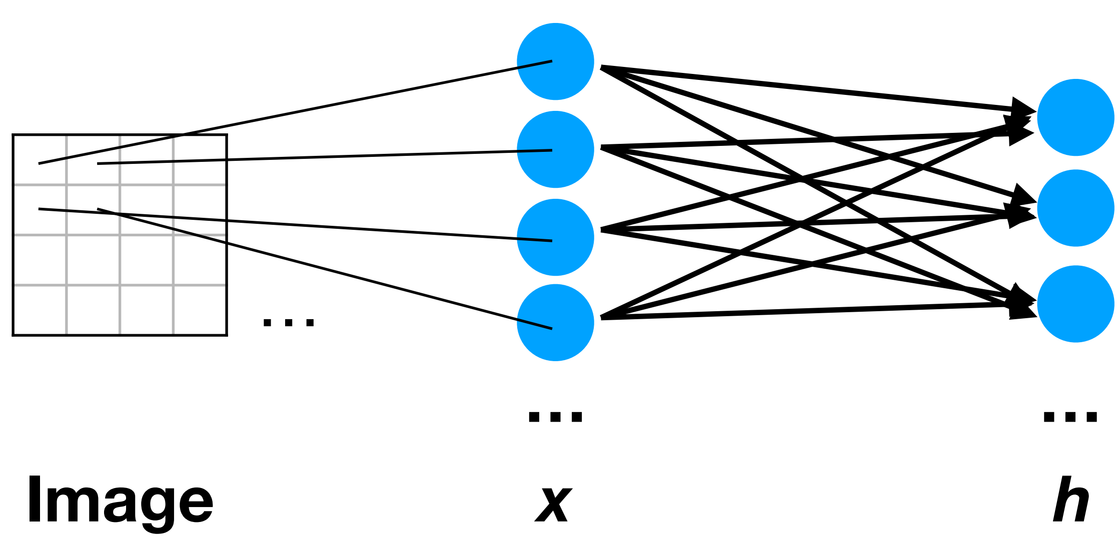
Proof idea

- Using pairs of sigmoid functions, we can construct delta functions that represent vertical “bars”.
- Using enough vertical bars, we can approximate any *f* (akin to the trapezoidal rule of calculus).
- See visual proof sketch here.

Image features

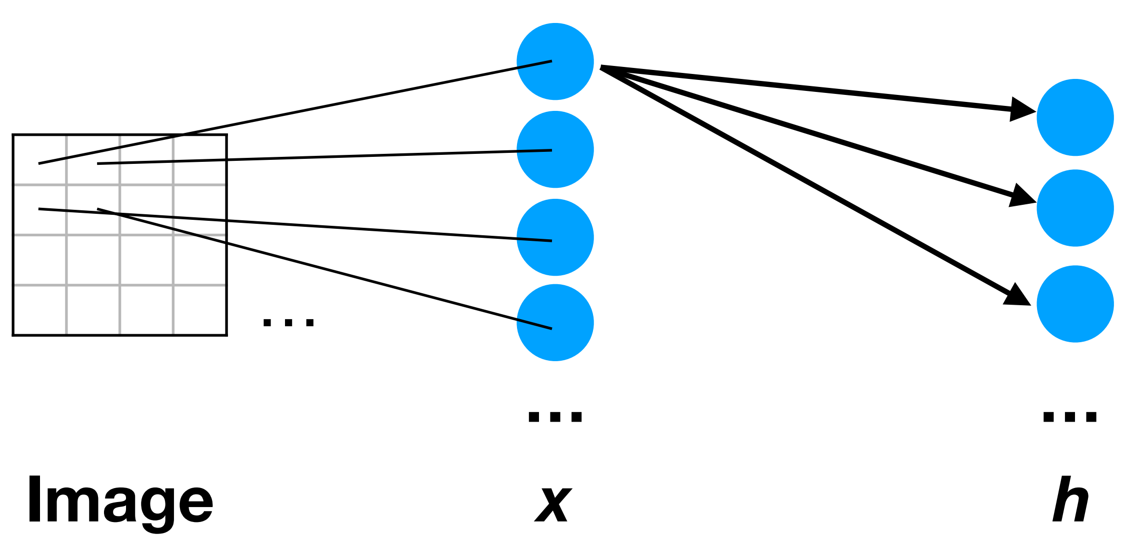
2-d spatial structure in images

- So far we have examined feed-forward neural networks that are fully connected, i.e., every neuron in layer *l* is connected to every neuron in layer *l*+1.



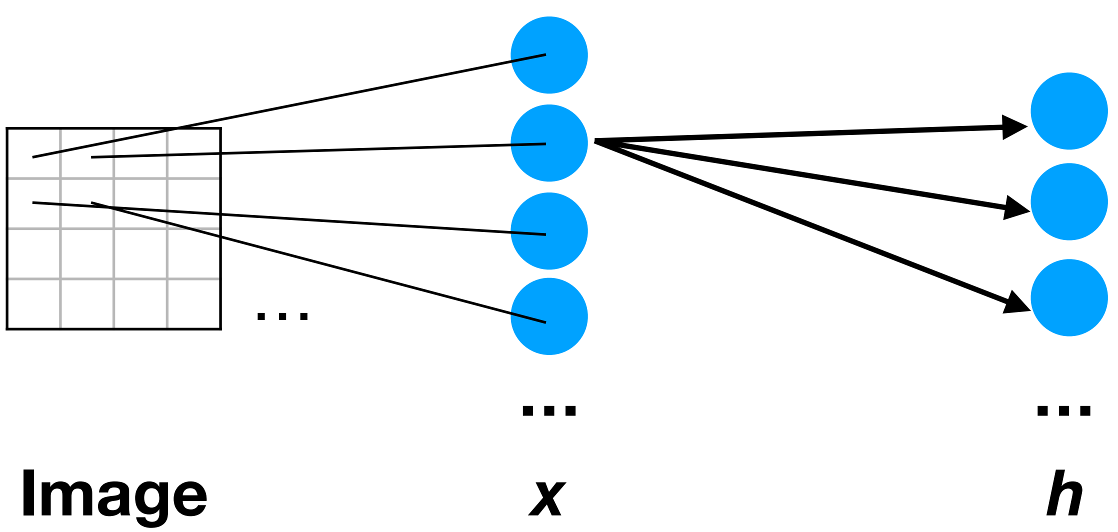
2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.



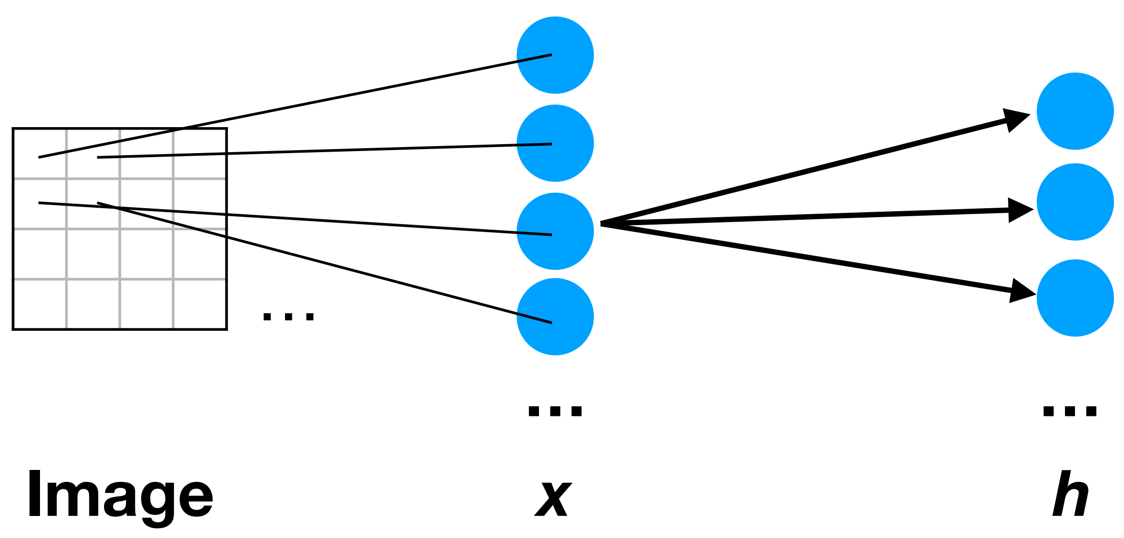
2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.



2-d spatial structure in images

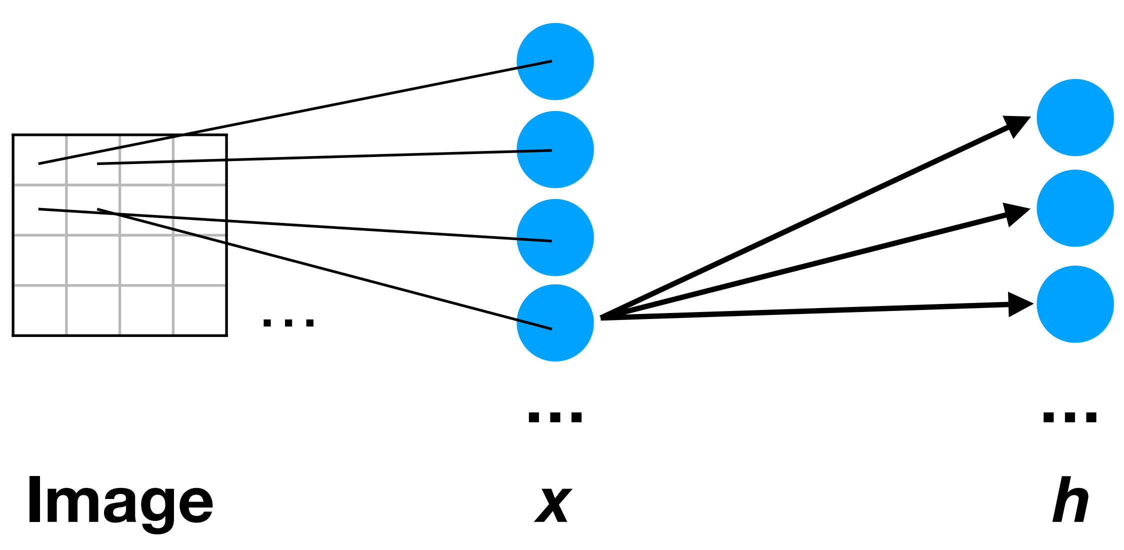
- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.



2-d spatial structure in images

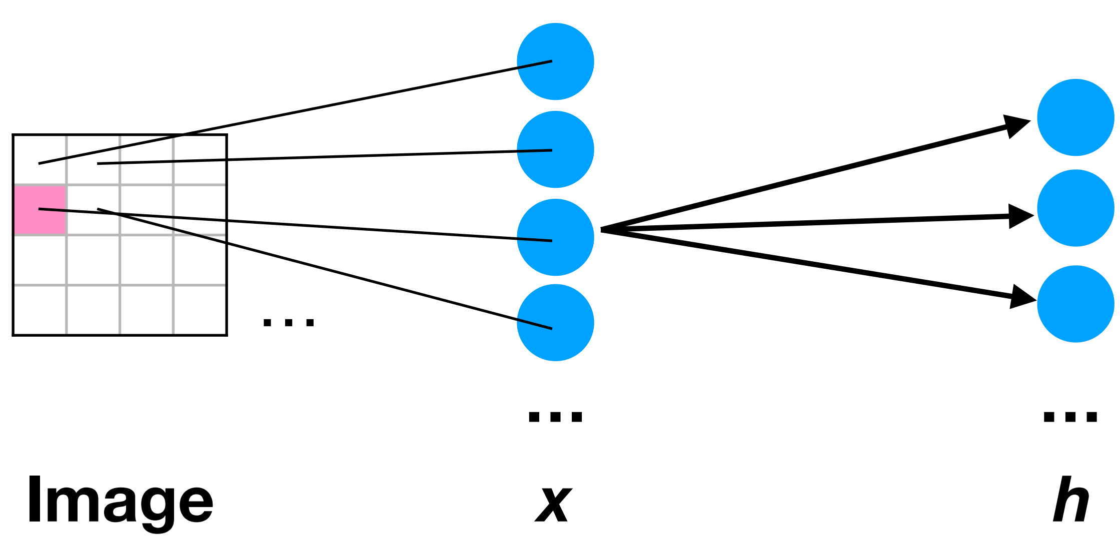
- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.

  - But does this make sense for images with a spatial structure?



2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.
  - A tiny shift in the input image can result in arbitrarily large change in the hidden unit activations.

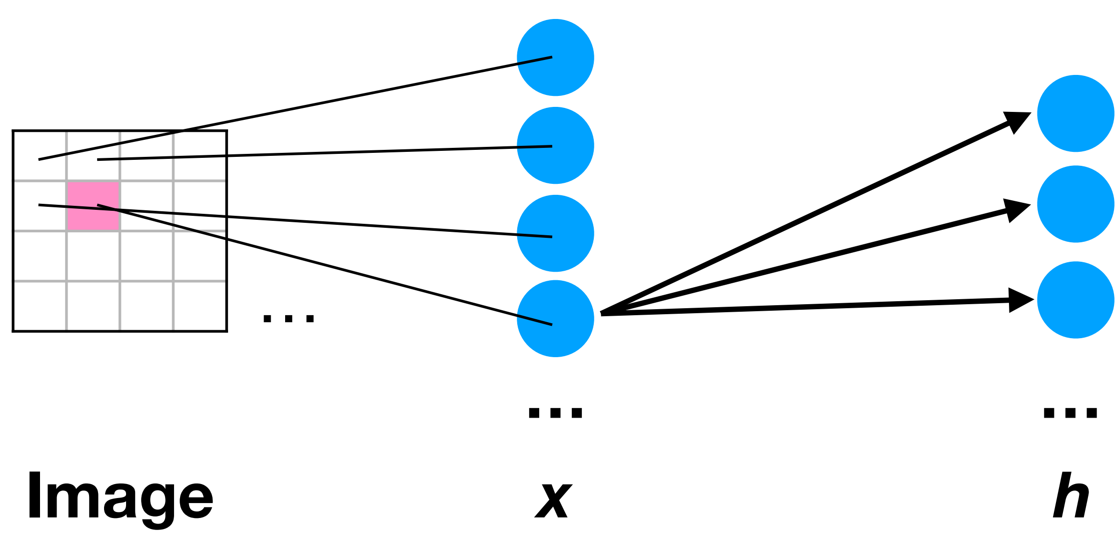


Before: the hidden units receive input from x_5.

2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.

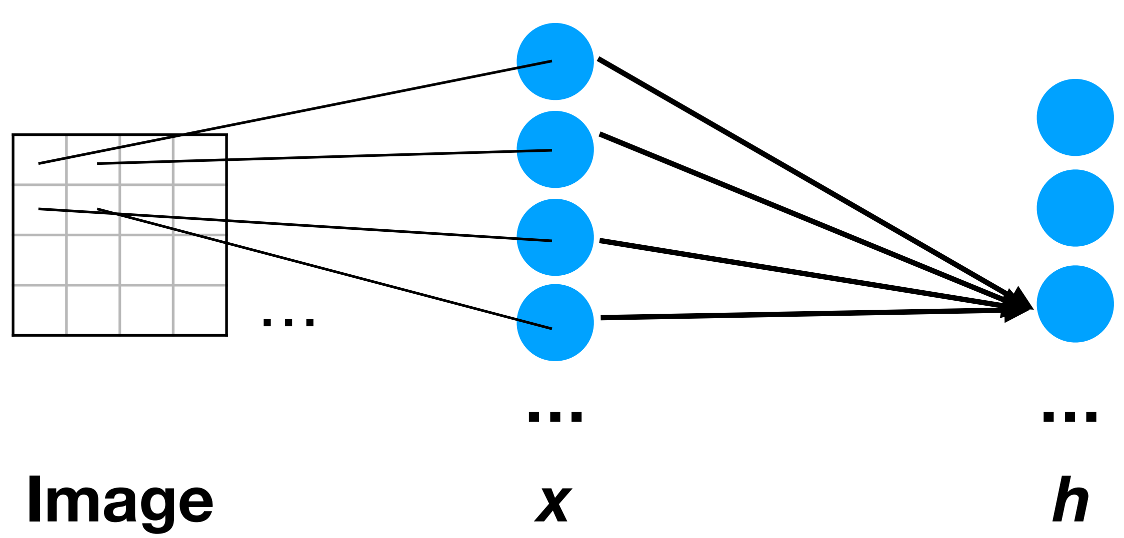
    - A tiny shift in the input image can result in arbitrarily large change in the hidden unit activations.



After: the hidden units receive input from x_6.

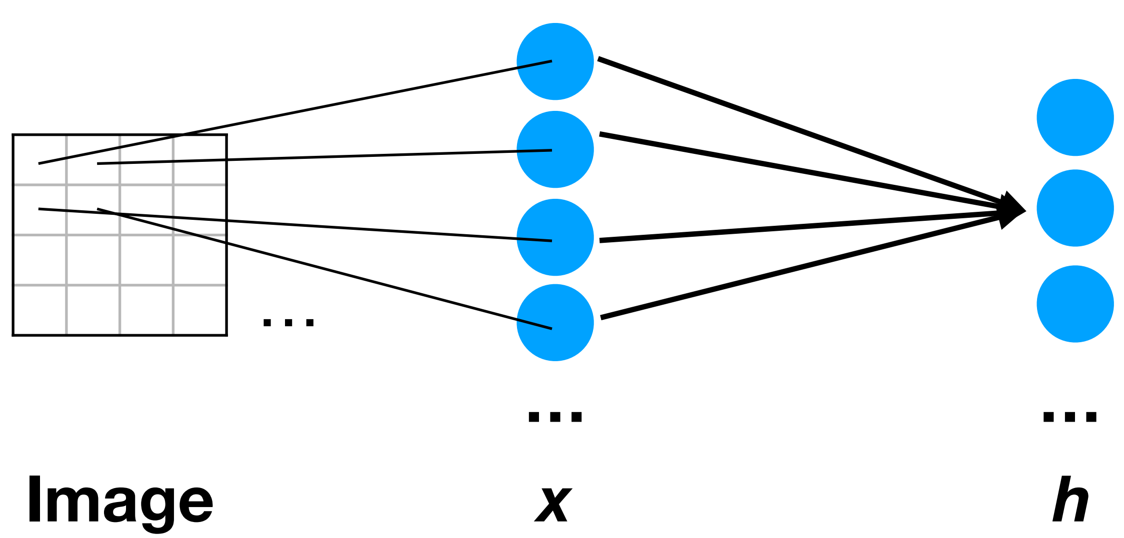
2-d spatial structure in images

- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.



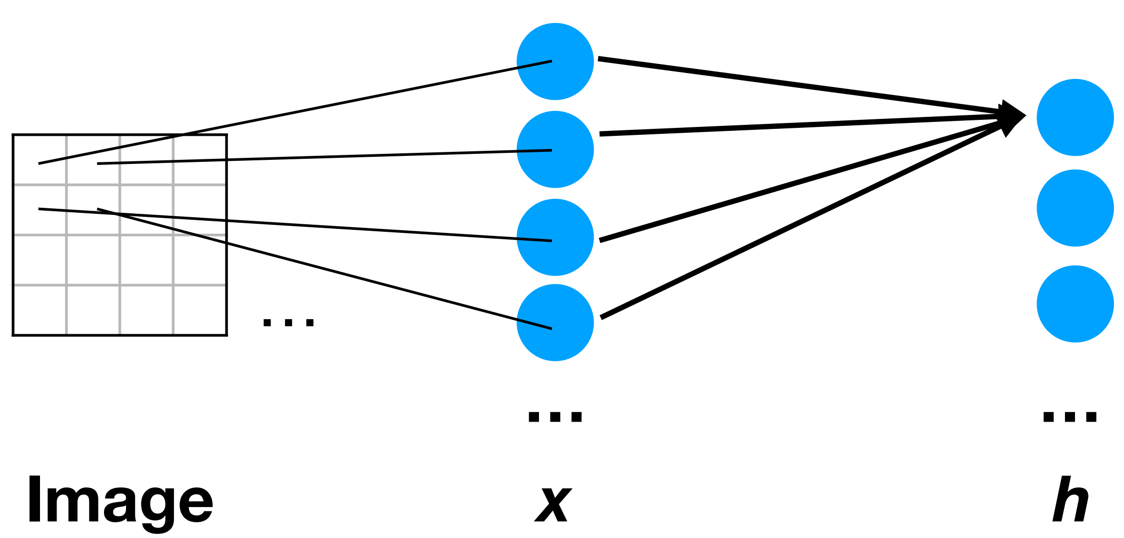
2-d spatial structure in images

- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.



2-d spatial structure in images

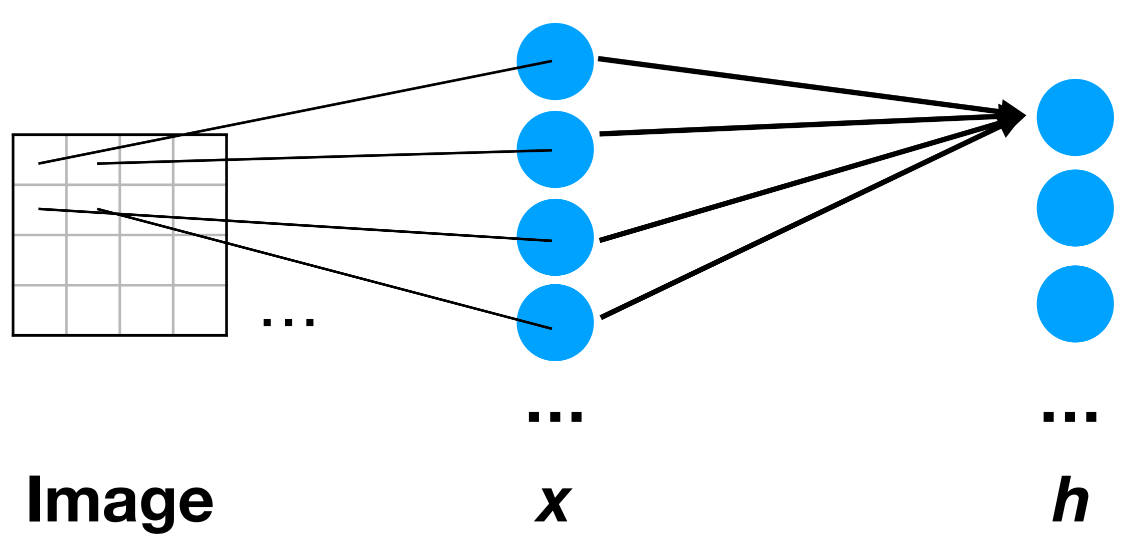
- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.



2-d spatial structure in images

- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.

  - I.e., the hidden layer(s) have no spatial structure — the index *i* of each hidden unit is meaningless.



2-d spatial structure in images

- Throwing away the 2-d spatial structure is akin to trying to classify an image after applying some arbitrary (but fixed) permutation to it:

Pixel order:
1, 2, 3, …, 576

2-d spatial structure in images

- Throwing away the 2-d spatial structure is akin to trying to classify an image after applying some arbitrary (but fixed) permutation to it:

Pixel order:
1, 2, 3, …, 576

Pixel order:
25, 305, 213, …, 509

2-d spatial structure in images

- Images tend to contain the *same* visual features — lines and corners of different colors (see MNIST demo) — at *many* locations and scales.

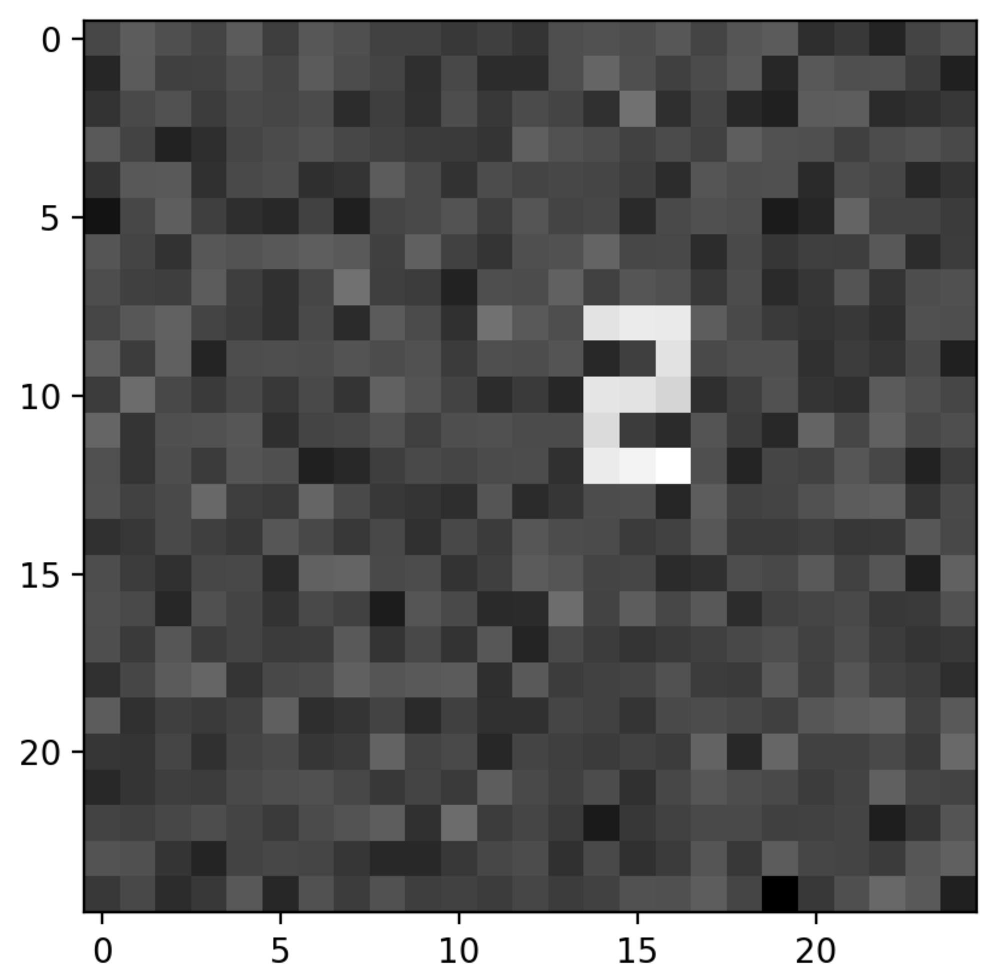
- By harnessing the spatial structure of images, we can train NNs with far fewer weights.

    - This is a powerful form of regularization.

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?

Jacob Whitehill, WPI

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.

Jacob Whitehill, WPI

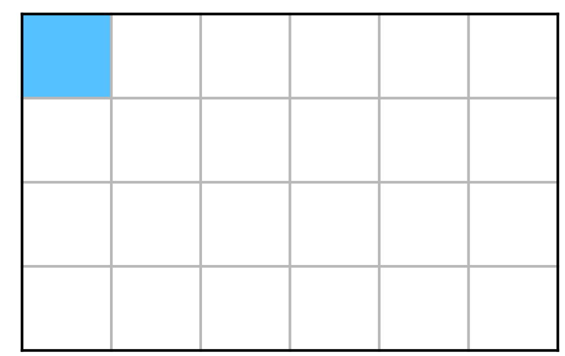
Convolution

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.

Jacob Whitehill, WPI

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

- At each (r, c), this can be thought of as a 2-D "dot product" between the image and the template. Over the whole image, 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

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

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

Jacob Whitehill, WPI

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

0*1

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

0*1+1*1

Jacob Whitehill, WPI

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

0*1+1*1+1*1

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

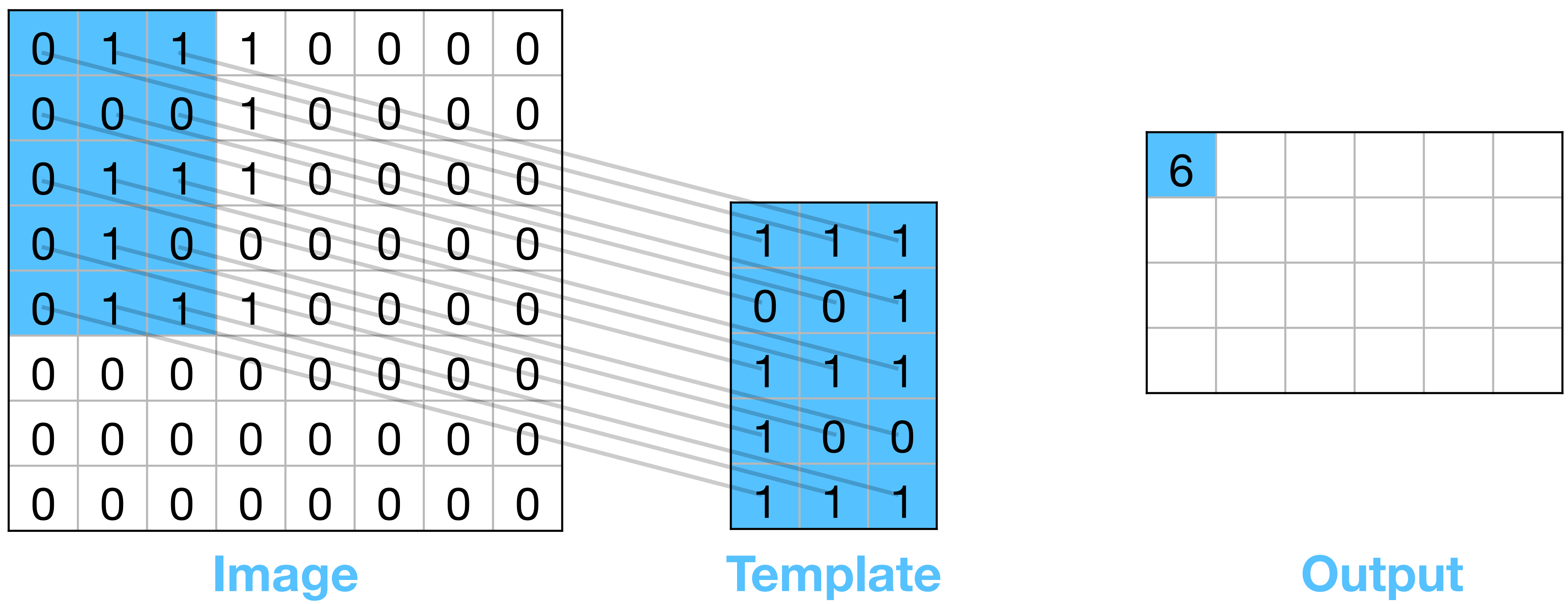
1	1	1
0	0	1
1	1	1
1	0	0
1	1	1

Template

Output

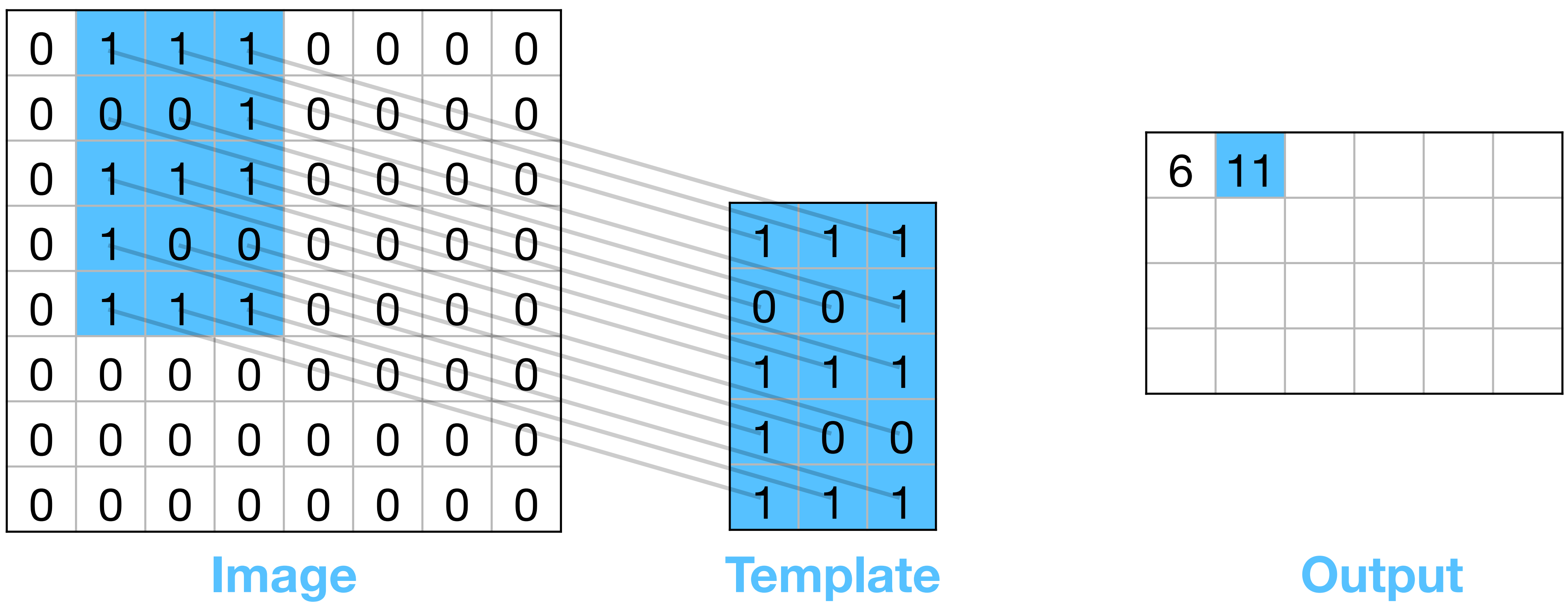
0*1+1*1+1*1+
0*0+0*0+0*1

Convolution



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



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

Jacob Whitehill, WPI

Convolution: exercise

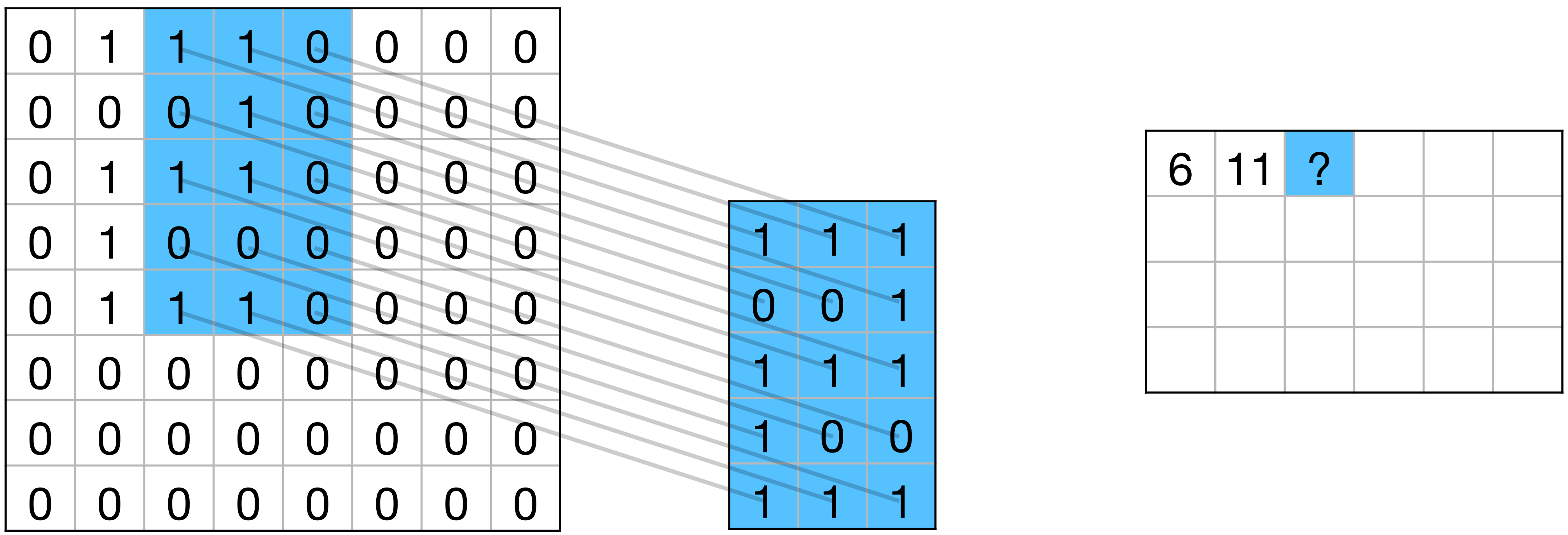


Image Template Output

Jacob Whitehill, WPI

Convolution: exercise

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			

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

Jacob Whitehill, WPI

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

...

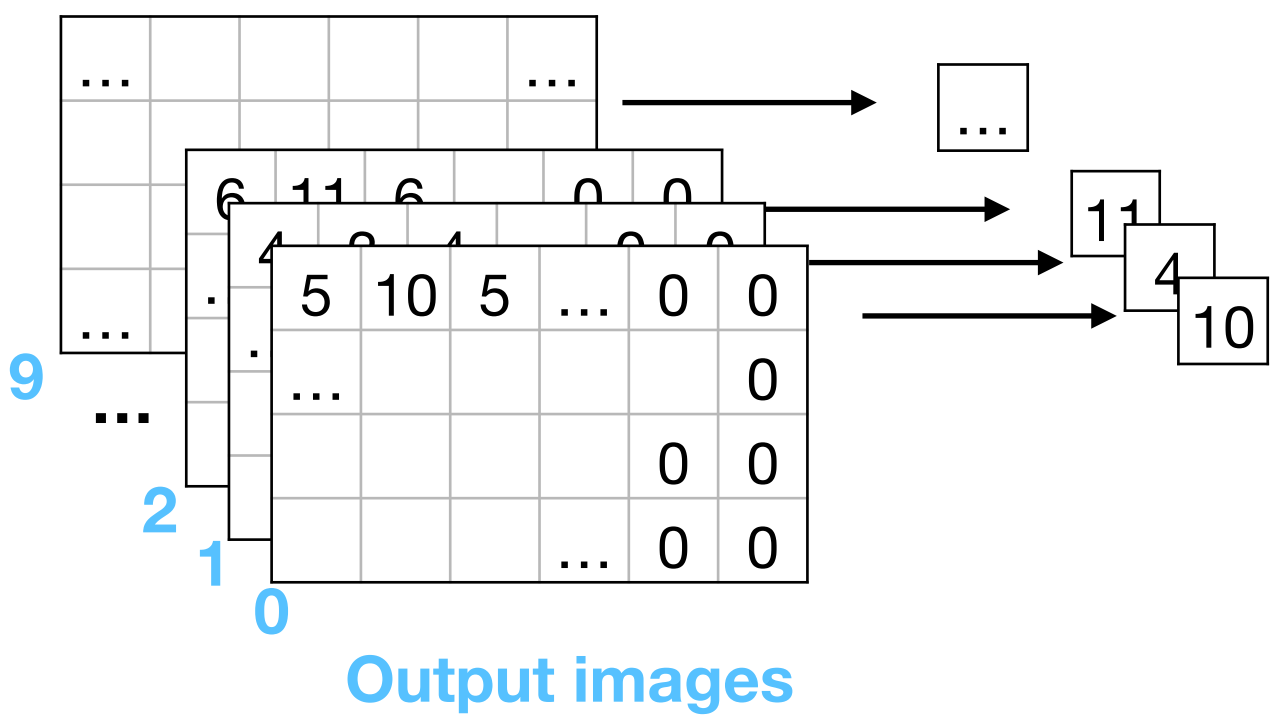
Template

…					…
…					…

Output

Convolution for classification

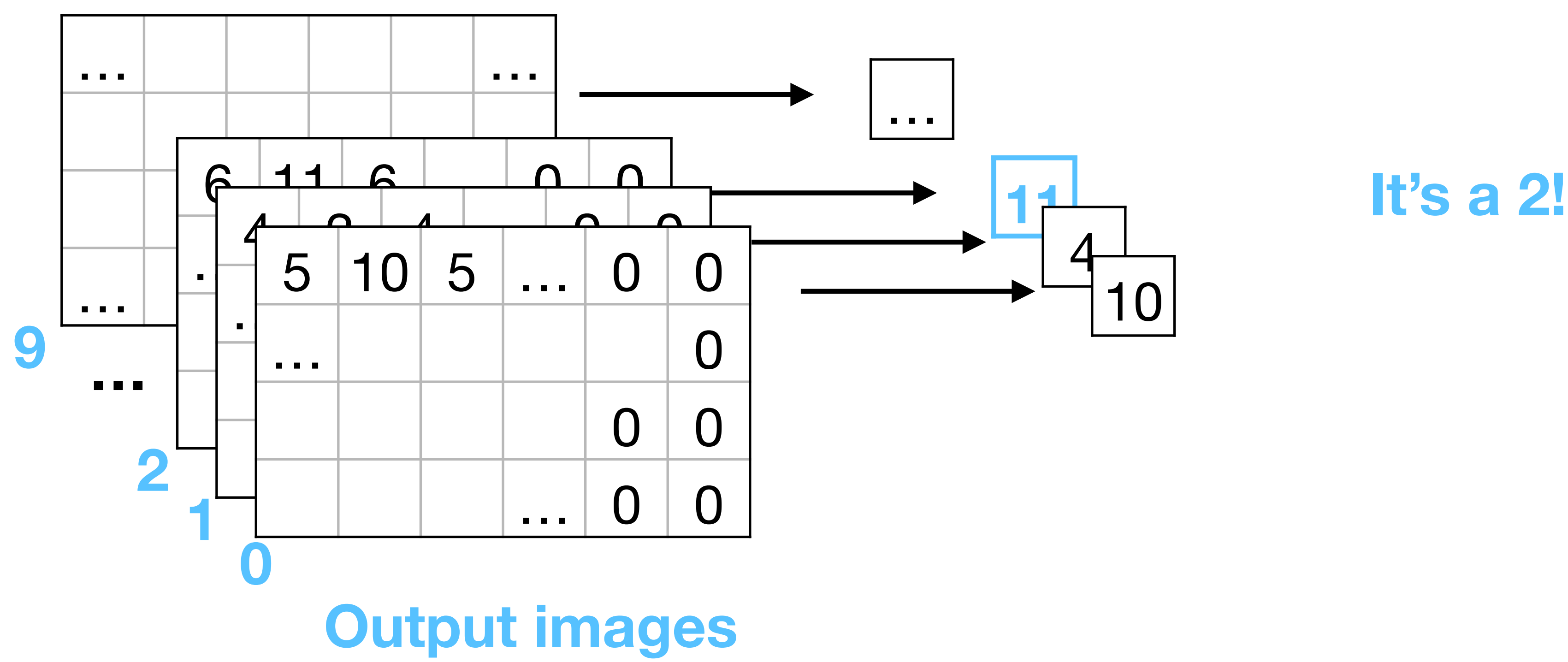
- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:
  - Compute the maximum over each output image.



Jacob Whitehill, WPI

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:

12 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

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.

12 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

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

Kernel

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.

12 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

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

Kernel

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.

12 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

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

Kernel

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.

12 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

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:

12 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

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

Kernel

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:

12 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

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:

12 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

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:

12 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

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

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:

12 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

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

Dilated convolution

- To expand the spatial extent of the filter without increasing the number of parameters, we can dilate it by adding "spaces" (with dilation k=2) between filter elements.

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

Image

2	1
-3	9

Filter

1*2+5*1+3*-3+-5*9 = -47

-47		

Feature map

Dilated convolution

- To expand the spatial extent of the filter without increasing the number of parameters, we can dilate it by adding "spaces" (with dilation *k*=2) between filter elements.

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

Image

2	1
-3	9

Filter

9*2+-2*1+7*-3+-3*9 = -32

-47	-32	
	…	

Feature map

Dilated convolution

- Dilated convolution is sometimes called **à trous** convolution (“with holes”).

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

Image

2	1
-3	9

Filter

9*2+-2*1+7*-3+-3*9 = -32

-47	-32	
	…	

Feature map

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:

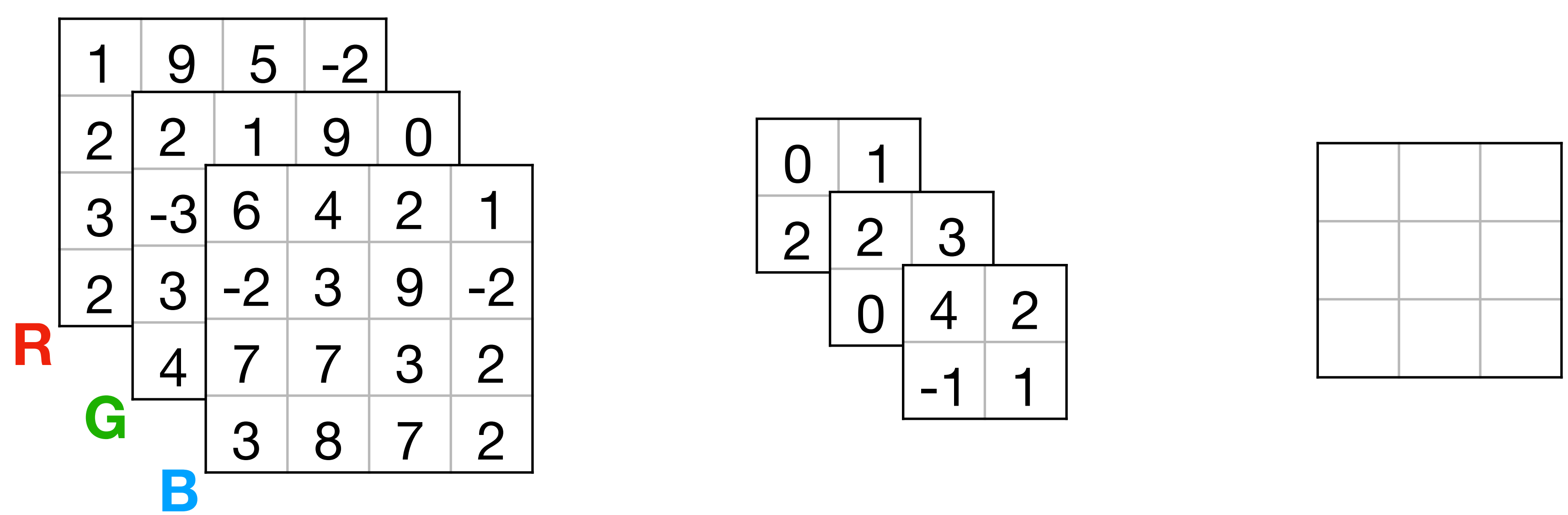


Image Filter Feature map

Jacob Whitehill, WPI

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

Jacob Whitehill, WPI

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		

Jacob Whitehill, WPI

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

49	…	

Jacob Whitehill, WPI

Convolution: multiple output channels

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

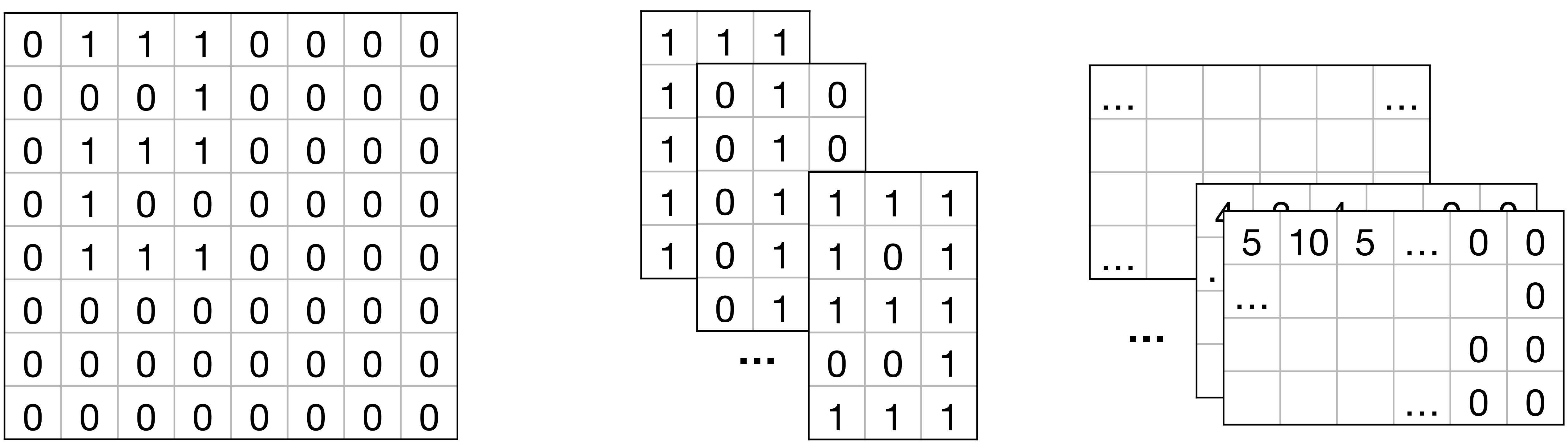


Image **Multiple filters** **Multiple feature maps**

Convolution: multiple output channels

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

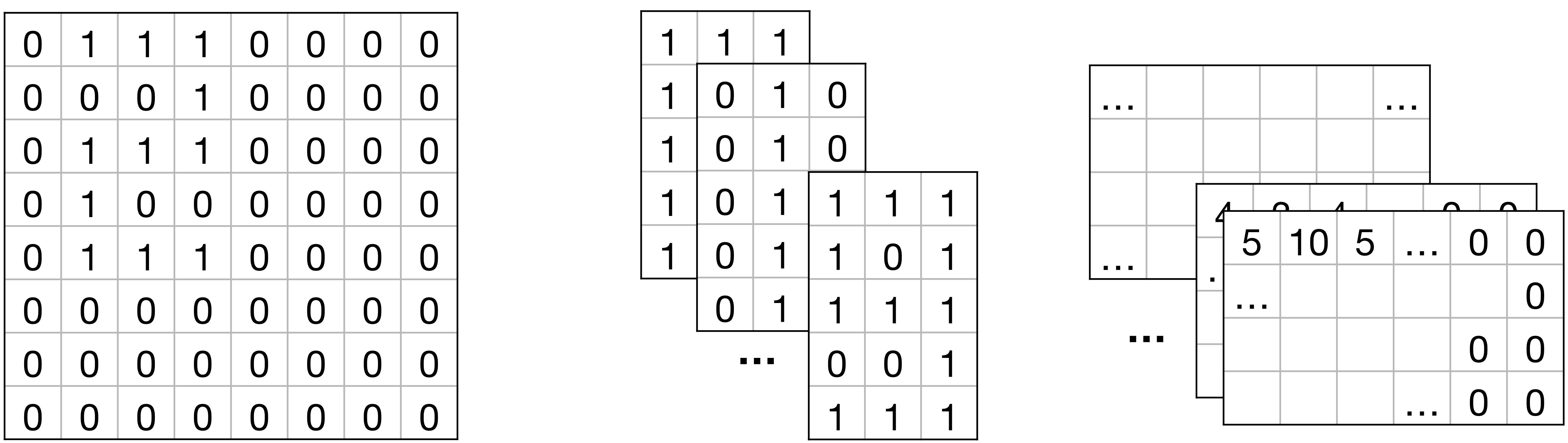
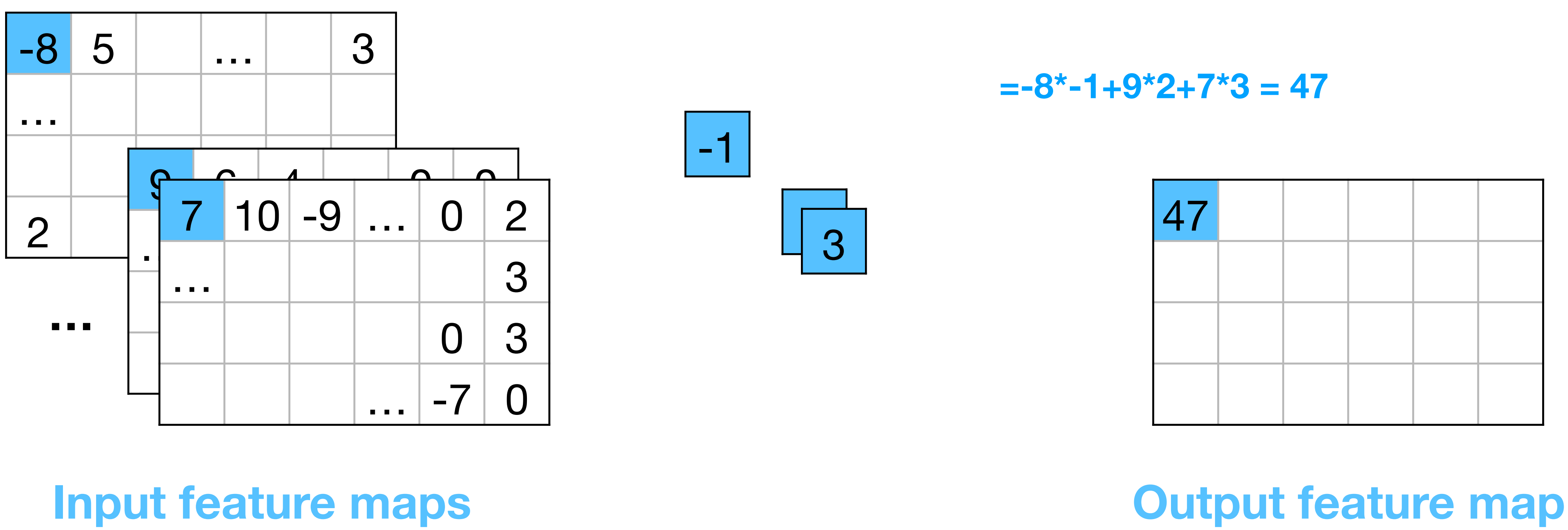


Image　　Multiple filters　　Multiple feature maps

Jacob Whitehill, WPI

1-d convolution

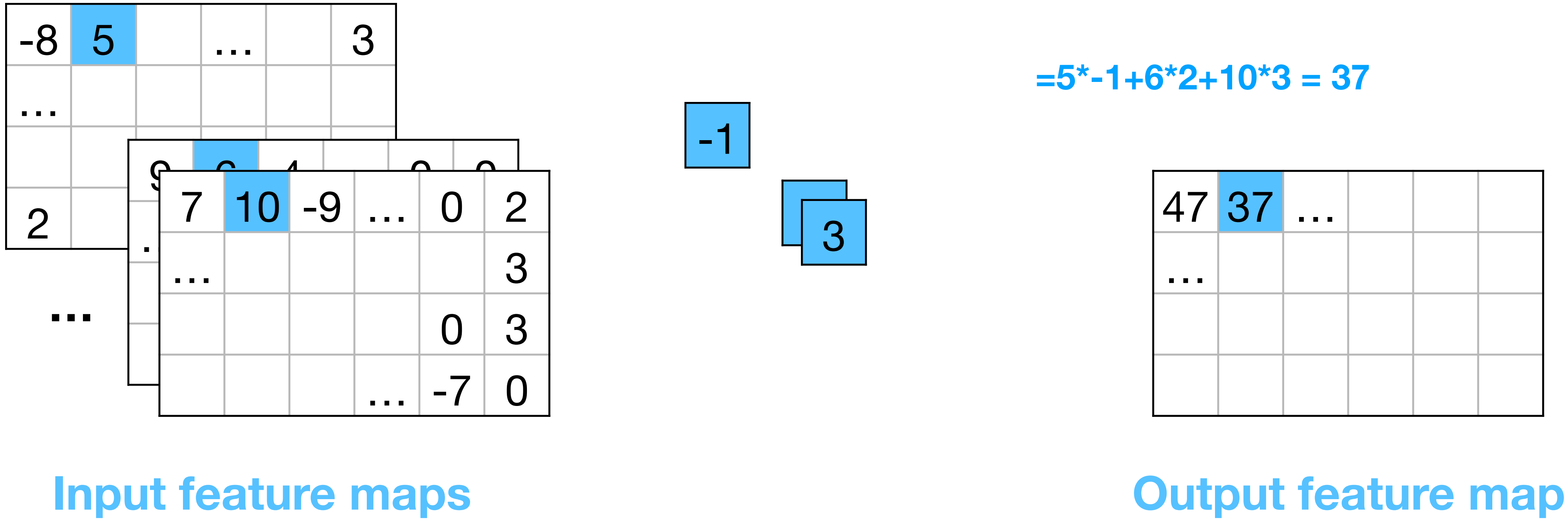
- We can also perform "1-d convolution".
- This is equivalent to a weighted sum among the input feature maps.
- It is useful to reduce the number of feature maps at the current NN layer.



Jacob Whitehill, WPI

1-d convolution

- We can also perform “1-d convolution”.
- This is equivalent to a weighted sum among the input feature maps.
- It is useful to reduce the number of feature maps at the current NN layer.

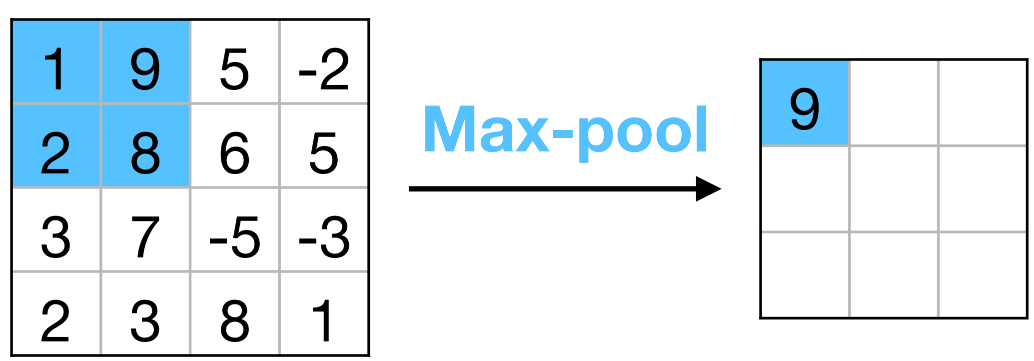


Jacob Whitehill, WPI

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.



Jacob Whitehill, WPI

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.

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

$\xrightarrow{\text{Max-pool}}$

9	9	

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.

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

Max-pool →

9	9	6

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.

$$\begin{bmatrix} 1 & 9 & 5 & -2 \\ 2 & 8 & 6 & 5 \\ 3 & 7 & -5 & -3 \\ 2 & 3 & 8 & 1 \end{bmatrix} \xrightarrow{\textbf{Max-pool}} \begin{bmatrix} 9 & 9 & 6 \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Jacob Whitehill, WPI

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.

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

$\xrightarrow{\text{Max-pool}}$

9	9	6
8	8	6
7	8	8

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.

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

$\xrightarrow{\textbf{Max-pool}}$

?	?
?	?

Jacob Whitehill, WPI

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.

$$\begin{array}{|c|c|c|c|}\hline 1 & 9 & 5 & -2 \\ \hline 2 & 8 & 6 & 5 \\ \hline 3 & 7 & -5 & -3 \\ \hline 2 & 3 & 8 & 1 \\ \hline \end{array} \xrightarrow{\text{Max-pool}} \begin{array}{|c|c|}\hline 9 & 6 \\ \hline 7 & 8 \\ \hline \end{array}$$

Convolutional neural networks (CNNs)

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).

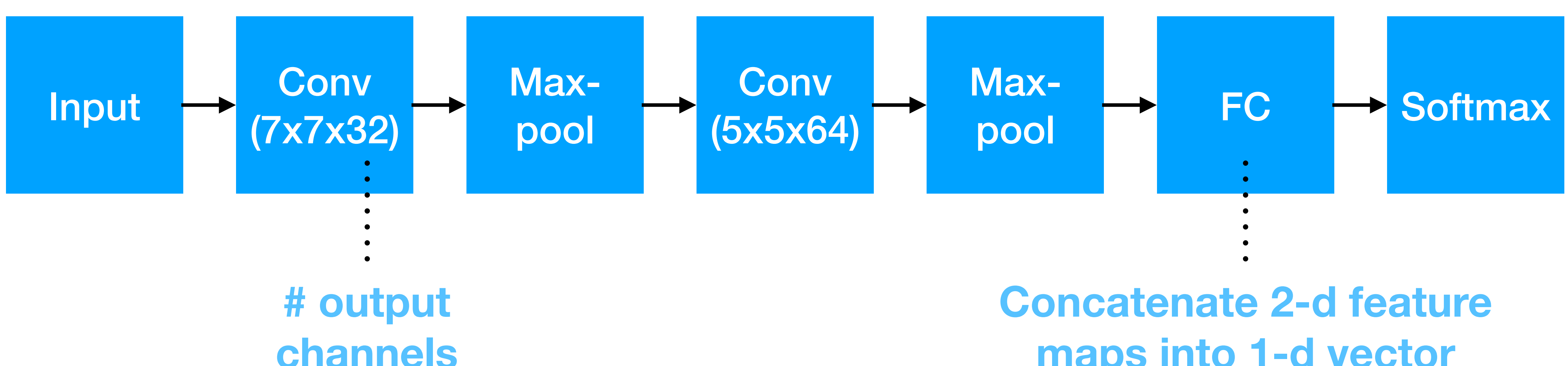
- CNNs have revolutionized computer vision, especially in object detection, object recognition, semantic segmentation, activity recognition, and other tasks.

Convolutional neural networks

- While the filters in the examples so far were built by hand, this is almost never done in practice.
- Instead, we train the weights using back-propagation, just like with feed-forward neural networks.
- With CNNs, the learned weights represent the elements of the convolution kernels.

CNN architecture

- CNNs (since ~2012) often employ:
  - Multiple convolutional layers
  - Pooling layers (e.g., max-pool) between some of the convolutional layers.
  - Fully-connected (FC) layers at the end.
  - Non-linear activation functions between each layer.
- Example:



CNN architecture

- Trend (~2016+):
  - Less pooling
  - More convolutional layers
  - Bottlenecks
  - Residual connections
- Show ImageNet (2012), VGG (2015) papers.

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.

Image	Filter	Feature map
1	1	-1
2	2	-2
-2	3	7
0		
3		

$$\underbrace{\begin{bmatrix} -1 \\ -2 \\ 7 \end{bmatrix}}_{\mathbf{h}} = \underbrace{\begin{bmatrix} \boxed{\quad ? \quad} \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 1 \\ 2 \\ -2 \\ 0 \\ 3 \end{bmatrix}}_{\mathbf{x}}$$

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.

Image

1
2
-2
0
3

Filter

1
2
3

Feature map

-1
-2
7

$$\underbrace{\begin{bmatrix} -1 \\ -2 \\ 7 \end{bmatrix}}_{\mathbf{h}} = \underbrace{\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 1 \\ 2 \\ -2 \\ 0 \\ 3 \end{bmatrix}}_{\mathbf{x}}$$

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.

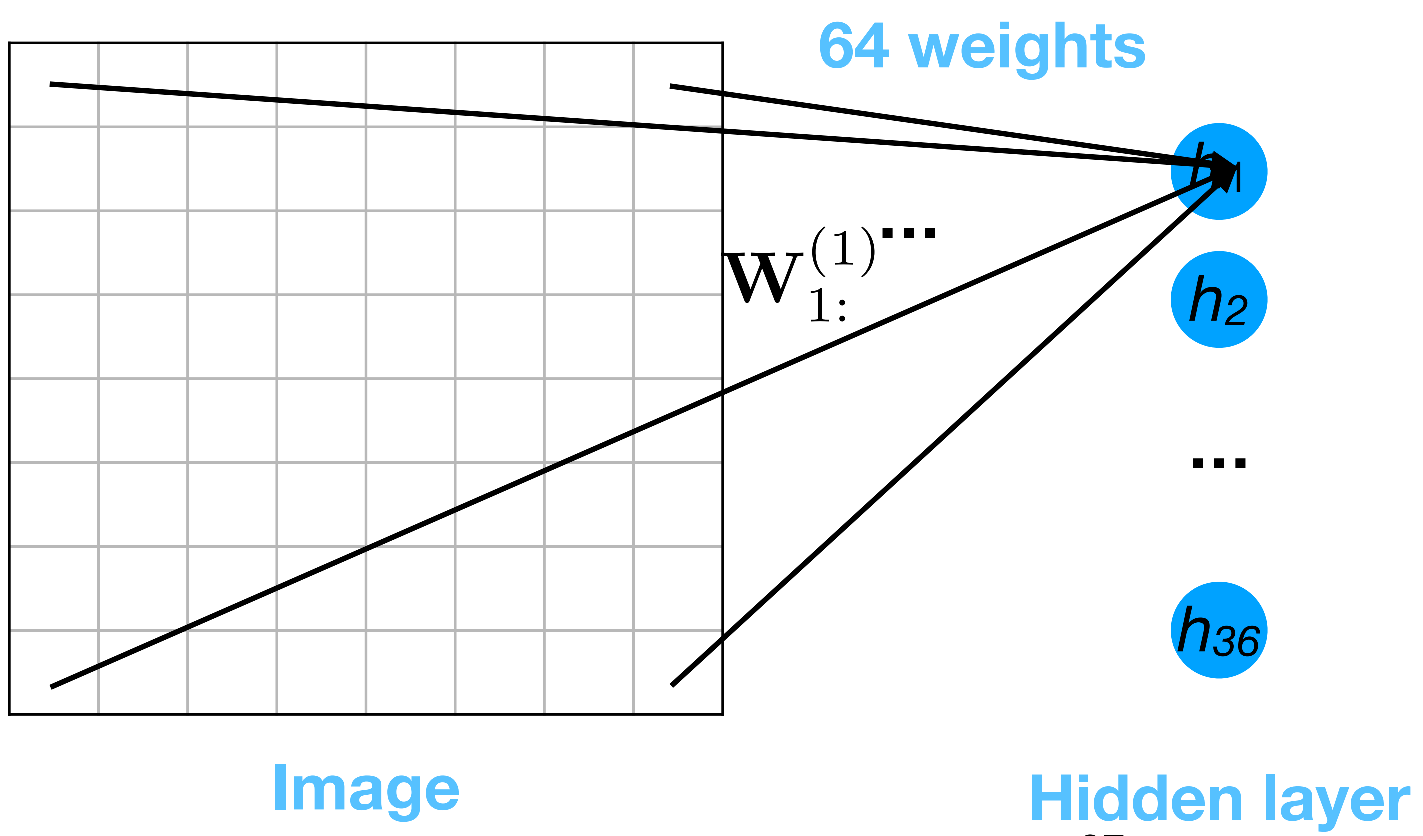
- This provides a strong *regularization effect* — if **W** performs a convolution, then it cannot vary as much as a general matrix **W**.

Image	Filter	Feature map
1	1	-1
2	2	-2
-2	3	7
0		
3		

$$\underbrace{\begin{bmatrix} -1 \\ -2 \\ 7 \end{bmatrix}}_{\mathbf{h}} = \underbrace{\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 1 \\ 2 \\ -2 \\ 0 \\ 3 \end{bmatrix}}_{\mathbf{x}}$$

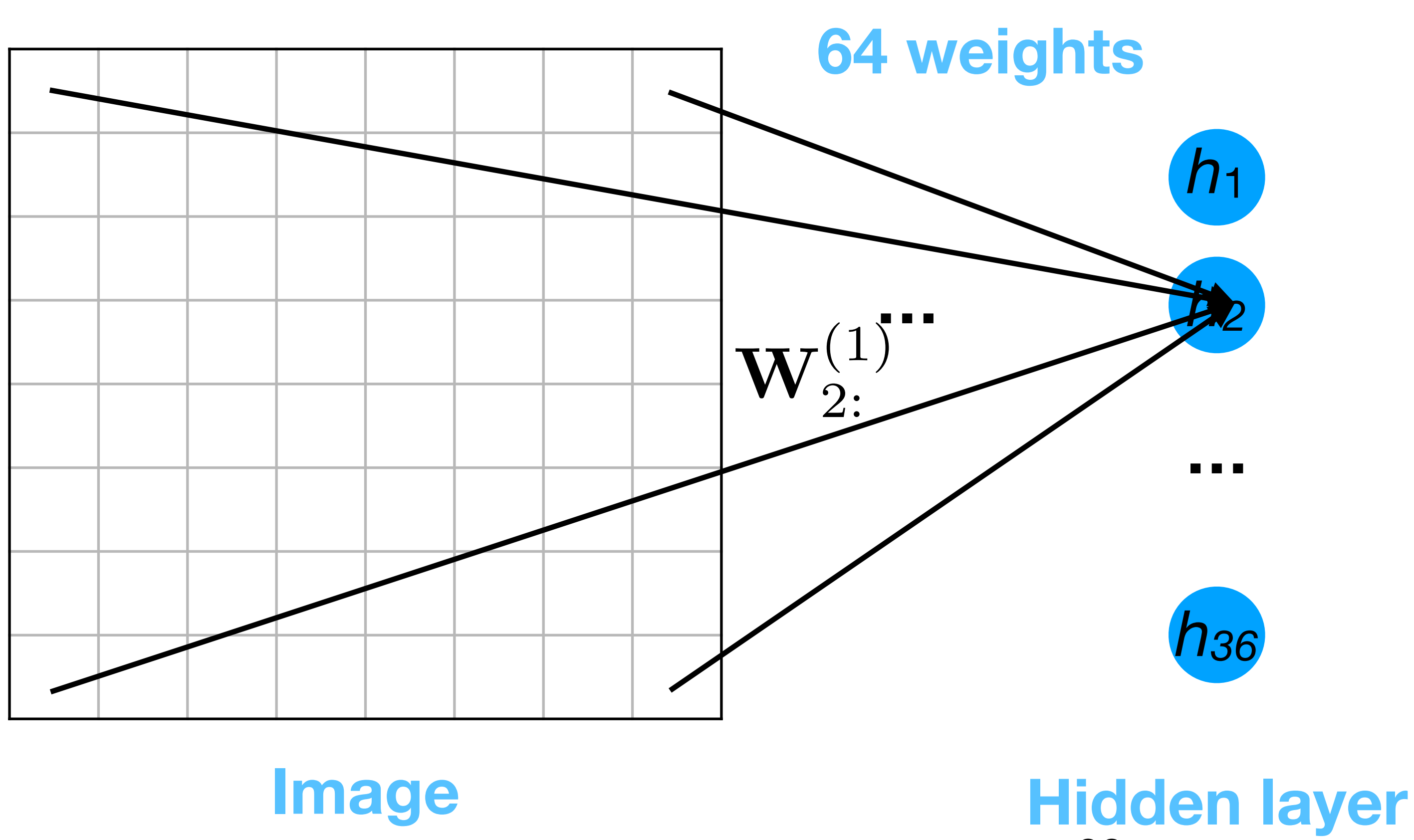
Convolutional neural networks

- CNNs are a special case of feed-forward (FF) NNs.
- In the FF NN below, each of the 36 hidden units is associated with 64 weights.



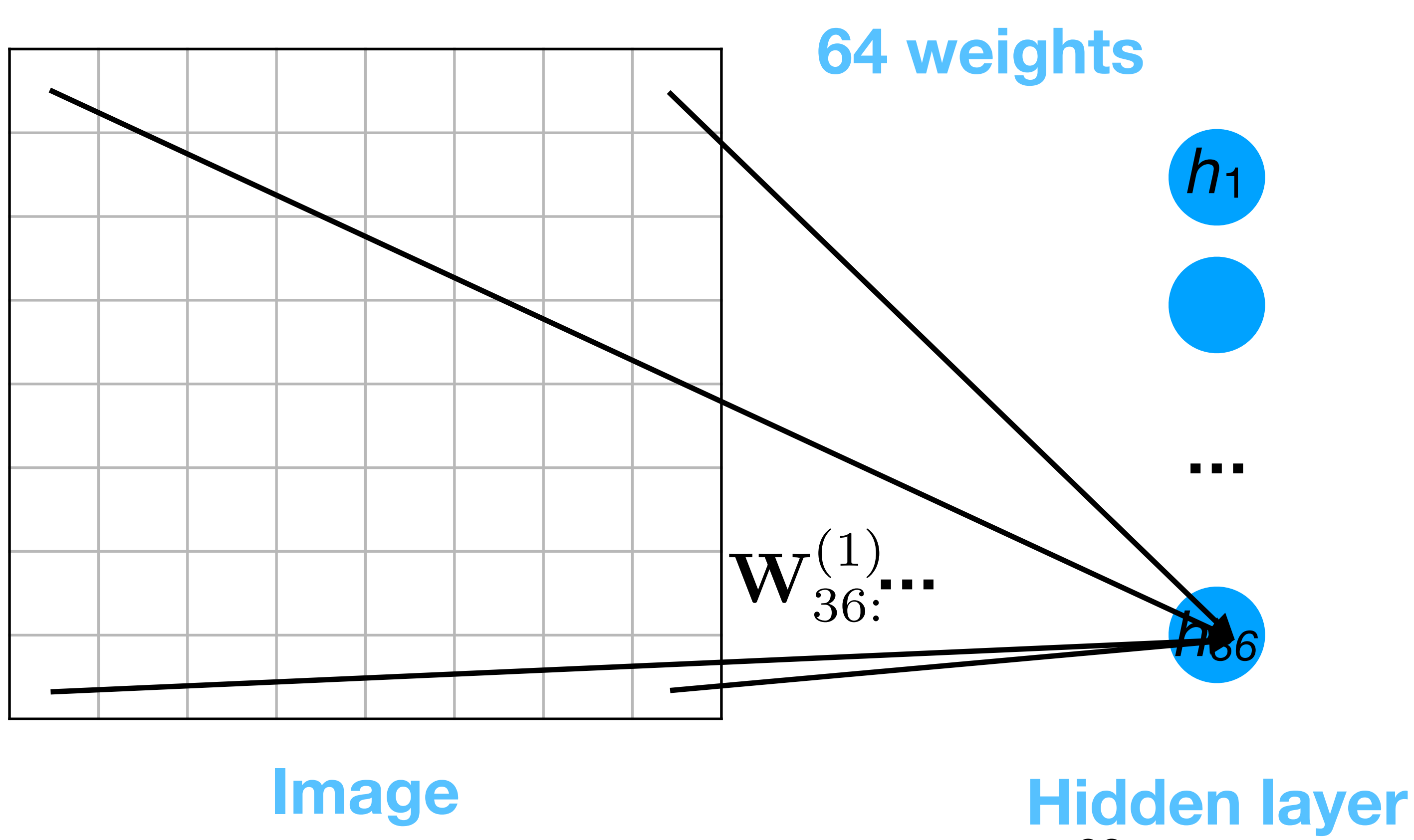
Convolutional neural networks

- CNNs are a special case of feed-forward (FF) NNs.
- In the FF NN below, each of the 36 hidden units is associated with 64 weights.



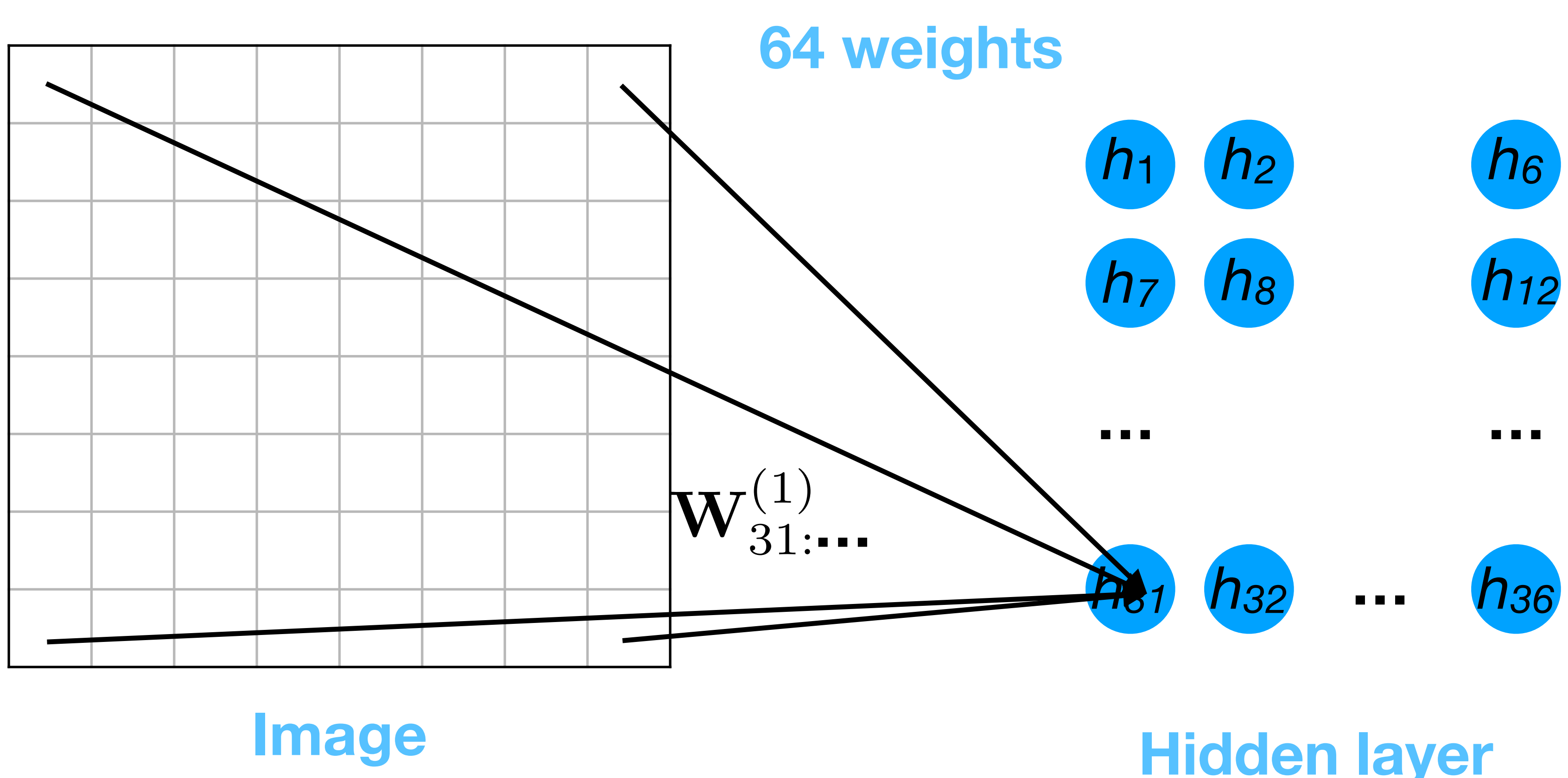
Convolutional neural networks

- CNNs are a special case of feed-forward (FF) NNs.
- In the FF NN below, each of the 36 hidden units is associated with 64 weights.



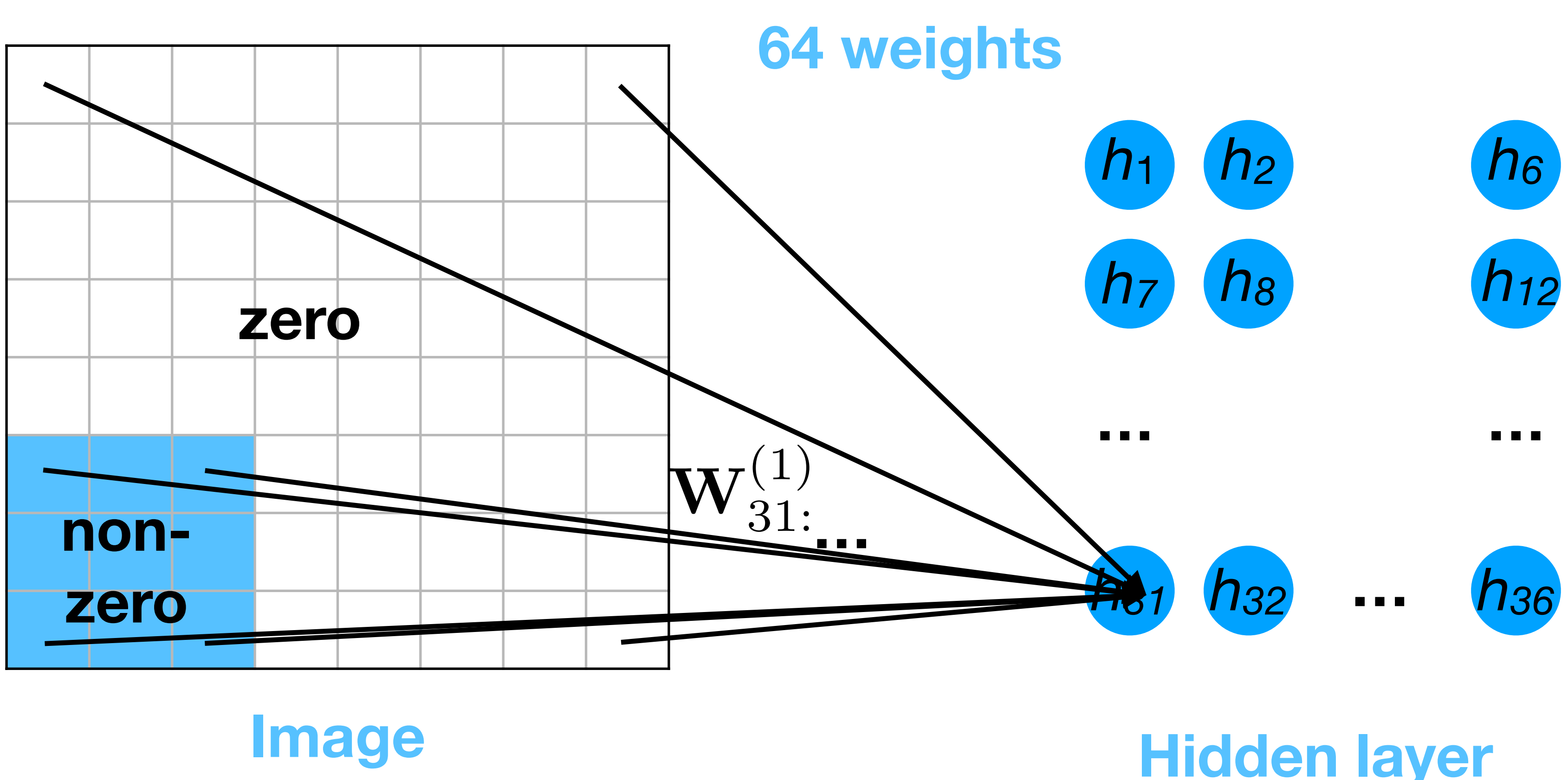
Convolutional neural networks

- We can re-arrange the hidden units into a grid (this just affects the visualization, not the computation).



Convolutional neural networks

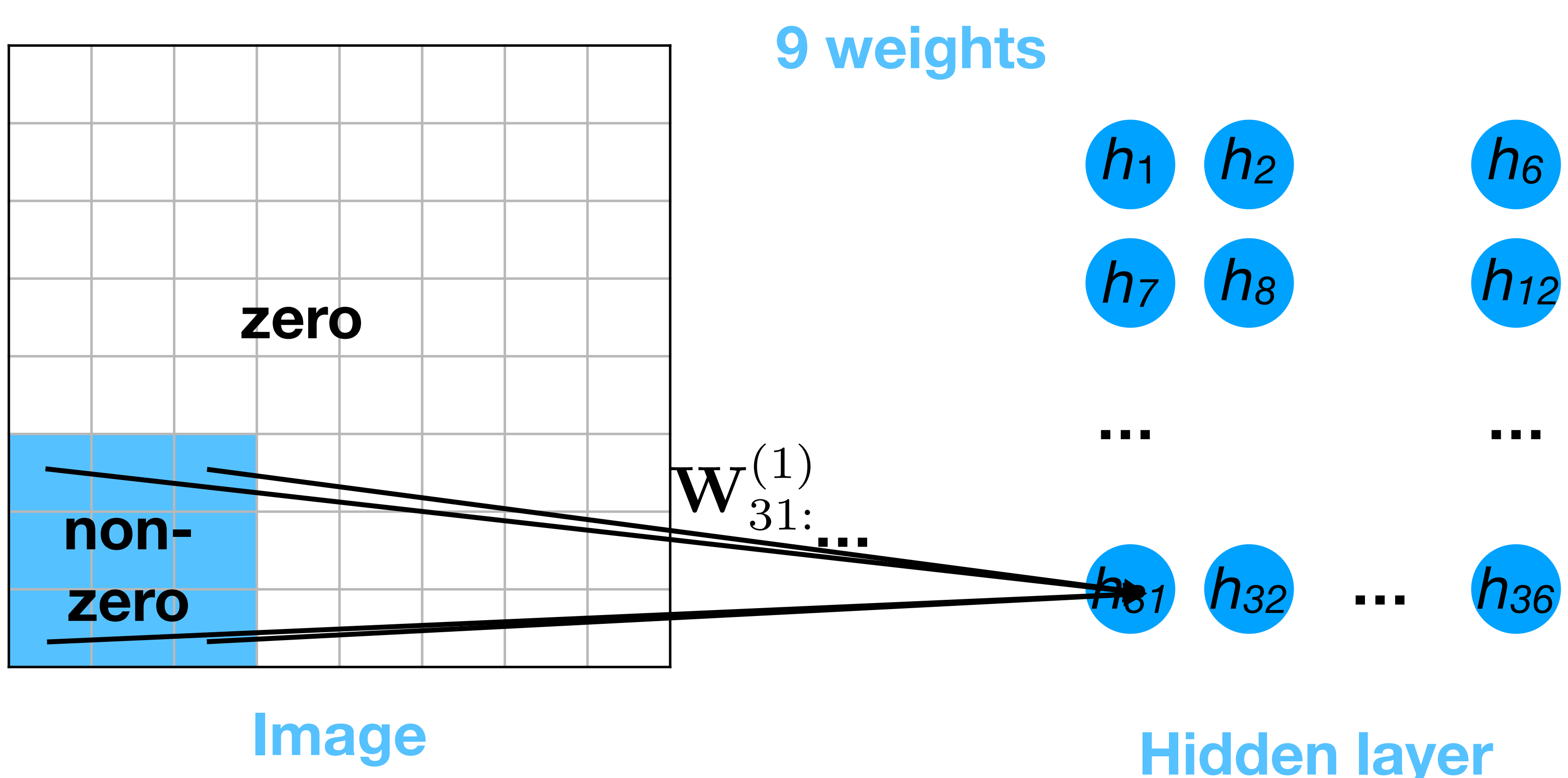
- We can also require that most of the weights for each hidden unit be 0.



Jacob Whitehill, WPI

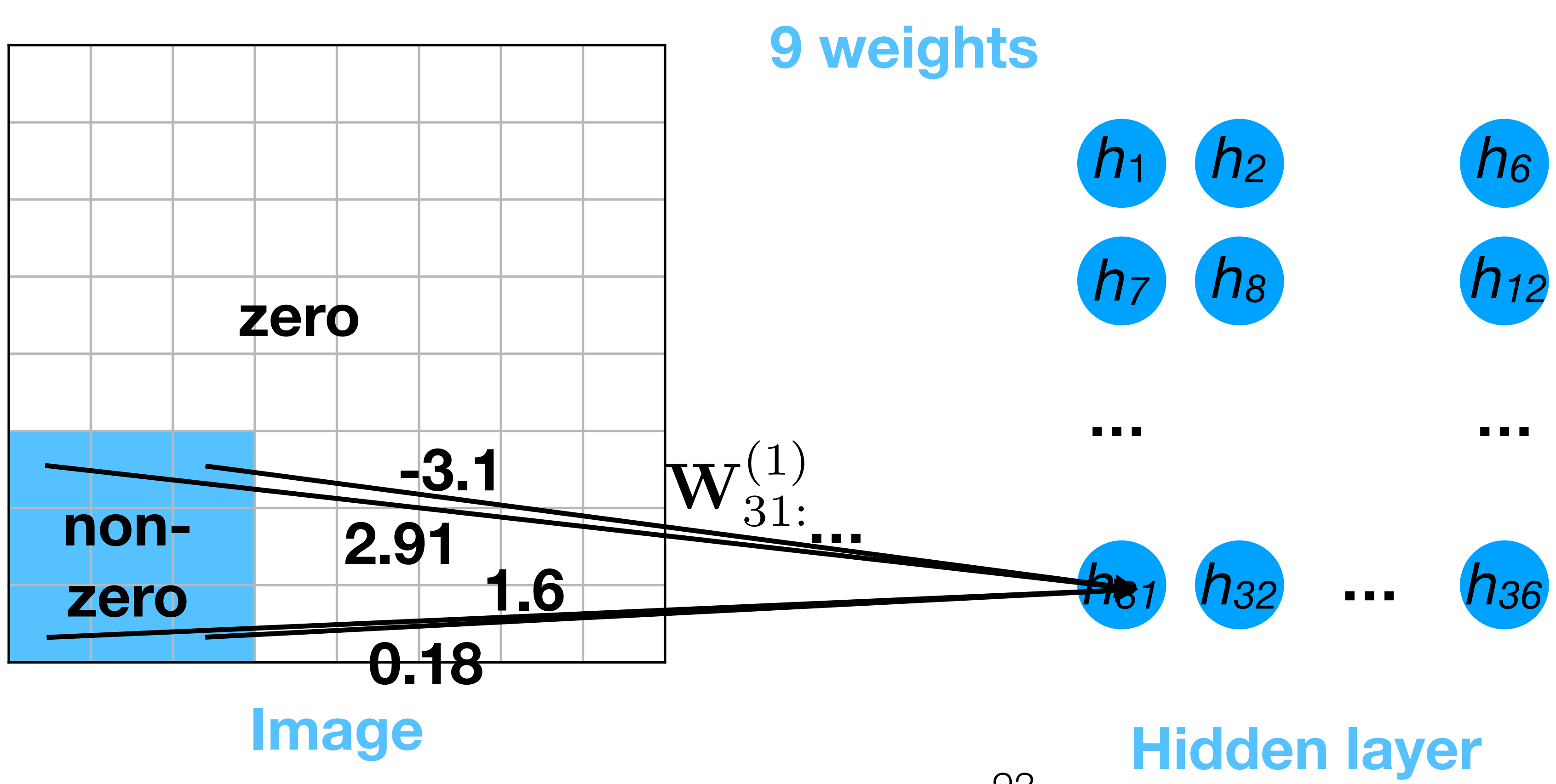
Convolutional neural networks

- Since the image pixels corresponding to the 0-weights contribute nothing to each hidden unit, they can be removed, resulting in just 9 weights per hidden unit.



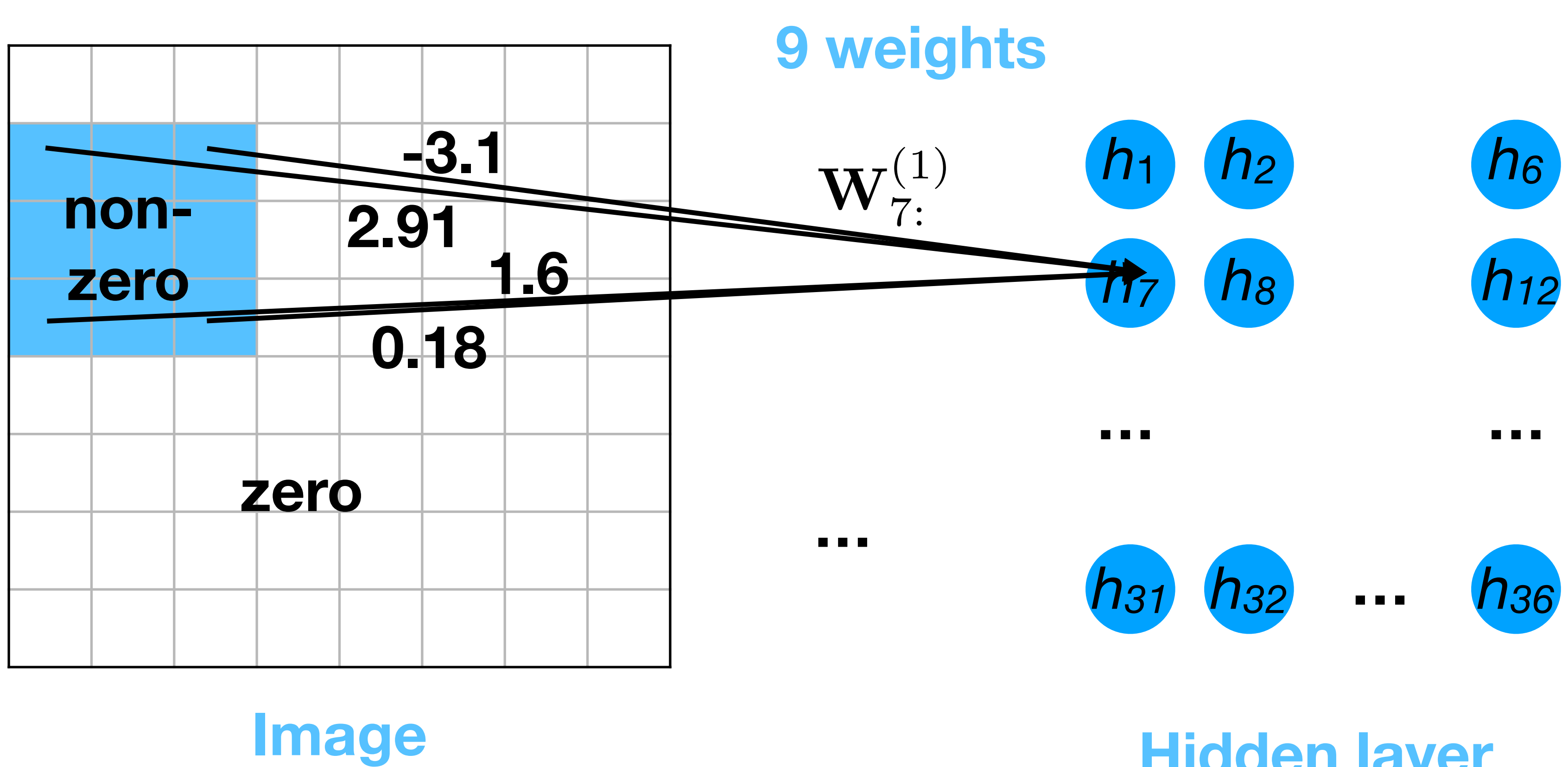
Convolutional neural networks

- Can can further constrain weight matrix $\mathbf{W}^{(1)}$ by requiring that each 3x3 set of non-zero weights be the *same* for each of the hidden units.



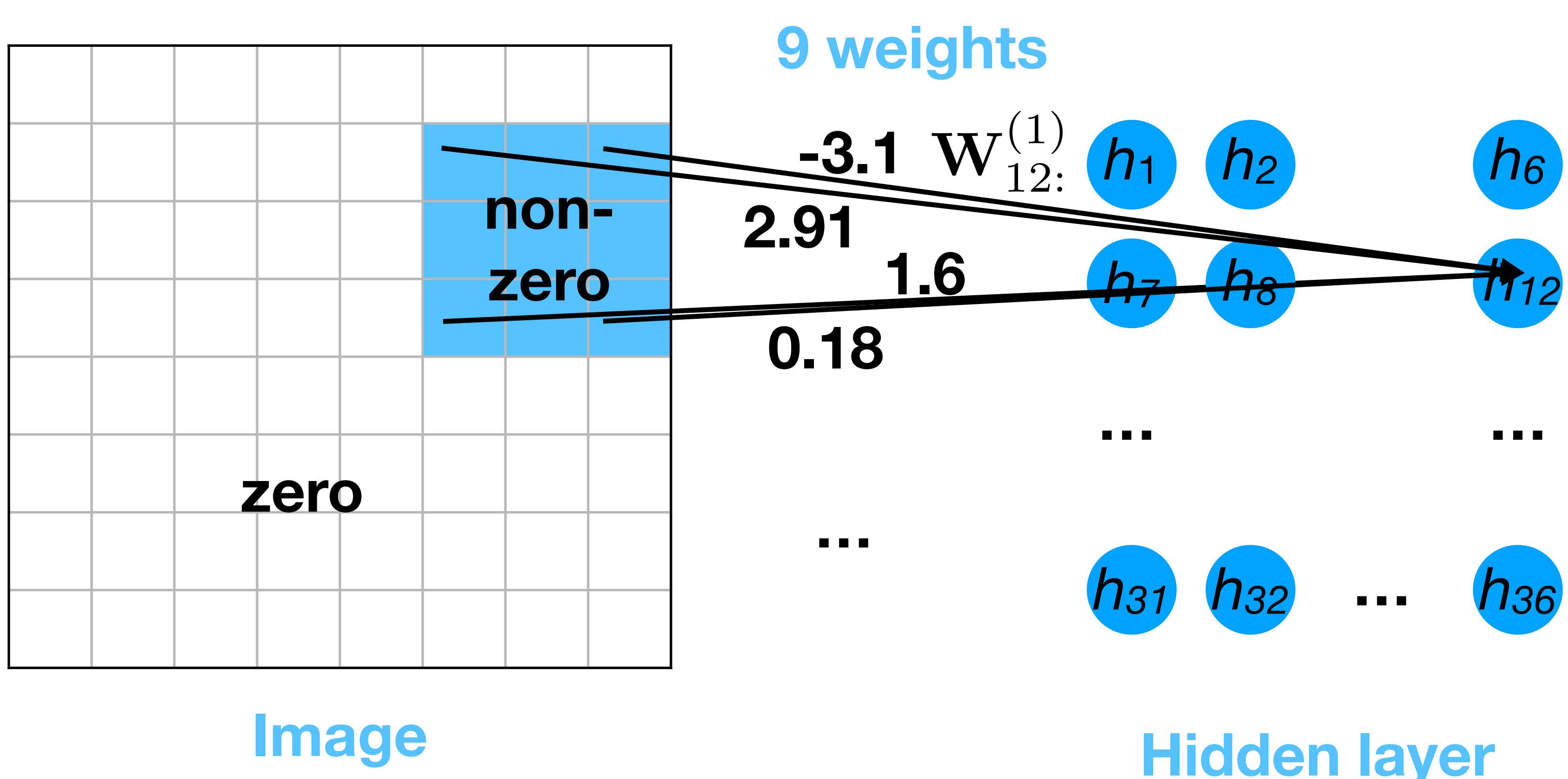
Convolutional neural networks

- Can can further constrain weight matrix $\mathbf{W}^{(1)}$ by requiring that each 3x3 set of non-zero weights be the *same* for each of the hidden units.



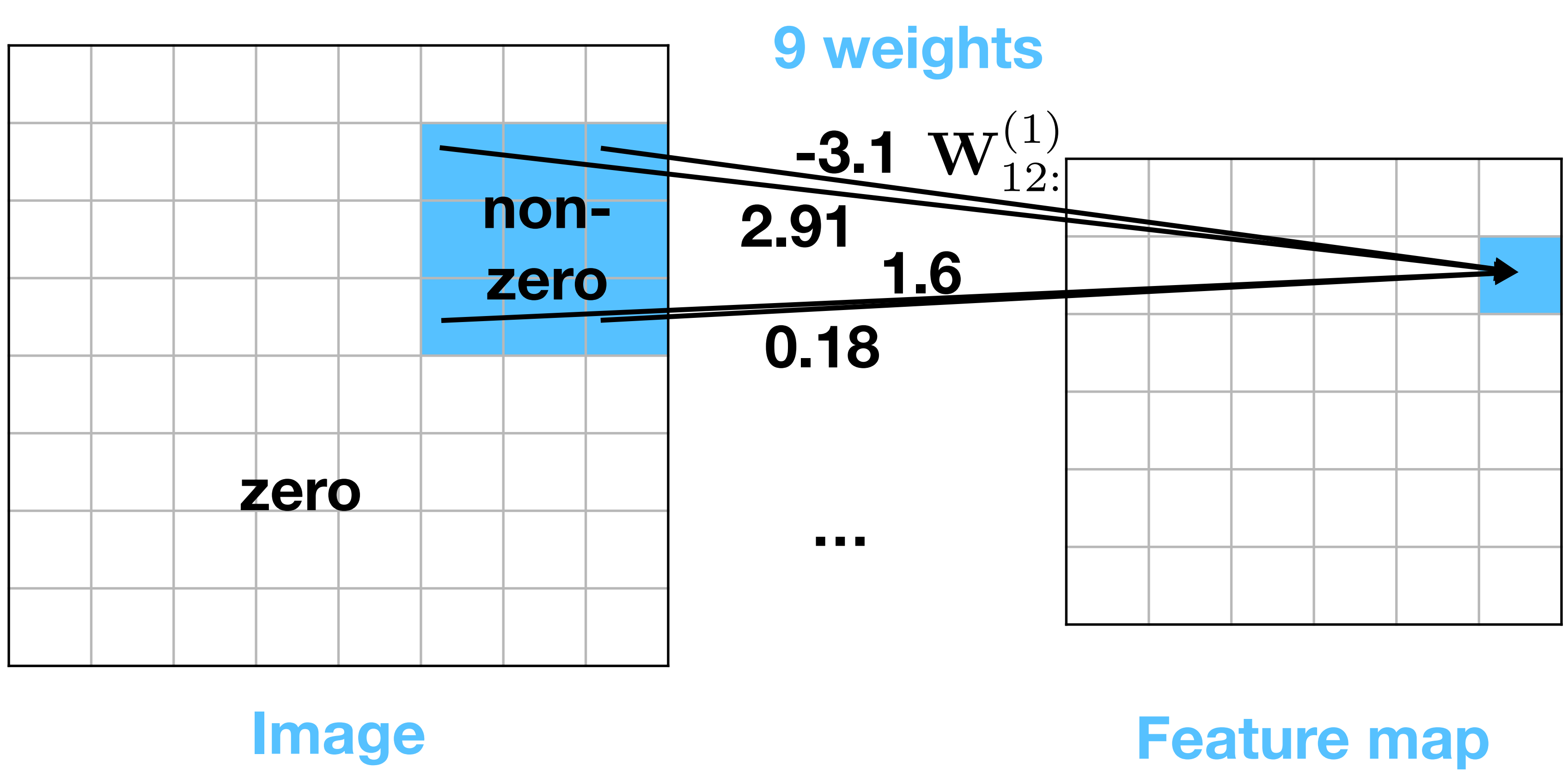
Convolutional neural networks

- Can can further constrain weight matrix $\mathbf{W}^{(1)}$ by requiring that each 3x3 set of non-zero weights be the *same* for each of the hidden units.



Convolutional neural networks

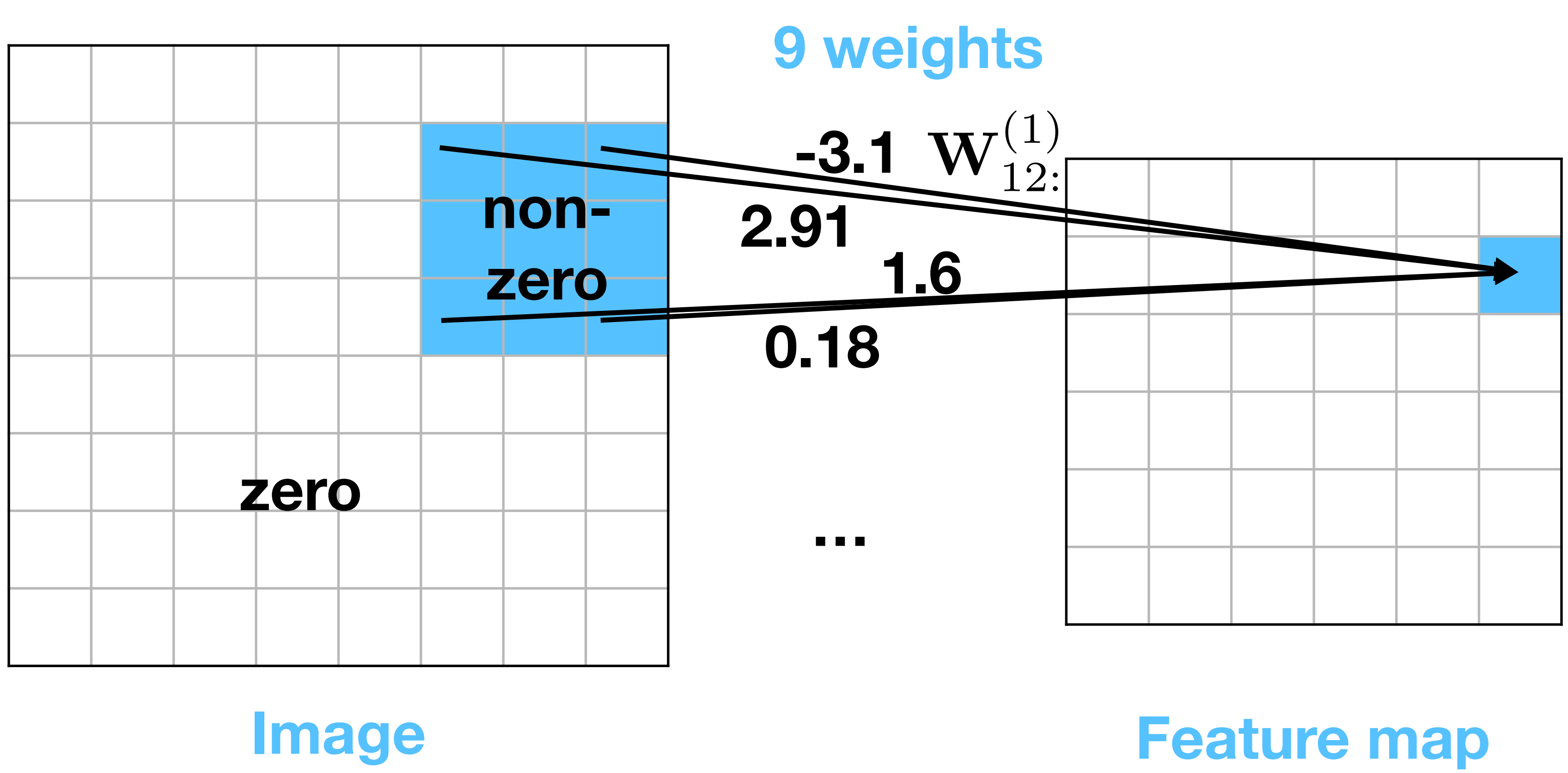
- This is now completely equivalent to a convolutional layer instead of a general feed-forward layer.



Jacob Whitehill, WPI

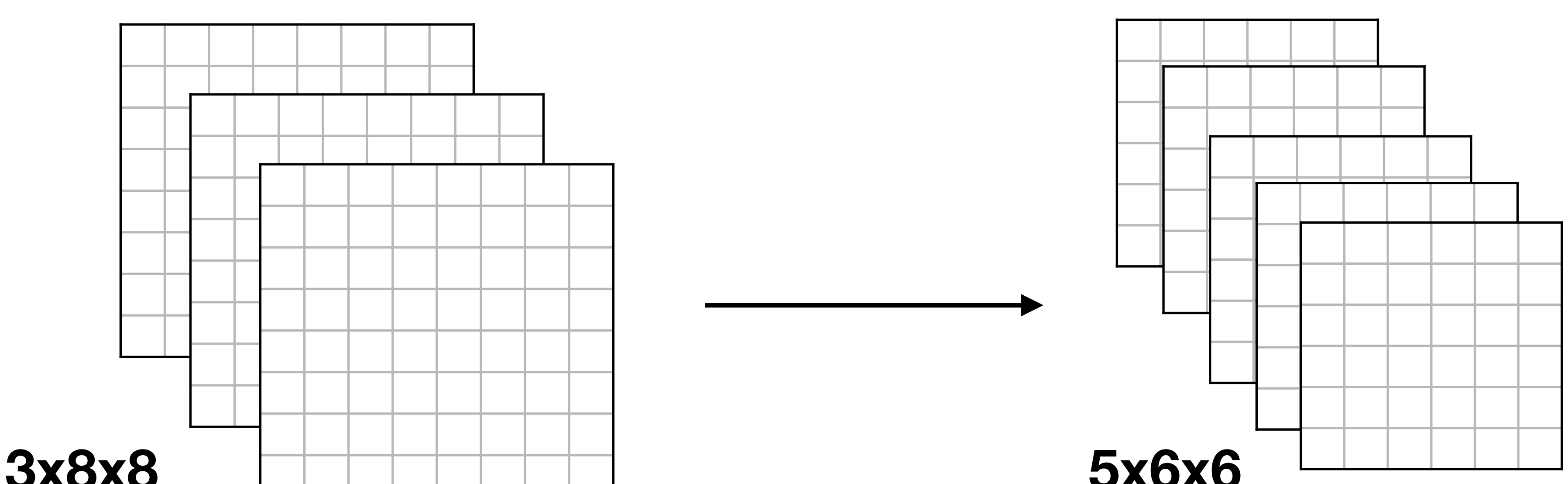
Convolutional neural networks

- We can thus use the exact same backpropagation algorithm to train CNNs as we did fully-connected NNs.



CNN vs. FCNN: Number of parameters

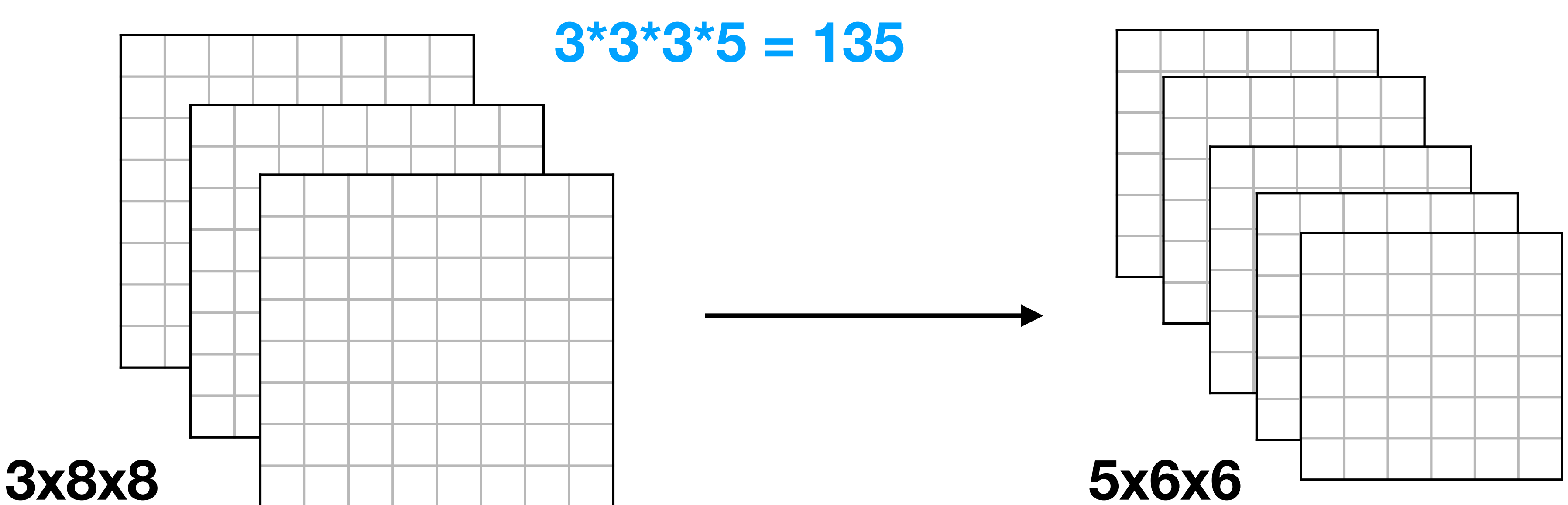
- Suppose layer l of a NN is 3x8x8 (i.e., 3 channels each of an 8x8 grid).
- We then convolve layer l with 5 different convolution filters, each of size 3x3x3; this produces layer l+1, whose size is 5x6x6.
- How many total weights do we have in a CNN?



Jacob Whitehill, WPI

CNN vs. FCNN: Number of parameters

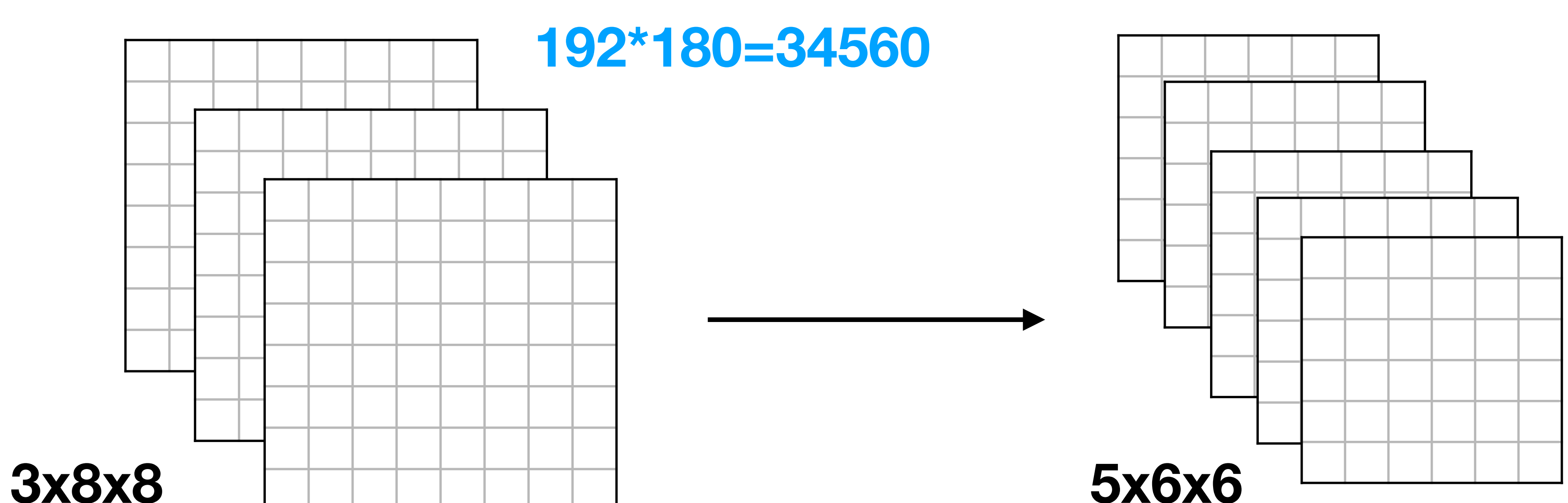
- Suppose layer *l* of a NN is 3x8x8 (i.e., 3 channels each of an 8x8 grid).
- We then convolve layer *l* with 5 different convolution filters, each of size 3x3x3; this produces layer *l*+1, whose size is 5x6x6.
- How many total weights do we have in a CNN?



Jacob Whitehill, WPI

CNN vs. FCNN: Number of parameters

- Suppose layer l of a NN has 192 neurons.
- We then connect all 192 neurons from layer l to all 180 neurons of layer l+1; i.e., the NN is **fully connected** (FC).
- How many total weights do we have in a FCNN?



Jacob Whitehill, WPI