CS/DS 541: Class 11

Jacob Whitehill

Convolution

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

0	1
2	-1

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

+

G

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

2	3
0	2

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

49		

+

B

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

4	2
-1	1

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

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

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).

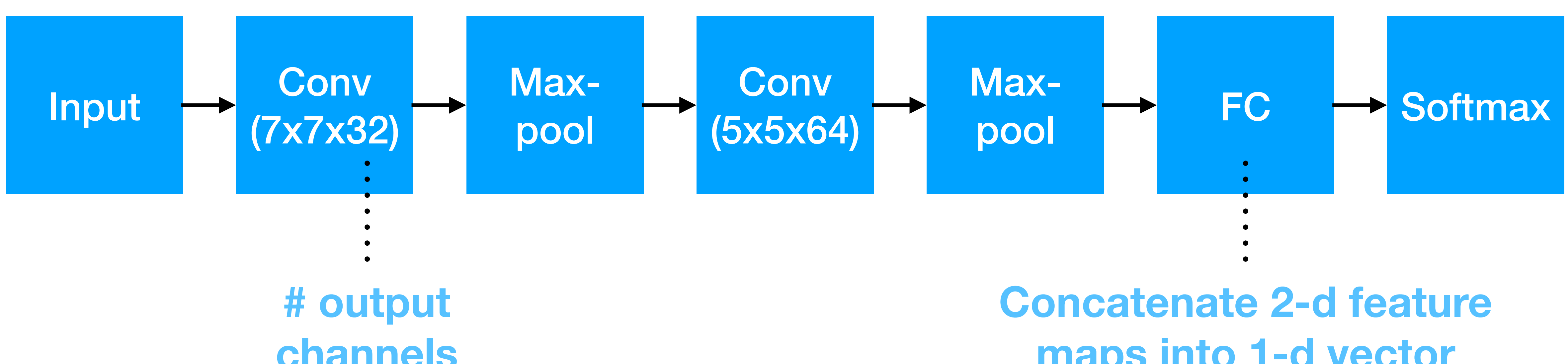
- CNNs have revolutionized computer vision, especially in object detection, object recognition, semantic segmentation, activity recognition, and other tasks.

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

1
2
-2
0
3

Filter

1
2
3

Feature map

-1
-2
7

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

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**.

1
2
-2
0
3

Image

1
2
3

Filter

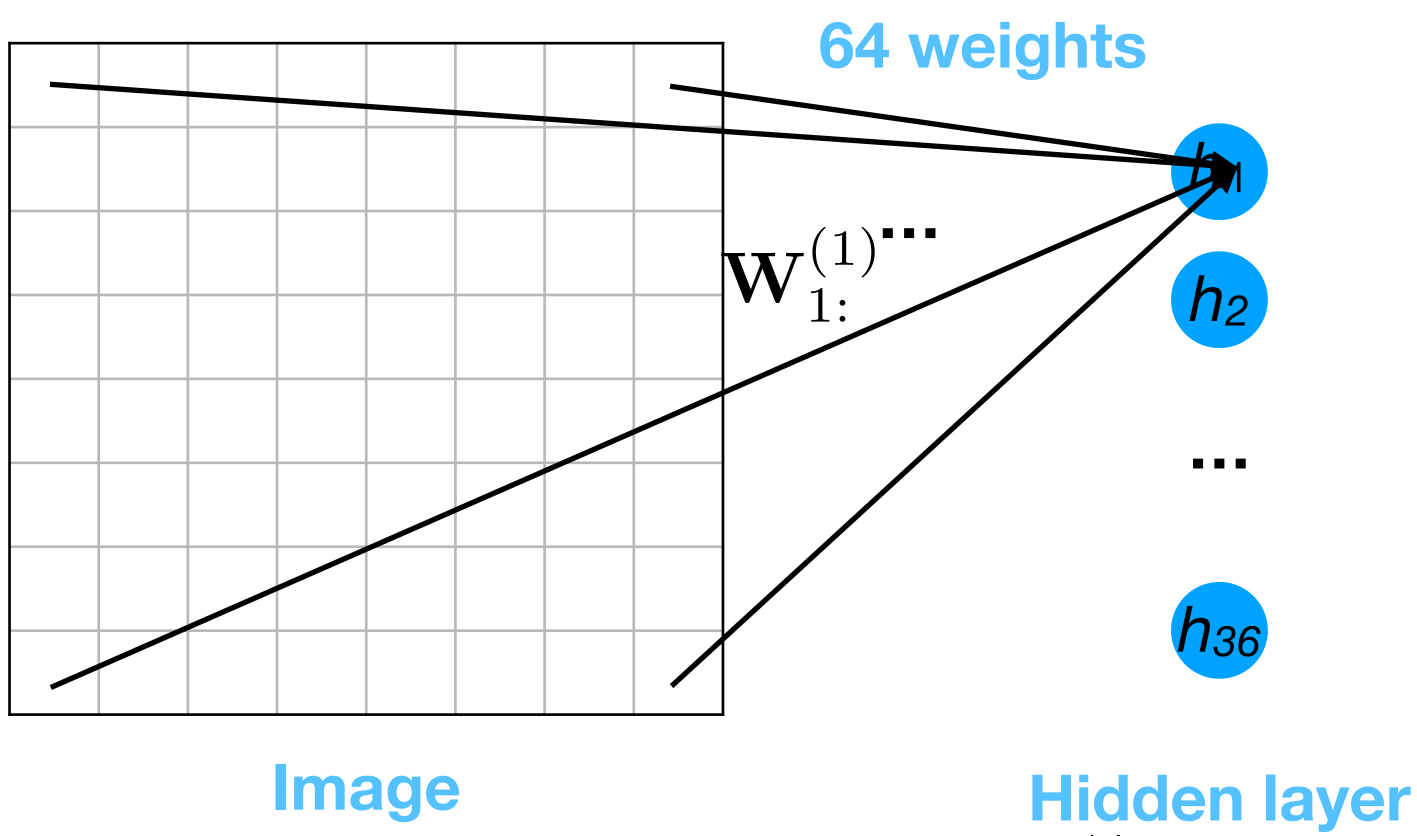
-1
-2
7

Feature map

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

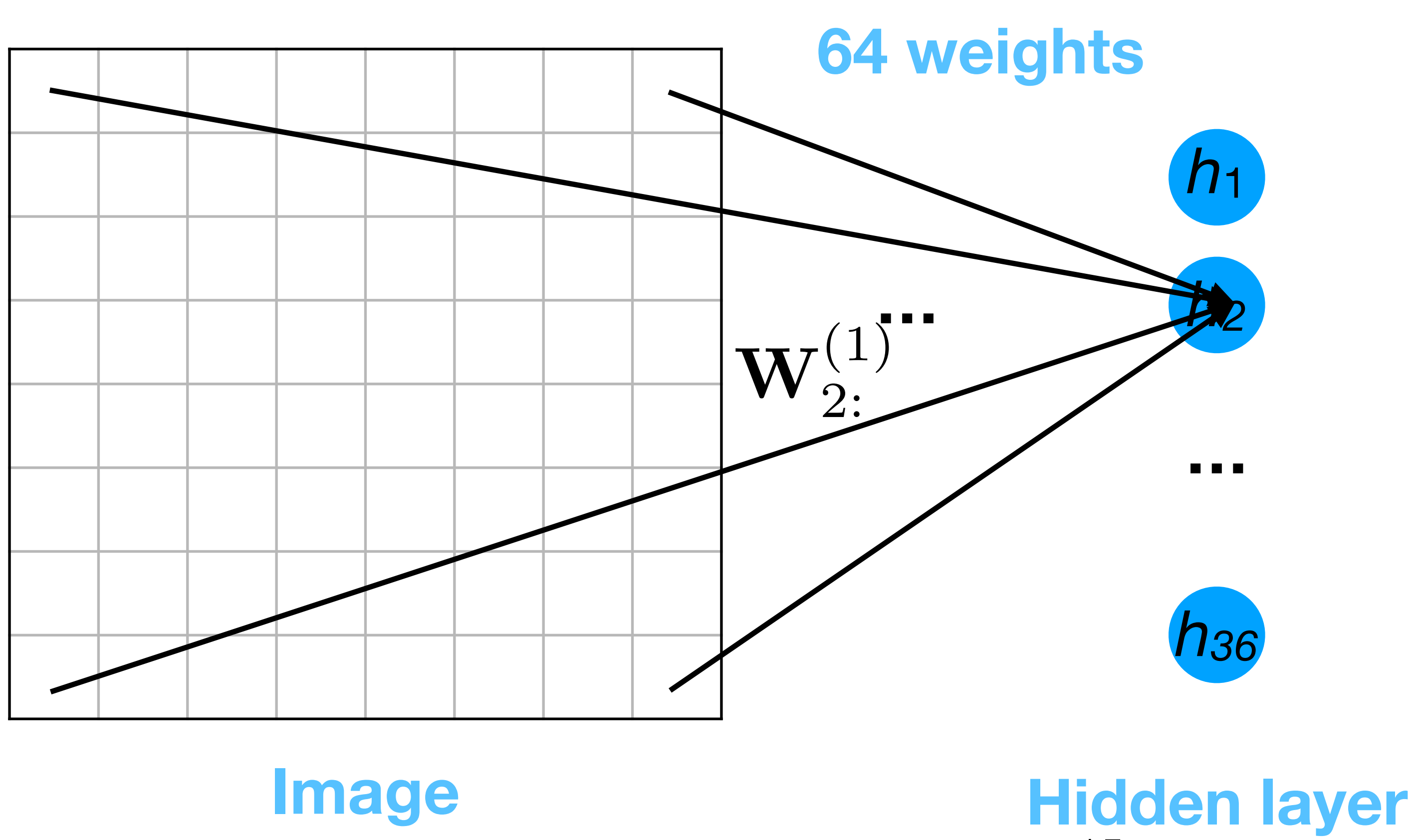
Convolutional neural networks

- 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.



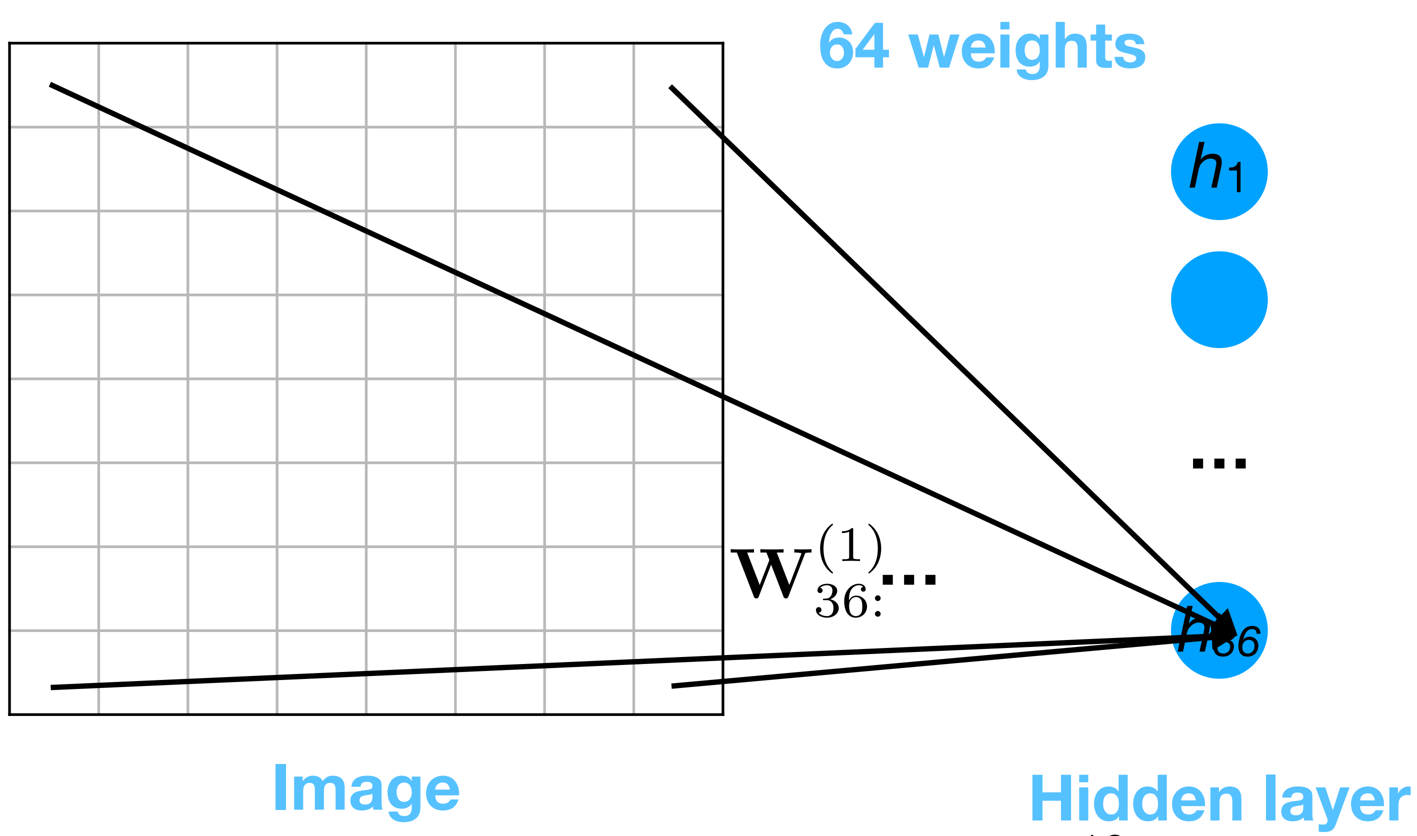
Convolutional neural networks

- 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.



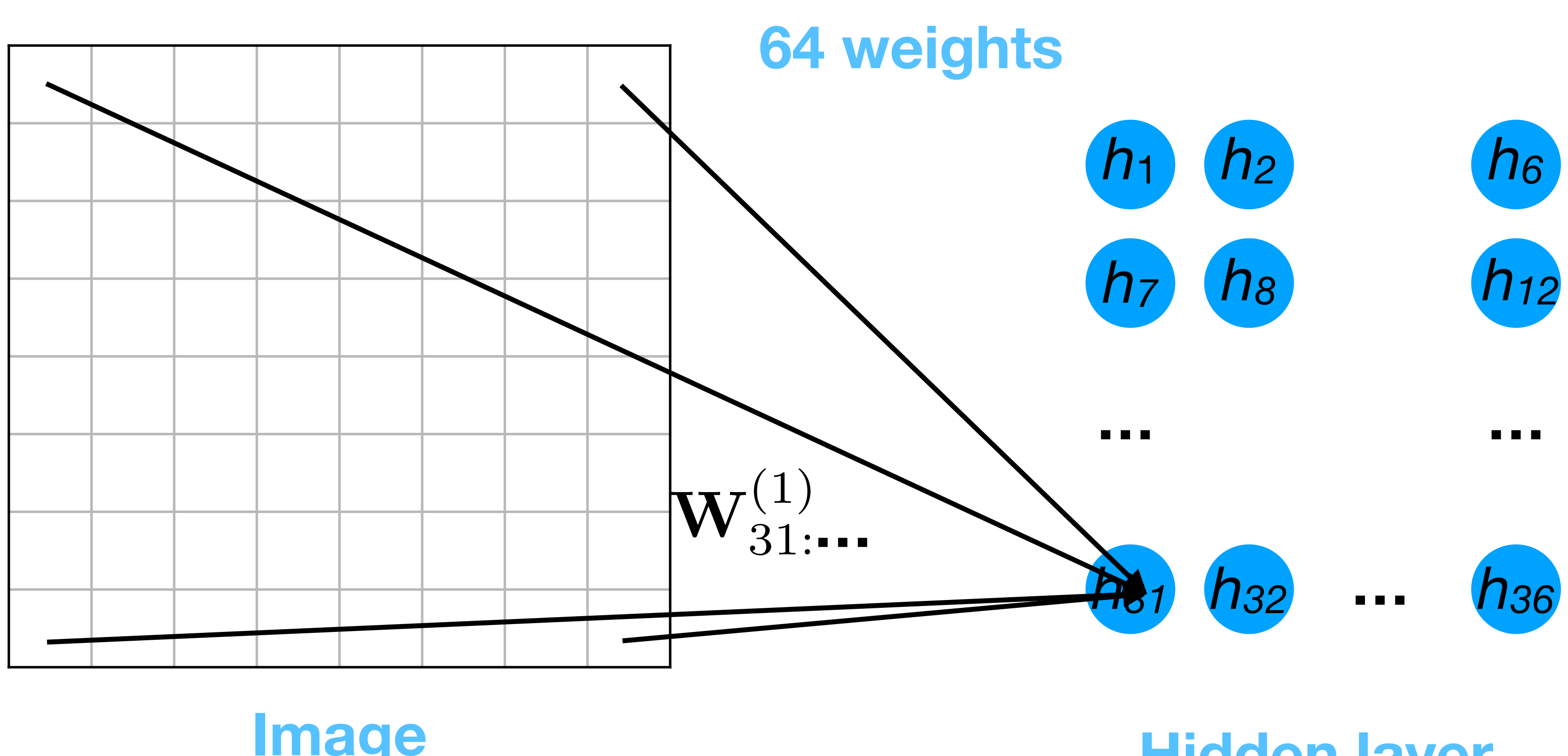
Convolutional neural networks

- 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.



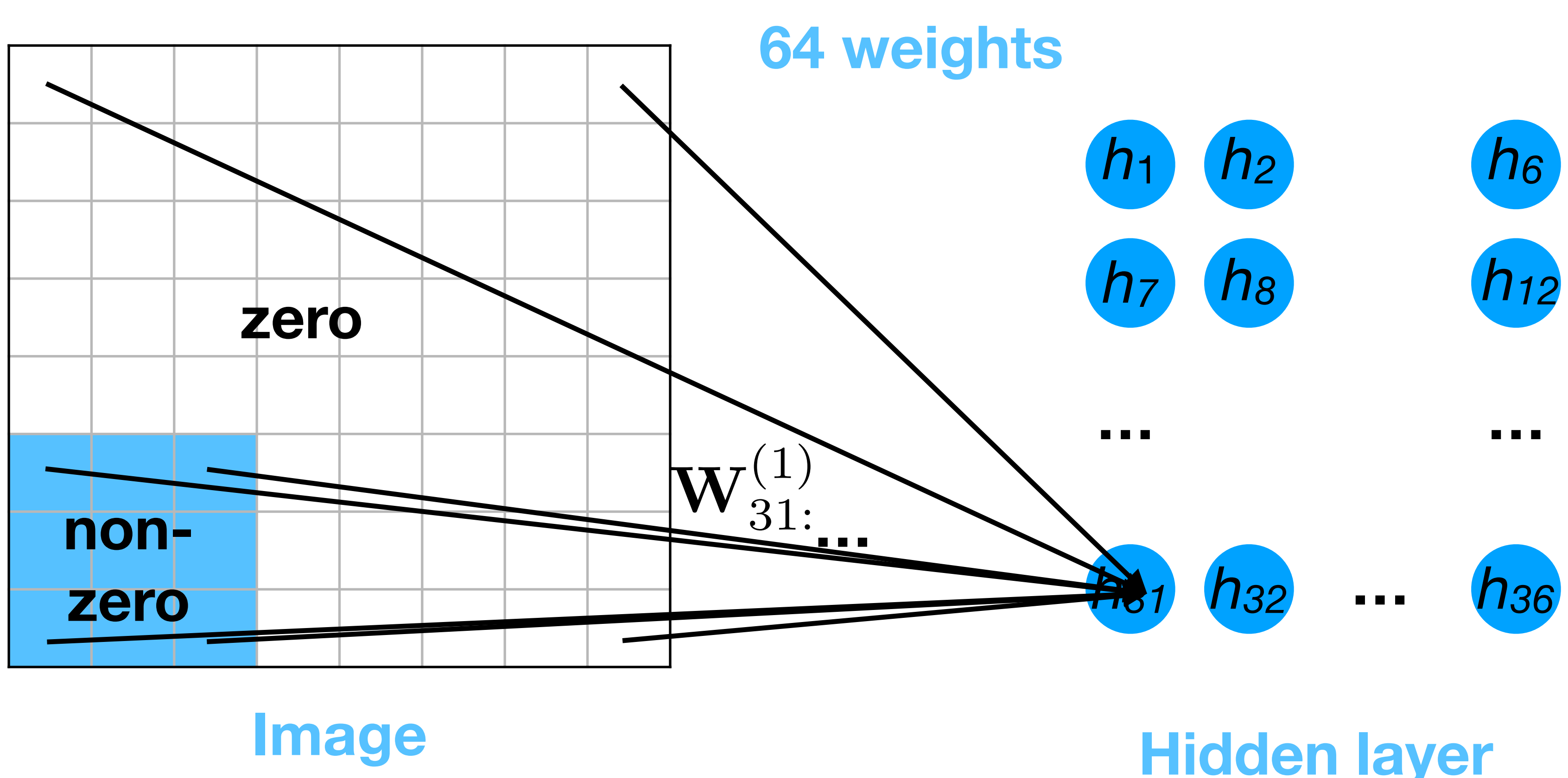
Convolutional neural networks

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



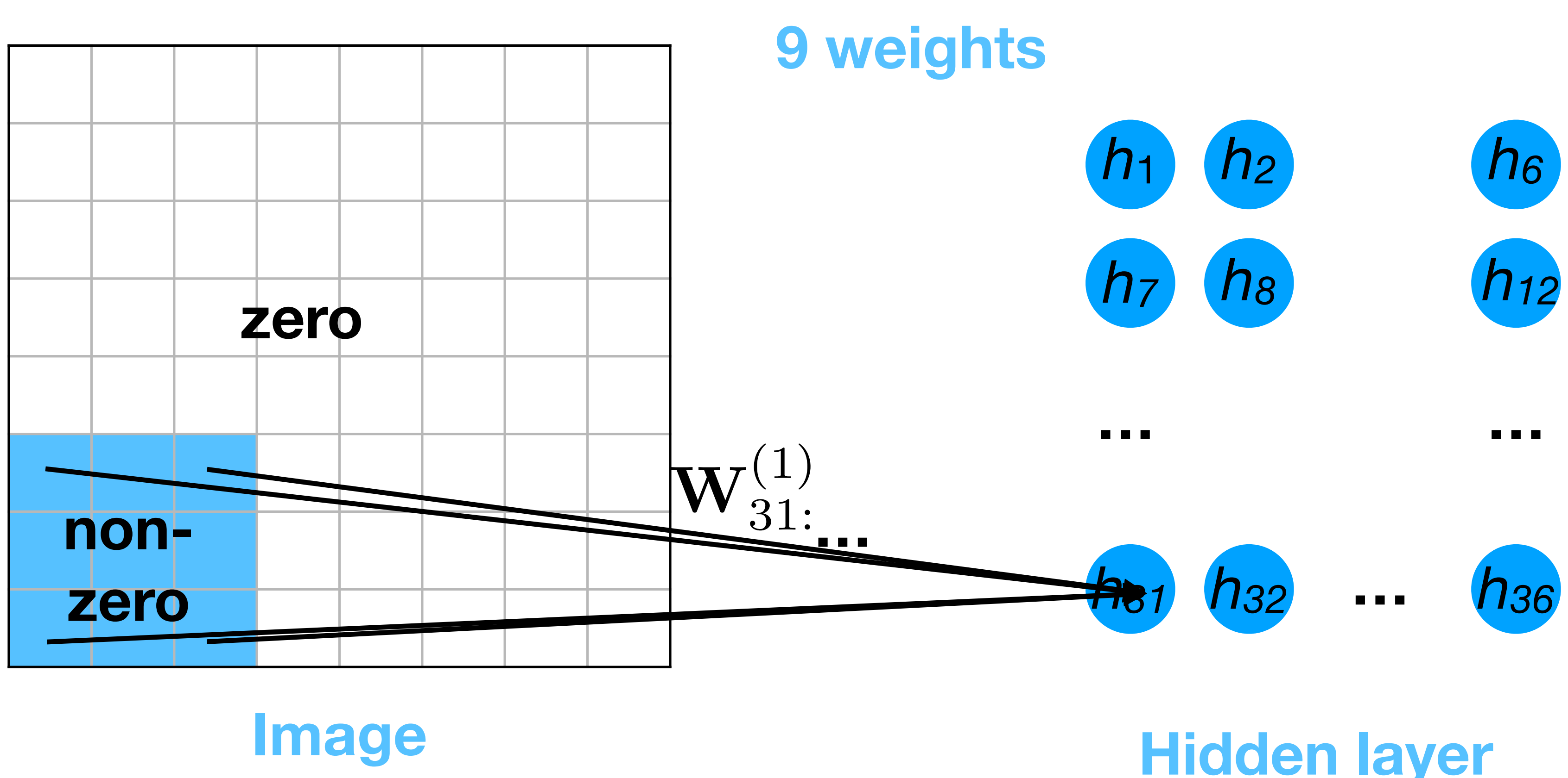
Convolutional neural networks

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



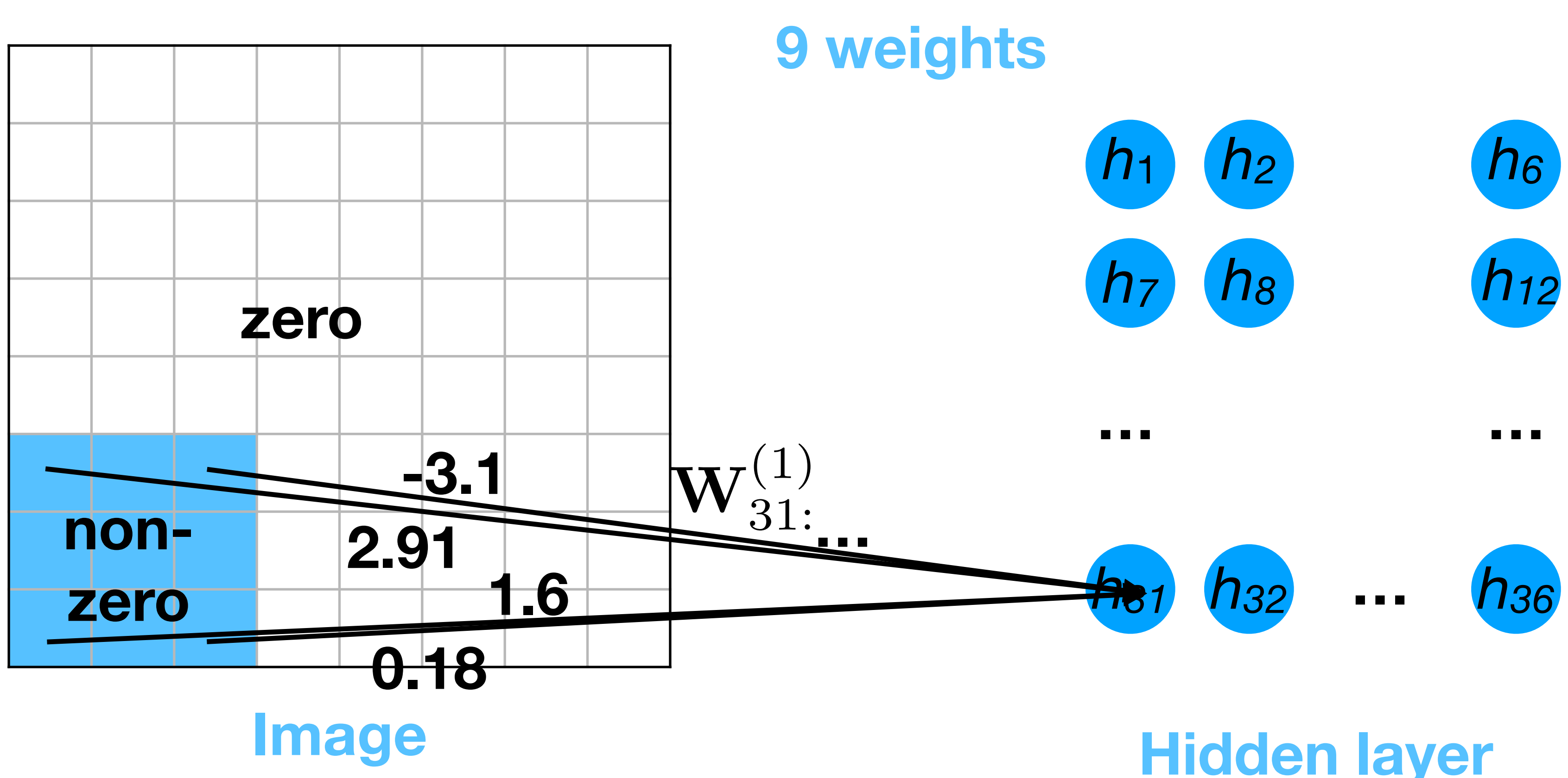
Convolutional neural networks

- 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.



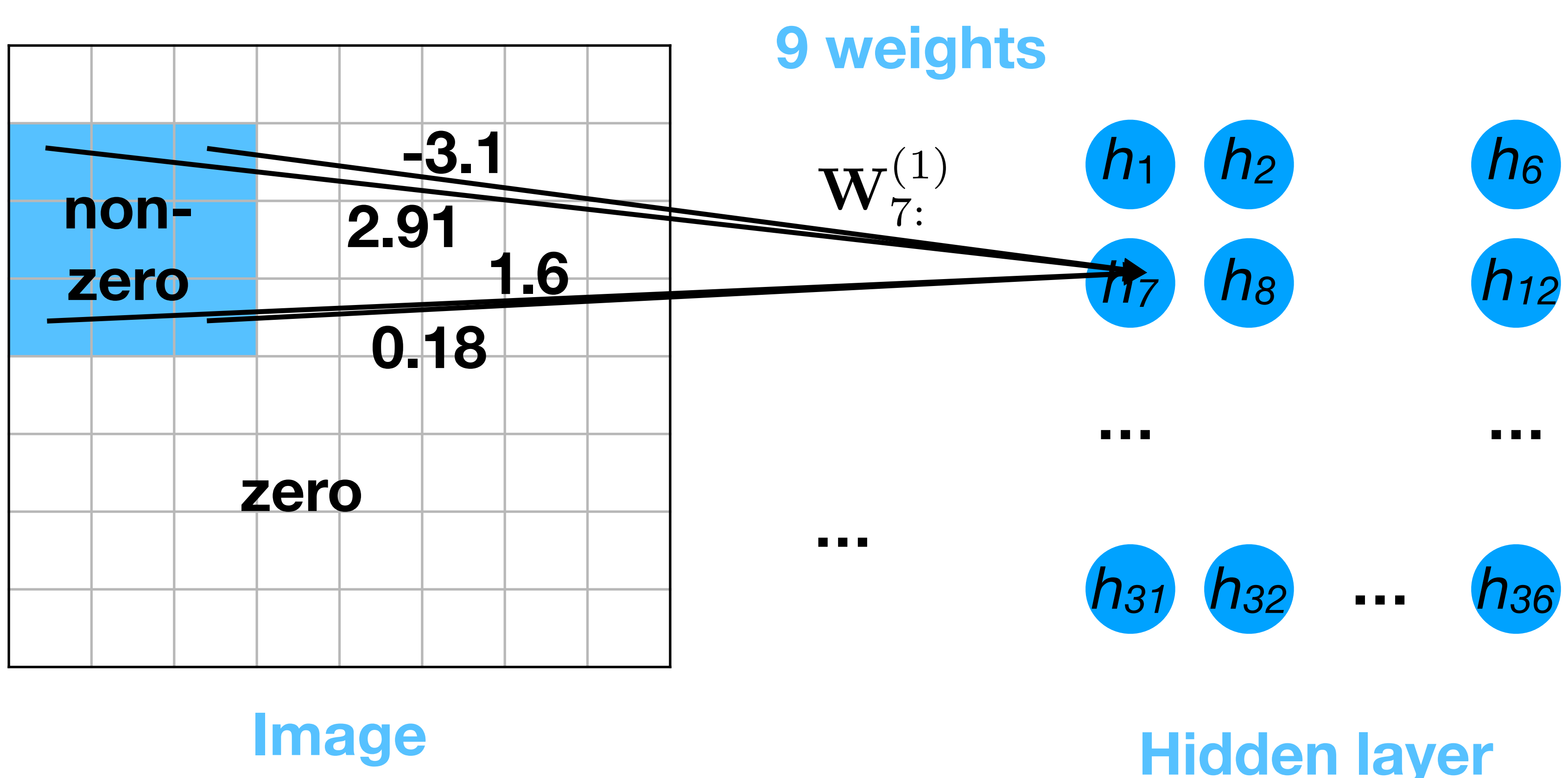
Convolutional neural networks

- 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

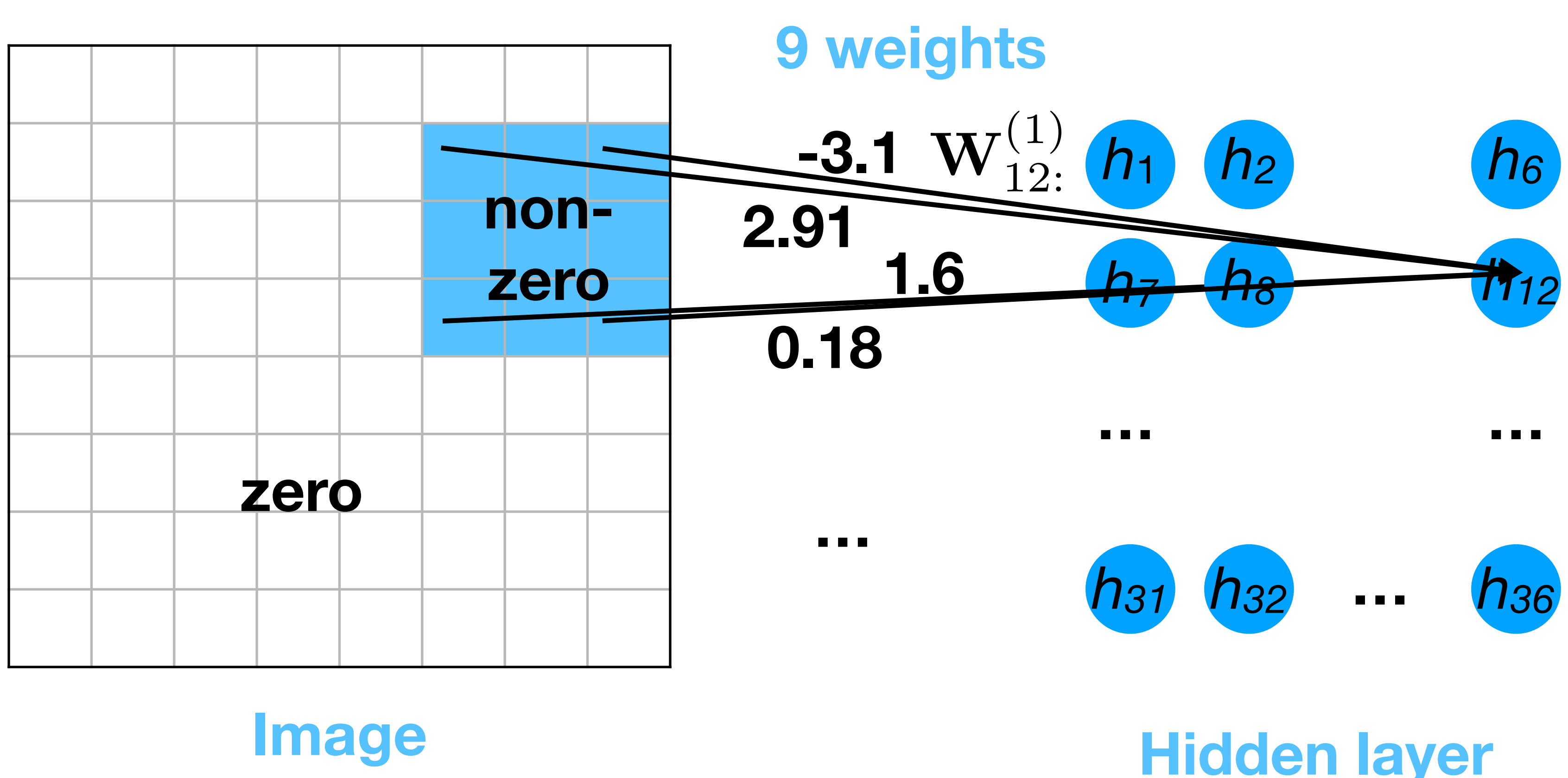
- Can can further constrain weight matrix $\mathbf{W}^{(1)}$ by requiring that each 3x3 set of non-zero weights be the *same* for each of the hidden units.



Jacob Whitehill, WPI

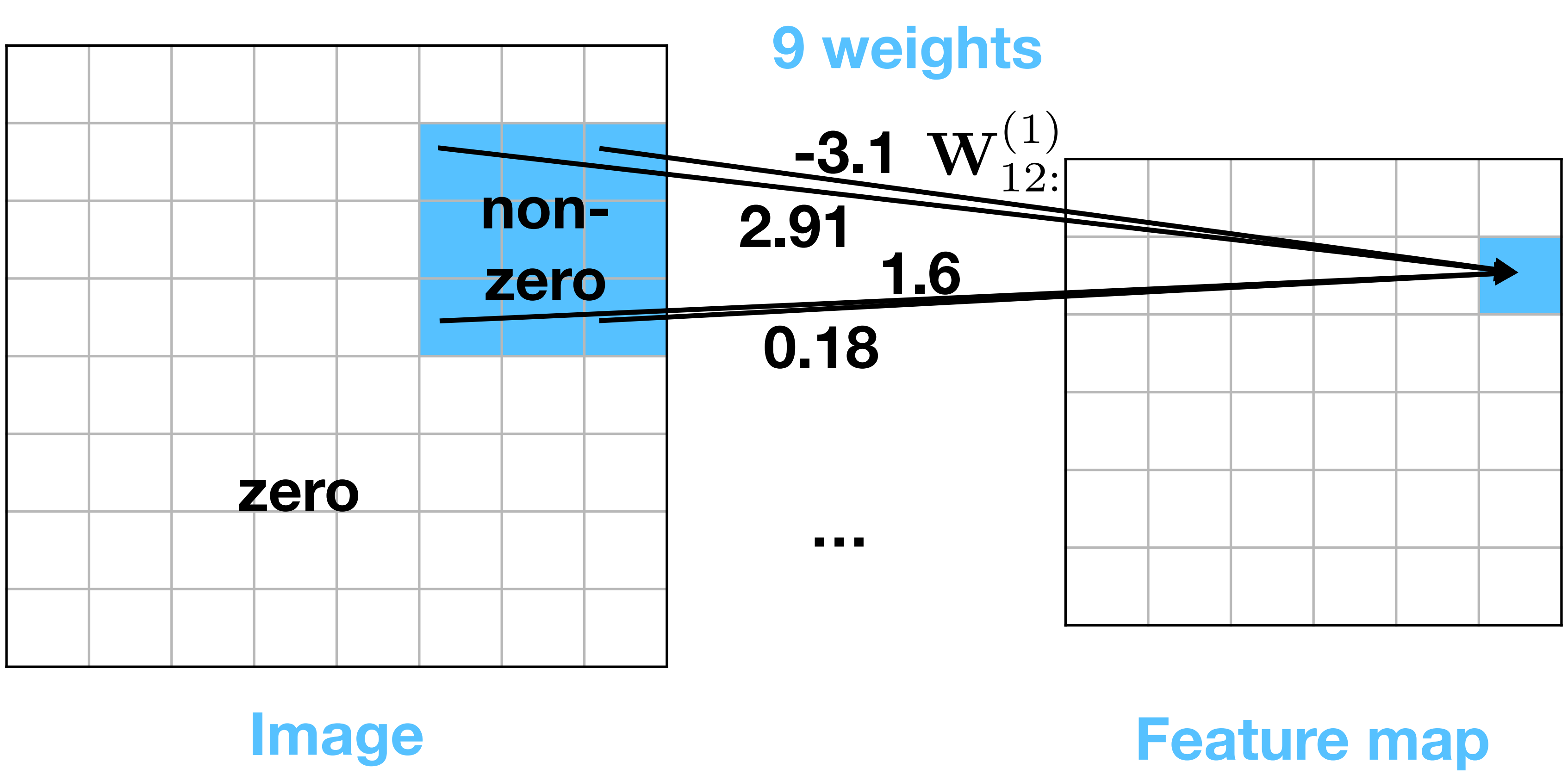
Convolutional neural networks

- 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.



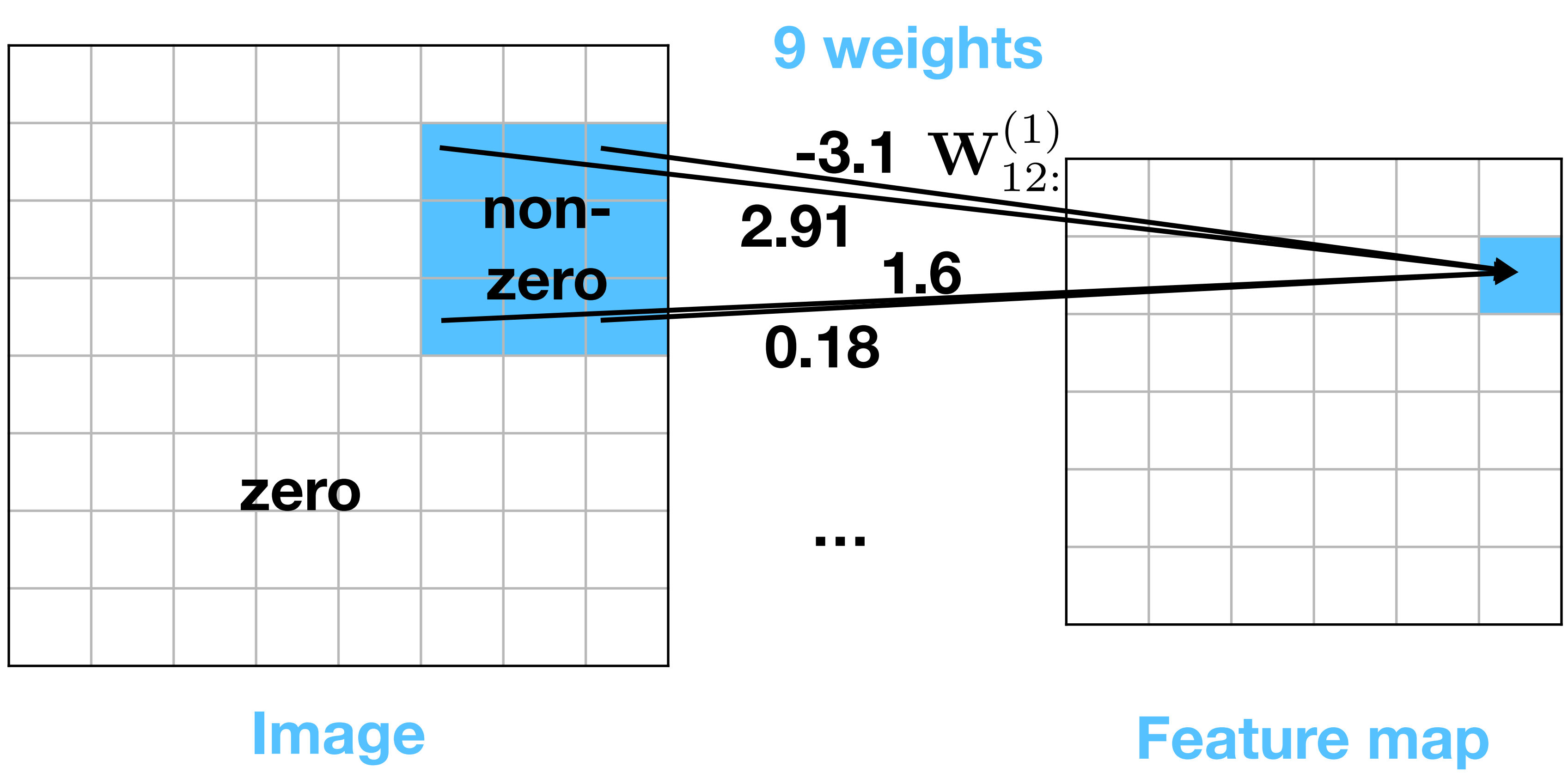
Convolutional neural networks

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



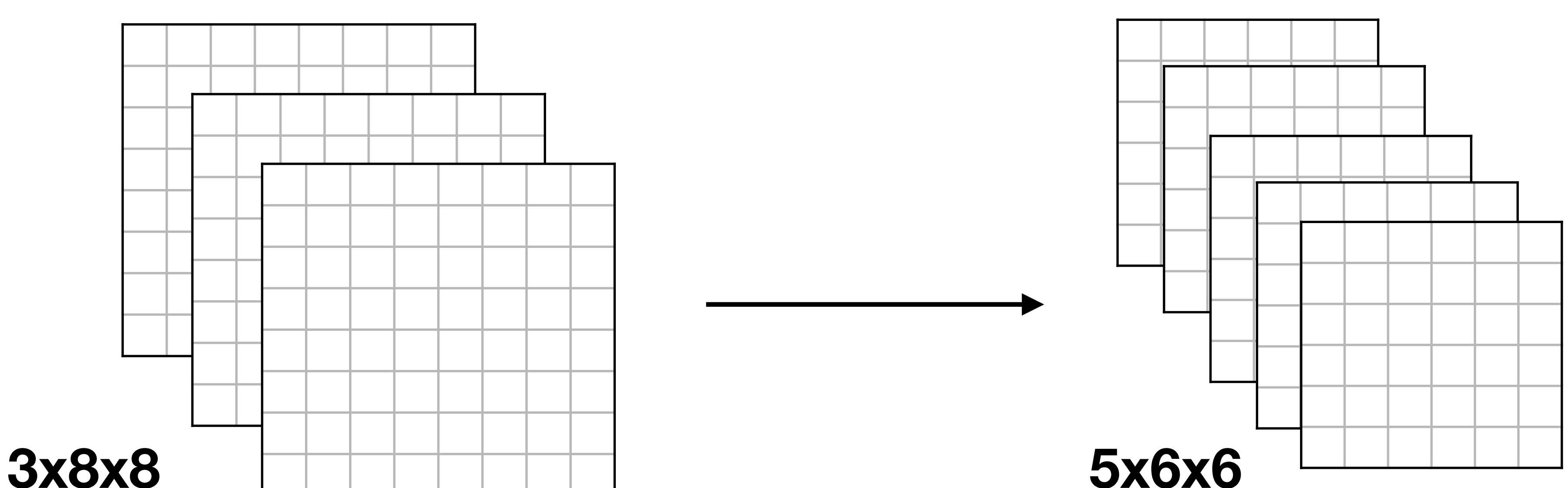
Convolutional neural networks

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



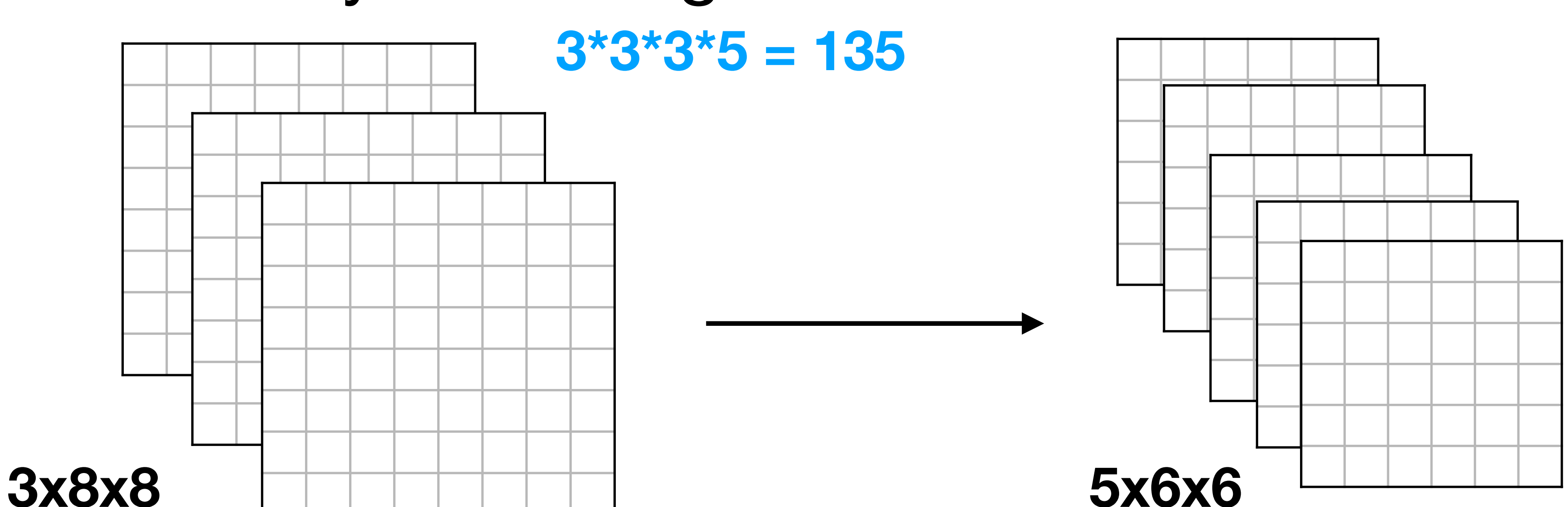
CNN vs. FCNN: Number of parameters

- 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?



CNN vs. FCNN: Number of parameters

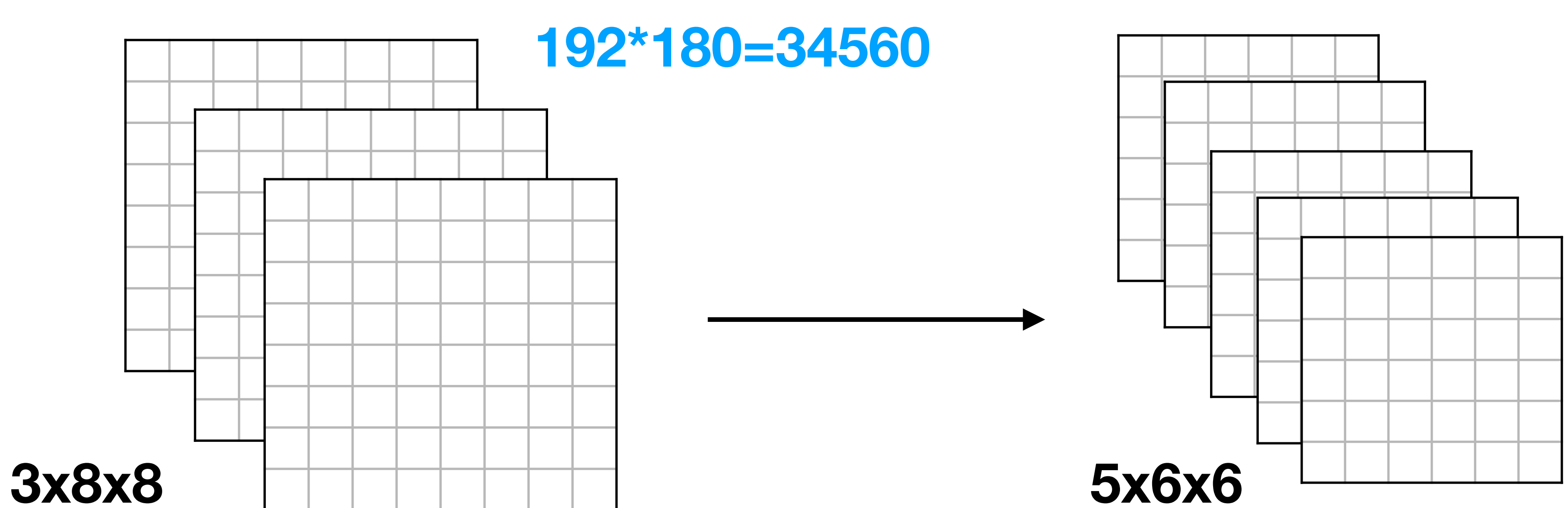
- Suppose layer *l* of a NN is 3x8x8 (i.e., 3 channels each of an 8x8 grid).
- We then convolve layer *l* with 5 different convolution filters, each of size 3x3x3; this produces layer *l*+1, whose size is 5x6x6.
- How many total weights do we have in a CNN?



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?



Exercise

- Consider the following convolutional layer (conv2-1).

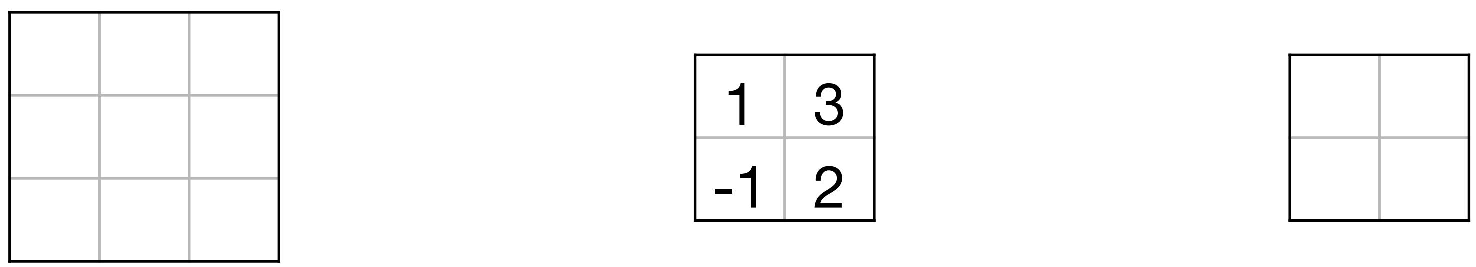
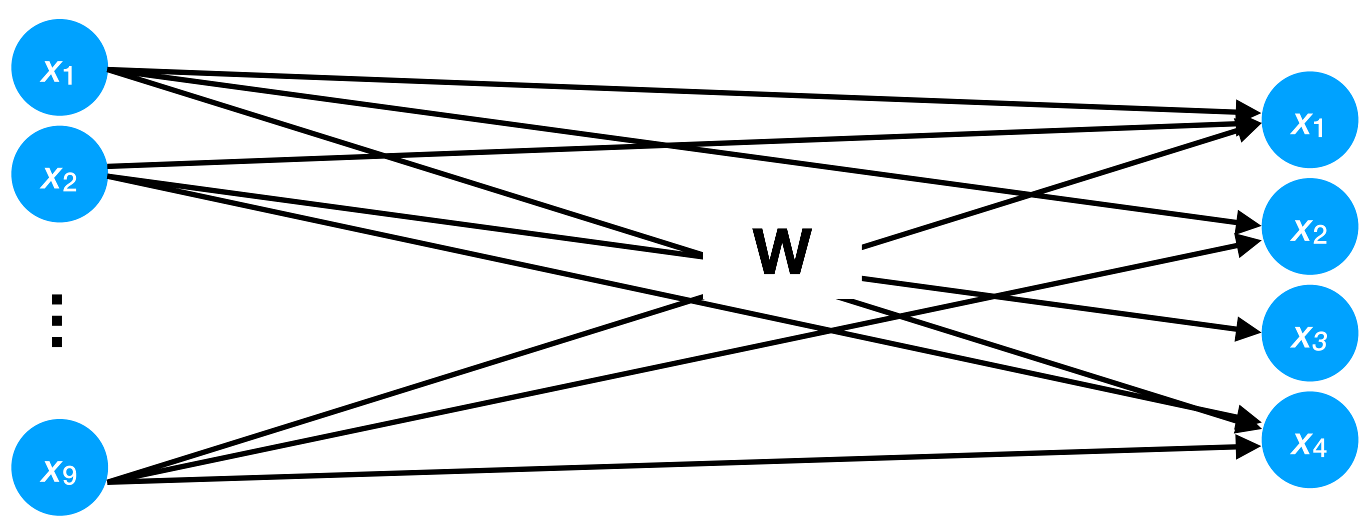


Image Filter Feature map

- How could we encode the same operation as a fully-connected layer with weights **W** (for **h**=**Wx**)?



Solution

- Consider the following convolutional layer (conv2-1).

Image

1	3
-1	2

Filter

Feature map

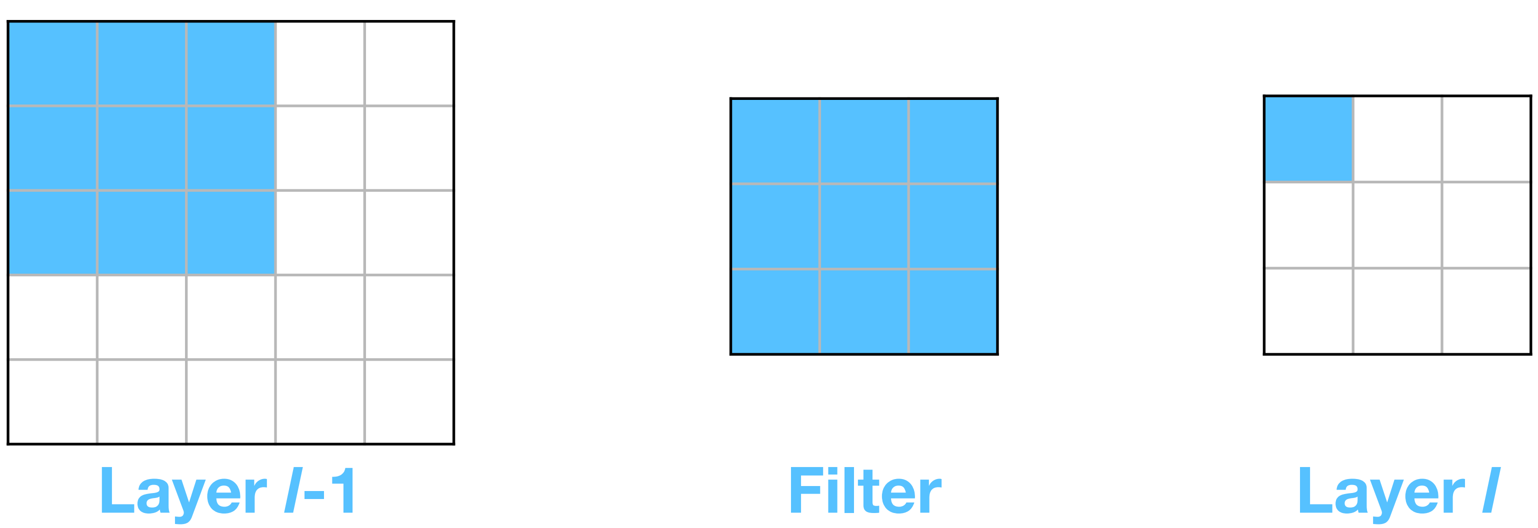
- How could we encode the same operation as a fully-connected layer with weights **W**? Doubly-circulant matrix:

$$\mathbf{W} = \begin{bmatrix} f_{11} & f_{12} & 0 & f_{21} & f_{22} & 0 & 0 & 0 & 0 \\ 0 & f_{11} & f_{12} & 0 & f_{21} & f_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & f_{11} & f_{12} & 0 & f_{21} & f_{22} & 0 \\ 0 & 0 & 0 & 0 & f_{11} & f_{12} & 0 & f_{21} & f_{22} \end{bmatrix}$$

Receptive fields in CNNs

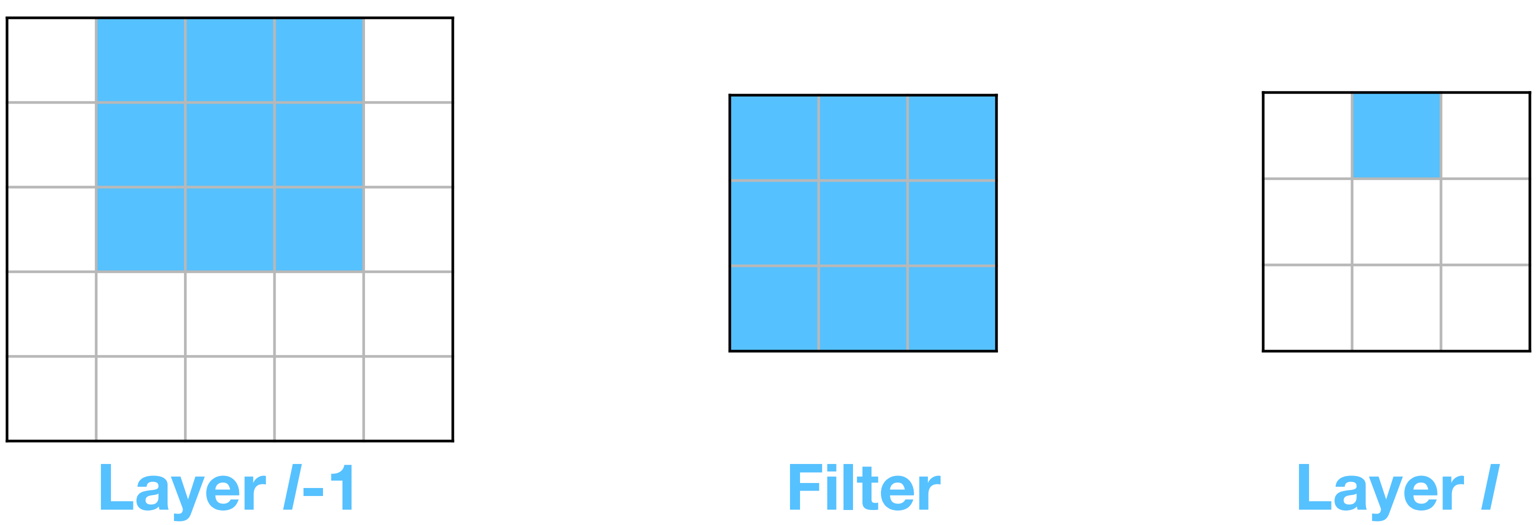
Receptive fields

- Each neuron *i* in a convolutional layer *l* depends only on neurons in a local region around *i* in the previous layer *l*-1.



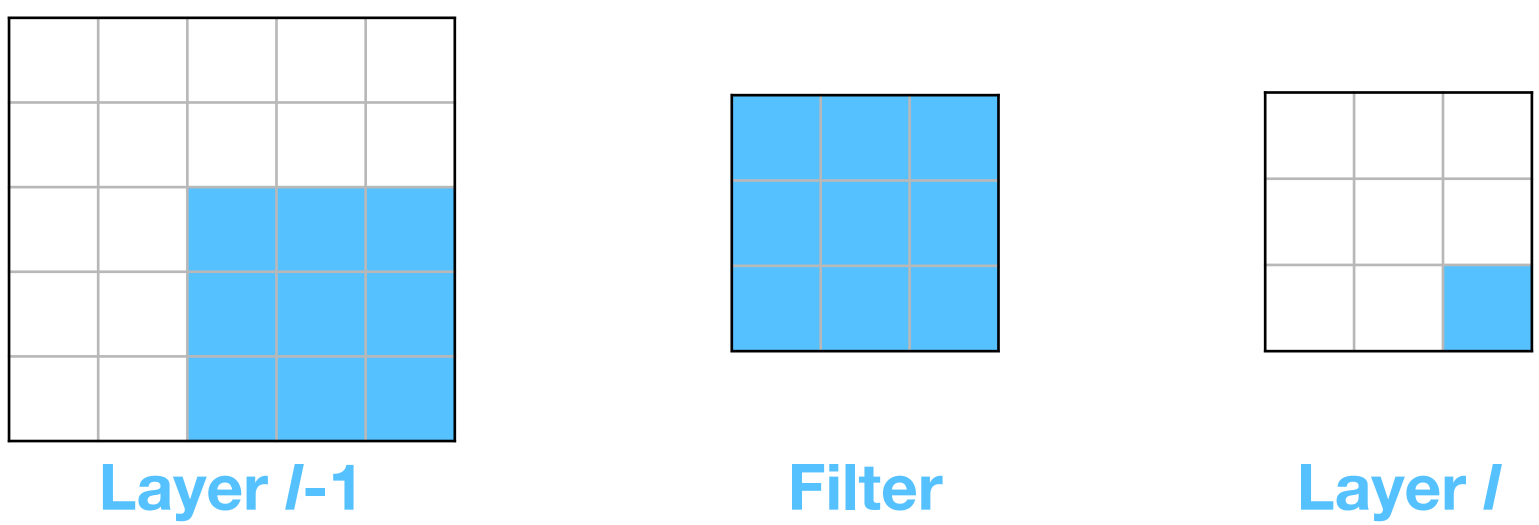
Receptive fields

- Each neuron *i* in a convolutional layer *l* depends only on neurons in a local region around *i* in the previous layer *l*-1.



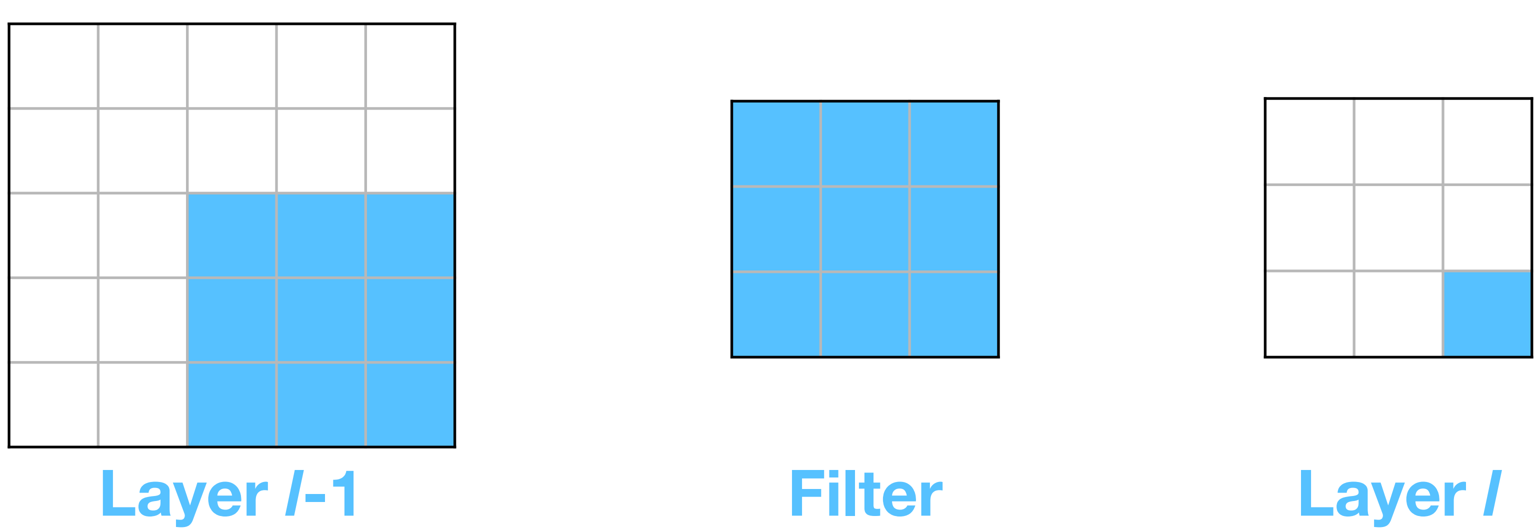
Receptive fields

- Each neuron *i* in a convolutional layer *l* depends only on neurons in a local region around *i* in the previous layer *l*-1.



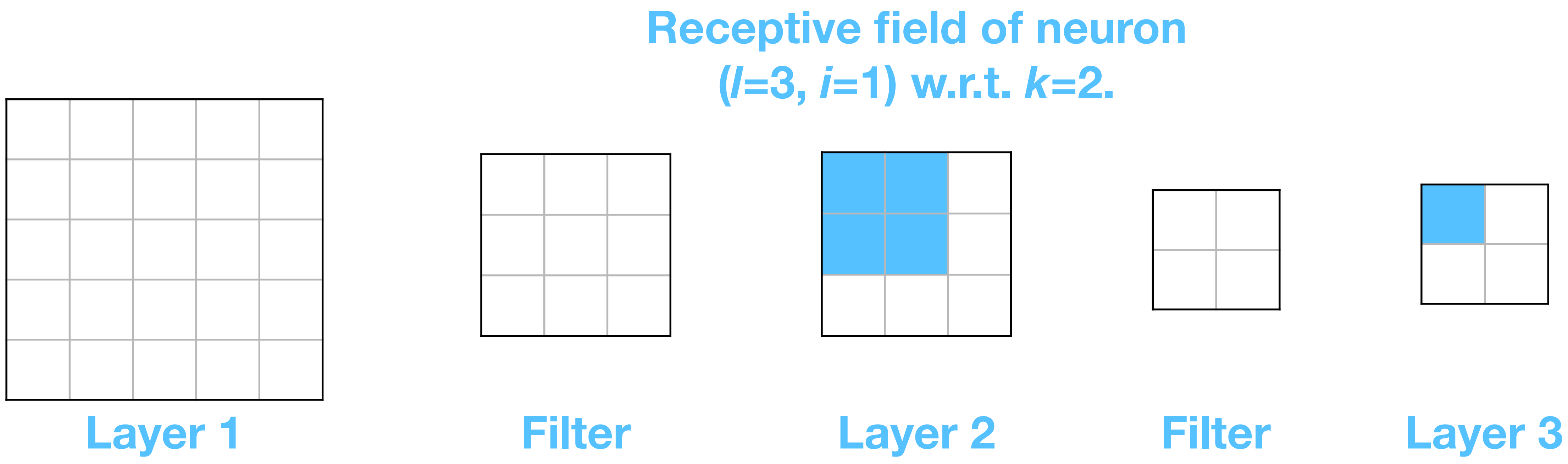
Receptive fields

- In general, we define the **receptive field** of neuron *i* in layer *l* w.r.t. layer *k* as the set of neurons in *k* that influence *i* in *l*.



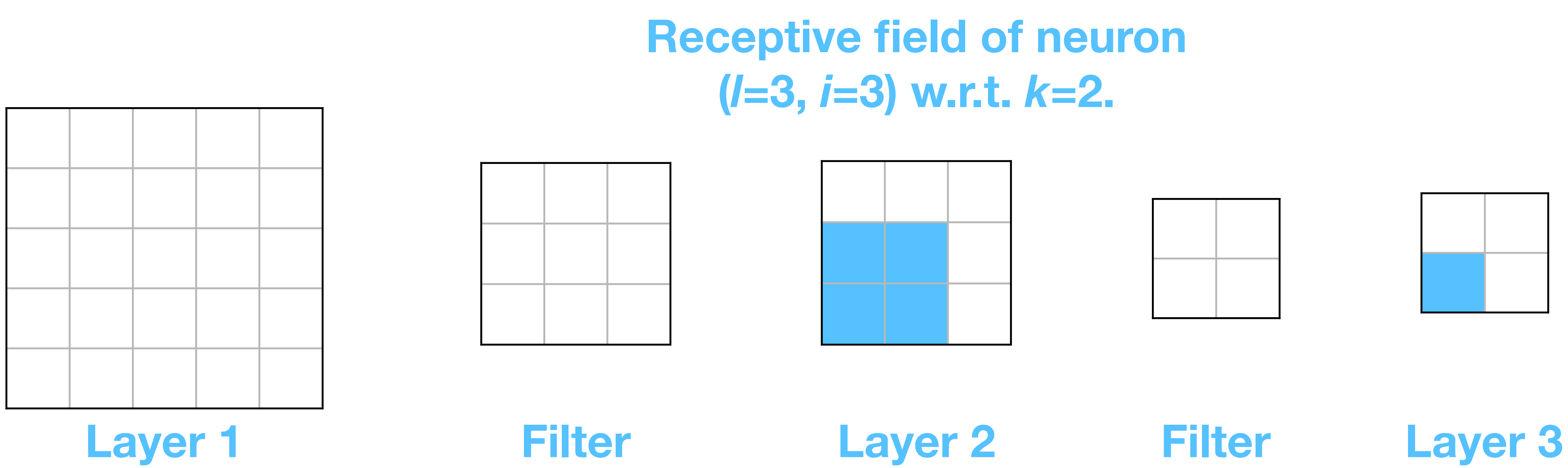
Receptive fields

- In general, we define the **receptive field** of neuron *i* in layer *l* w.r.t. layer *k* as the set of neurons in *k* that influence *i* in *l*.



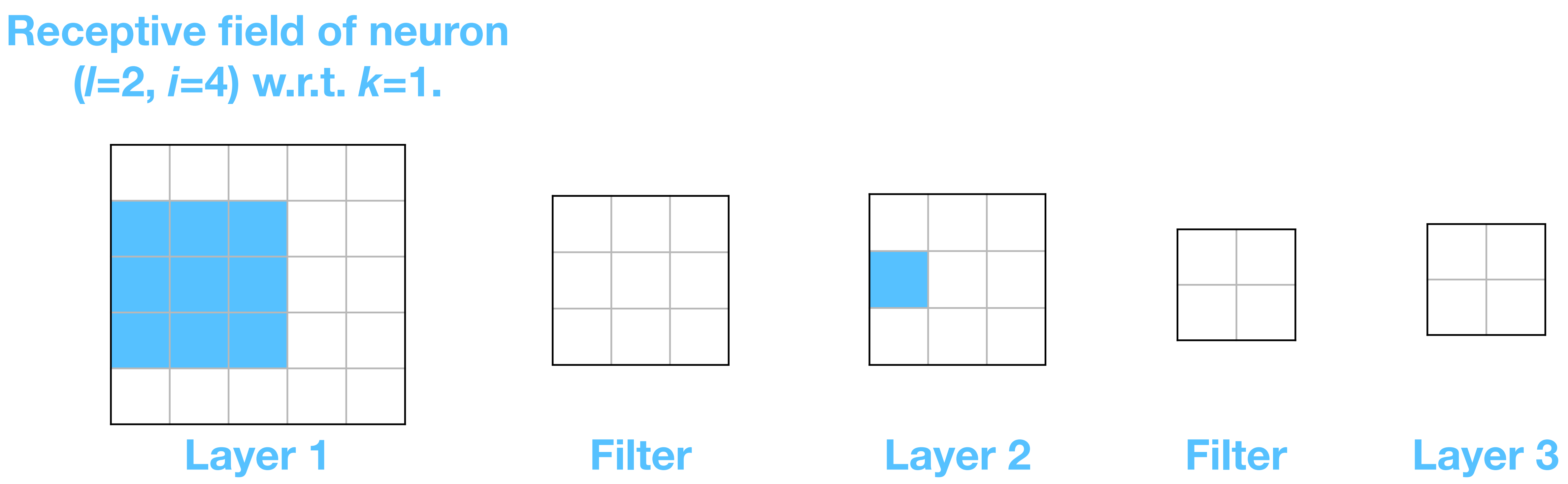
Receptive fields

- In general, we define the **receptive field** of neuron *i* in layer *l* w.r.t. layer *k* as the set of neurons in *k* that influence *i* in *l*.



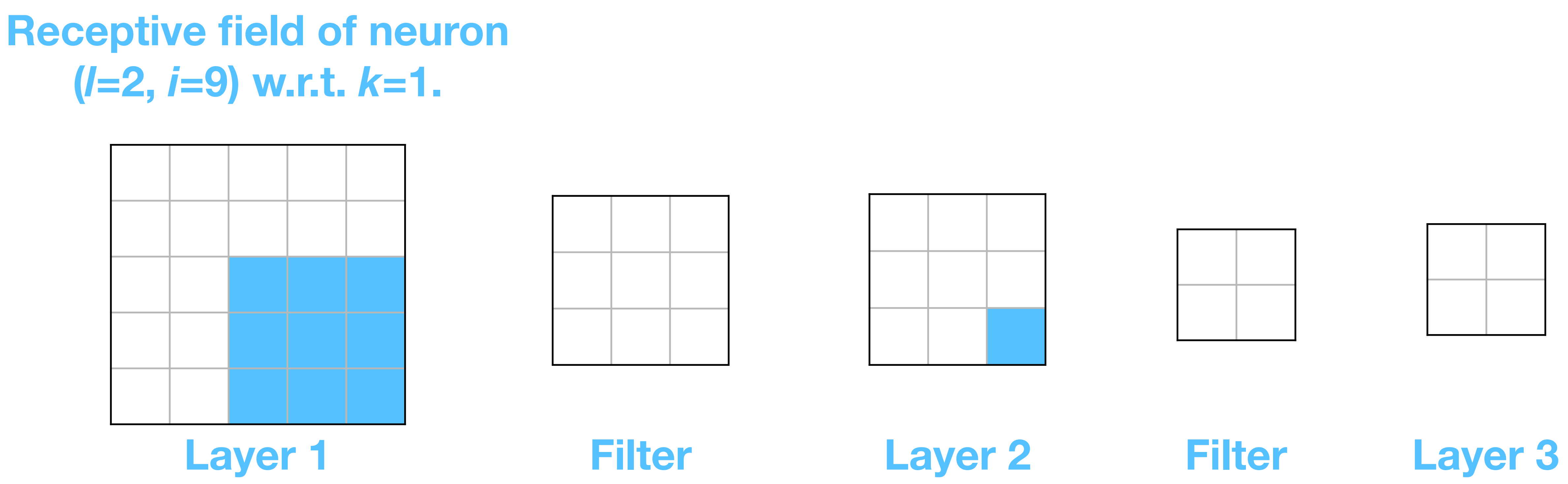
Receptive fields

- In general, we define the **receptive field** of neuron *i* in layer *l* w.r.t. layer *k* as the set of neurons in *k* that influence *i* in *l*.



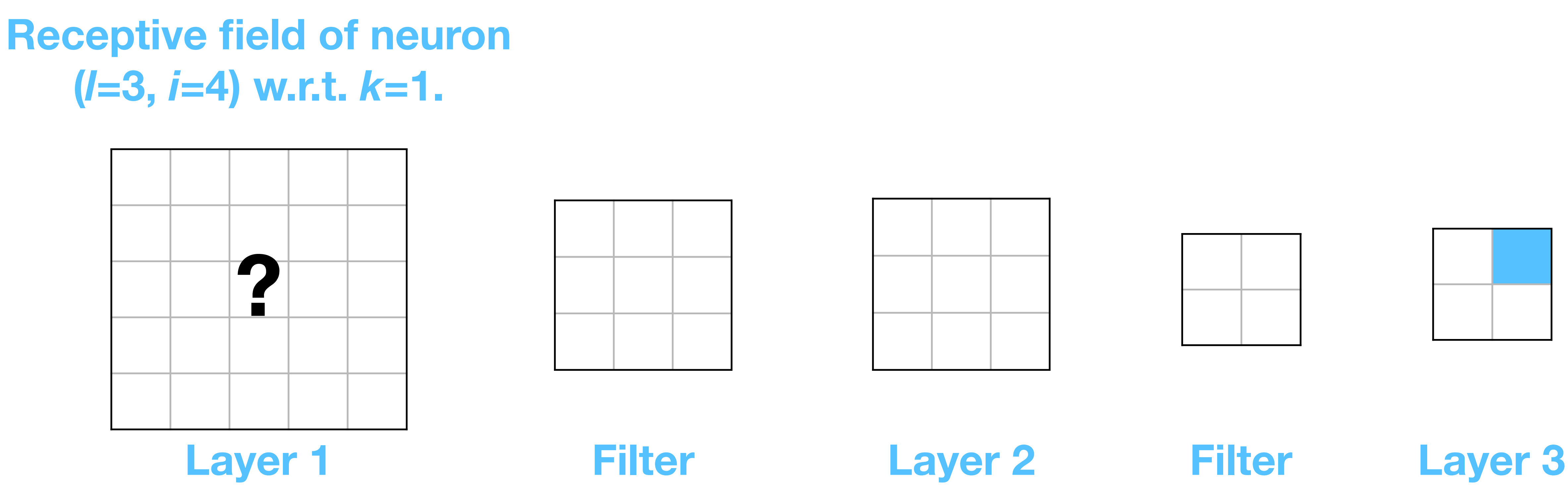
Receptive fields

- In general, we define the **receptive field** of neuron i in layer l w.r.t. layer k as the set of neurons in k that influence i in l.



Exercise

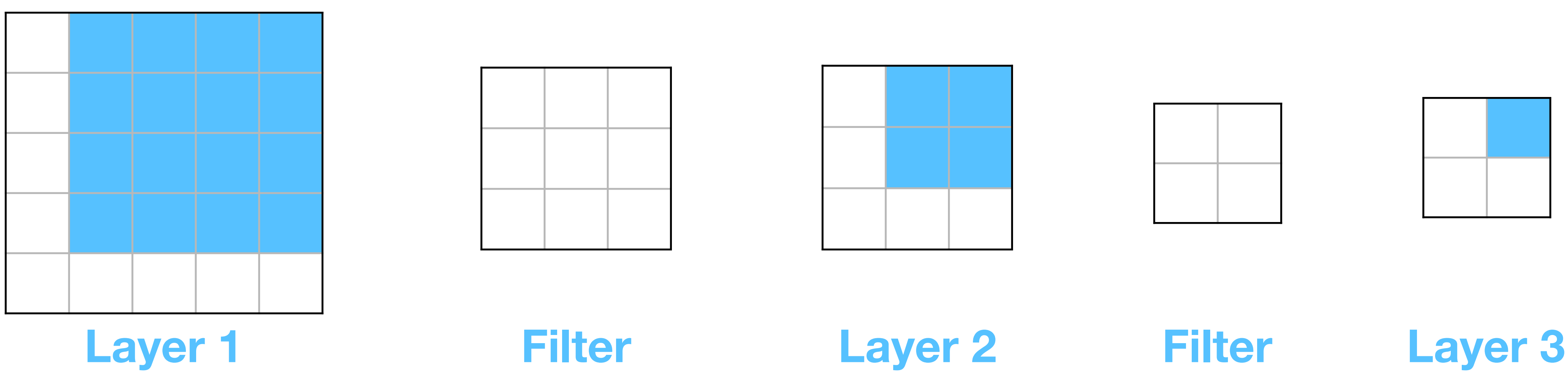
- What is the receptive field of neuron (l=3, i=2) w.r.t. k=1?



Solution

- What is the receptive field of neuron (*l*=3, *i*=2) w.r.t. *k*=1?

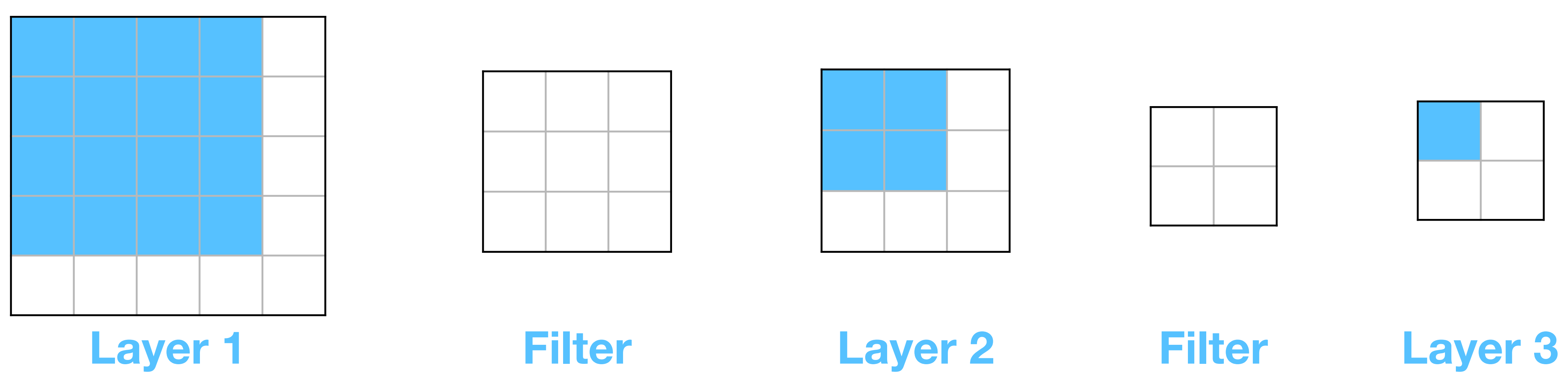
Receptive field of neuron (*l*=3, *i*=4) w.r.t. *k*=1.



Receptive fields

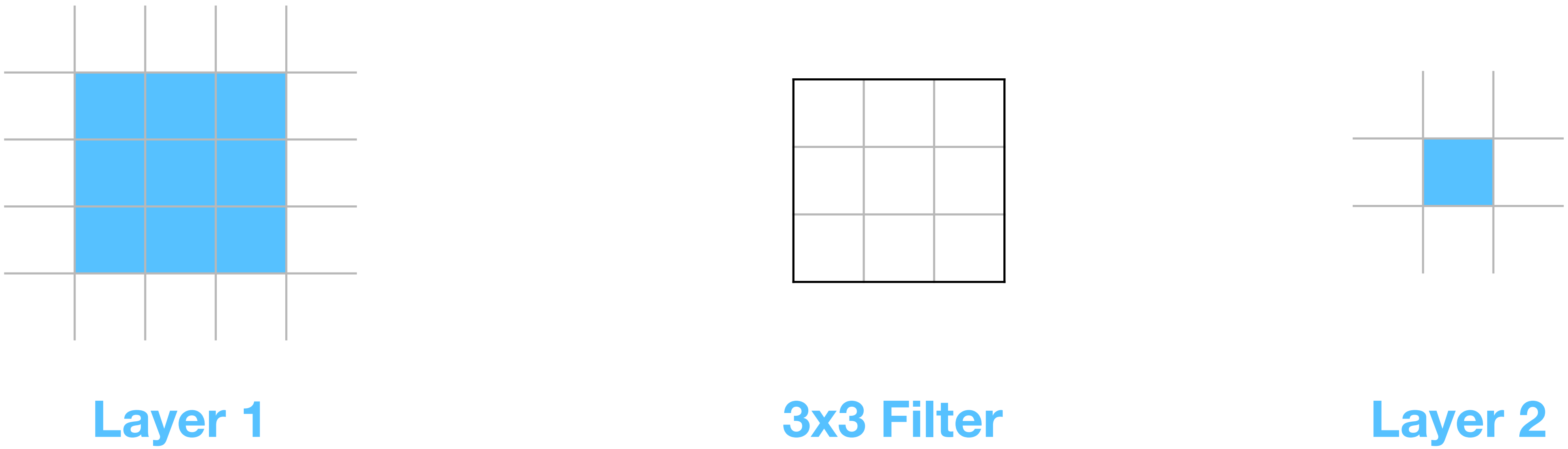
- Note that, if k is not mentioned explicitly, then typically assume k=1, i.e., the receptive field is the set of *input* neurons that affect a neuron i in layer l.

Receptive field of neuron (l=3, i=1).



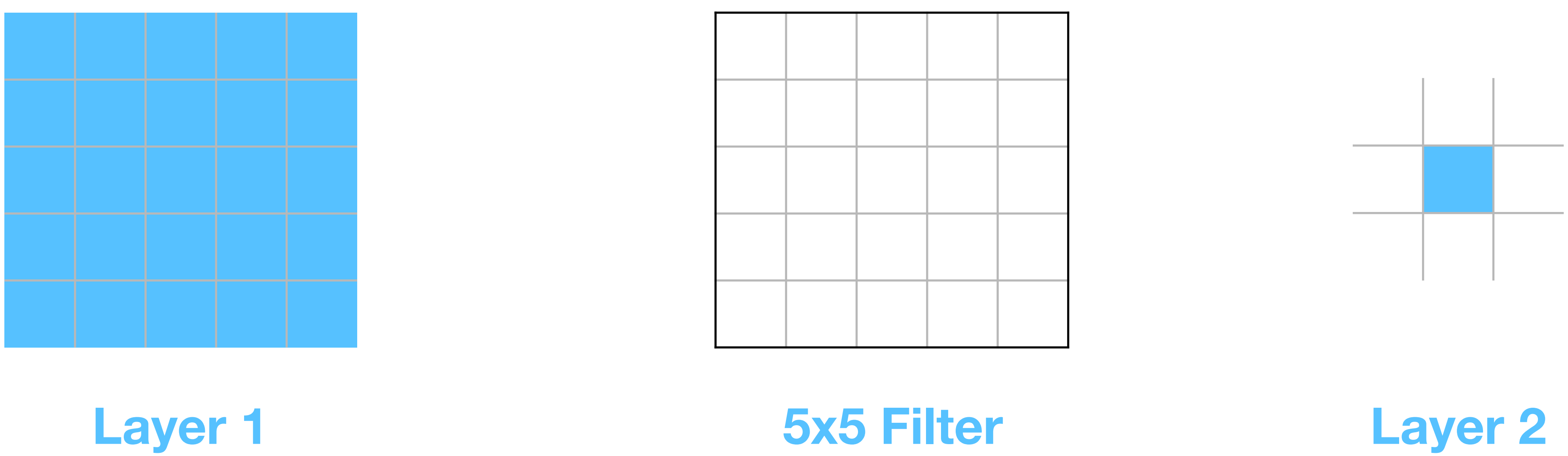
Receptive fields

- By using larger filter sizes, we can expand the receptive field, e.g.:



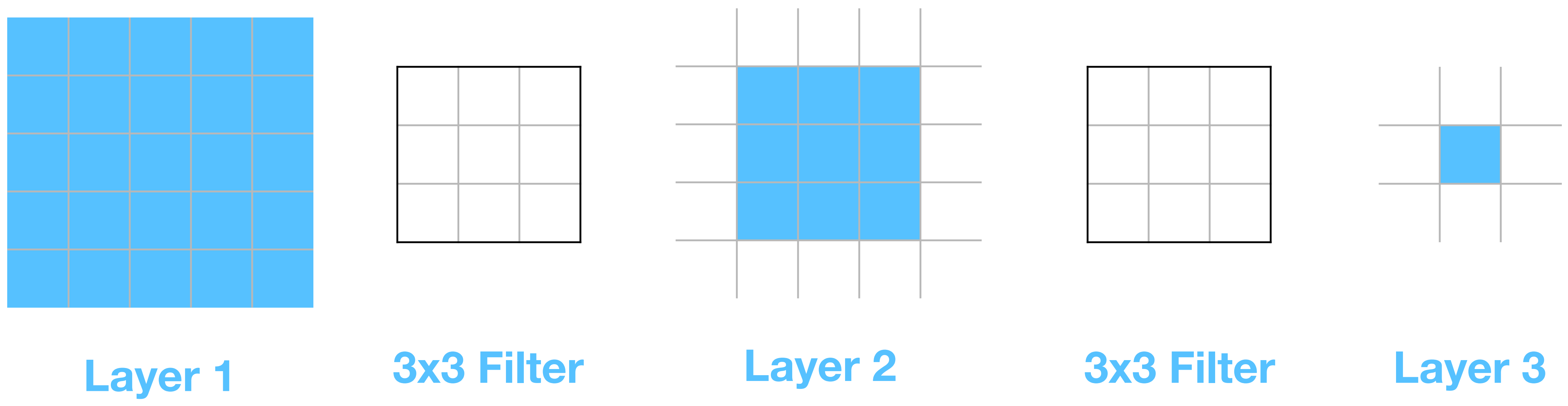
Receptive fields

- By using larger filter sizes, we can expand the receptive field, e.g.:



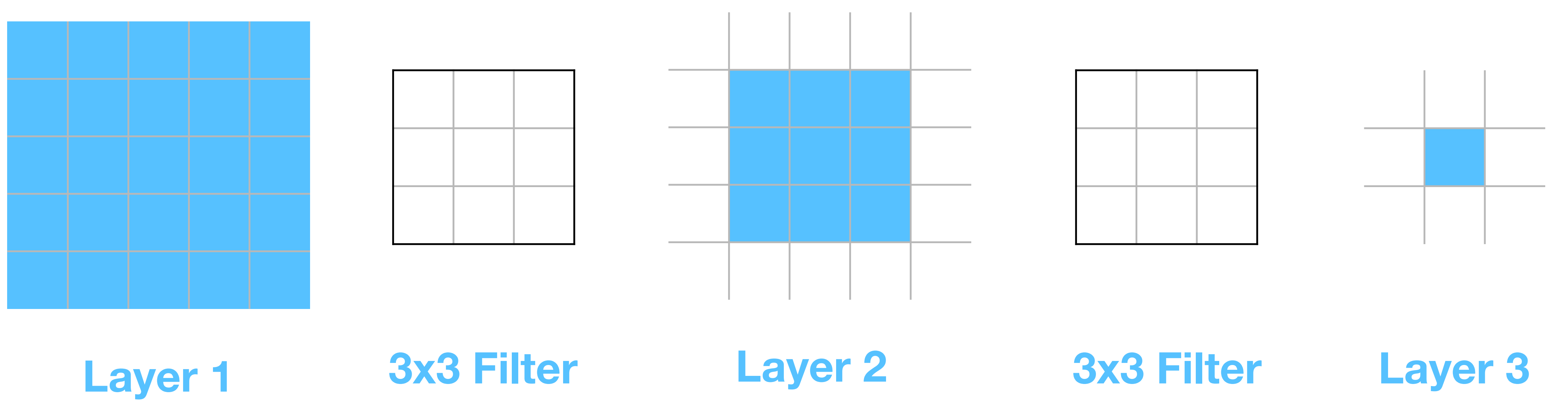
Composition of convolution

- Alternatively, we can **compose** multiple convolutional layers with small filters to achieve a similar effect:



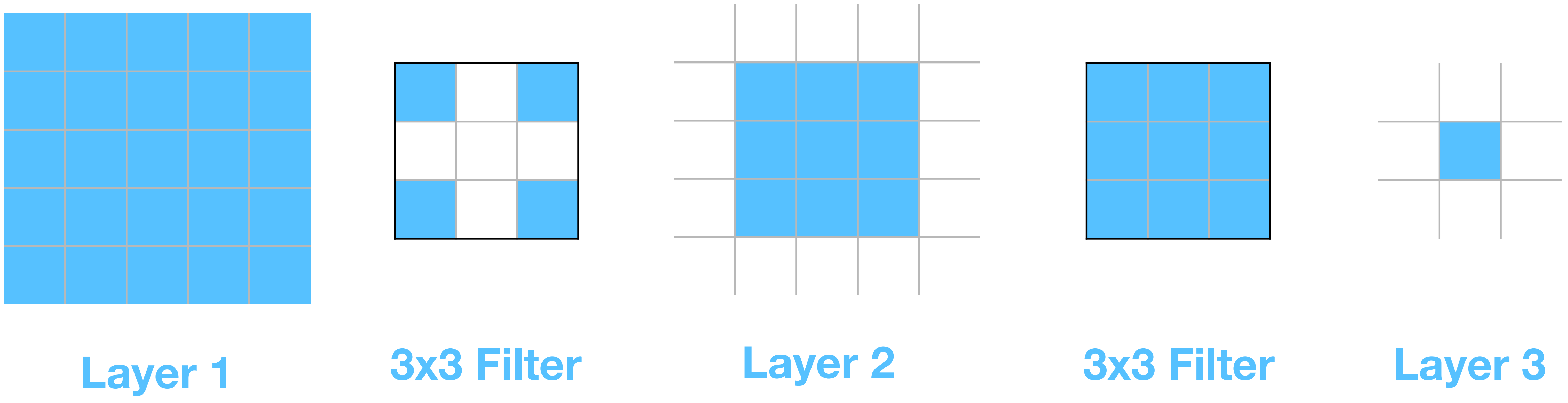
Composition of convolution

- Instead of 5*5=25 parameters, we have just 2*3*3=18 parameters.
- Through a non-linear activation function in layer 2, we can also learn more complicated functions.



Dilated convolution

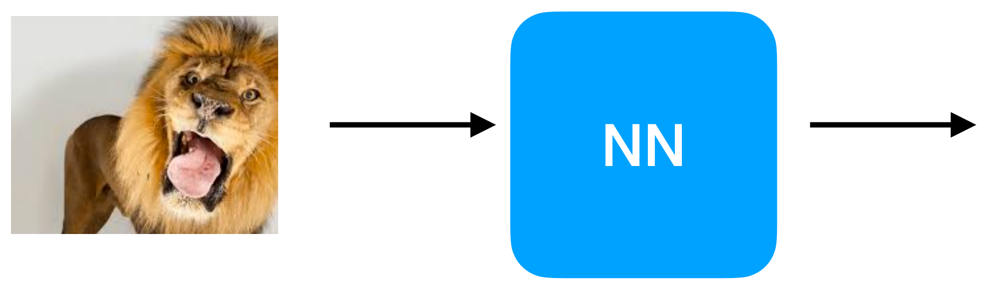
- Dilated convolution (aka **à trous**, “with holes”) can also expand the receptive field without increasing the number of parameters.
- Here we have 3*3 + 2*2 = 13 total parameters.



Translation invariance

Translation invariance

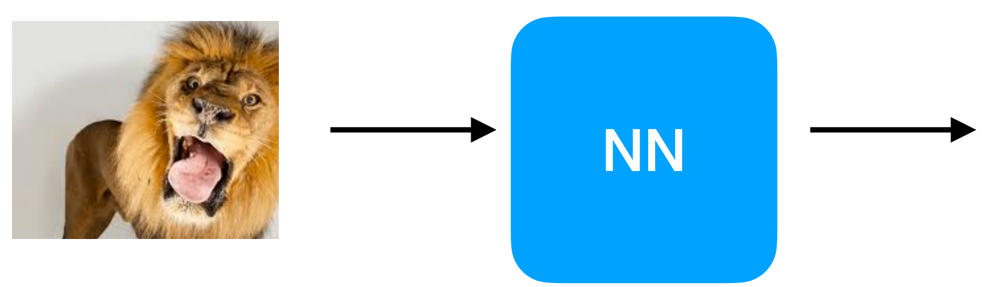
- Suppose we are training a NN to classify the contents of an image:



…
***P*(lion | image)=0.9**
P(person | image)=0.02
P(dog | image)=0.01
…

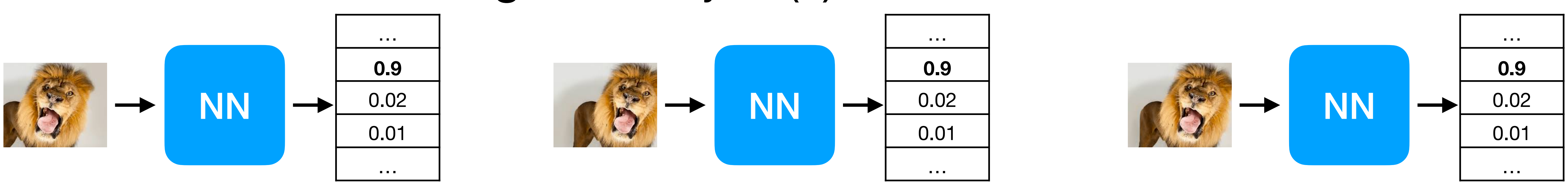
Translation invariance

- Suppose we are training a NN to classify the contents of an image:



…
***P*(lion | image)=0.9**
P(person | image)=0.02
P(dog | image)=0.01
…

- We often want the NN to produce the same output no matter where in the image the object(s) is/are located:



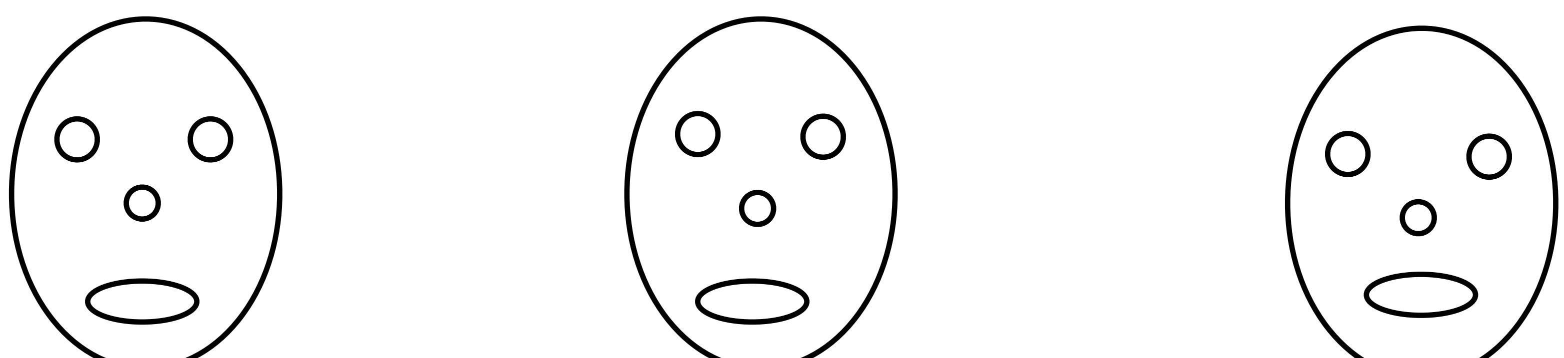
- This is called **translation invariance**.

Translation invariance

- When recognizing an object as a constellation of multiple parts, *local* translation invariance can help to smooth over tiny differences in the input.

- Consider how a NN might recognize a face as a constellation of two eyes, a nose, and a mouth.

- We might want the network output to be invariant to *small changes* in the locations of the parts:

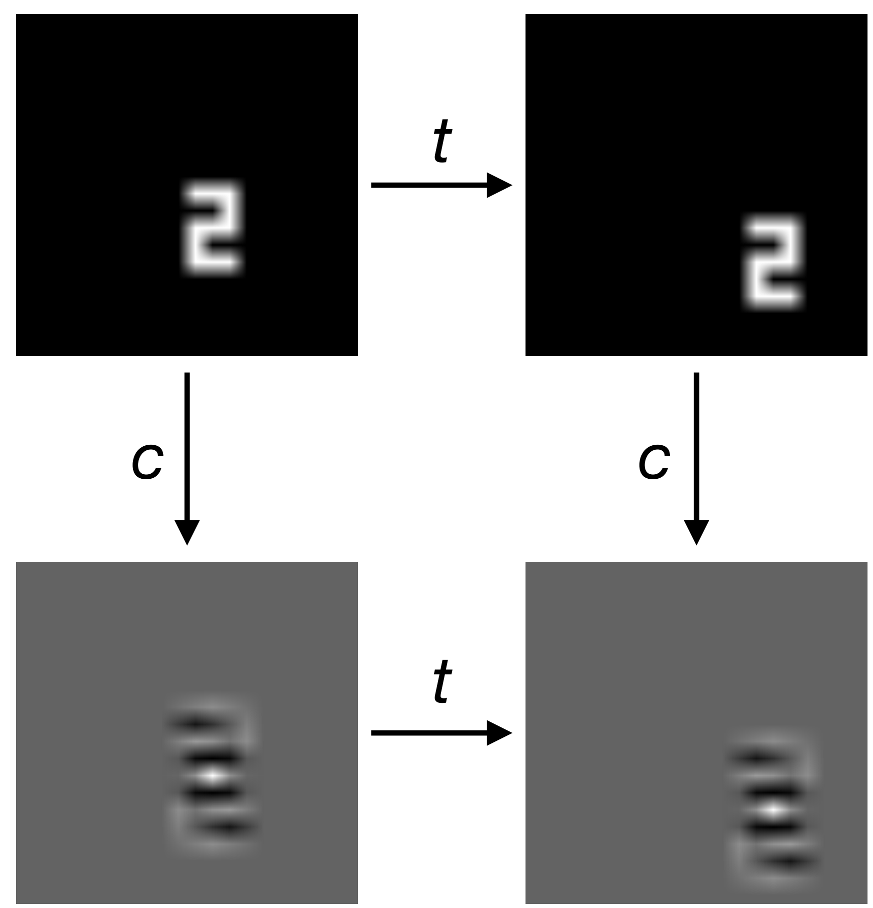


Translation invariance

- Convolution is a useful function that, when combined with an aggregation mechanism (e.g., max-pooling), can help provide translation invariance.

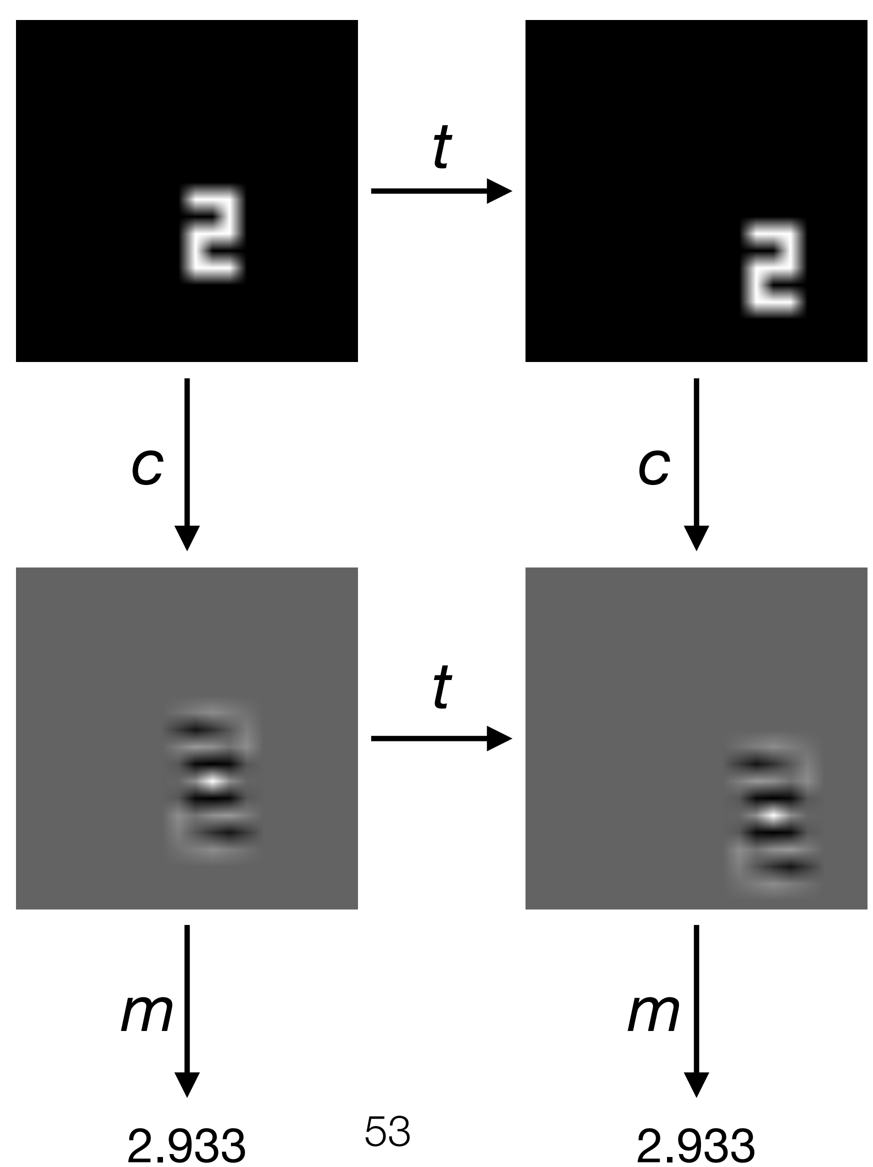
Equivariance

- Note that convolution is actually **equivariant**, i.e.:
  - The convolution (*c*) of the translation (*t*)
    =
    The translation (*t*) of the convolution (*c*)



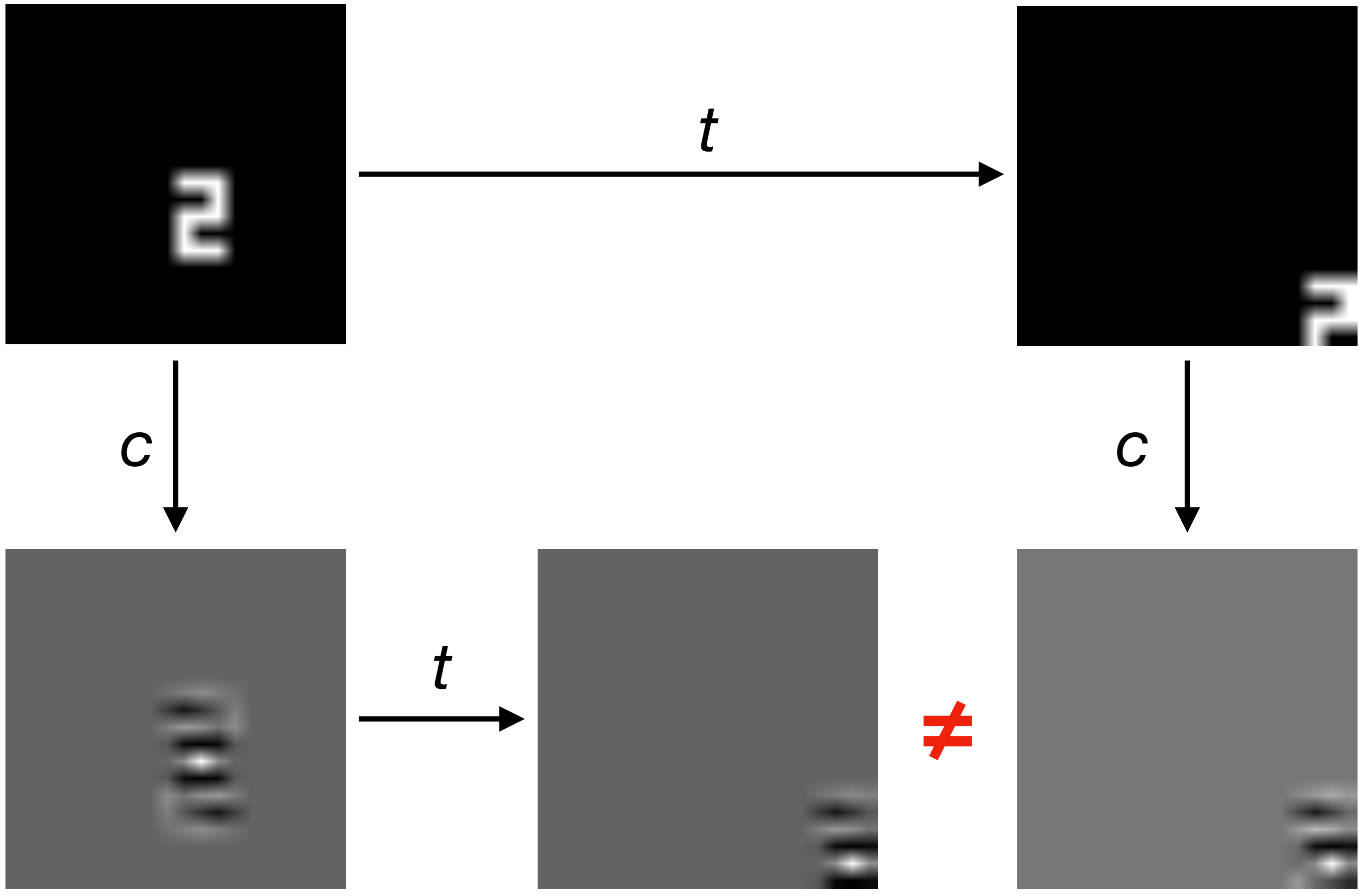
Equivariance

- To obtain translation **invariance**, we can take the maximum (or minimum) (*m*) over the filter response:



Equivariance

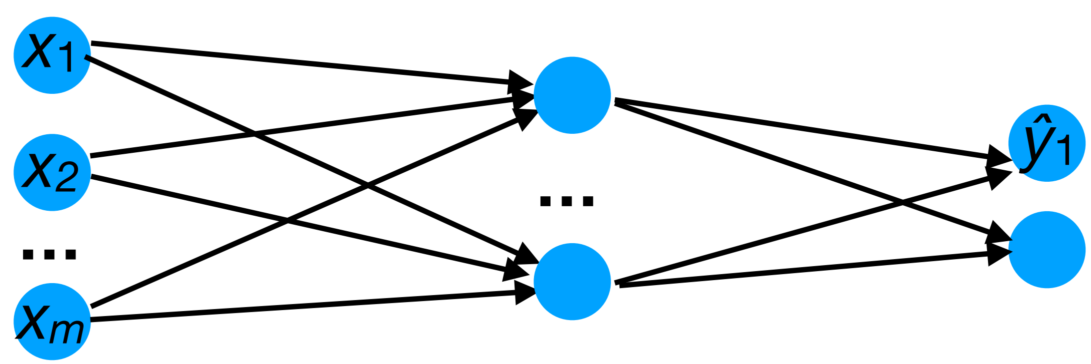
- Also, note that, due to edge effects, finite convolution is not perfectly equivariant, e.g.:



Recurrent neural networks (RNNs)

Fixed length versus variable length

- Up till now, all the ML models we have considered have processed inputs **x** of some **fixed length** *m* to predict some target value *y*.



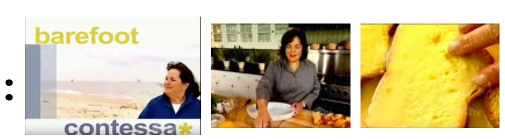
- But what if *m* varies from example to example?

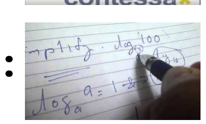
Jacob Whitehill, WPI

Fixed length versus variable length

- Examples of **variable-length** inputs:
  - Text passages: number of words.
  - Videos: number of frames.
  - Time series: number of events.

```
1: The dog is running.
2: The cat is eating the dog.
3: I like frogs.
```

1: 

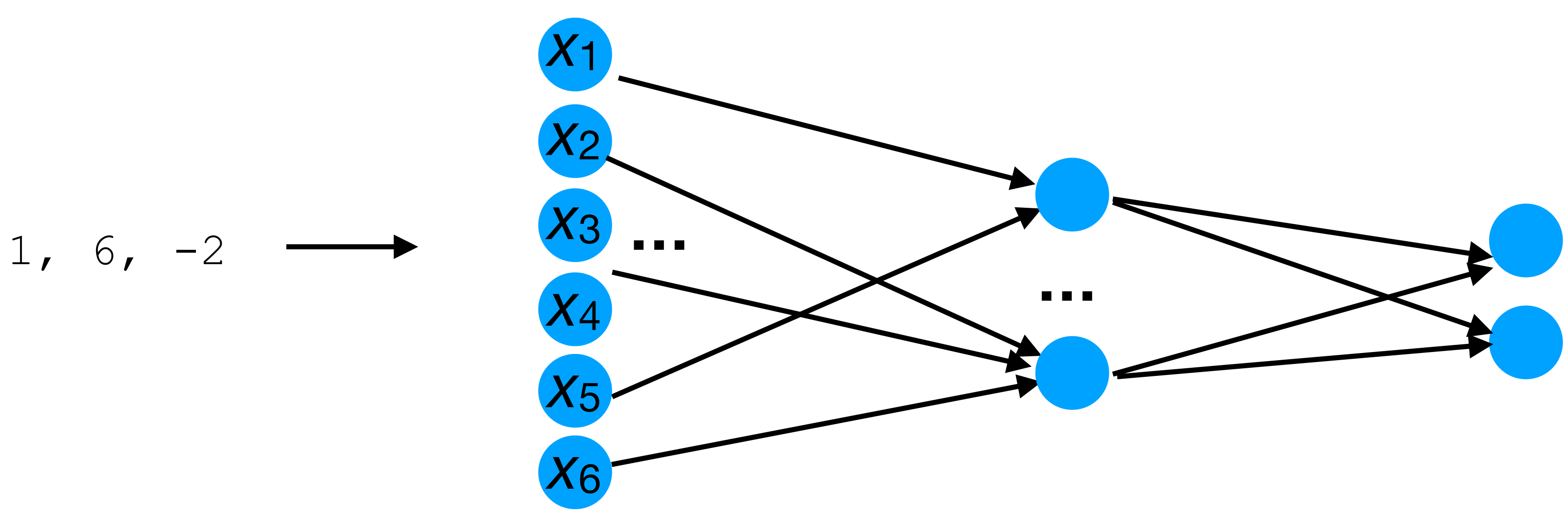
2: 

```
1: -5, 5, 7, -2, 0, 1
2: 5, 14, 23, 96
3: 1, 6, -2
```

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

  1. Decide what the maximum length should be, and "pad" the input to always reach that length.



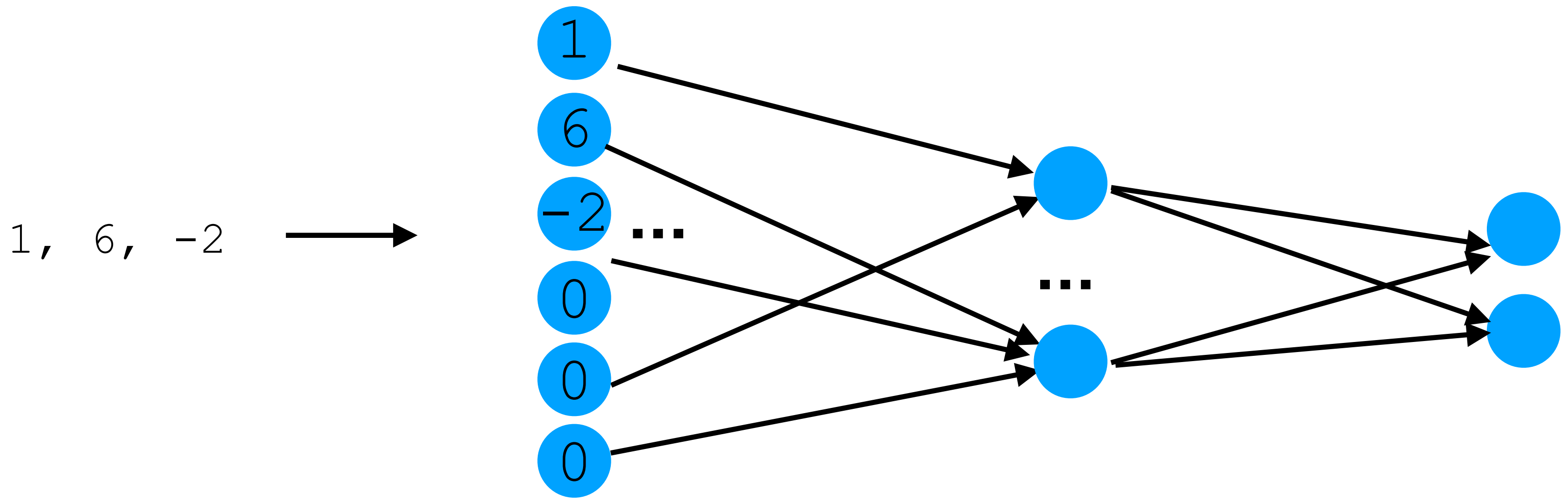
E.g., if max-length M=6, but actual length of some example m=3, then pad with 3 zeros.

Jacob Whitehill, WPI

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

    1. Decide what the maximum length should be, and "pad" the input to always reach that length.



E.g., if max-length M=6, but actual length of some example m=3, then pad with 3 zeros.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

  1. Decide what the maximum length should be, and "pad" the input to always reach that length.

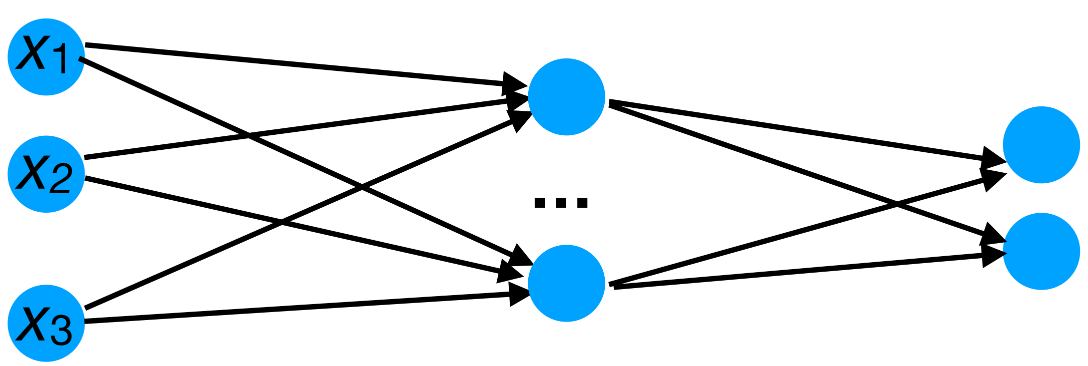
     - Advantages:
       - Simple to implement.
     - Disadvantages:
       - Wasteful of memory
       - Hard to learn weights from input units close to M.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

  2. Decide what the minimum length should be, and discard any features beyond this length.

```
-5, 5, 7, -2, 0, 1
```



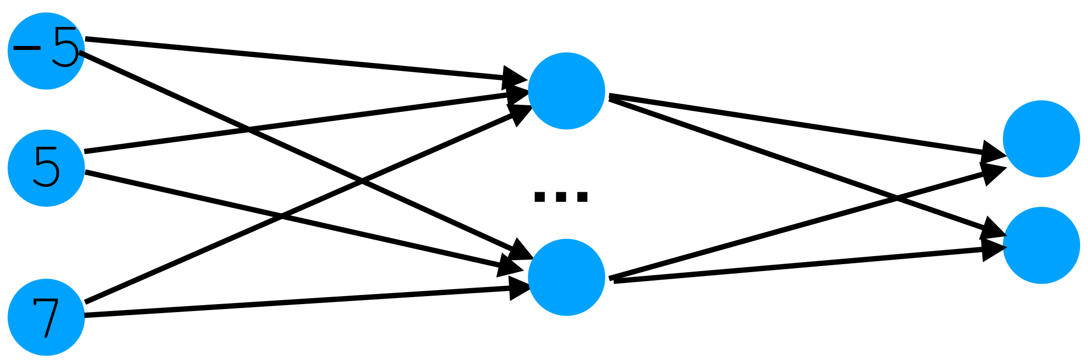
E.g., if min-length M=3, but actual length of some example m=6, then ignore the last 3 features.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

  2. Decide what the minimum length should be, and discard any features beyond this length.

`-5, 5, 7, -2, 0, 1`



E.g., if min-length *M*=3, but actual length of some example *m*=6, then ignore the last 3 features.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

  2. Decide what the minimum length should be, and discard any features beyond this length.

     - Advantages:
       - Simple to implement.

     - Disadvantages:
       - Throws away potentially valuable information.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

  3. Extract summary features from the sequence that do not depend on the input length.

E.g., extract aggregate color information, number of detected objects of fixed set of classes, etc.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

3. Extract summary features from the sequence that do not depend on the input length.

Judge the type of video based on the fixed-length feature representation.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

  3. Extract summary features from the sequence that do not depend on the input length.

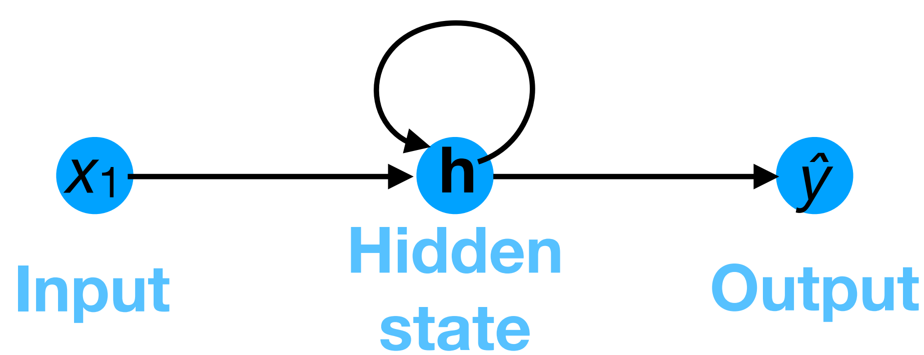
     - Advantages:
       - Simple to implement; can give high accuracy.
     - Disadvantages:
       - Requires manual feature engineering.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

 4. Scan the input one element at a time. *Update* and *remember* a "hidden state" *across time-steps* to the store most important information.

Recurrent neural network



Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

4. Scan the input one element at a time. *Update* and *remember* a “hidden state” *across time-steps* to the store most important information.

   - Advantages:
     - More powerful, often resulting in higher accuracy.
   - Disadvantages:
     - More complicated to implement & train.

Handling variable-length inputs

- Possible strategies for handling variable-length inputs:

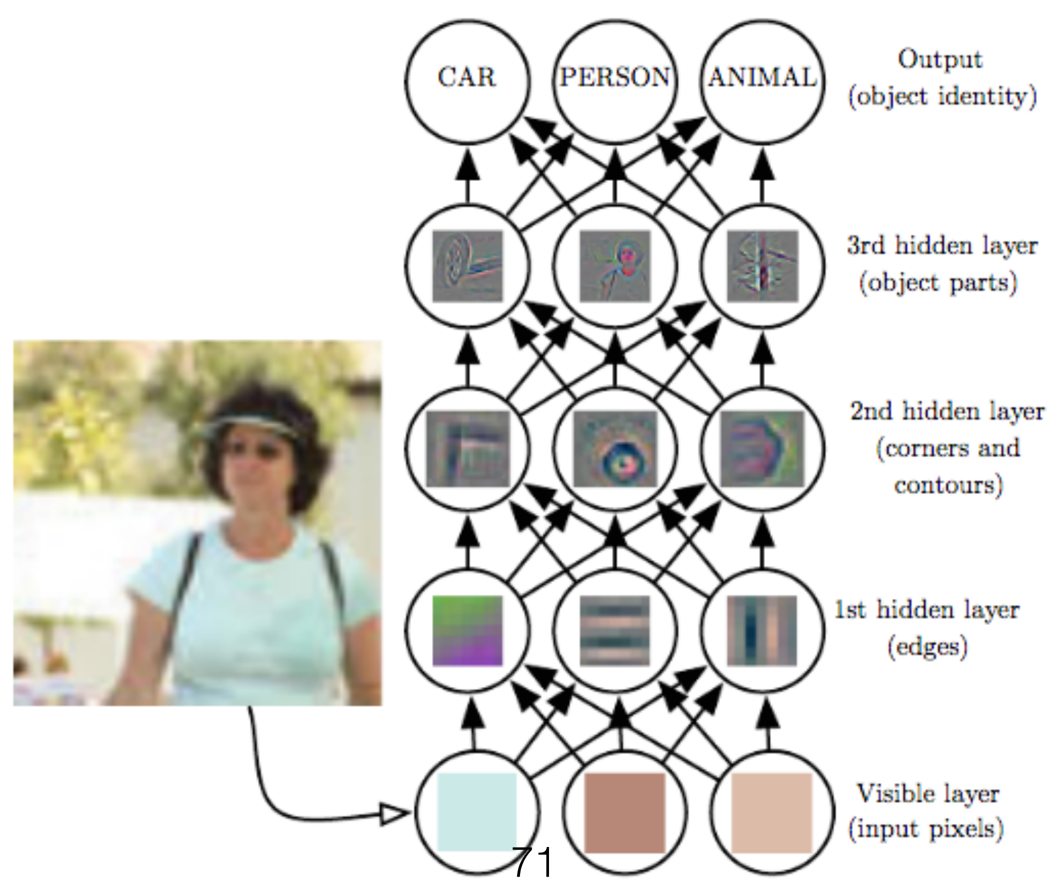
  5. Use an *attention mechanism* (more later) to determine at *test time* which elements of the input are most important.

     - Advantages:
       - More powerful, often resulting in higher accuracy. More parallelizable than step-wise approaches.
     - Disadvantages:
       - May be difficult to index the different elements of the input sequence.

Recurrent neural networks (RNNs)

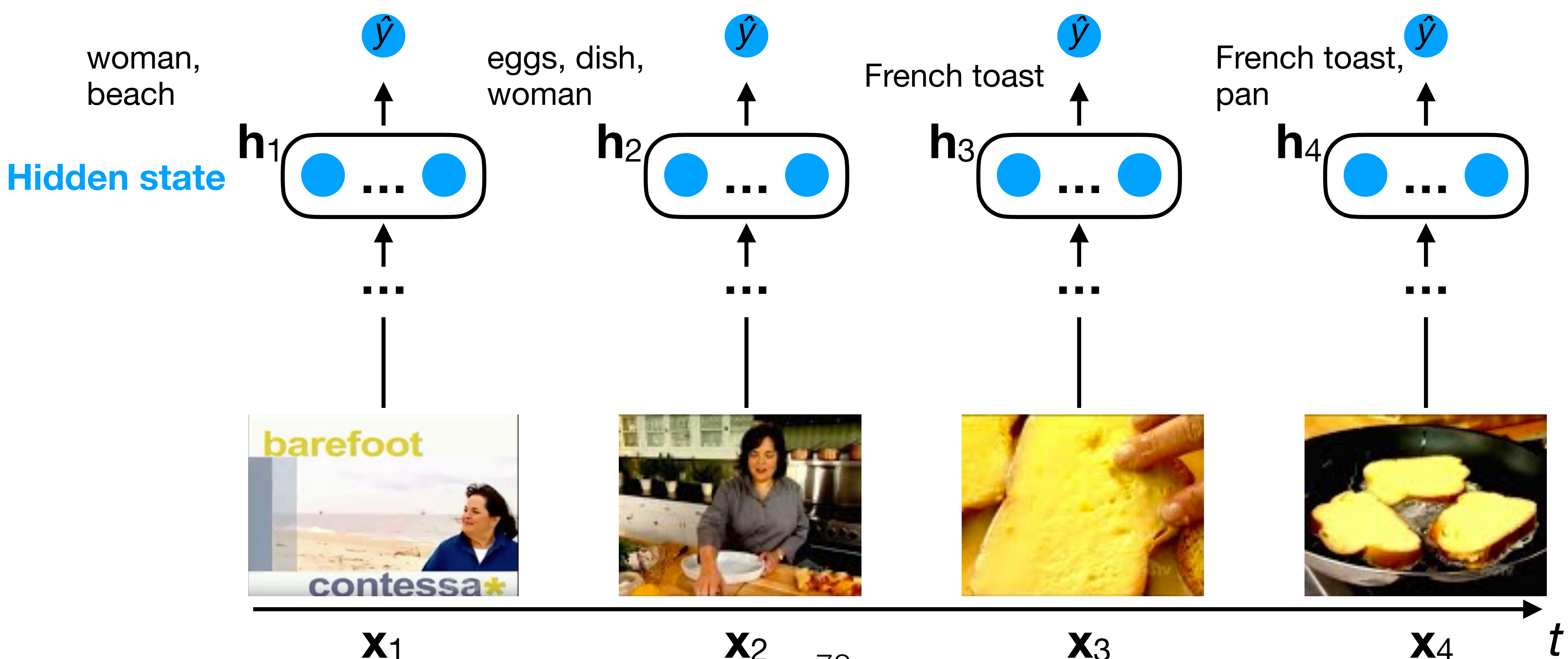
Role of the hidden state

- In feed-forward architectures (e.g., CNN), an effective network will produce hidden layer activations that capture the "essence" of the input.
- For instance, in the example below, we hope that the neurons in the top-most (i.e., last) hidden layer capture the locations and types of objects in the image:



Role of the hidden state

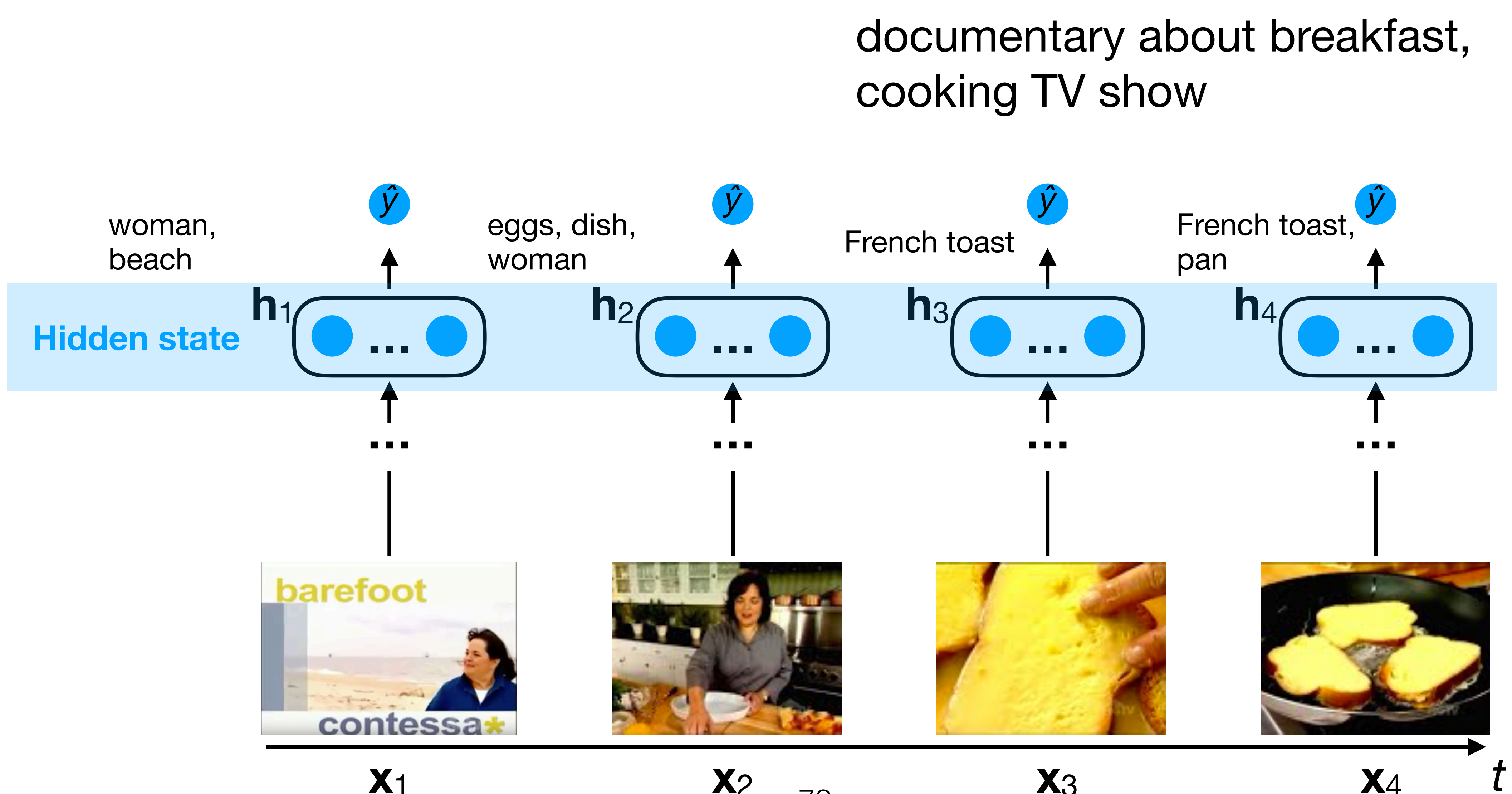
- A CNN applied to each image in the sequence can tell us about the *individual objects* contained in *each frame*.
- Here, the hidden state $\mathbf{h}_t$ of each input $\mathbf{x}_t$ is computed **independently**, i.e.: $\mathbf{h}_t = f(\mathbf{x}_t)$



Jacob Whitehill, WPI

Role of the hidden state

- However, sometimes we need to **aggregate over time** to understand how the individual objects relate to each other:

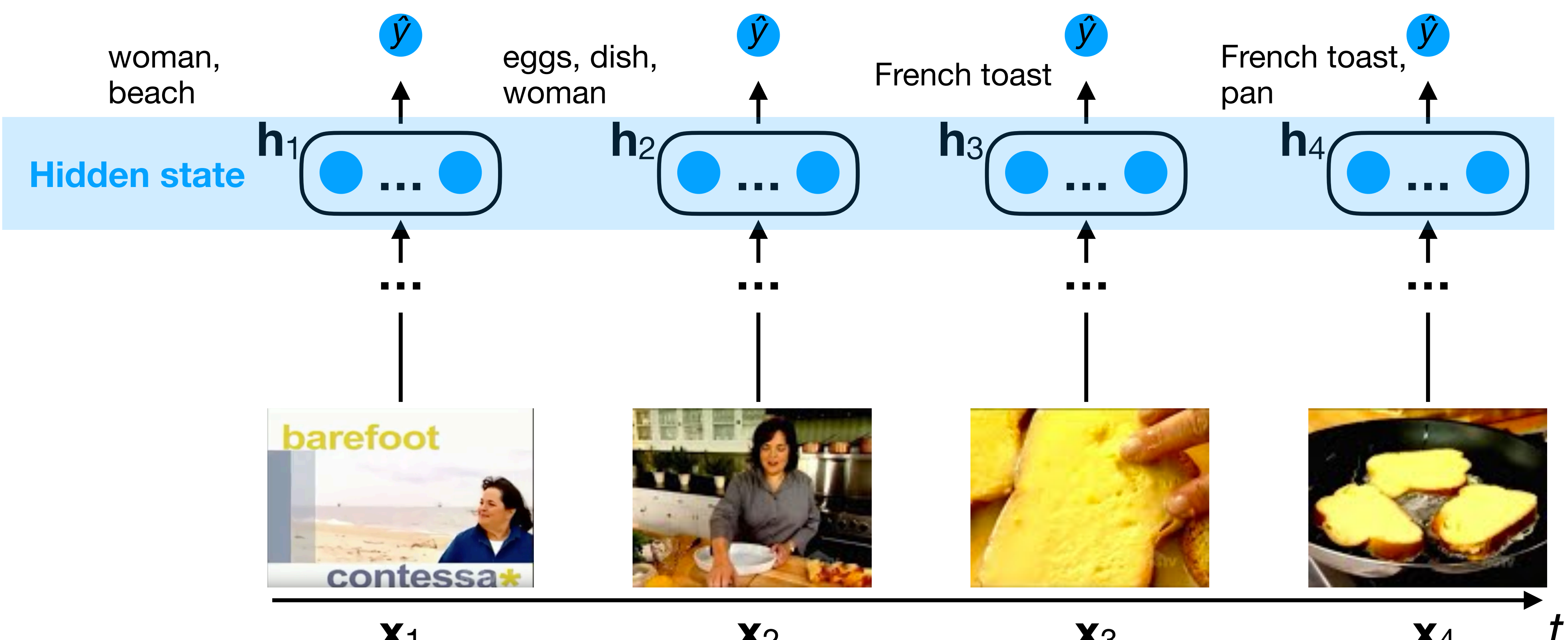


Jacob Whitehill, WPI

Role of the hidden state

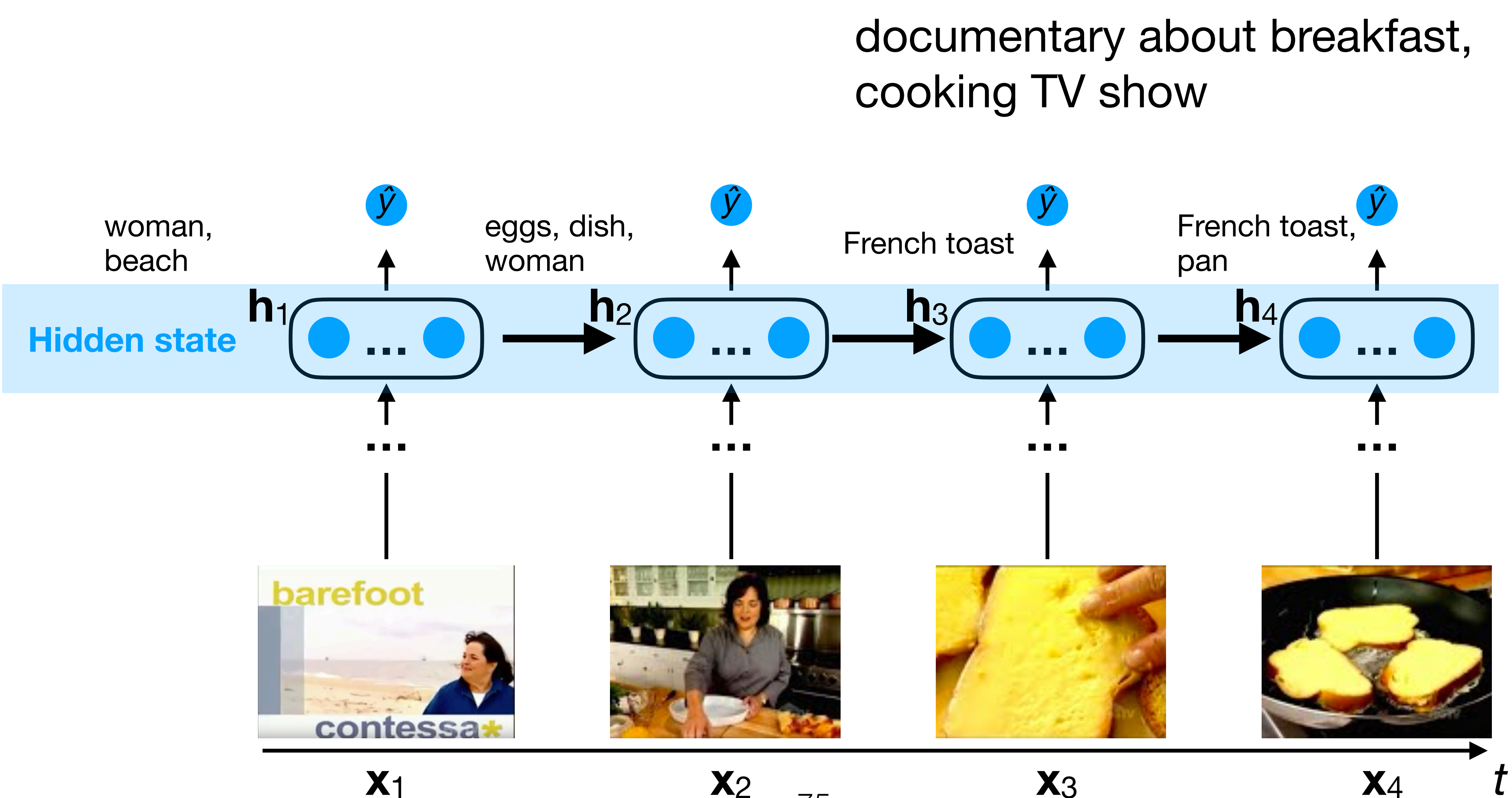
- It's also possible that past hidden state helps us to infer the current hidden state (e.g., the French toast looks blurry at time t=4 but was very clear at time t=3).

documentary about breakfast,
cooking TV show



Role of the hidden state

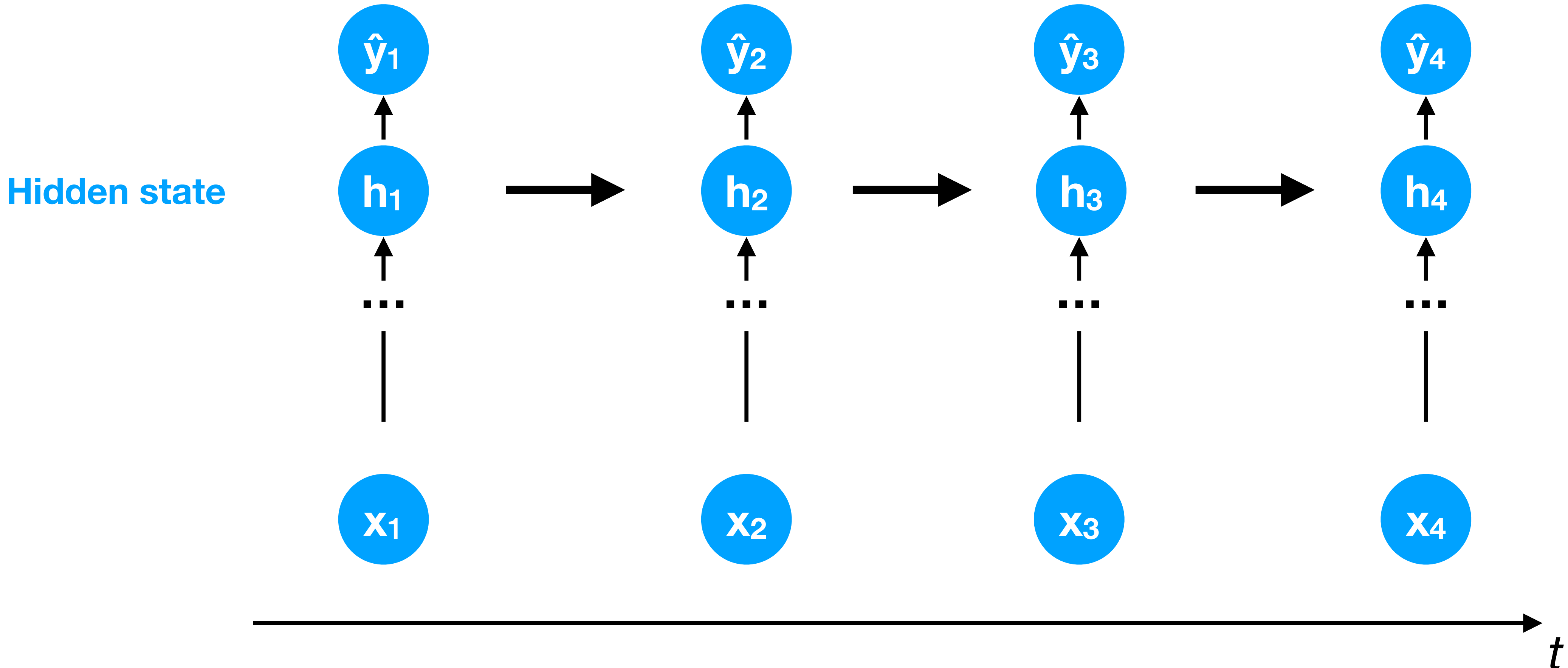
- To accomplish this, we can link the hidden state across the elements of the sequence, i.e.: $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$



Jacob Whitehill, WPI

Recurrent neural network

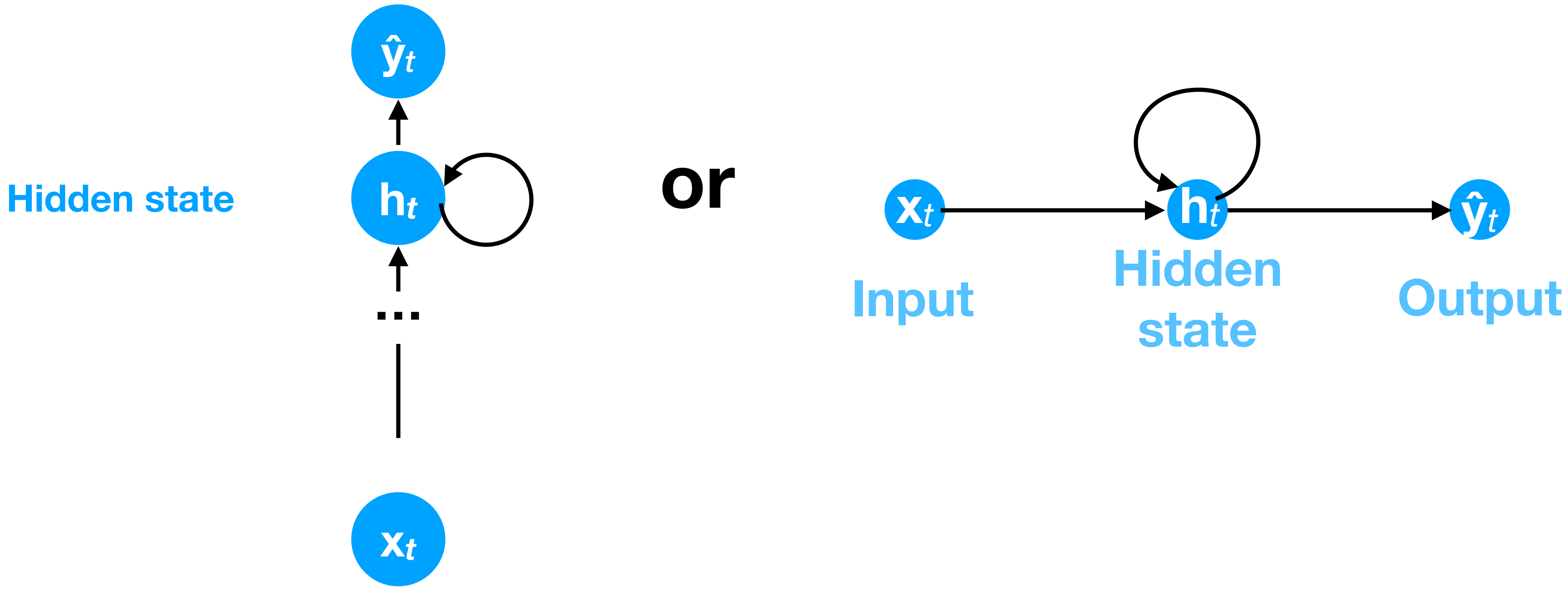
- This is the essence of a recurrent neural network (RNN) — the hidden states at time *t* depend on the hidden states of time *t*-1: $$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$



Jacob Whitehill, WPI

Recurrent neural network

- We sometimes represent the recurrent hidden state as a single node with an arrow to itself:



Jacob Whitehill, WPI