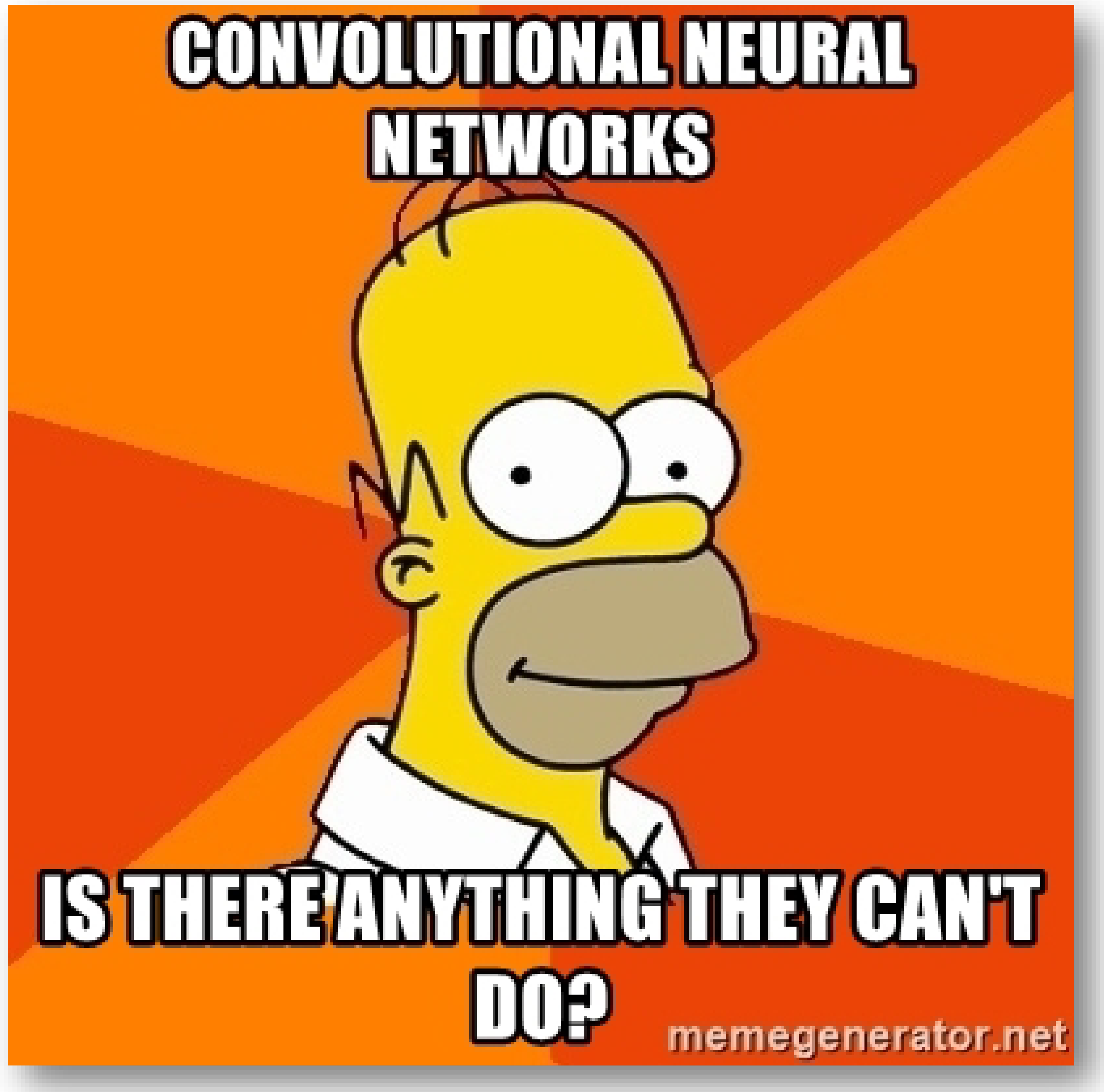
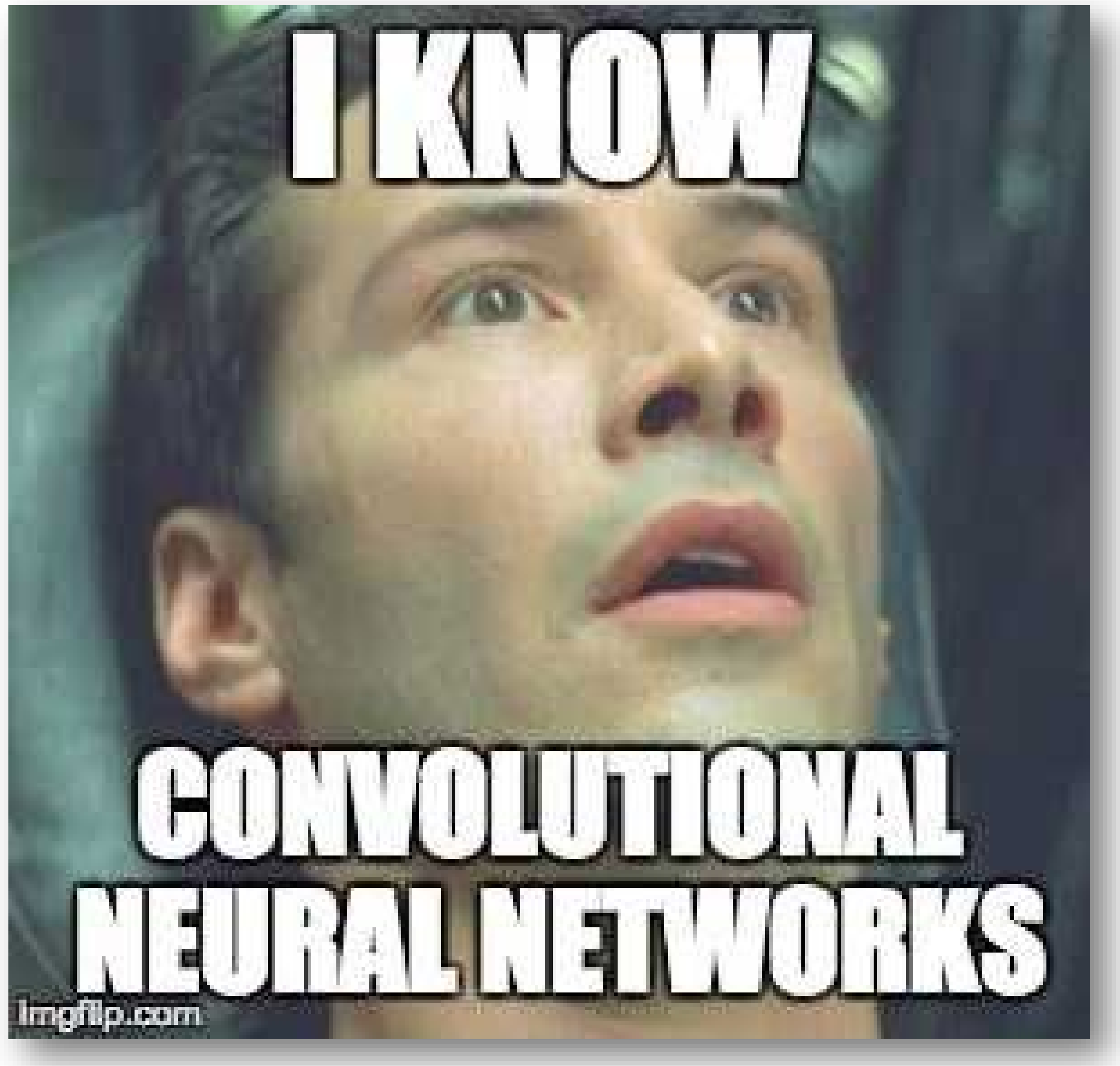
tejasgokhale.com

CMSC 475/675 Neural Networks

Lecture 4

Convolutional Neural Networks

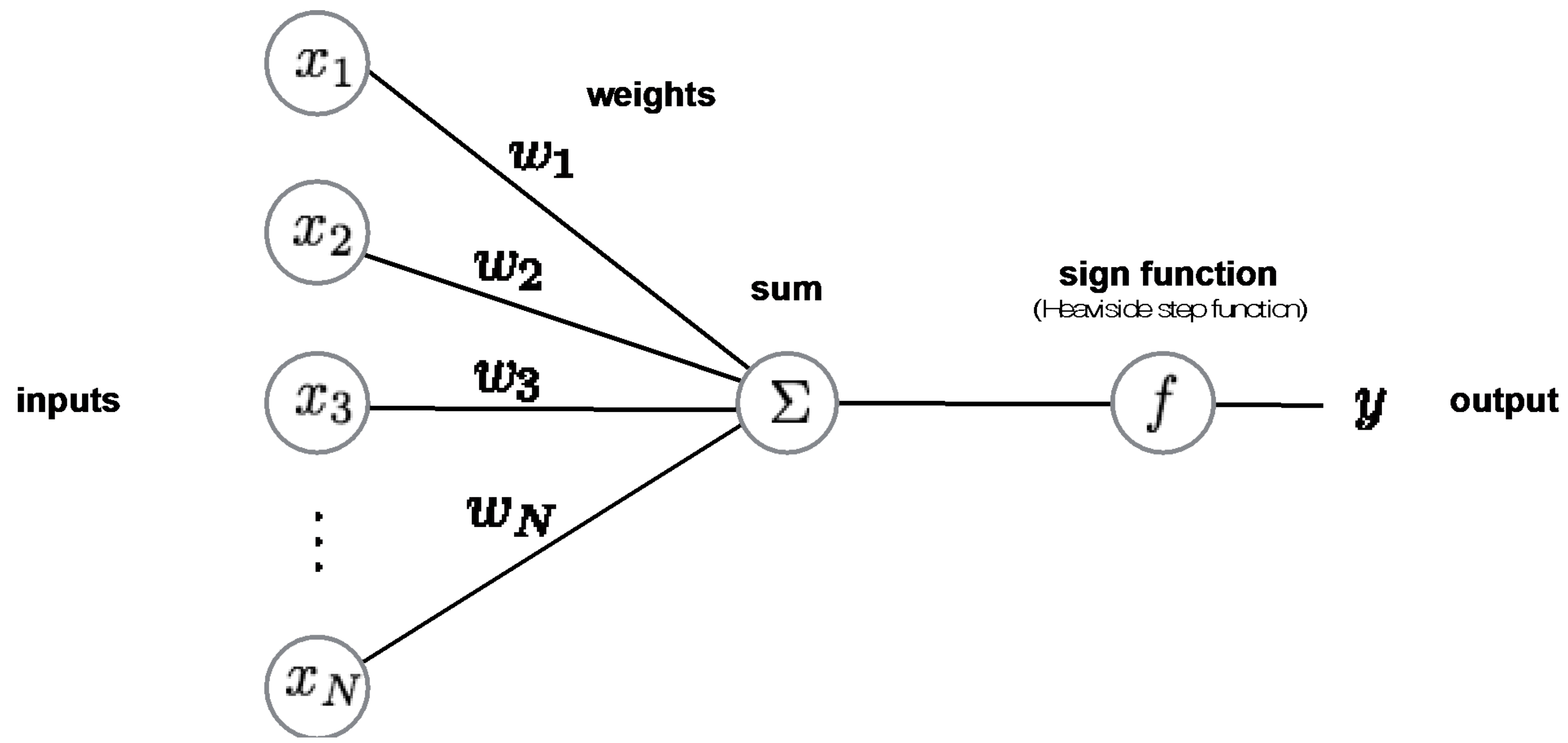


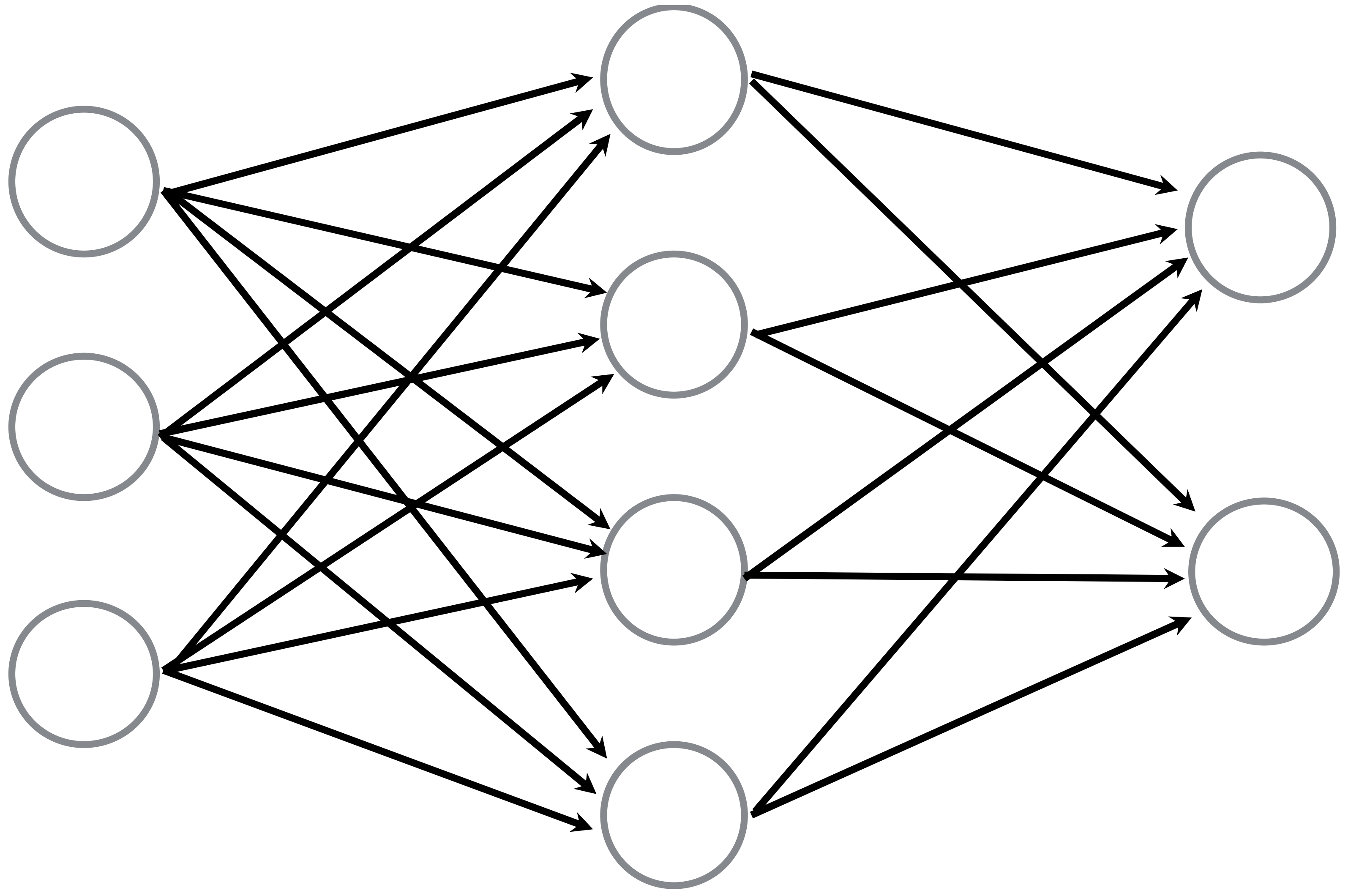


Some slides from Andrej Karpathy (Stanford), Suren Jayasuriya (ASU)

UMBC

Recap: Artificial Neural Networks: What's wrong on this slide?





Gradient Descent

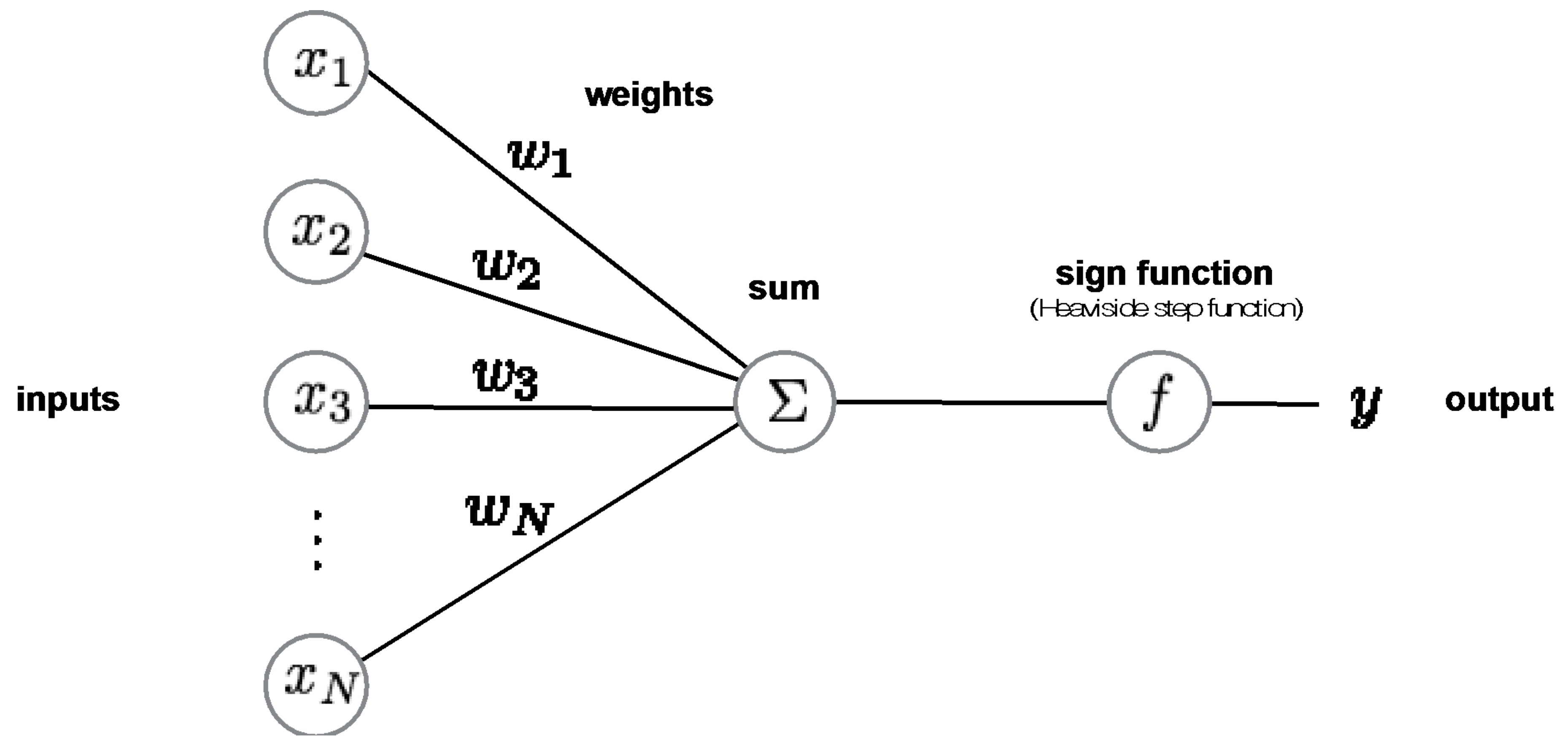
For each example sample $\{x_i, y_i\}$

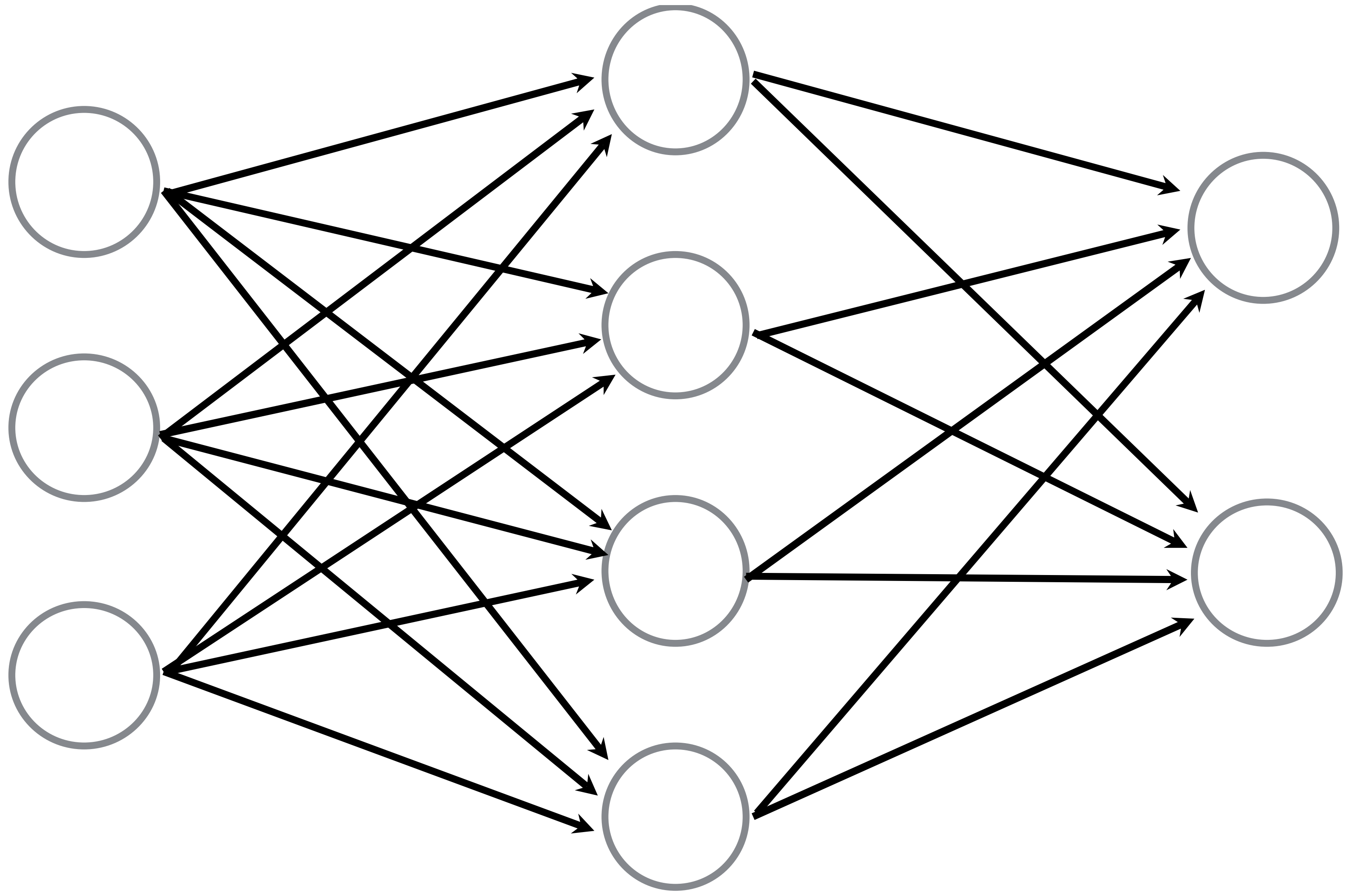
1. Predict
   a. Forward pass $\hat{y} = f_{MLP}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

Recap: Artificial Neural Networks: What's wrong on this slide?





Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{MLP}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

   vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

   vector of parameter update equations

Should be minus

Before Deep Learning

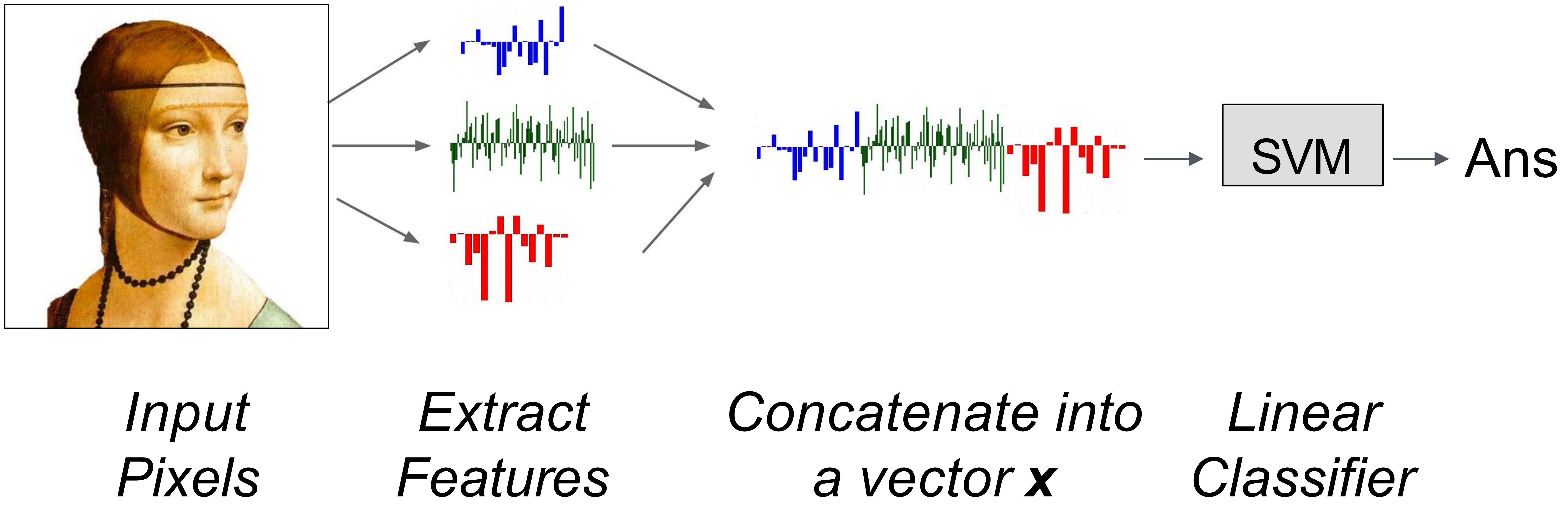


Figure: Karpathy 2016

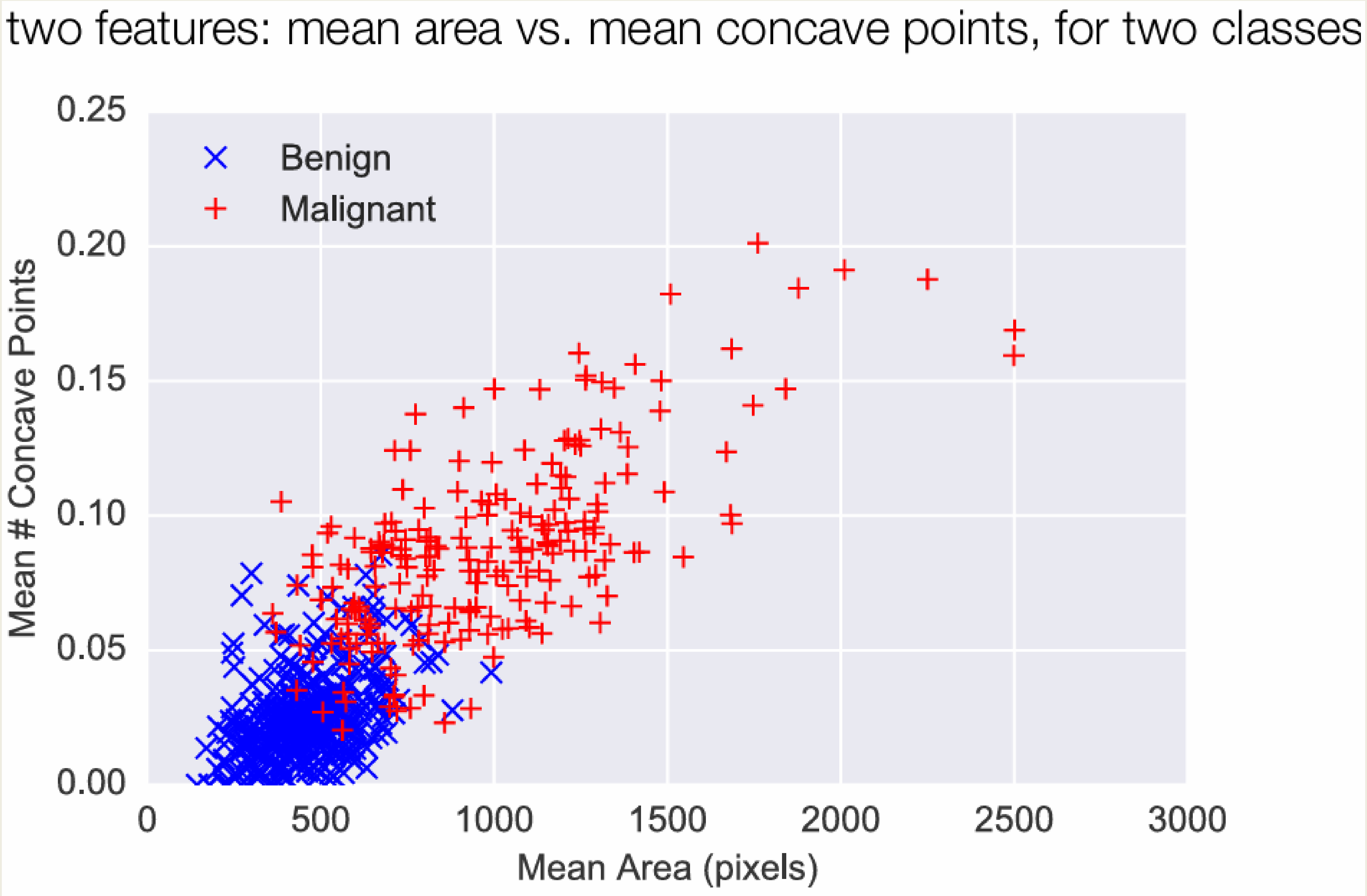
Recall: Tumor Classification

USING MACHINE LEARNING TO DIAGNOSE WHETHER A TUMOR IS BENIGN OR MALIGNANT

- Setting:
  - physician extracts a sample of fluid from tumor
  - Stains the cell → creates a "slide"
  - Computes features for each cell such as

    area, perimeter, concavity, texture etc.

- Want:
  - A system that can process the **"features"** and predict whether the tumor is benign or malignant



example from Zico Kolter

Before Deep Learning

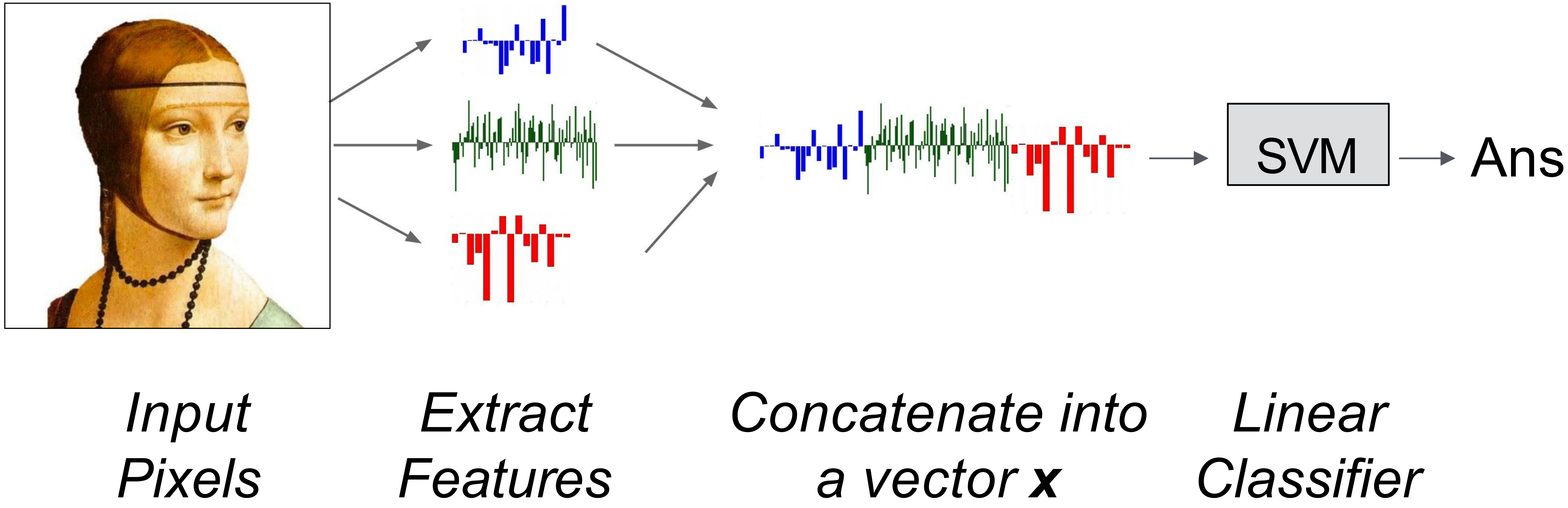


Figure: Karpathy 2016

Convolutional Neural Networks

Prerequisite:

What is a convolution?

Convolution

Convolution for 1D ***discrete*** signals

Definition of filtering as convolution:

$$(f * g)(i) = \sum_{j=1}^{m} g(j) \cdot f(i - j + m/2)$$

1D Convolution. Example

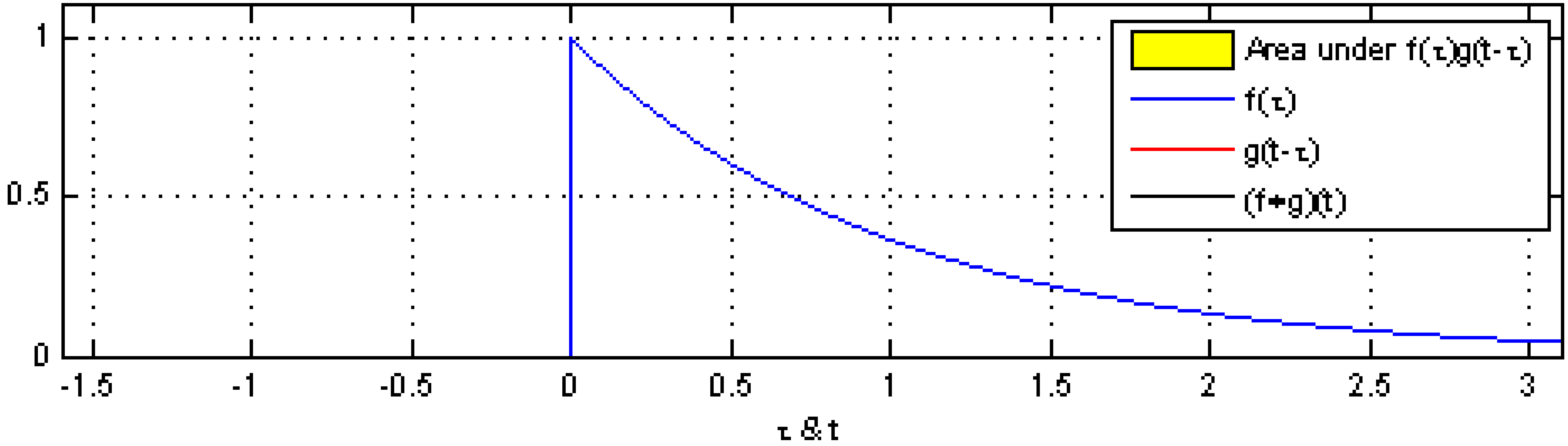
Suppose our input 1D image is:

$$f = \begin{array}{|c|c|c|c|c|c|c|} \hline 10 & 50 & 60 & 10 & 20 & 40 & 30 \\ \hline \end{array}$$

and our kernel is:

$$g = \begin{array}{|c|c|c|} \hline 1/3 & 1/3 & 1/3 \\ \hline \end{array}$$

Let's call the output image h. What is the value of $h(3)$?



1
0.5
0
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
τ & t
Area under f(τ)g(t-τ)
f(τ)
g(t-τ)
(f*g)(t)

1D Convolution. Example

Suppose our input 1D image is:

$$f = \begin{array}{|c|c|c|c|c|c|c|}\hline 10 & 50 & 60 & 10 & 20 & 40 & 30 \\ \hline\end{array}$$

and our kernel is:

"Box" Filter that causes "Blur" or "Smoothing"

$$g = \begin{array}{|c|c|c|}\hline 1/3 & 1/3 & 1/3 \\ \hline\end{array}$$

Let's call the output image h. What is the value of $h(3)$?

$$h = \begin{array}{|c|c|c|c|c|c|c|}\hline 20 & 40 & 40 & 30 & 20 & 30 & 23.333 \\ \hline\end{array}$$

Convolution for 2D discrete signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

filtered image

filter

input image

Convolution for 2D discrete signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

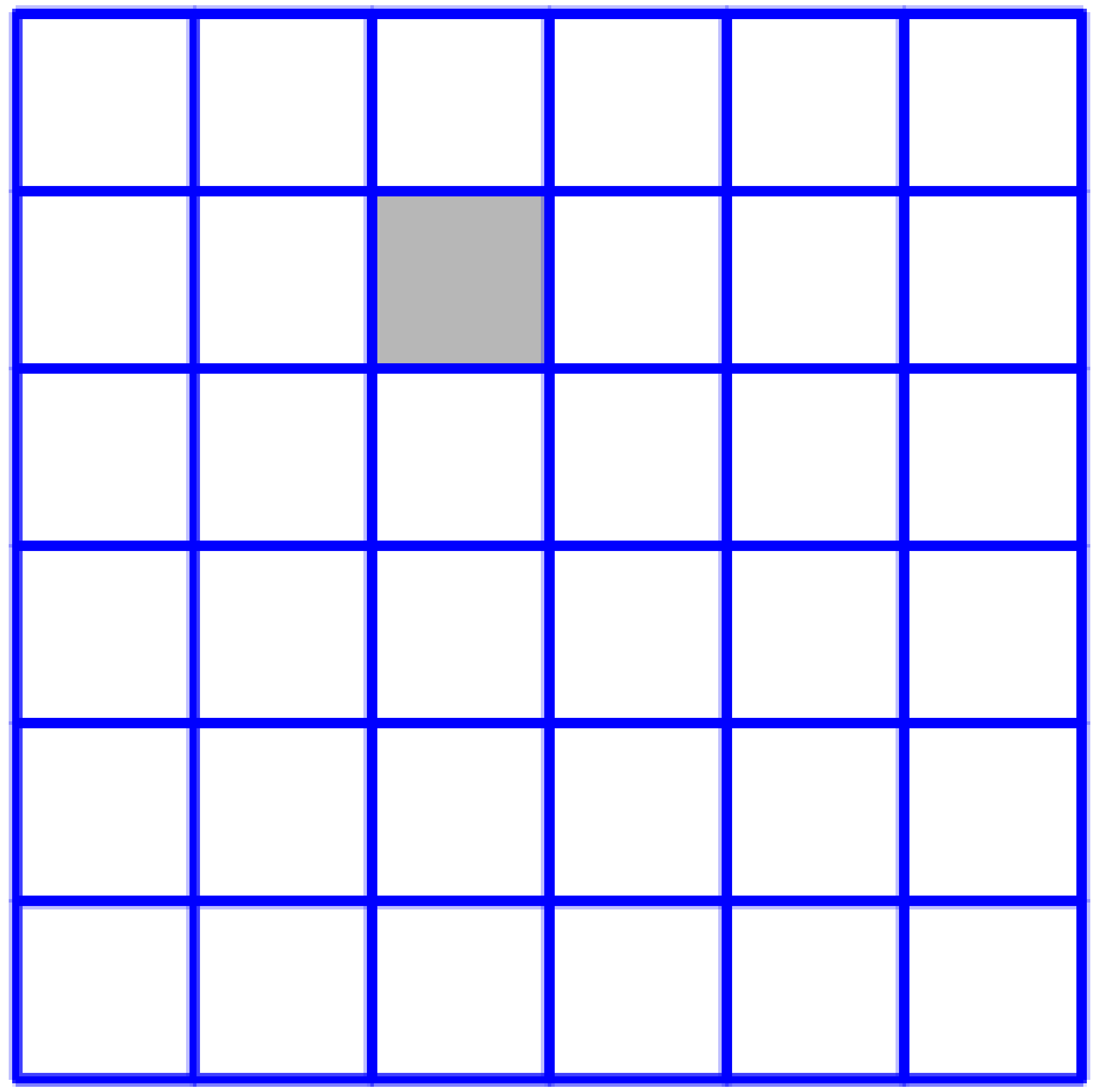
filtered image   filter   input image

If the filter $f(i,j)$ is non-zero only within $-1 \leq i, j \leq 1$,
then

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i, j) I(x - i, y - j)$$

The kernel we saw earlier is the 3x3 matrix representation of $f(i,j)$.

An image is a matrix of pixels

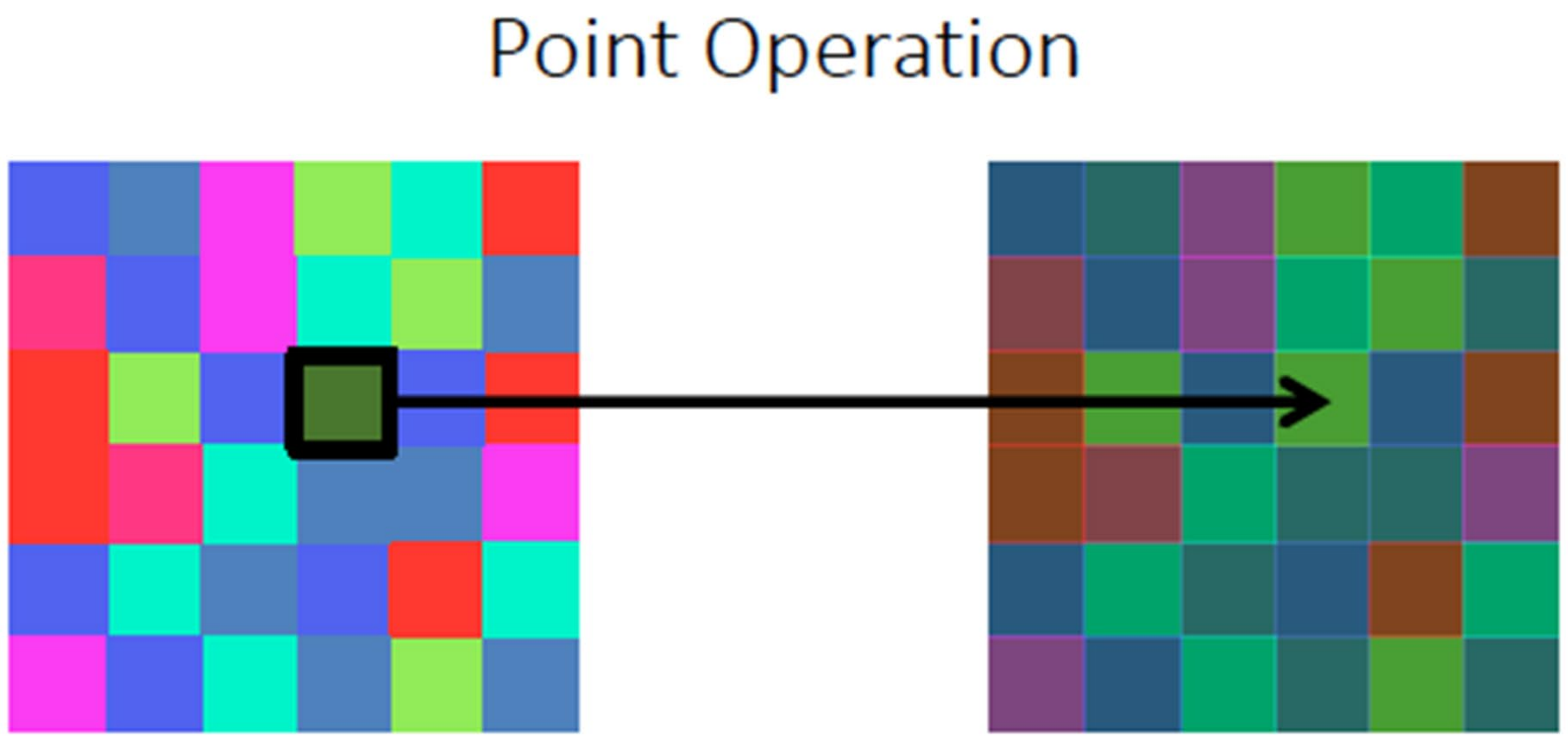


F[x,y]

An array of numbers (“pixels”)

x,y are integer column/row indices

Point Processing vs Image Filtering



point processing

Examples of point processing

original

$$x$$

darken

$$x - 128$$

lower contrast

$$\frac{x}{2}$$

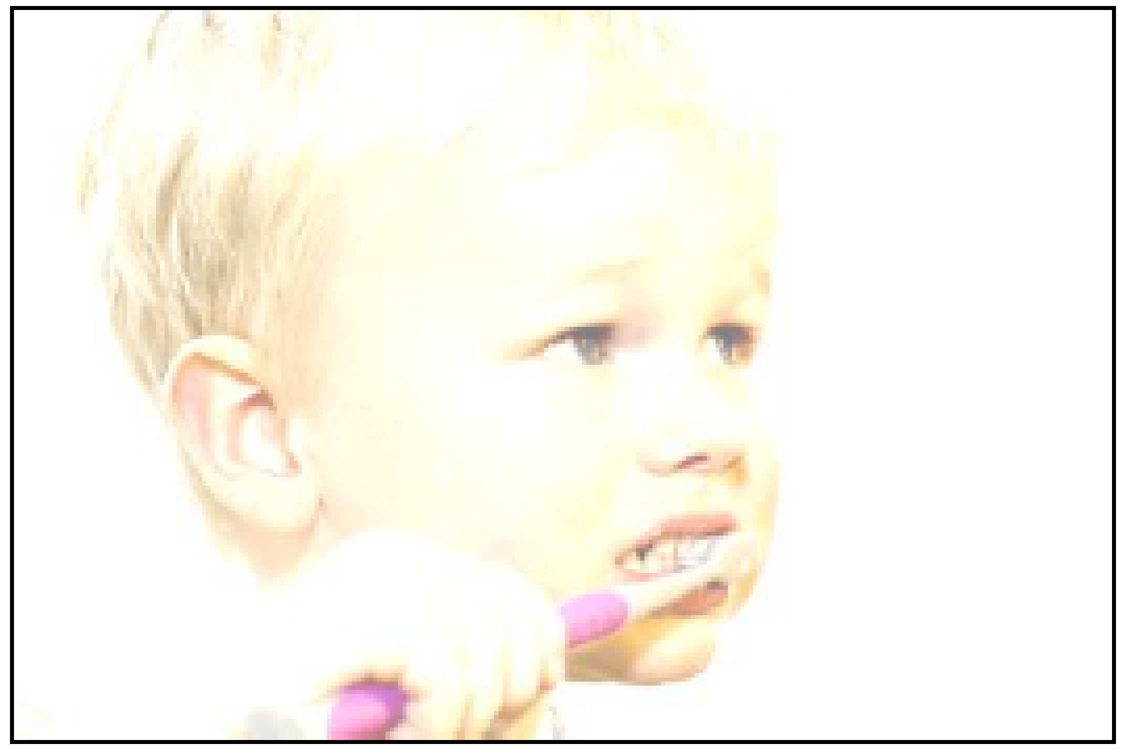
non-linear lower contrast

$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

invert

$$255 - x$$

lighten



$$x + 128$$

raise contrast

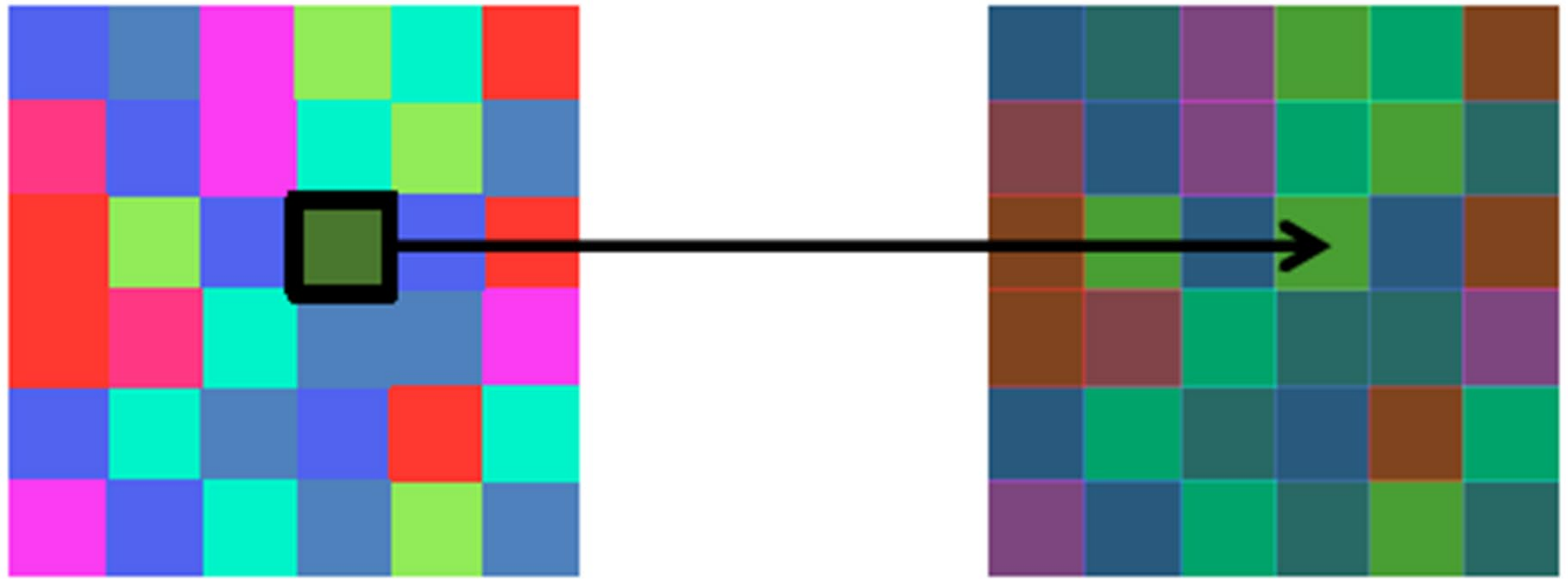
$$x \times 2$$

non-linear raise contrast

$$\left(\frac{x}{255}\right)^{2} \times 255$$

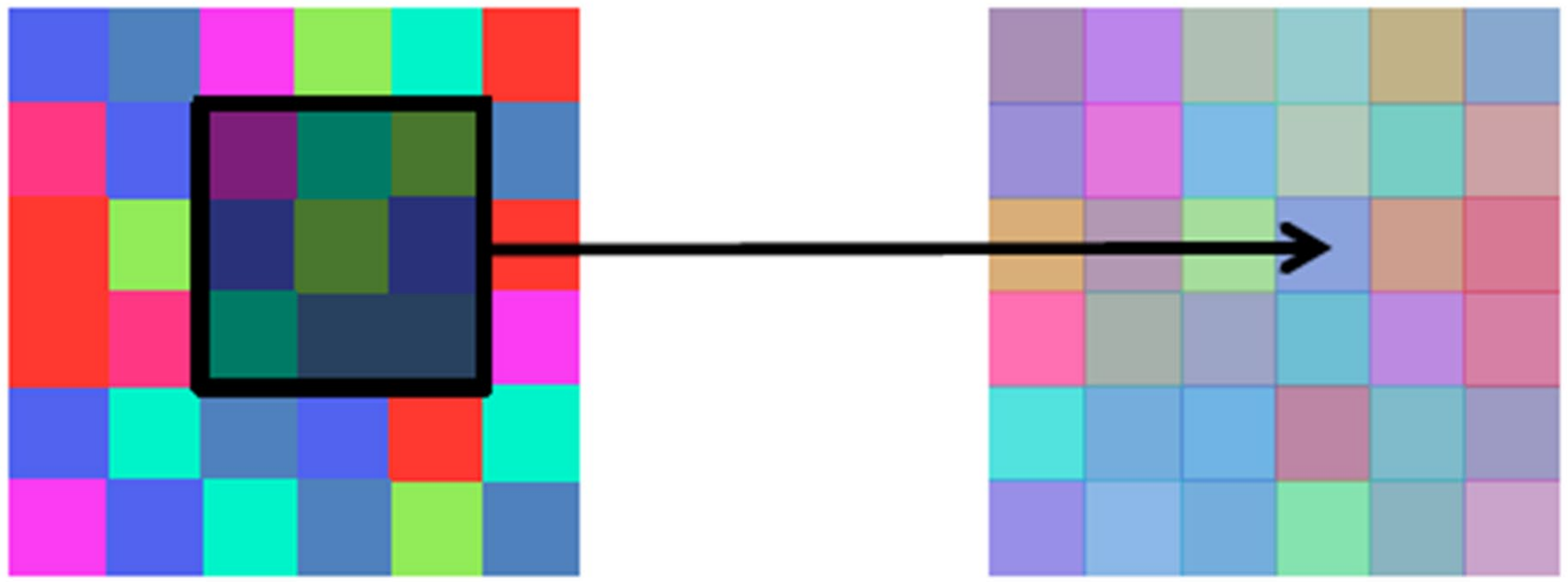
Point Processing vs Image Filtering

Point Operation



point processing

Neighborhood Operation



“filtering”

Image Filtering is Convolution

Image Convolution Examples

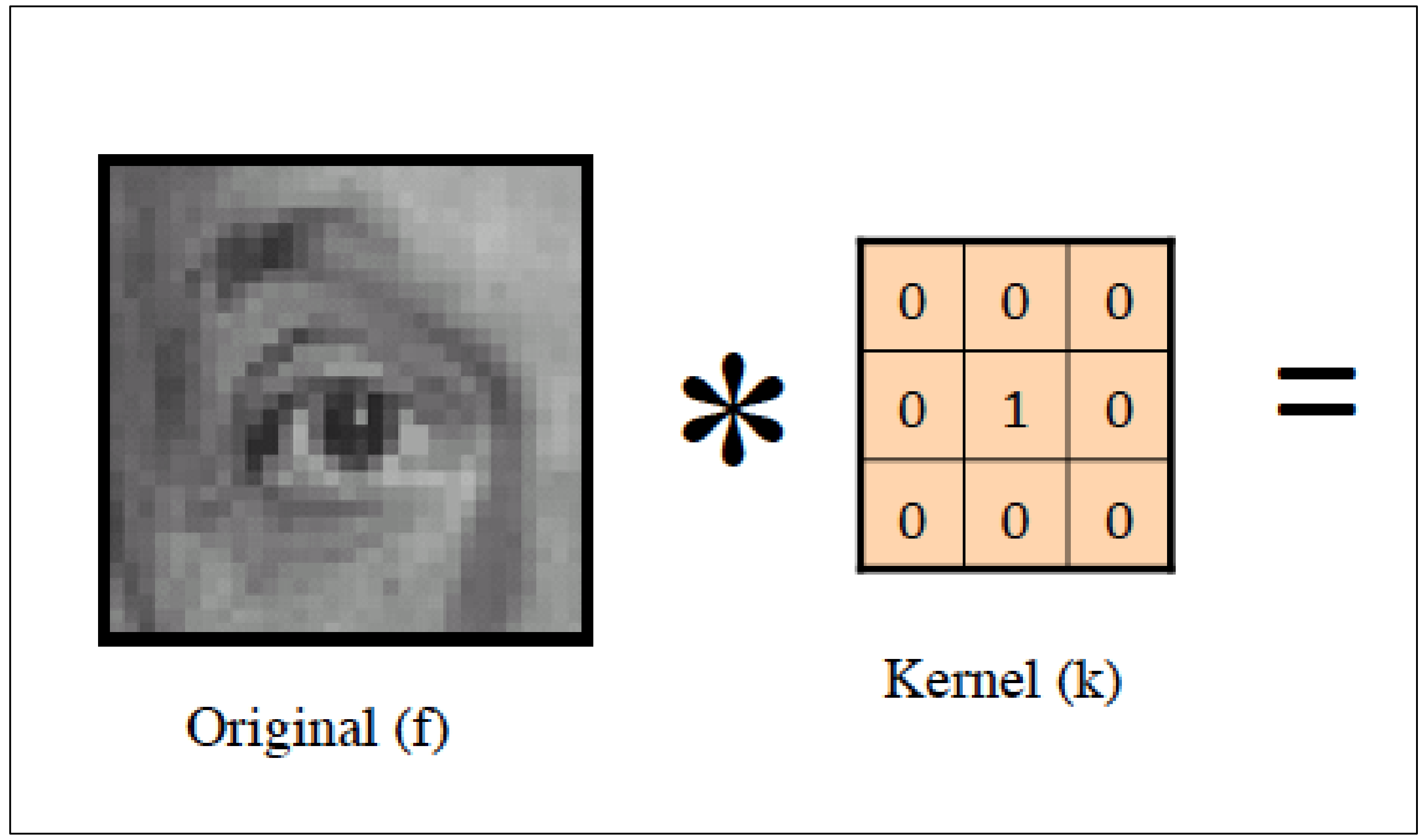


Image Convolution Examples

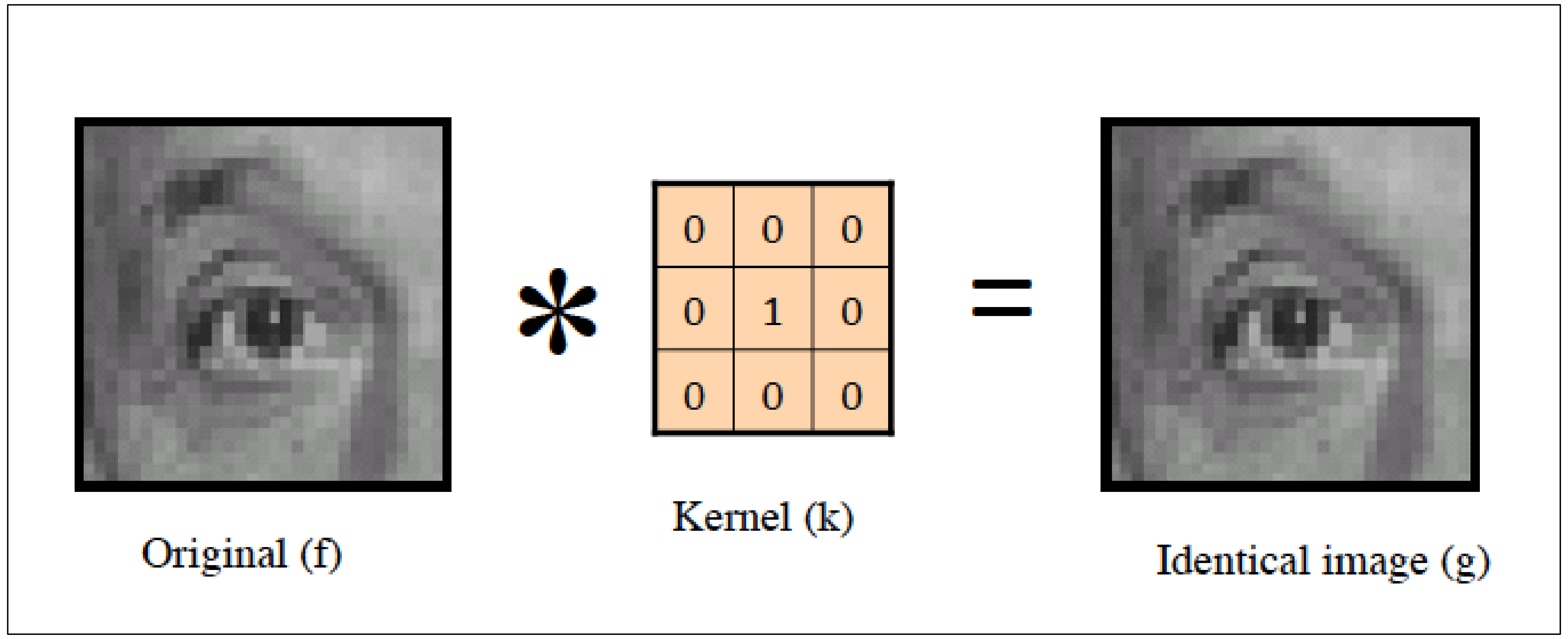
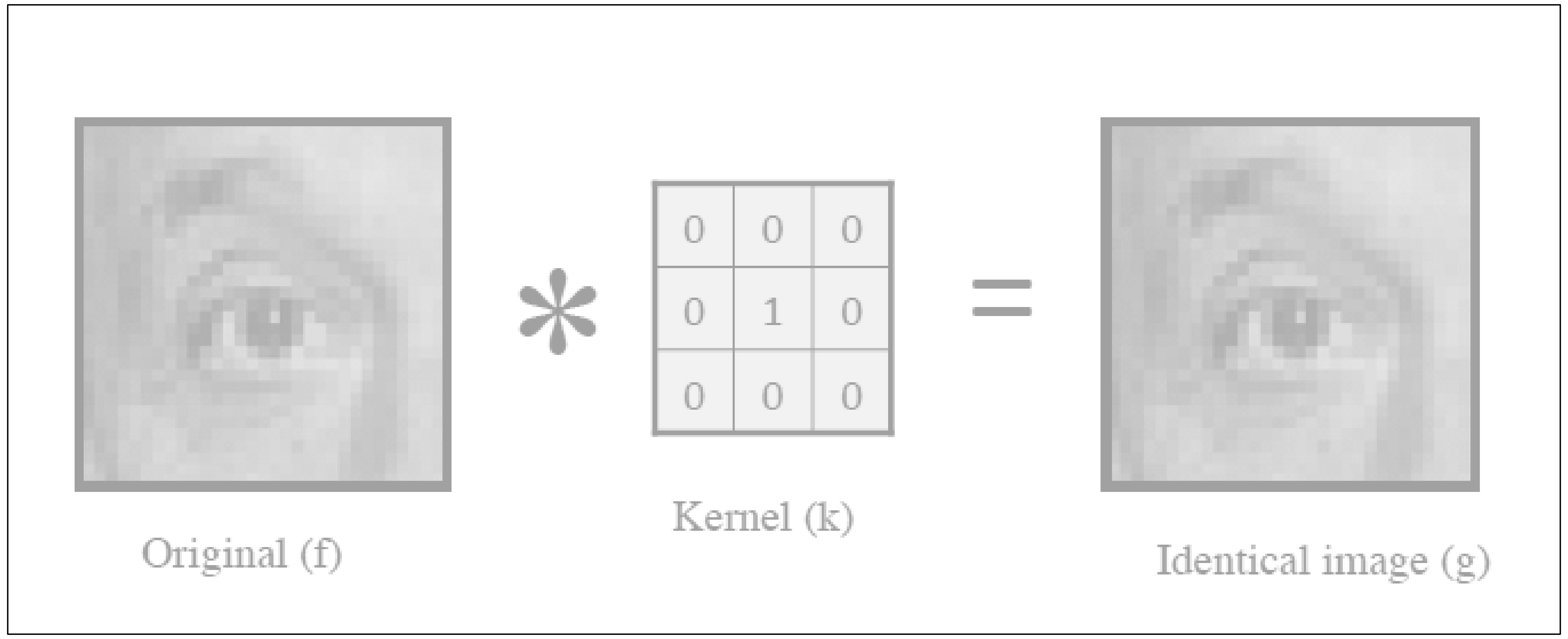


Image Convolution Examples



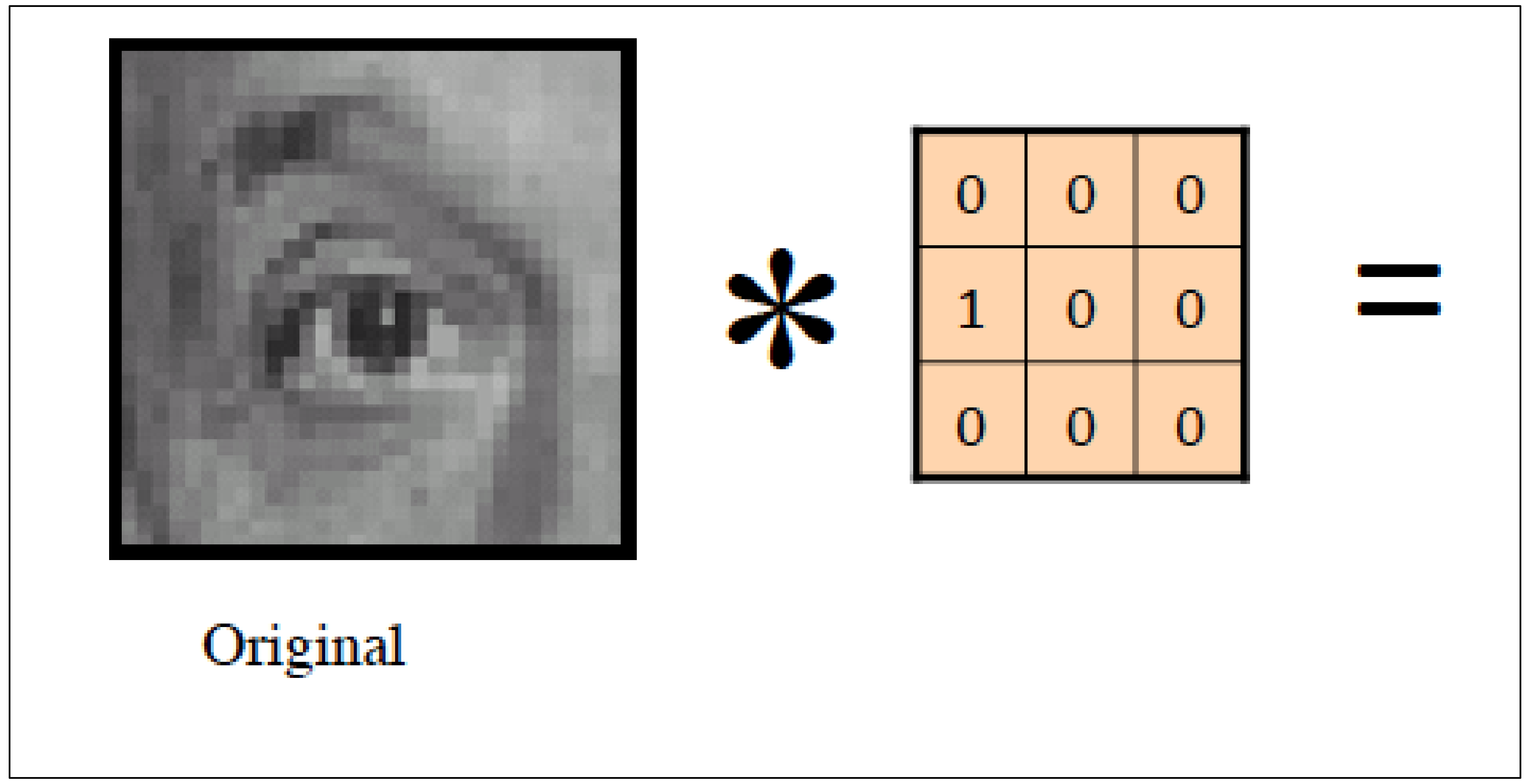
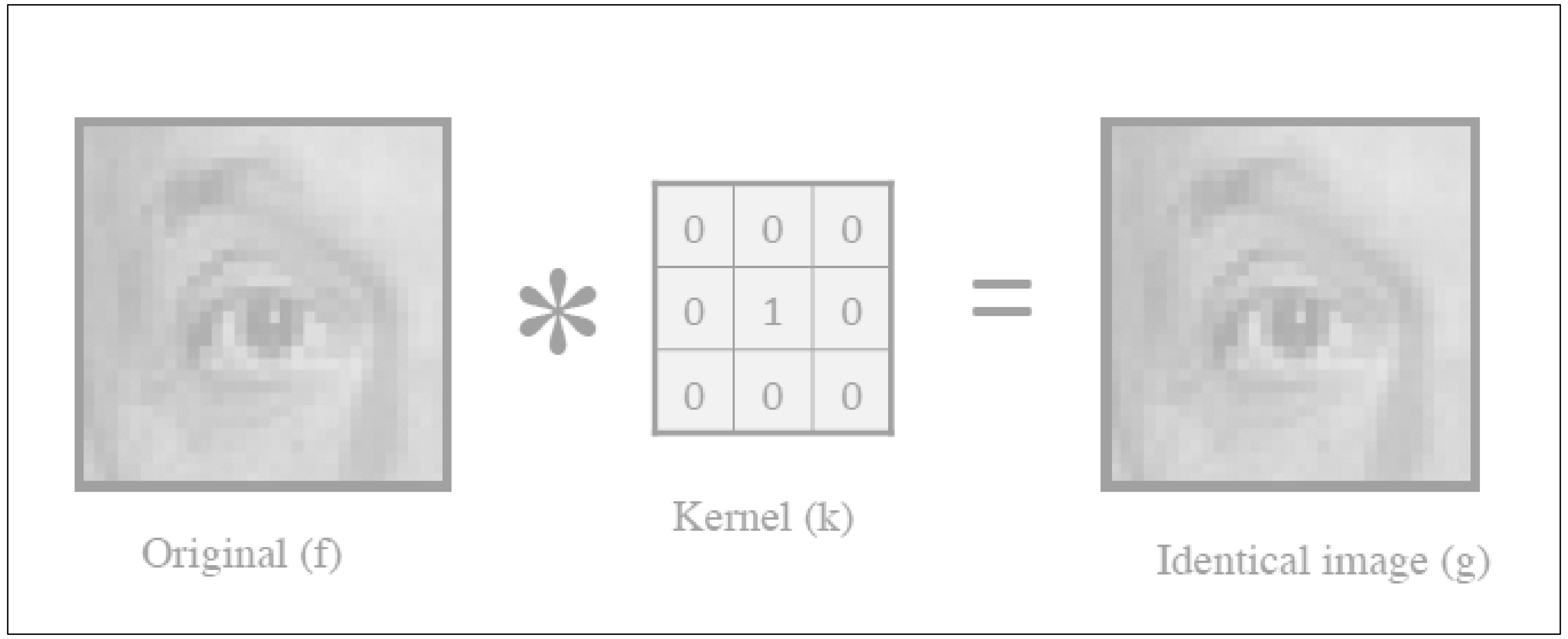


Image Convolution Examples



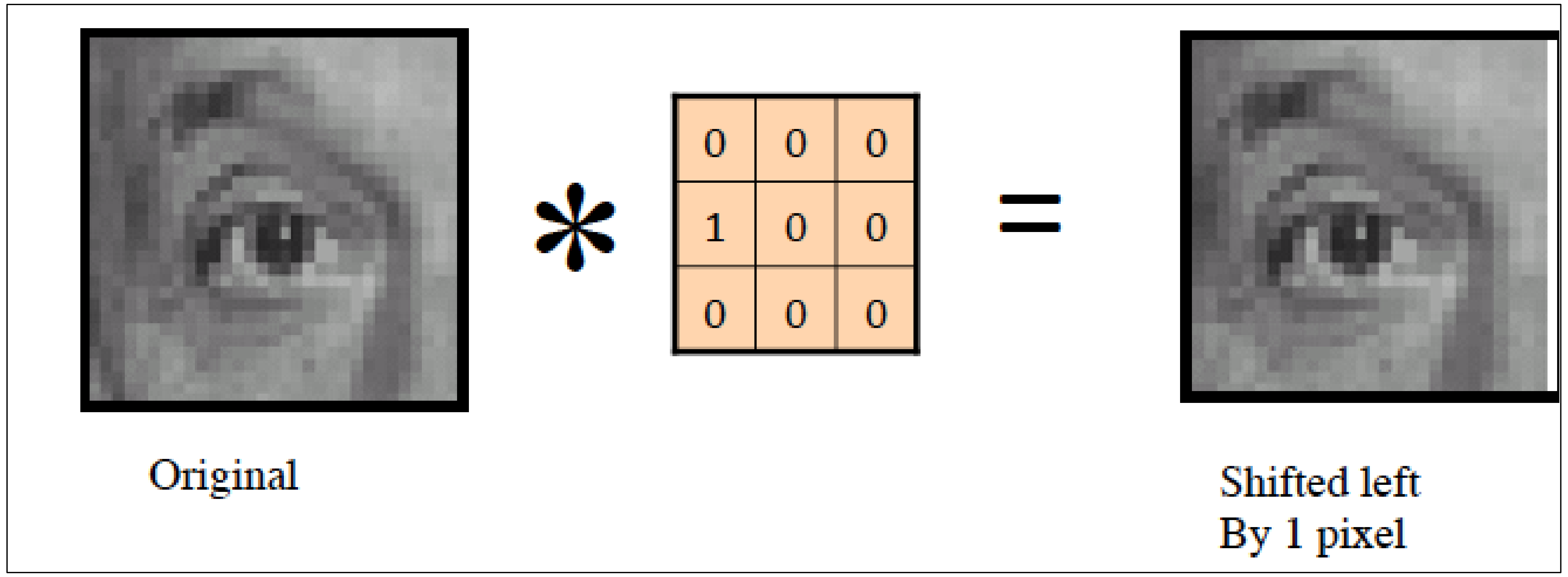
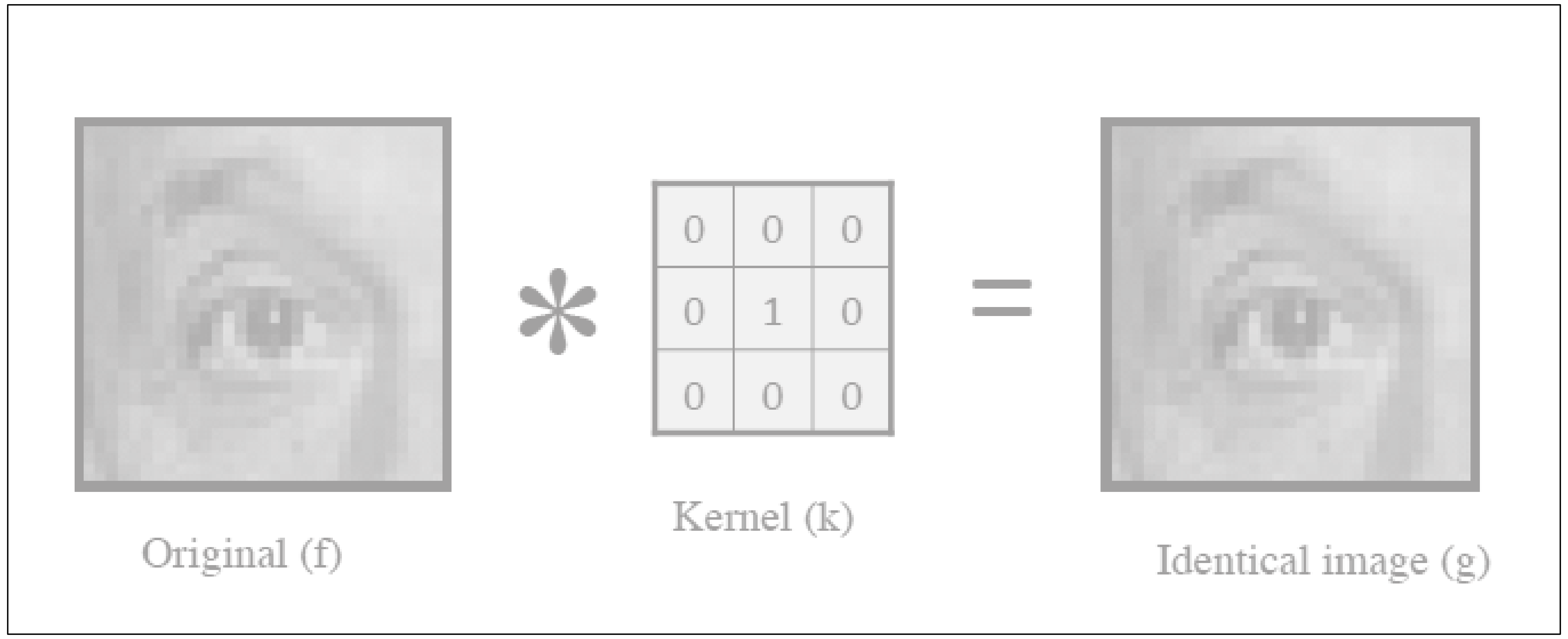
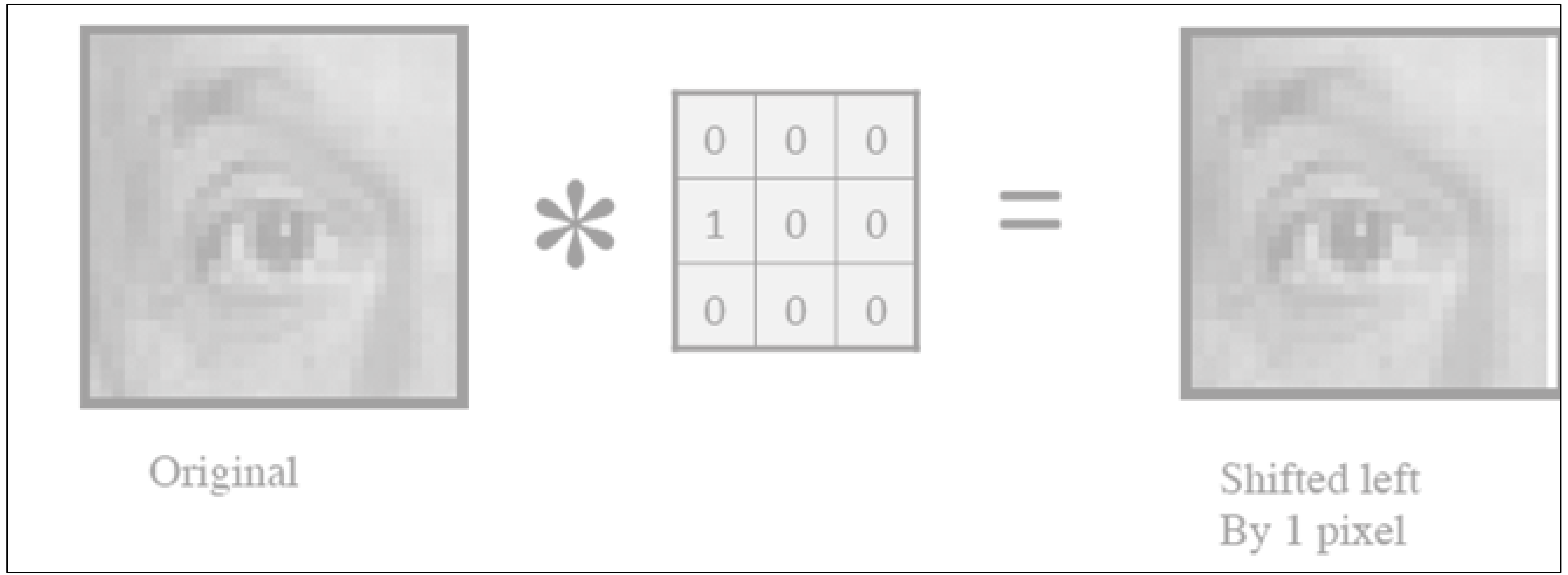


Image Convolution Examples





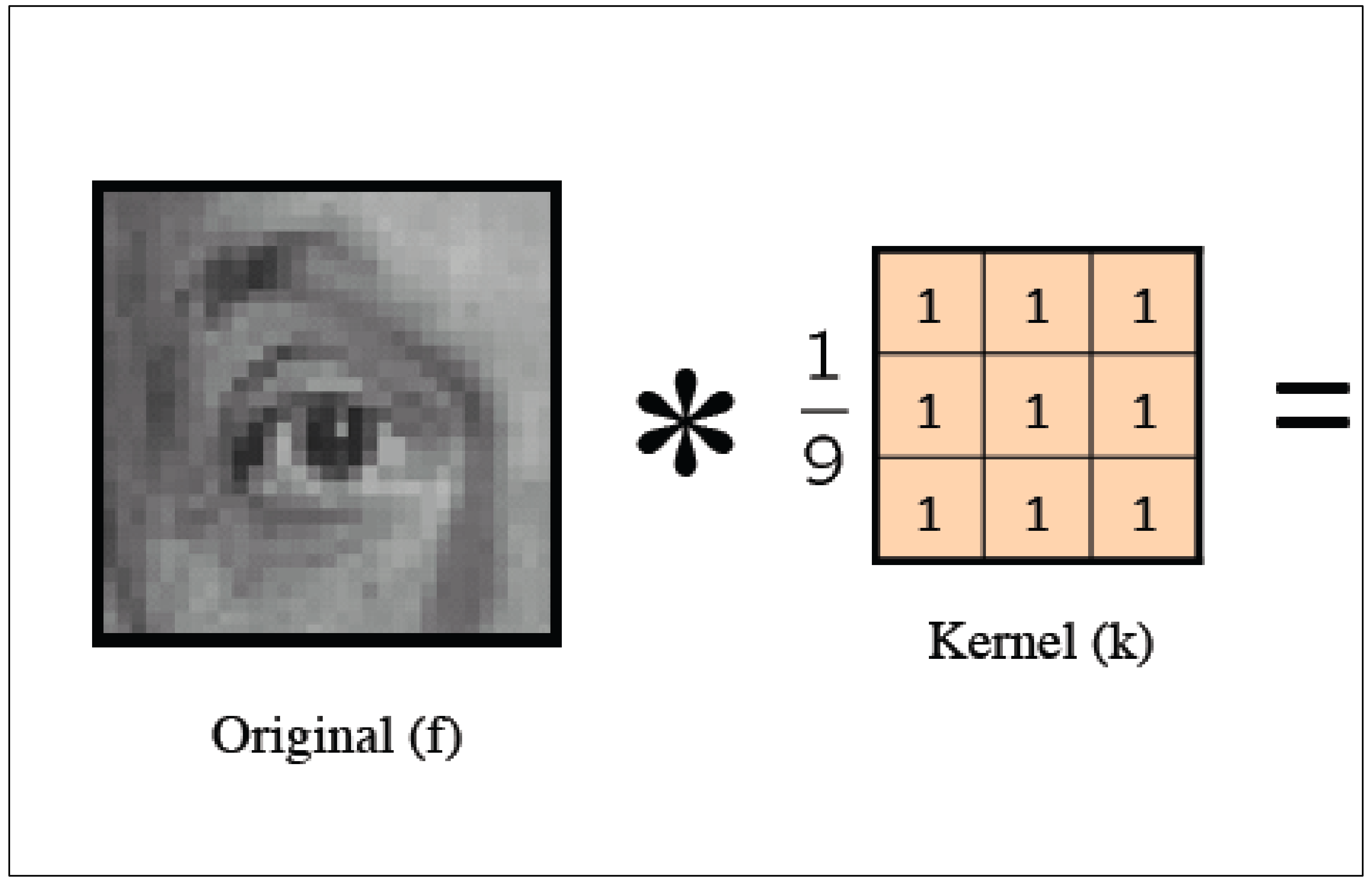
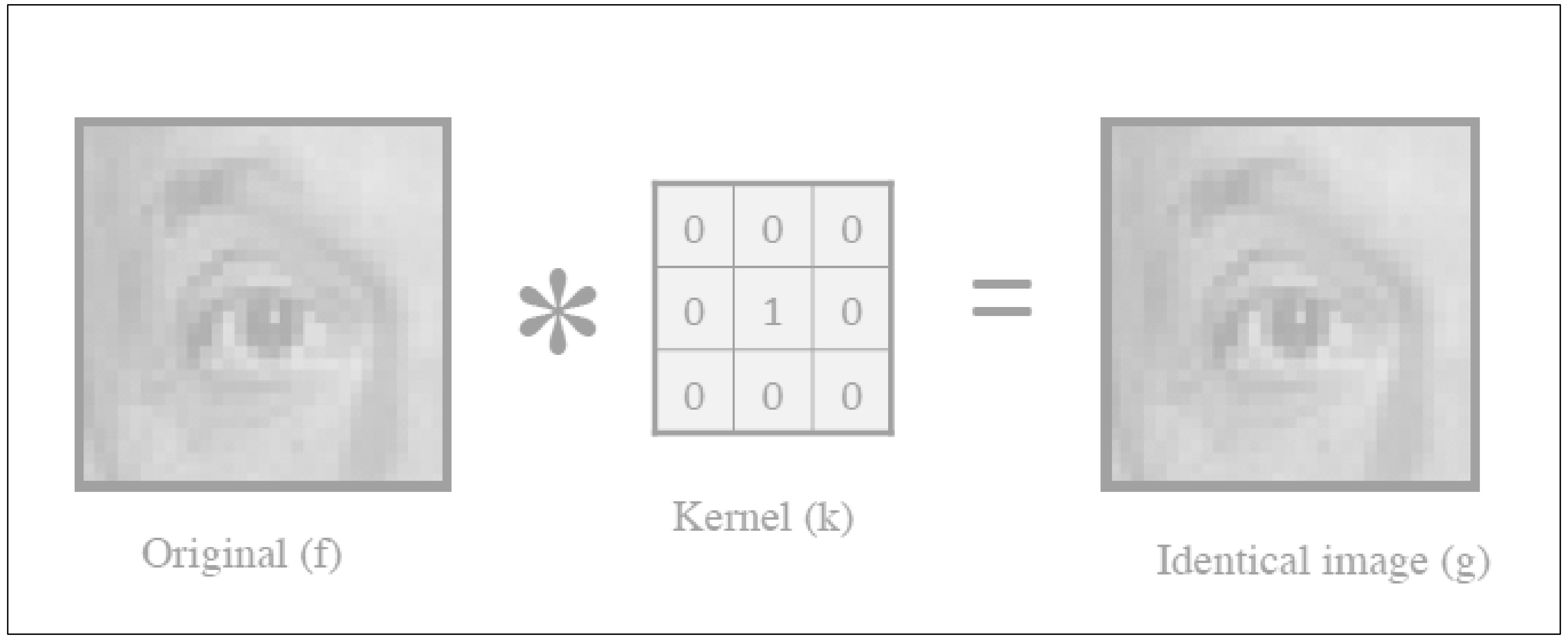
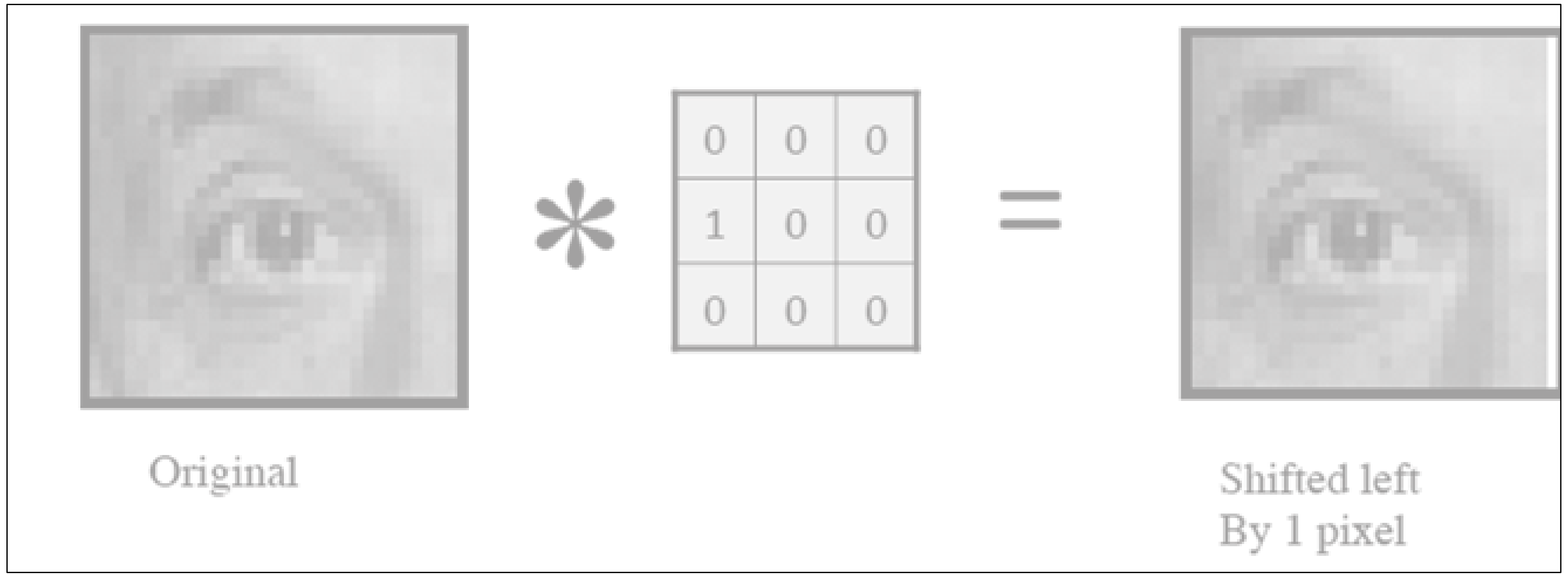


Image Convolution Examples





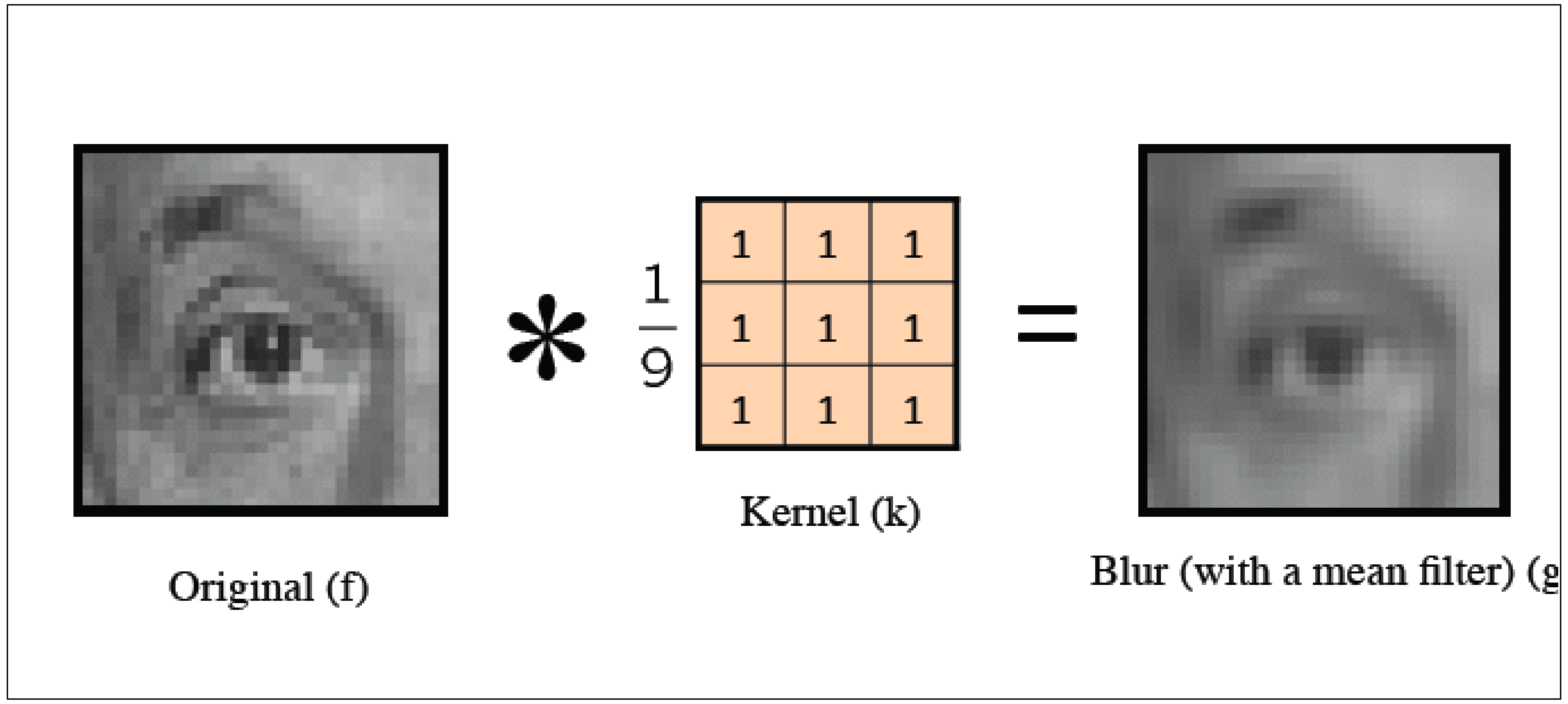
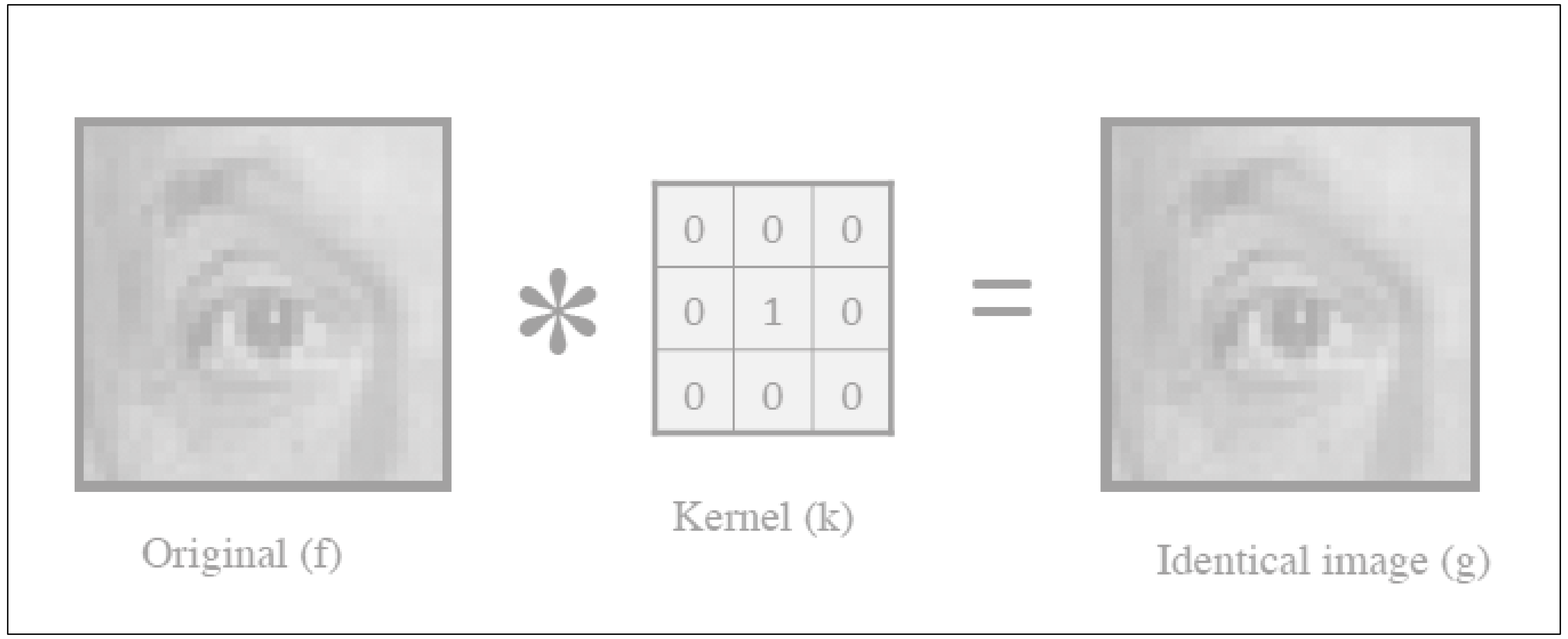
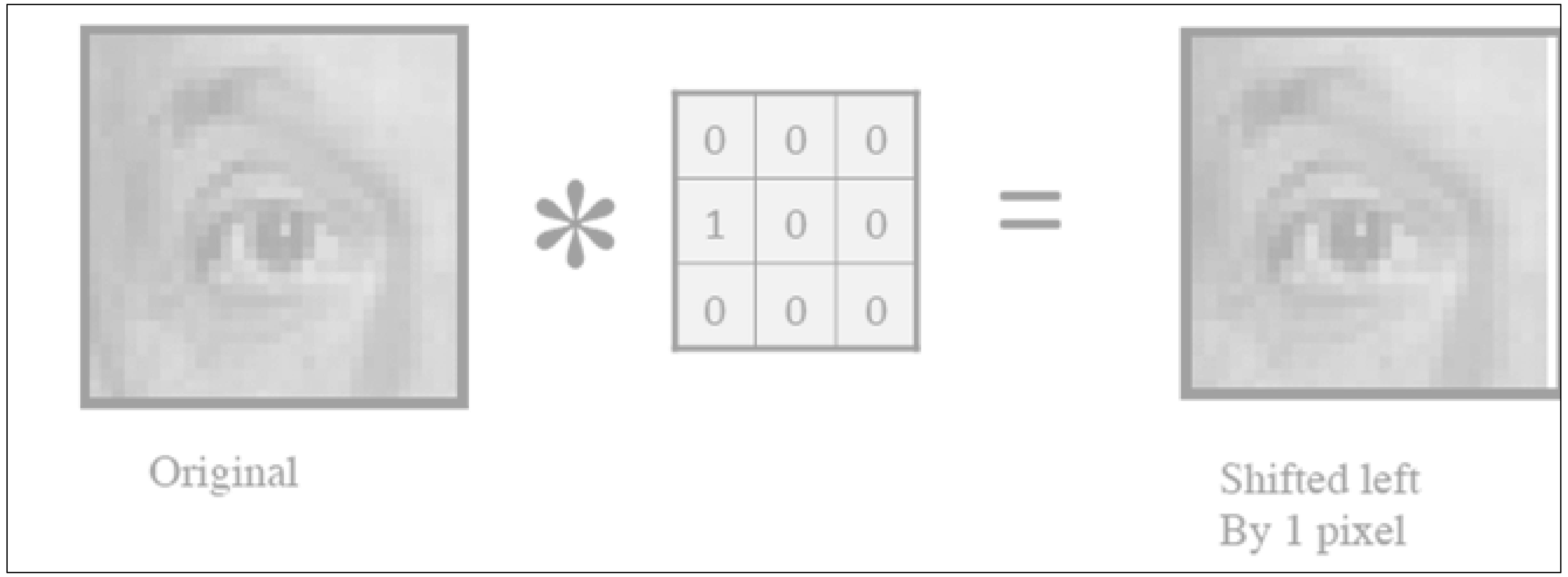
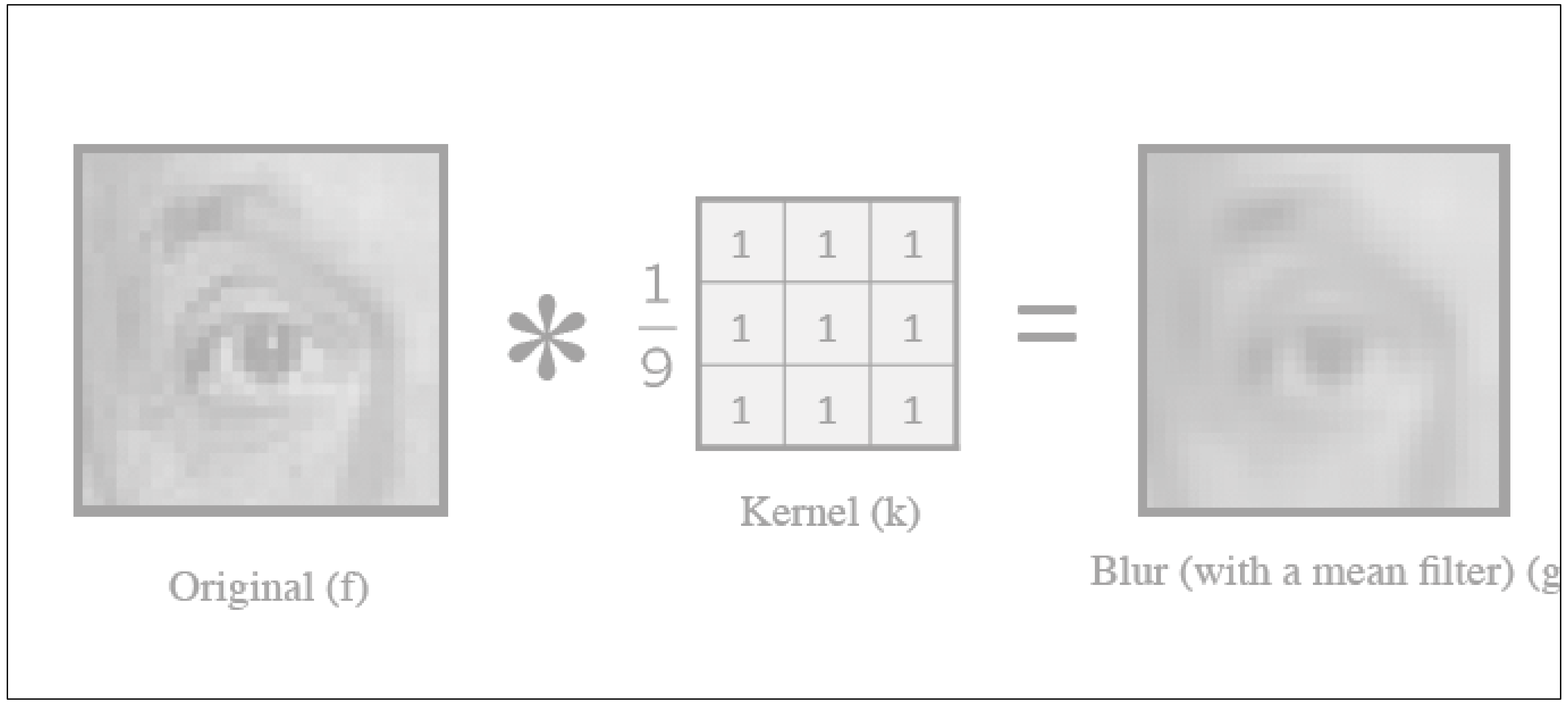


Image Convolution Examples







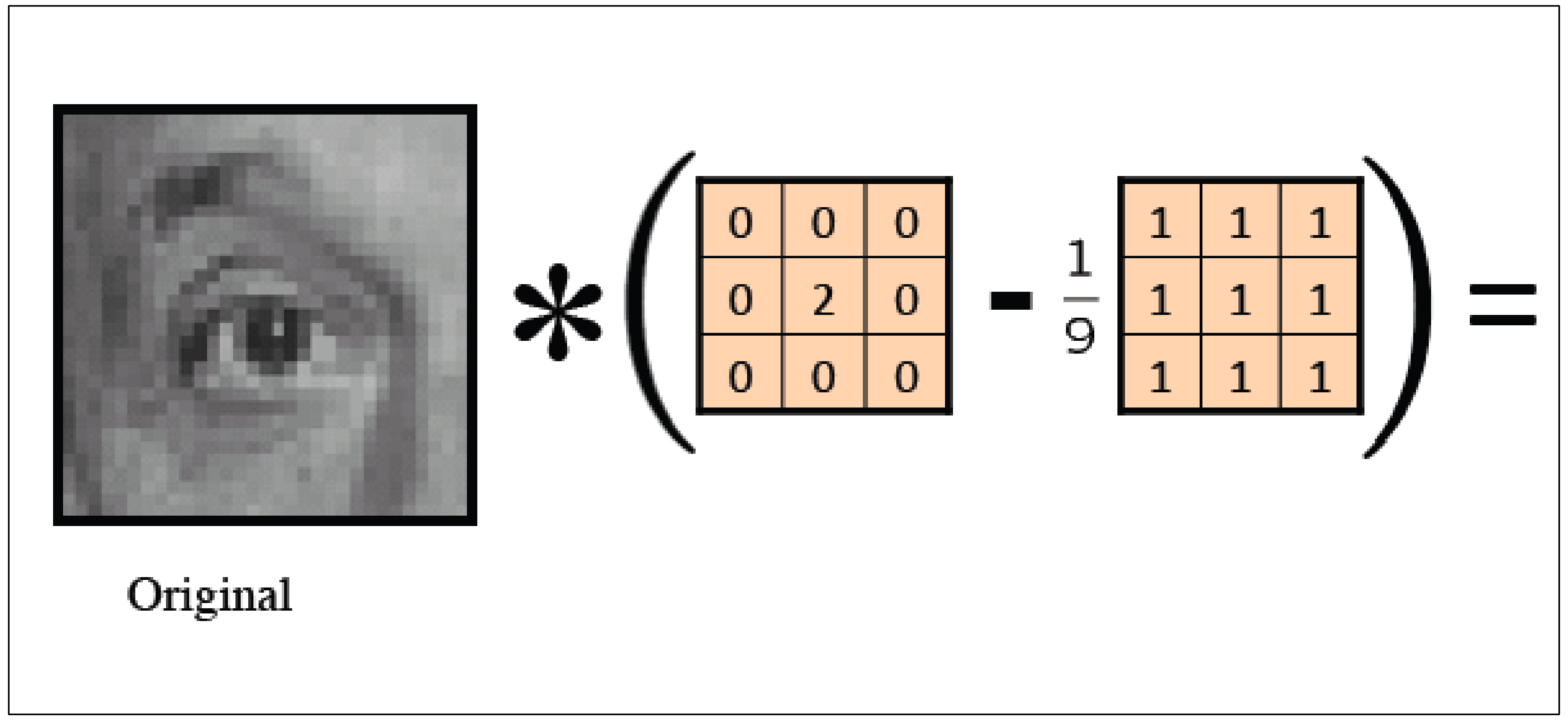
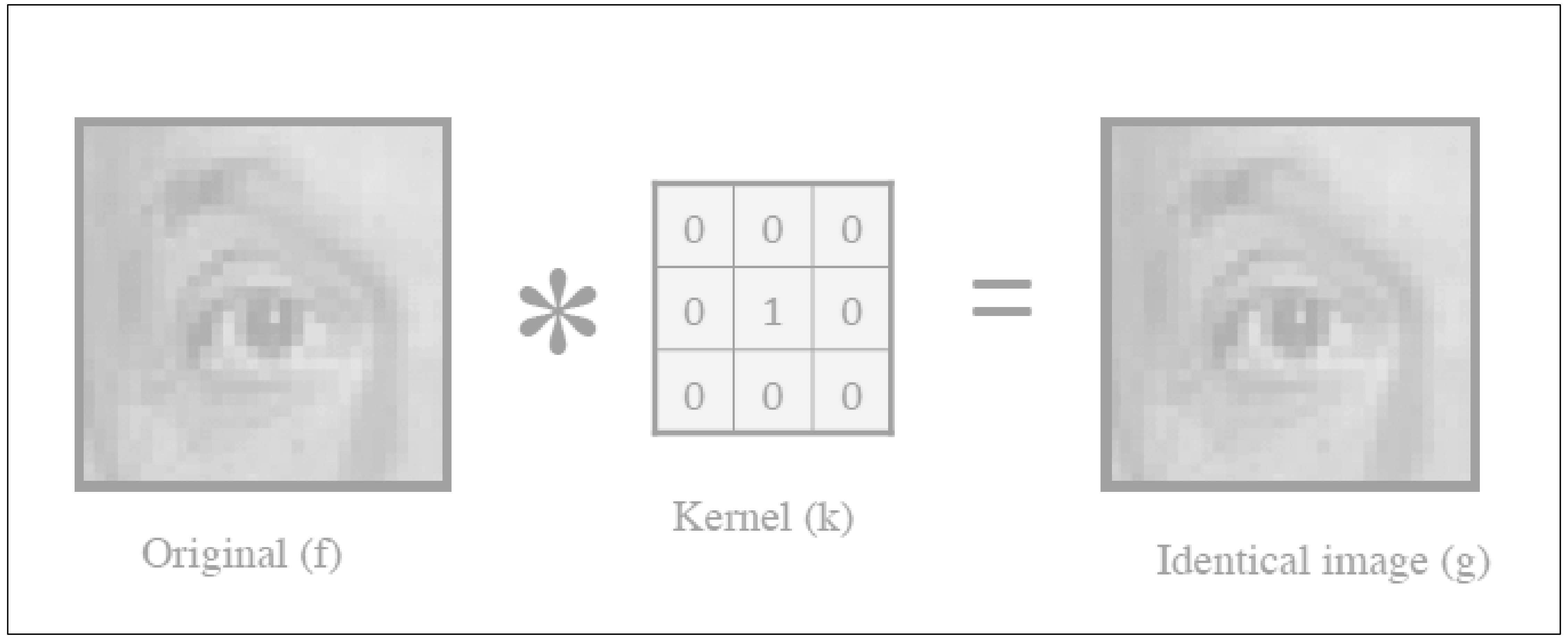
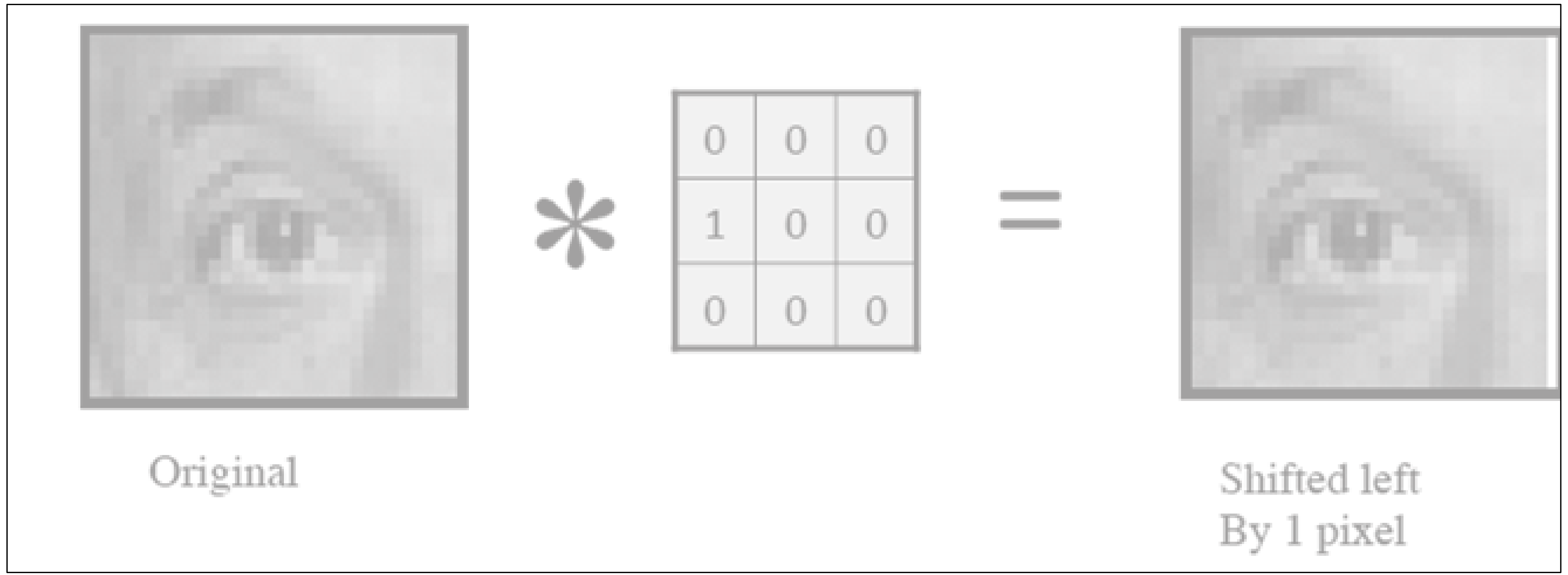
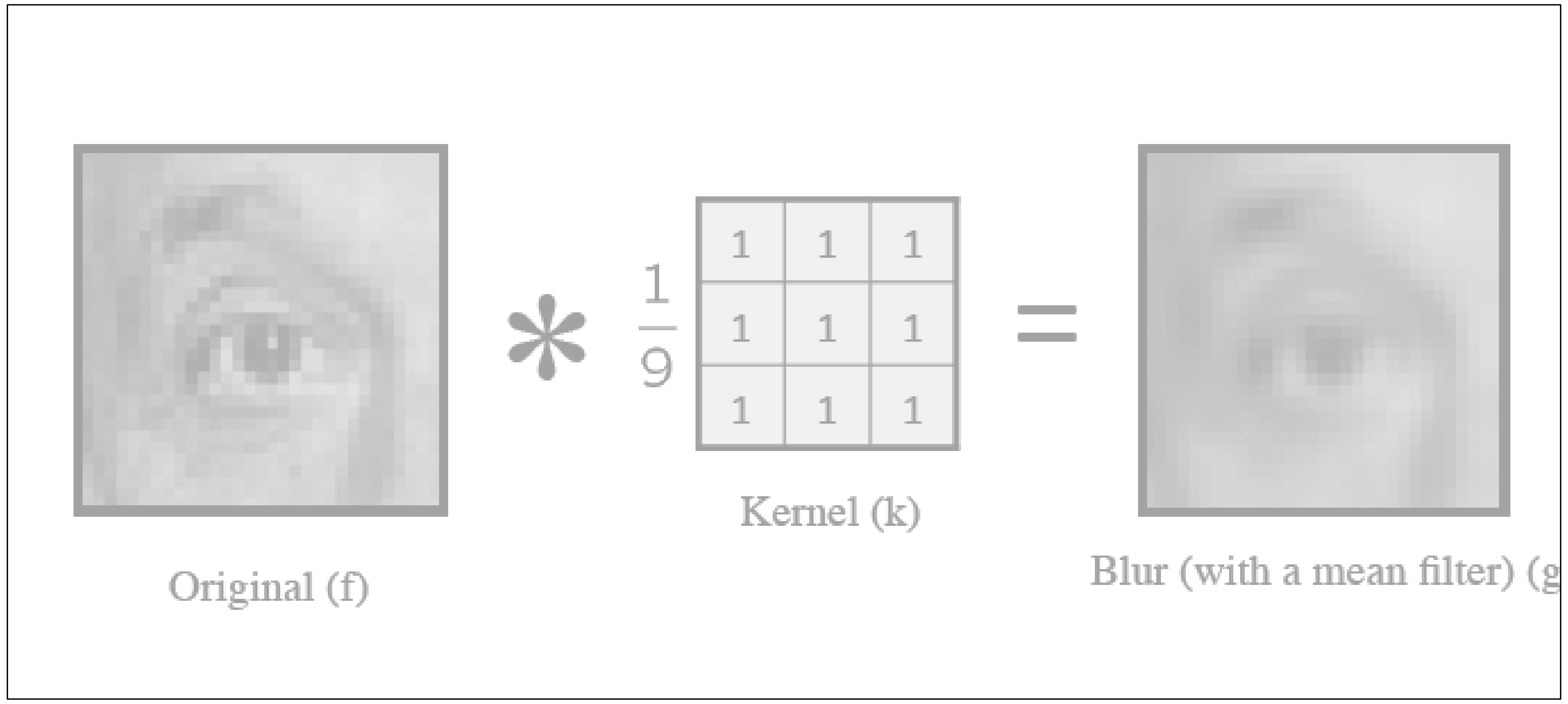
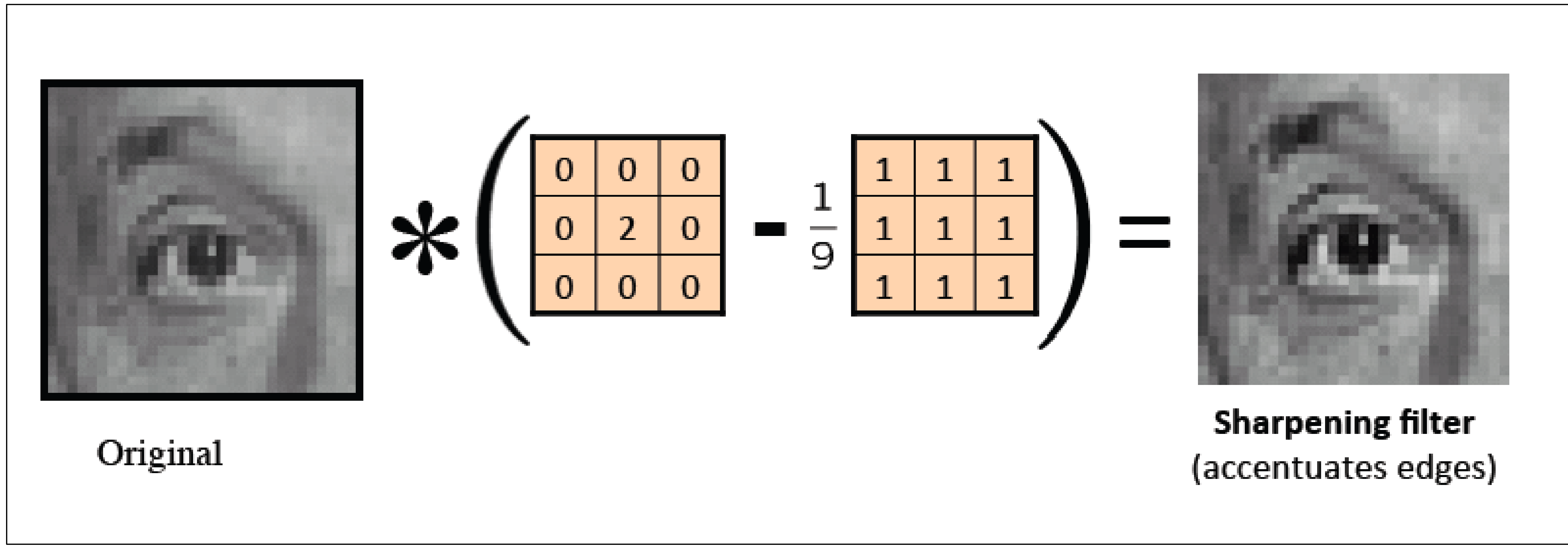


Image Convolution Examples









Example: box filter

$$g[\cdot\,,\cdot\,] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slide credit: David Lowe (UBC)

Image filtering

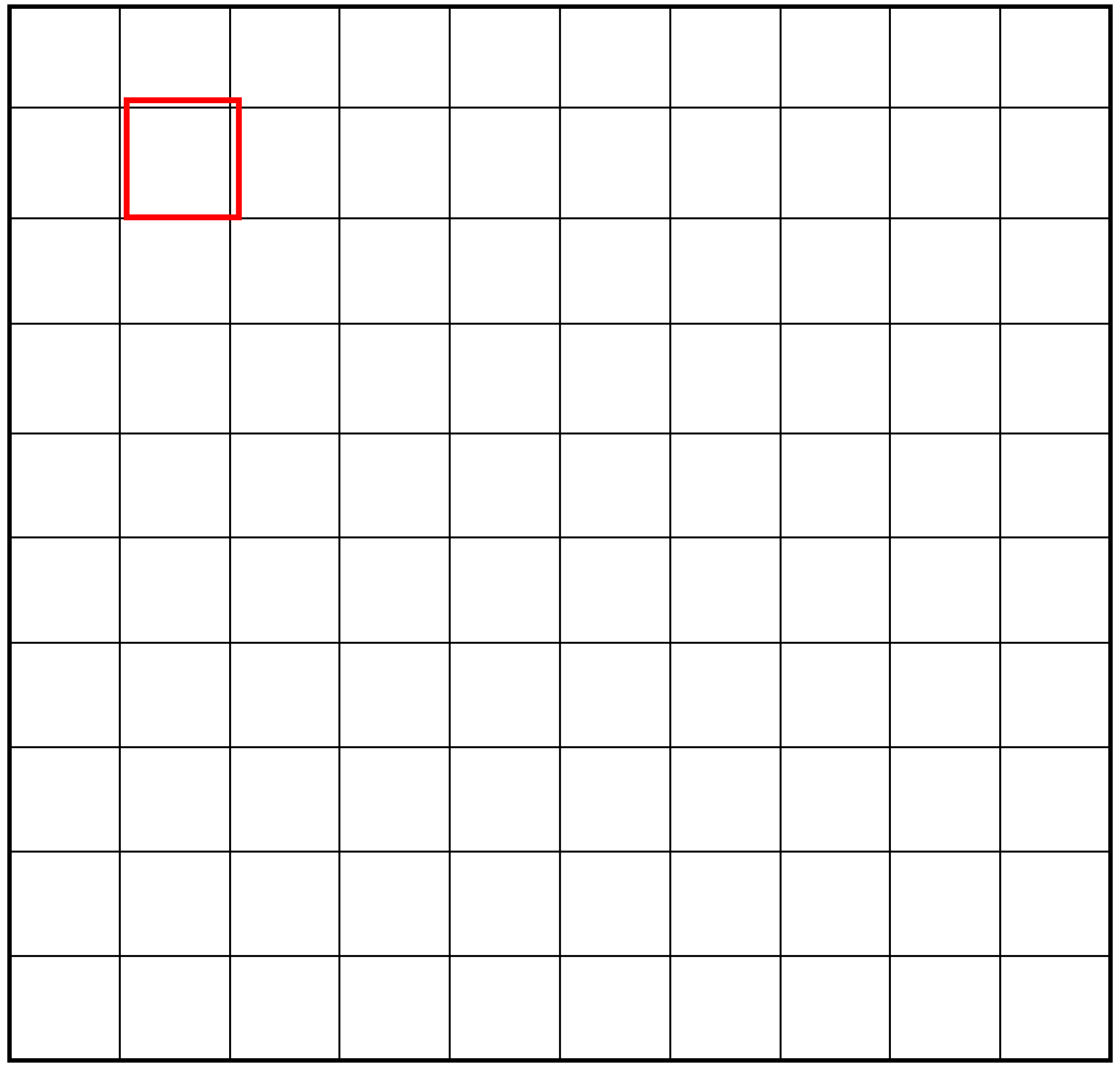
$g[\cdot,\cdot]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$



$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Image filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10							

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Image filtering

$g[\cdot\,,\cdot\,]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20						

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Image filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30					

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Image filtering

$g[\cdot\,,\cdot\,]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Image filtering

$g[\cdot,\cdot]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				
				?					

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Image filtering

$g[\cdot,\cdot]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

$$h[m,n]=\sum_{k,l} g[k,l]\, f[m+k,n+l]$$

Image filtering

$g[\cdot,\cdot]$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

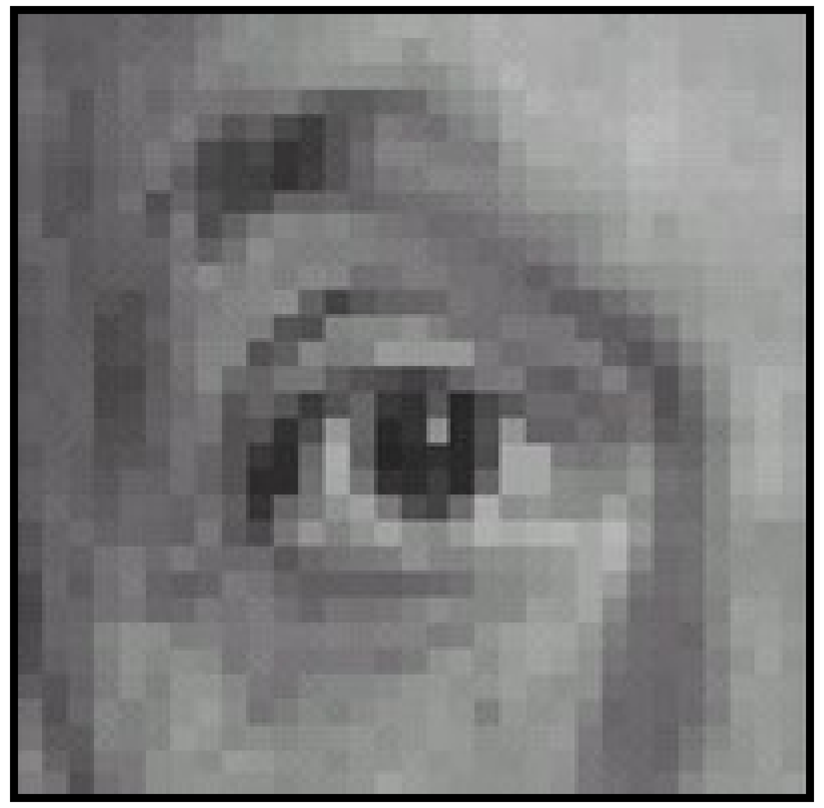
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Practice with linear filters



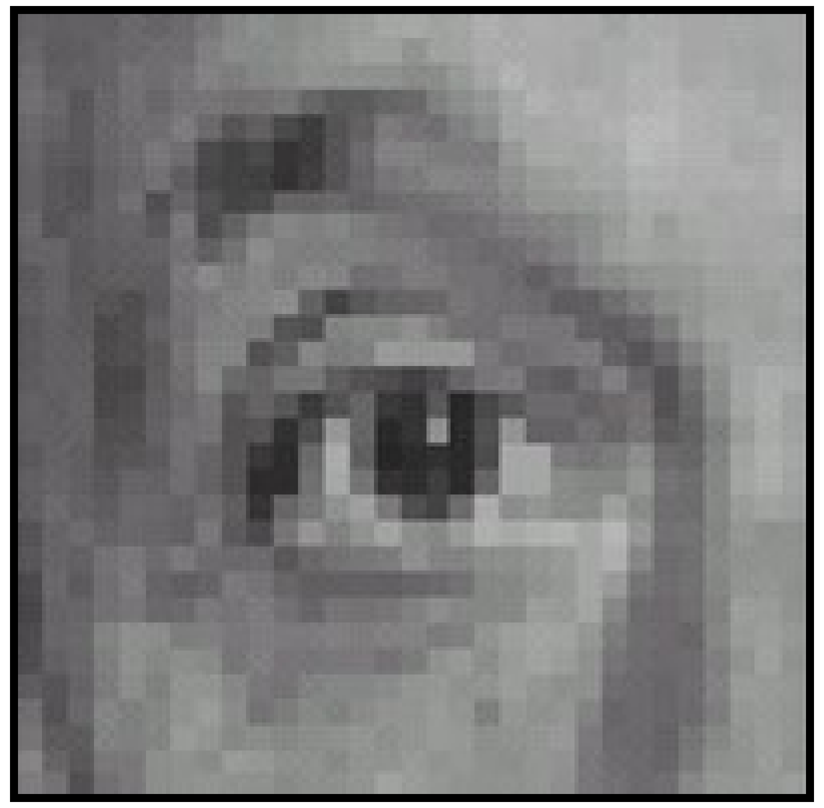
Original

0	0	0
0	1	0
0	0	0

?

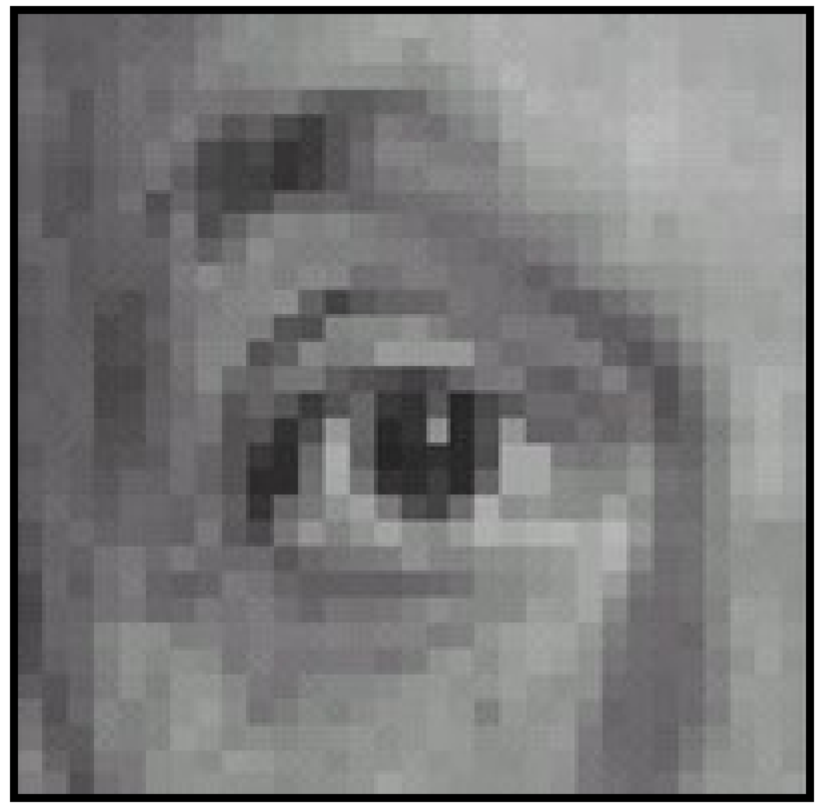
Source: D. Lowe

Practice with linear filters



Original

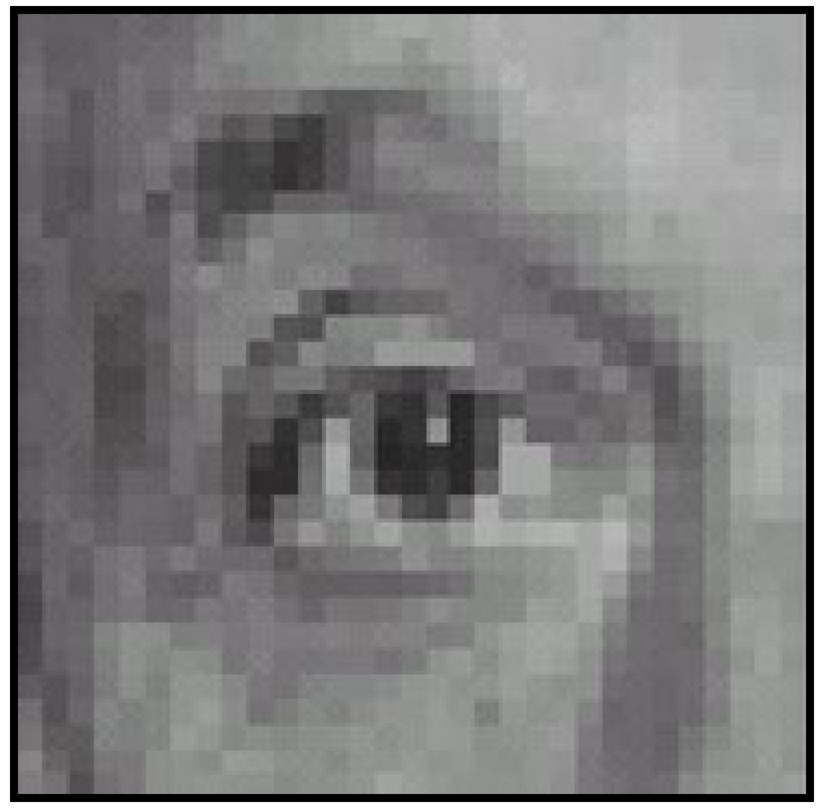
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Source: D. Lowe

Practice with linear filters



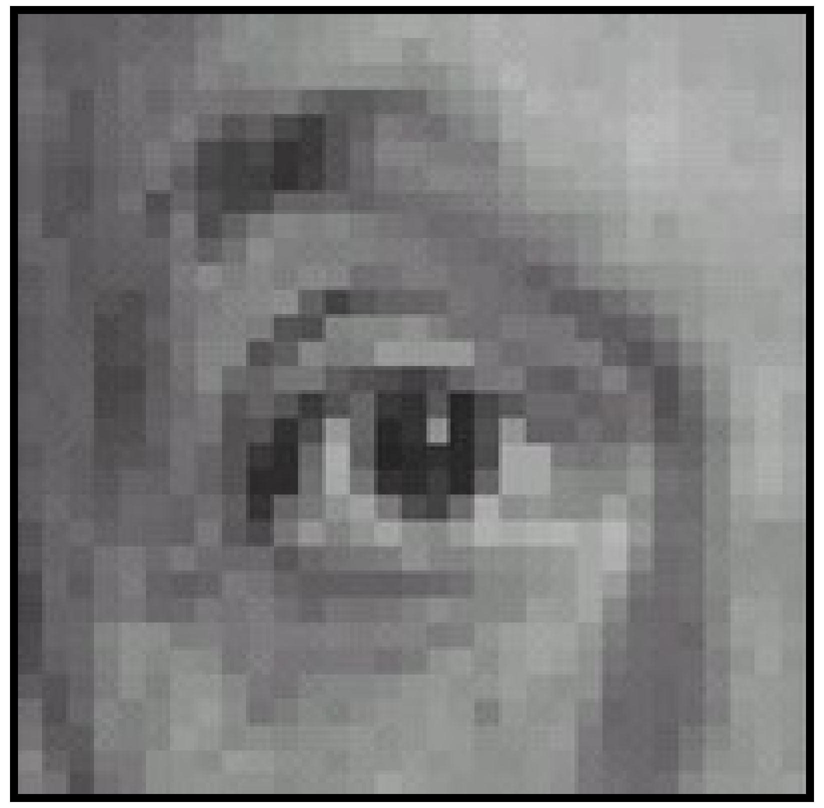
Original

0	0	0
0	0	1
0	0	0

?

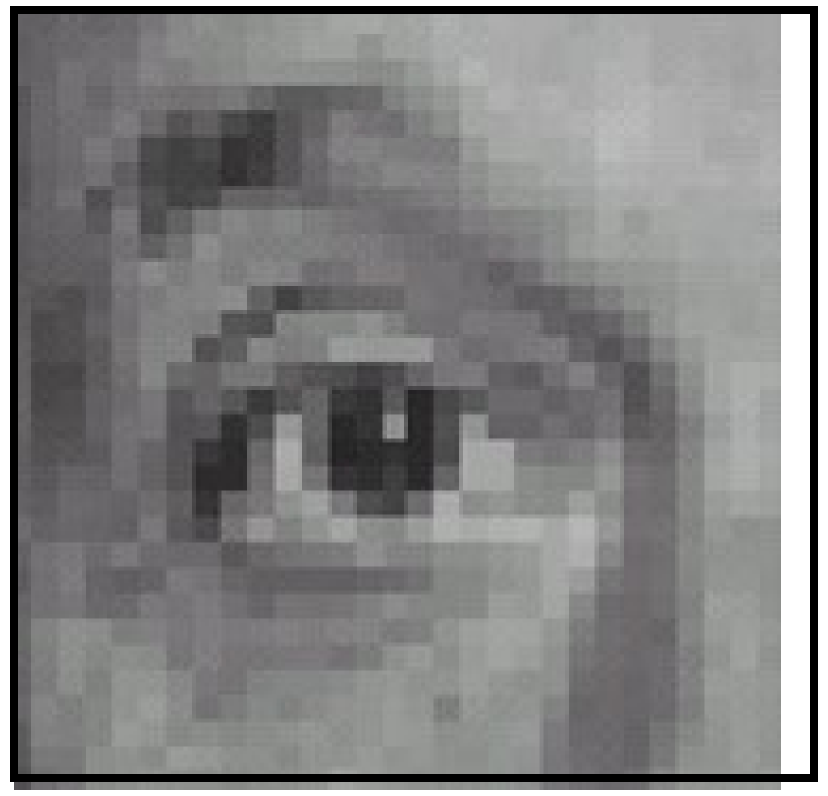
Source: D. Lowe

Practice with linear filters



Original

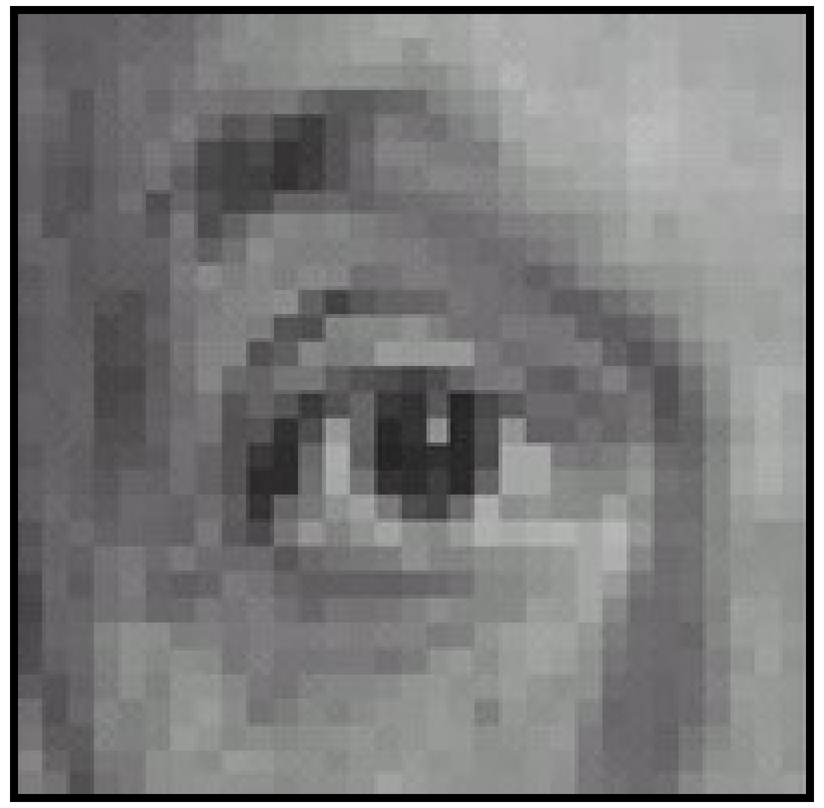
0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters



Original

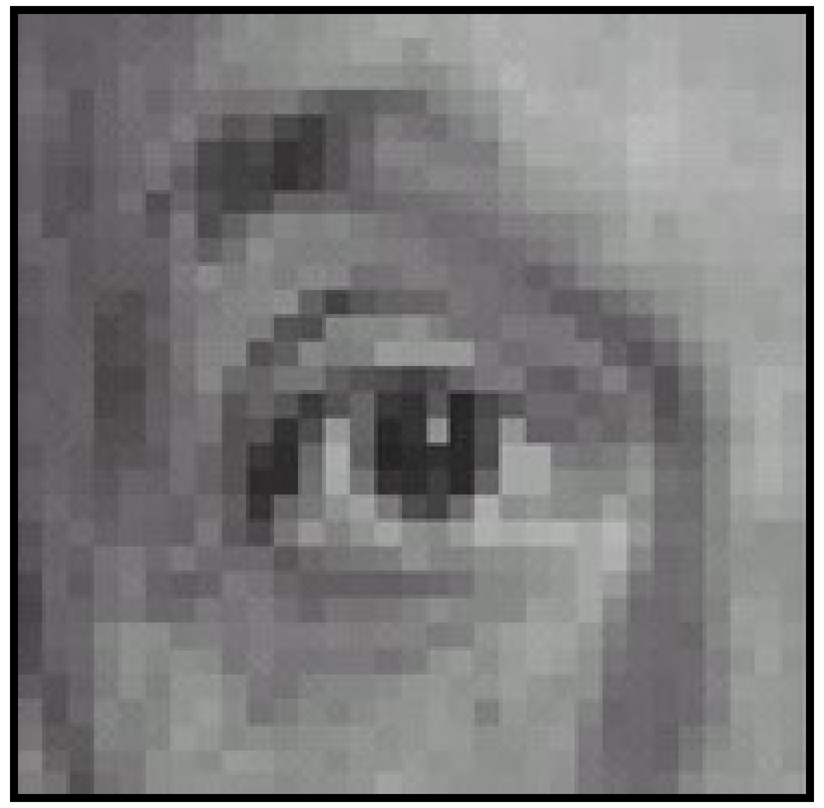
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

?

(Note that filter sums to 1)

Source: D. Lowe

Practice with linear filters



Original

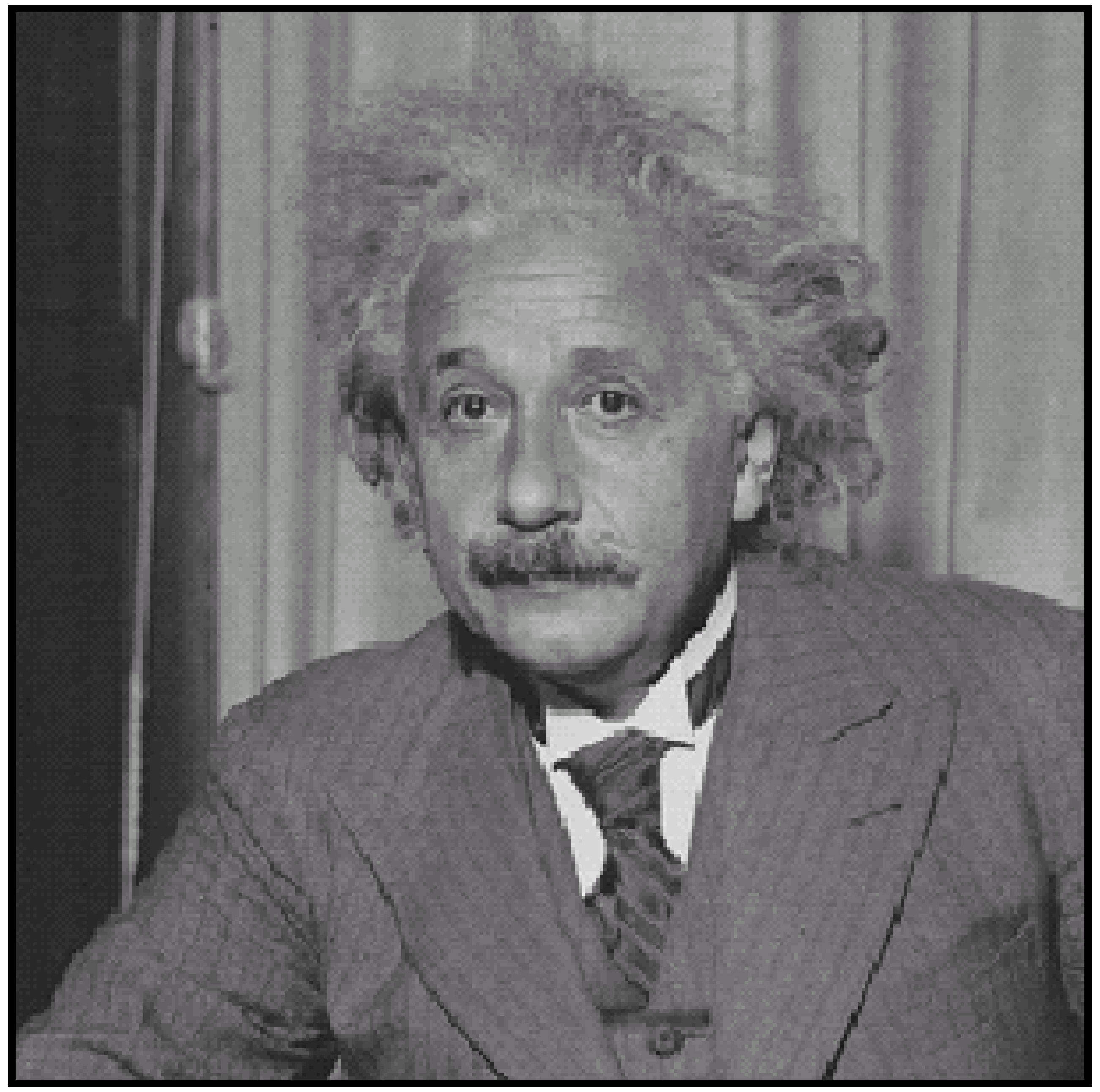
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

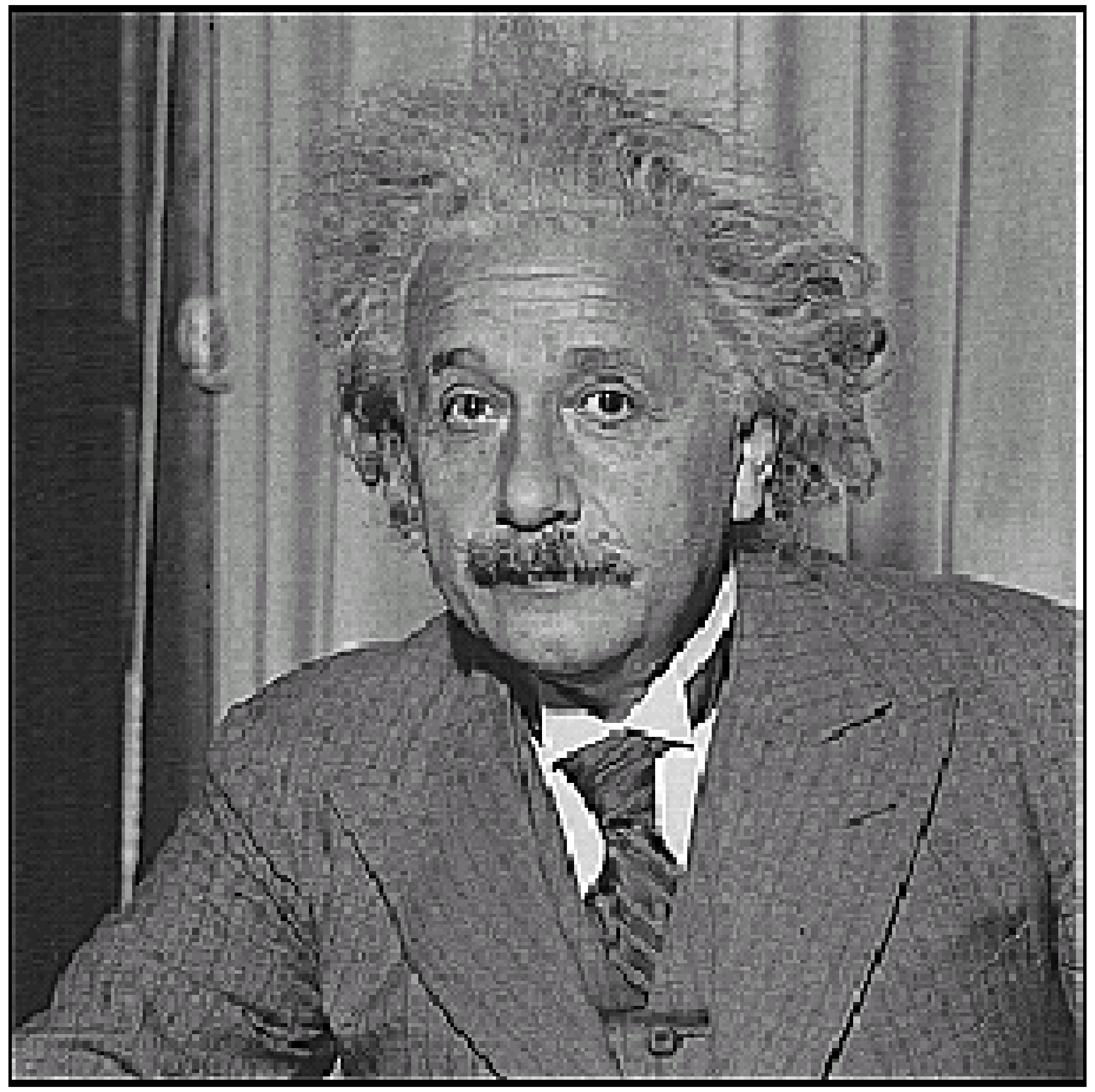
- Accentuates differences with local average

Source: D. Lowe

Sharpening



before



after

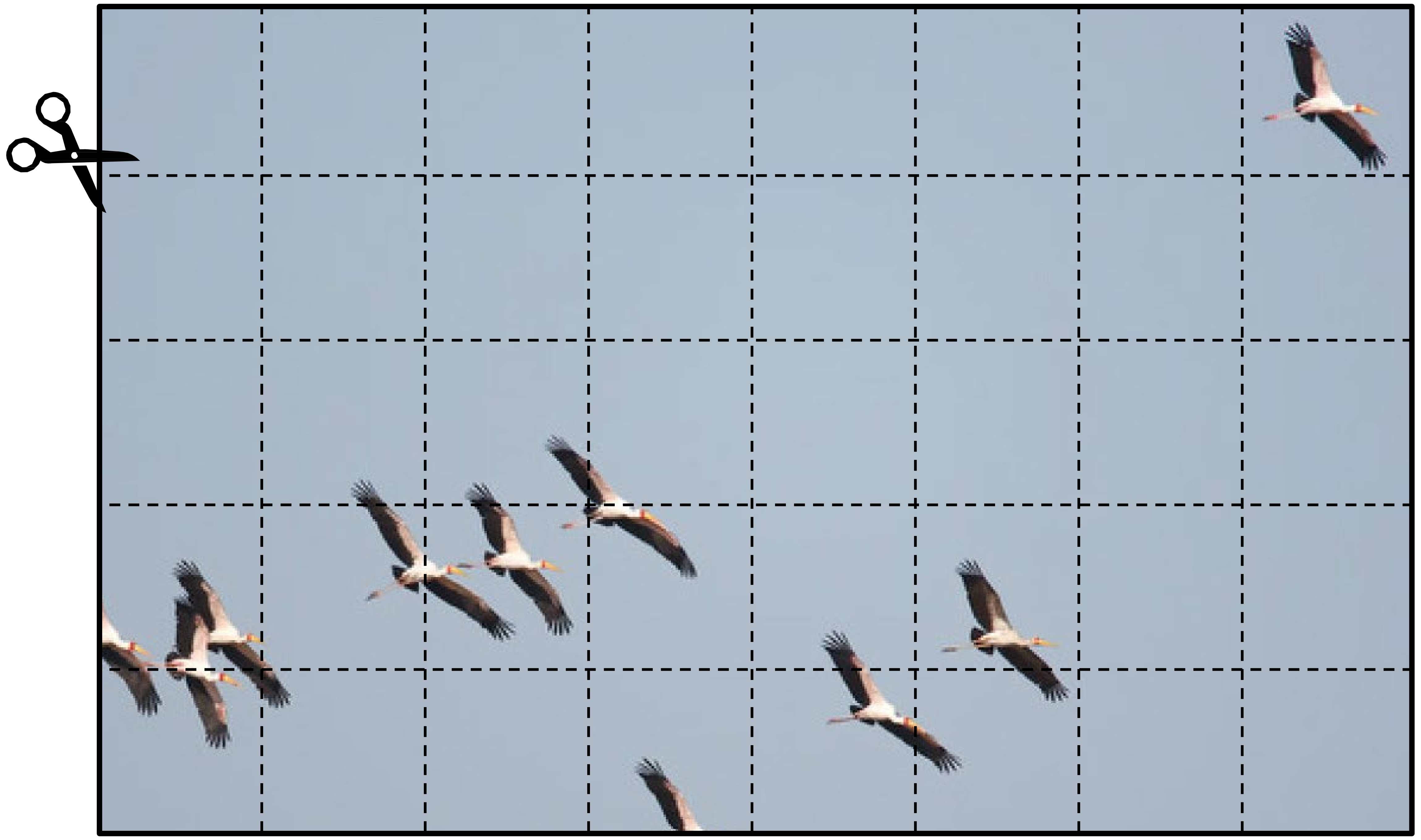
Source: D. Lowe

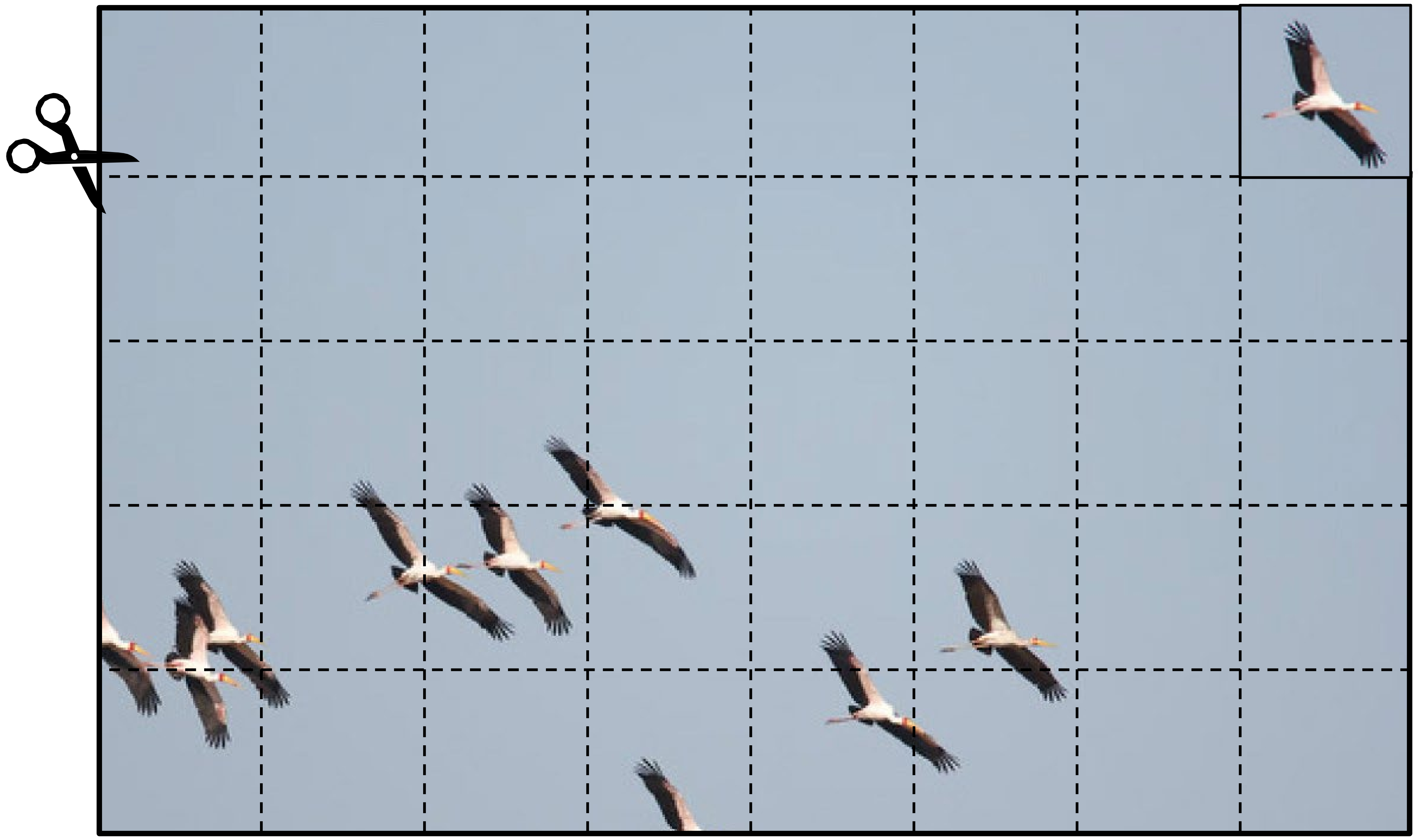
Ok now we know what a Convolution is.

Next:

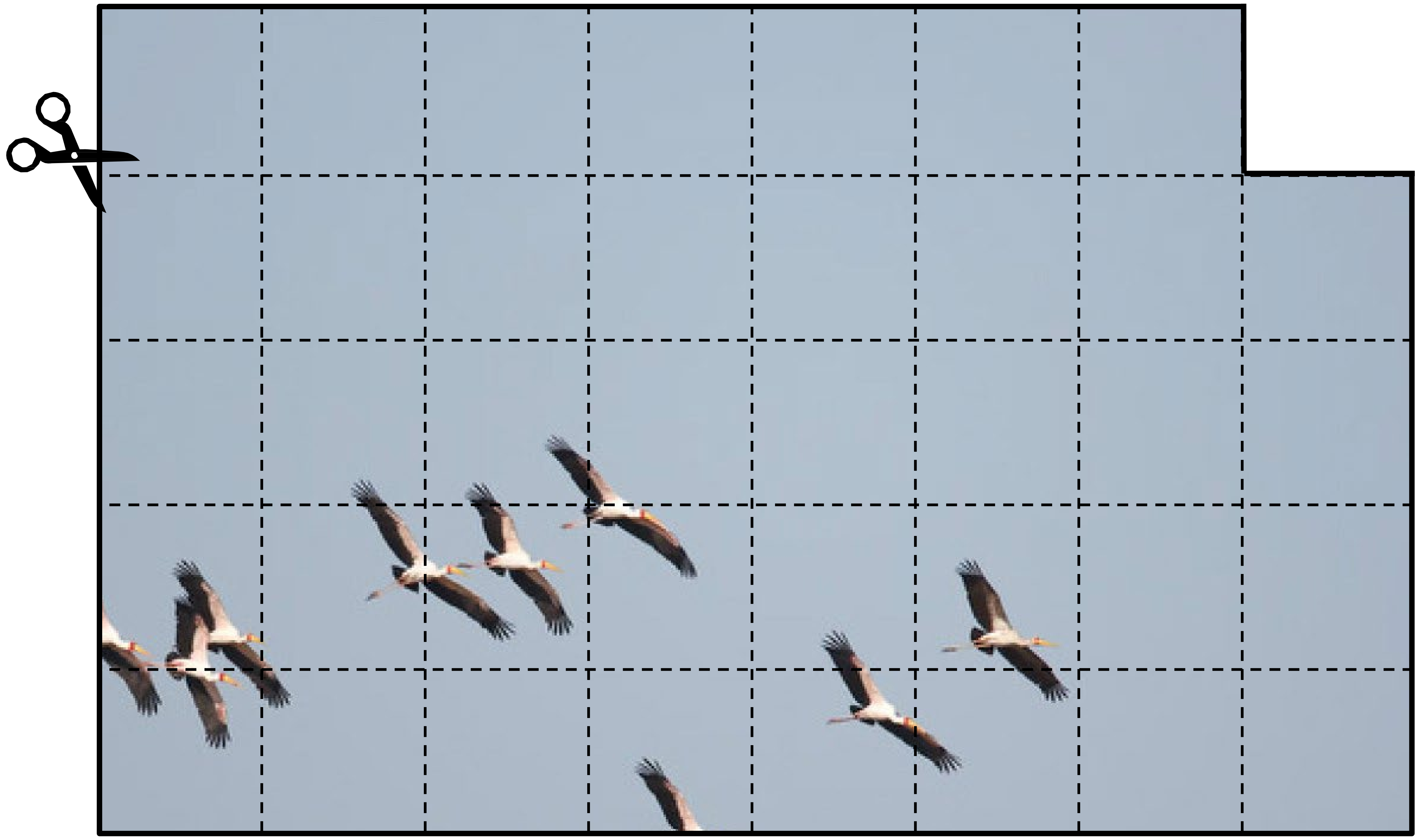
Convolutional Neural Networks

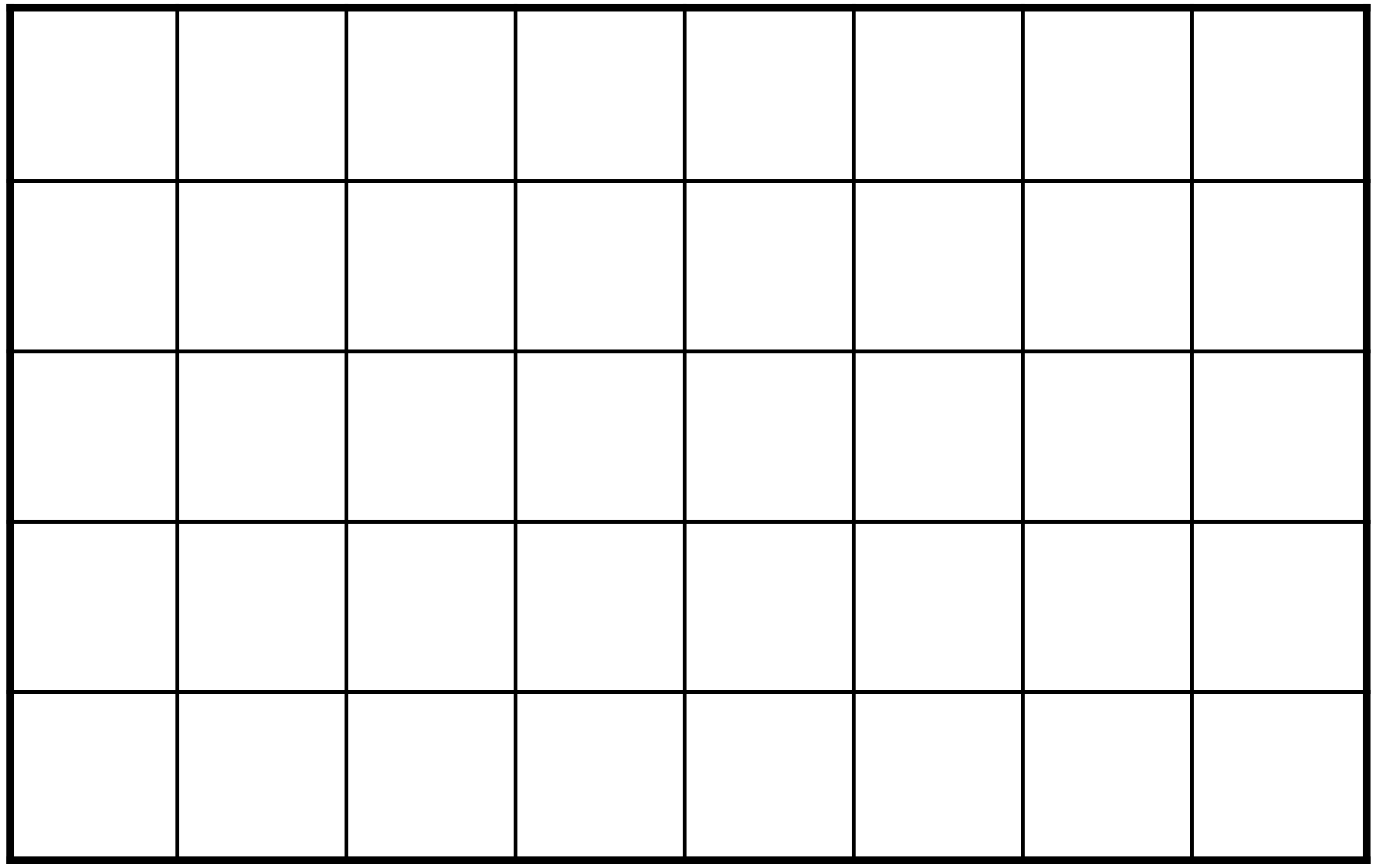
Photo credit: Fredo Durand



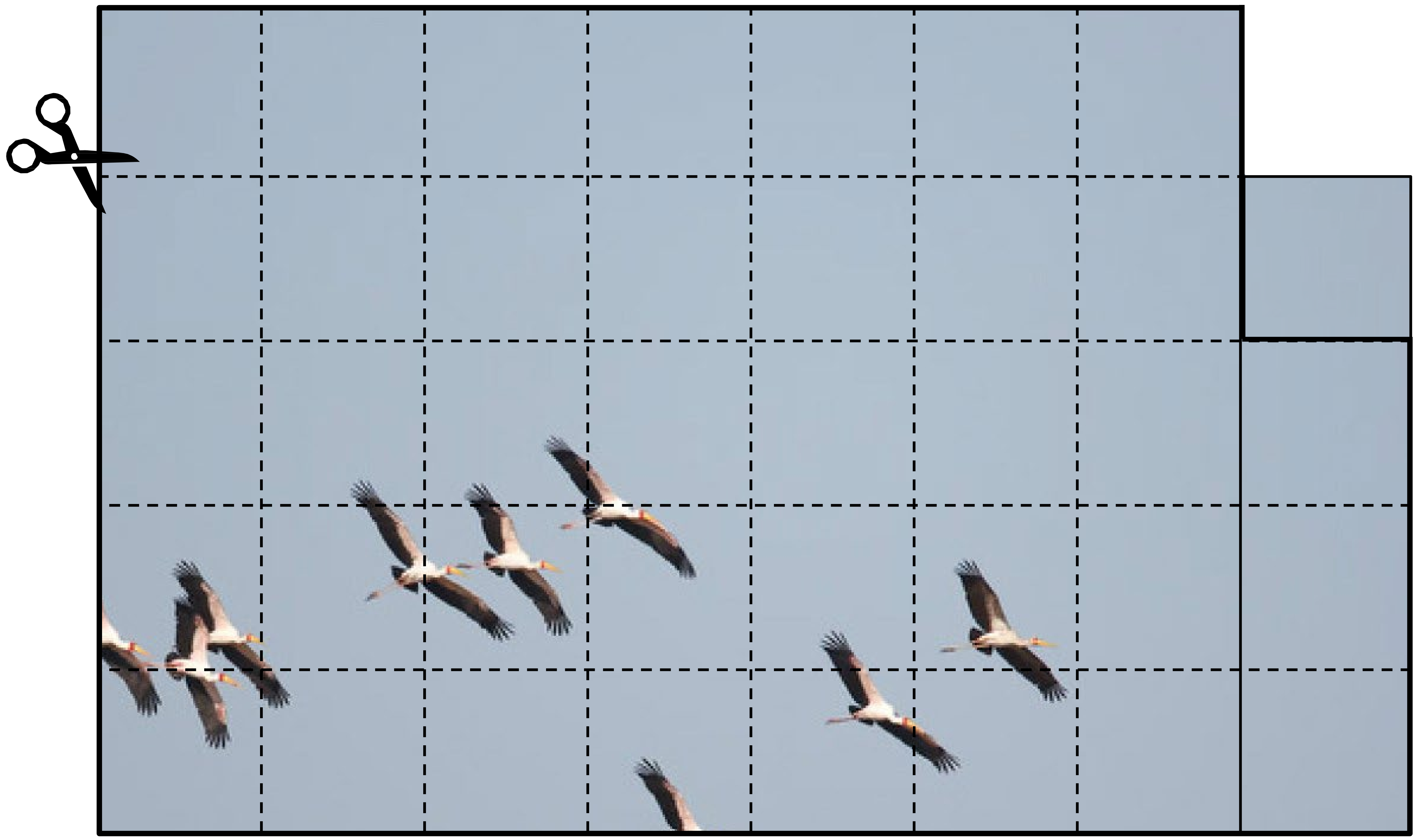


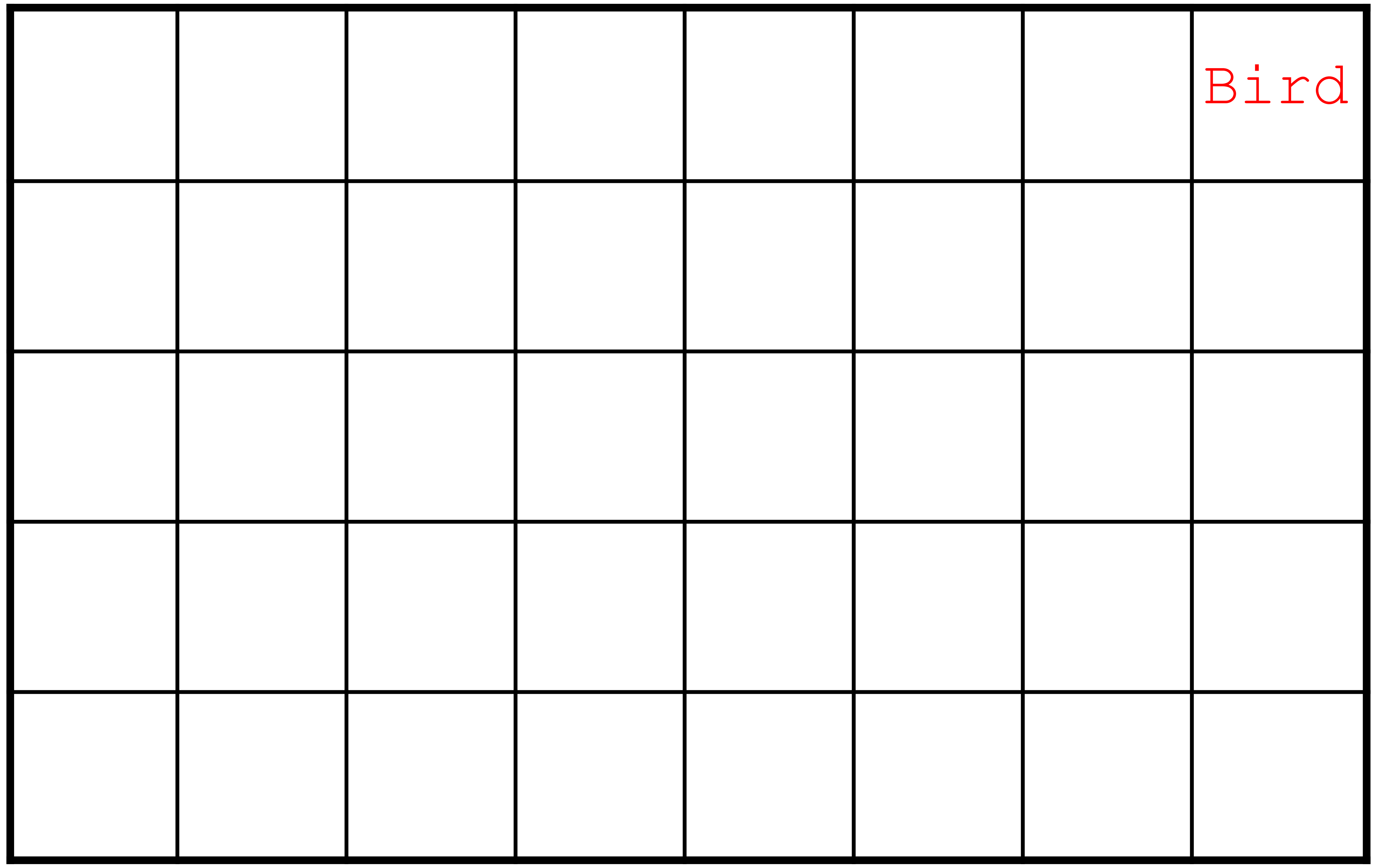

Classifier
Bird



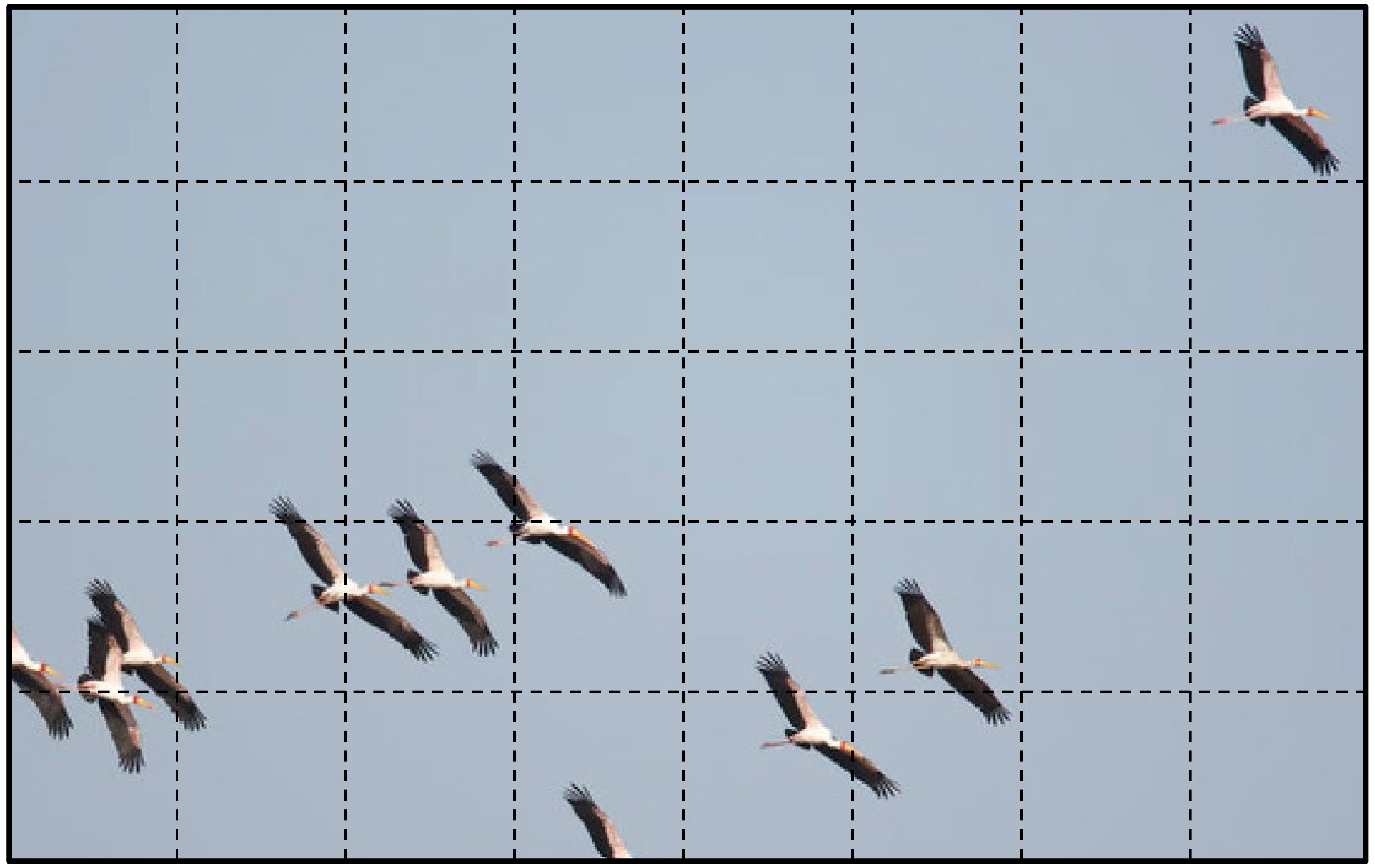



Classifier
Bird

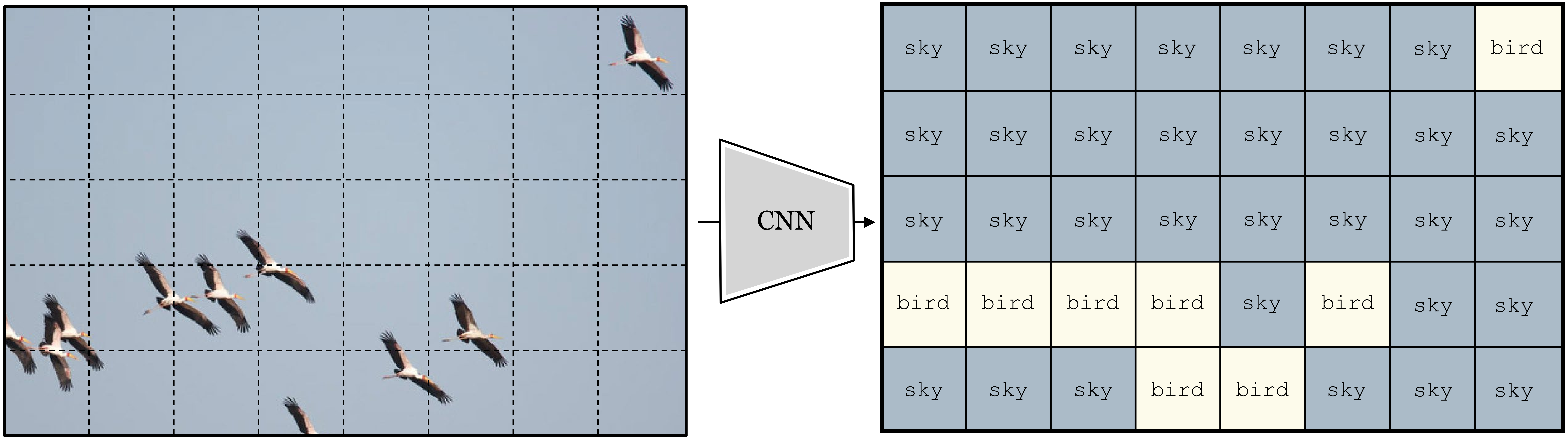



Bird


Classifier
Sky



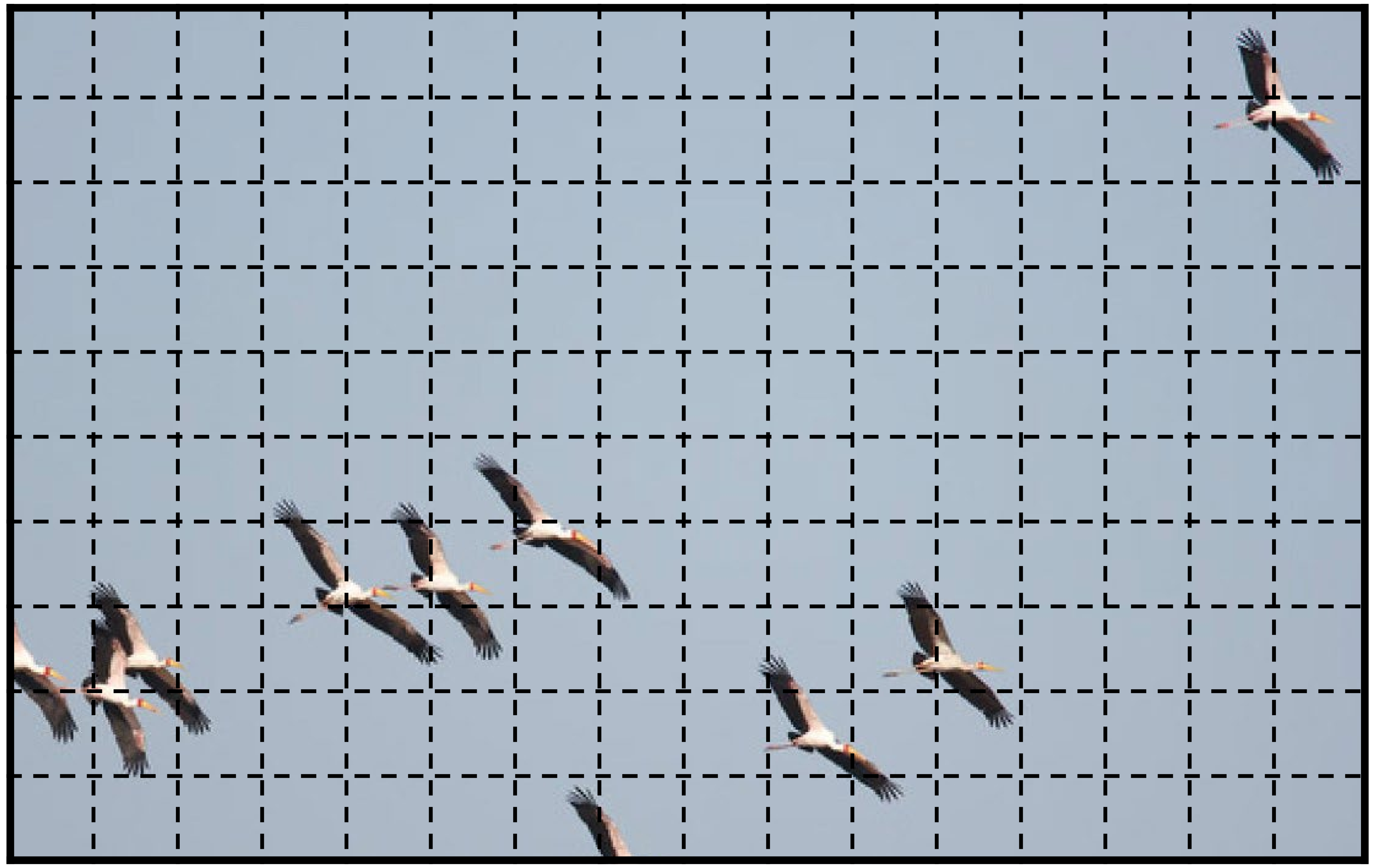
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Bird
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Sky	Sky	Sky	Sky	Sky	Sky	Sky	Sky
Bird	Bird	Bird	Sky	Bird	Sky	Sky	Sky
Sky	Sky	Sky	Bird	Sky	Sky	Sky	Sky



Problem:

What if objects don’t fit neatly into these patches?

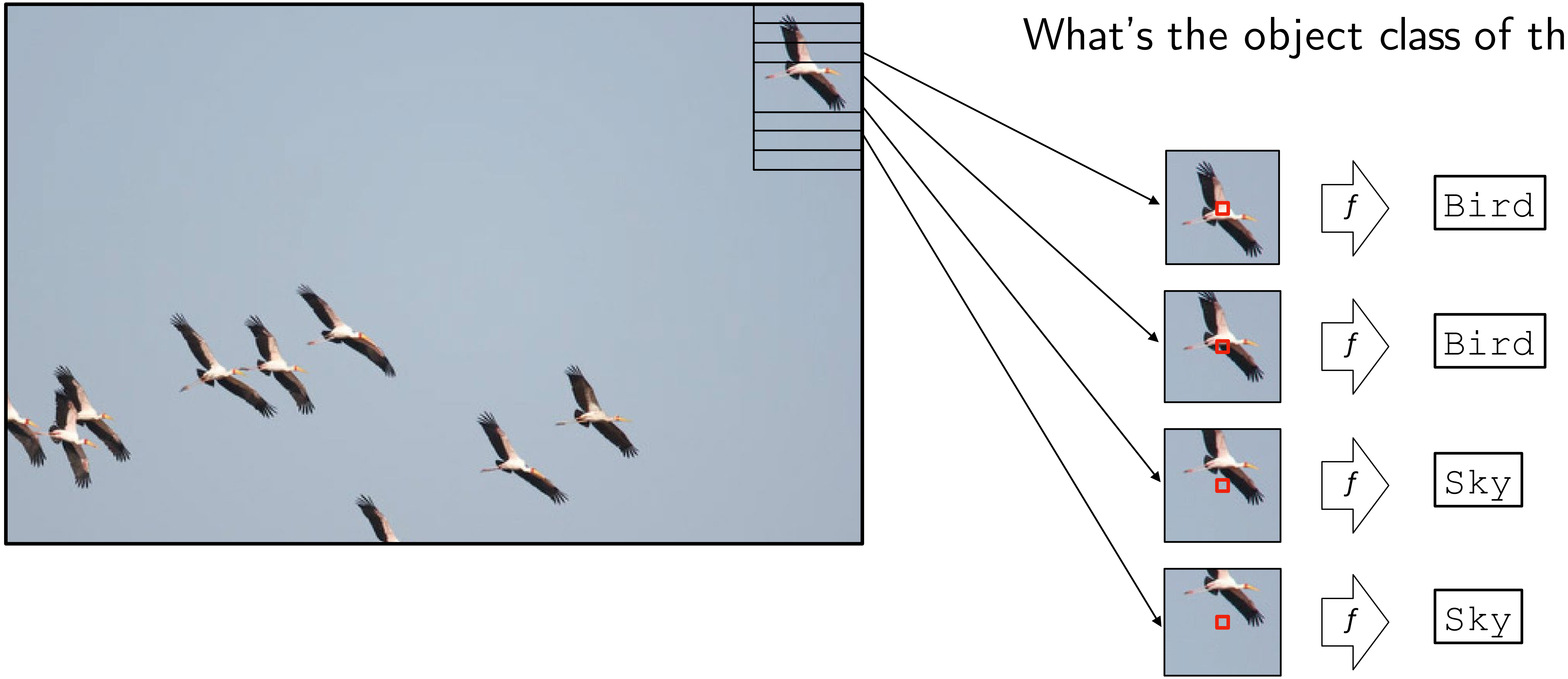
How to increase the resolution of the output map?

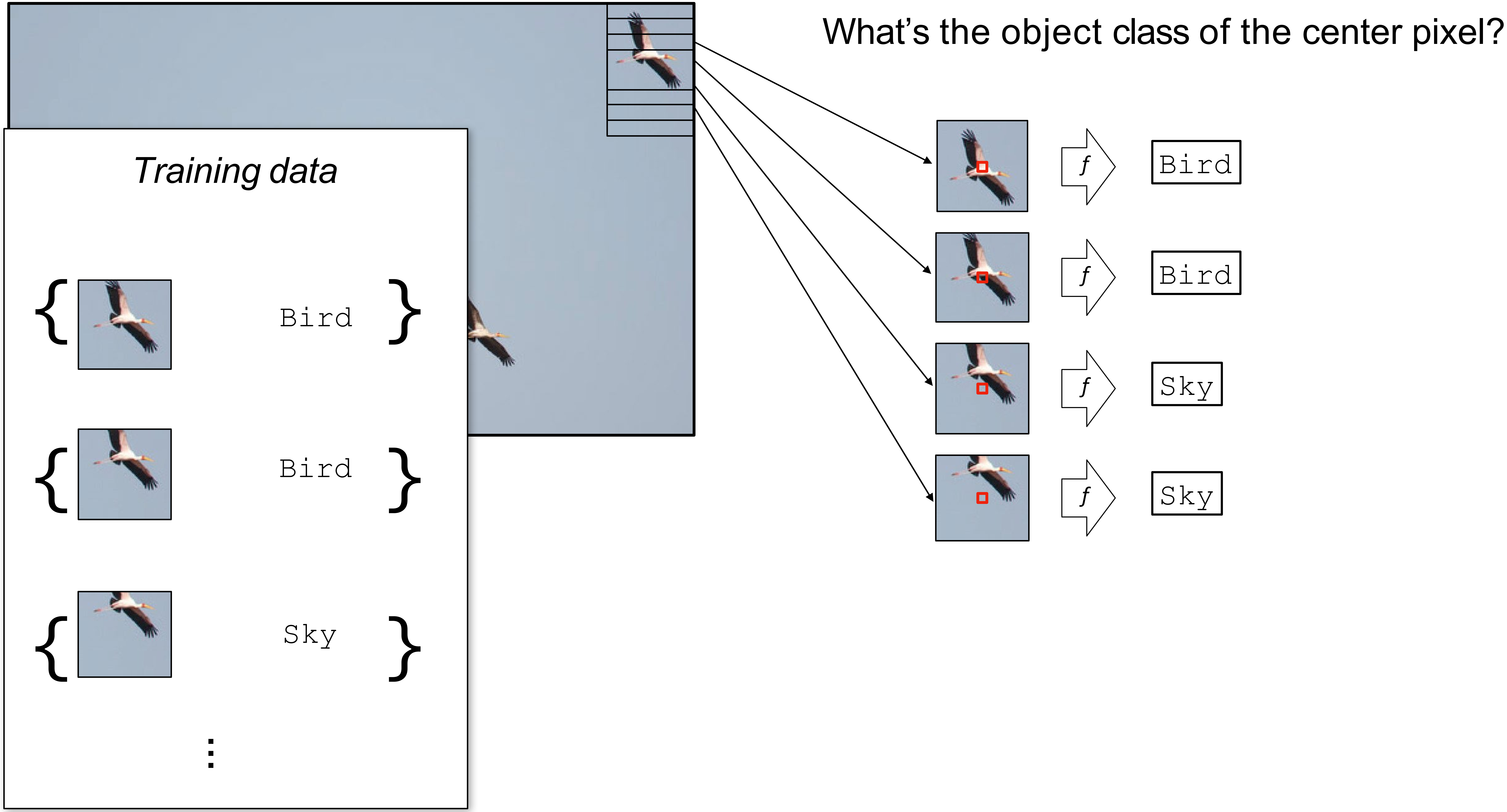


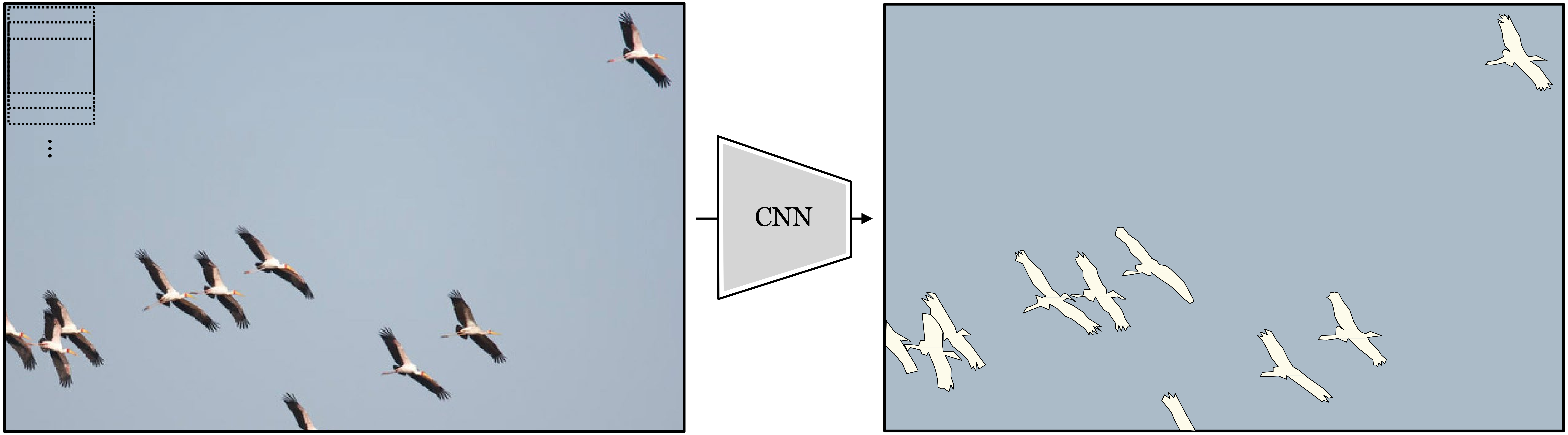
Smaller patches increase resolution
but not easy to recognize content in small each patch

Instead: we will use large but *overlapping* patches

What's the object class of the center pixel?



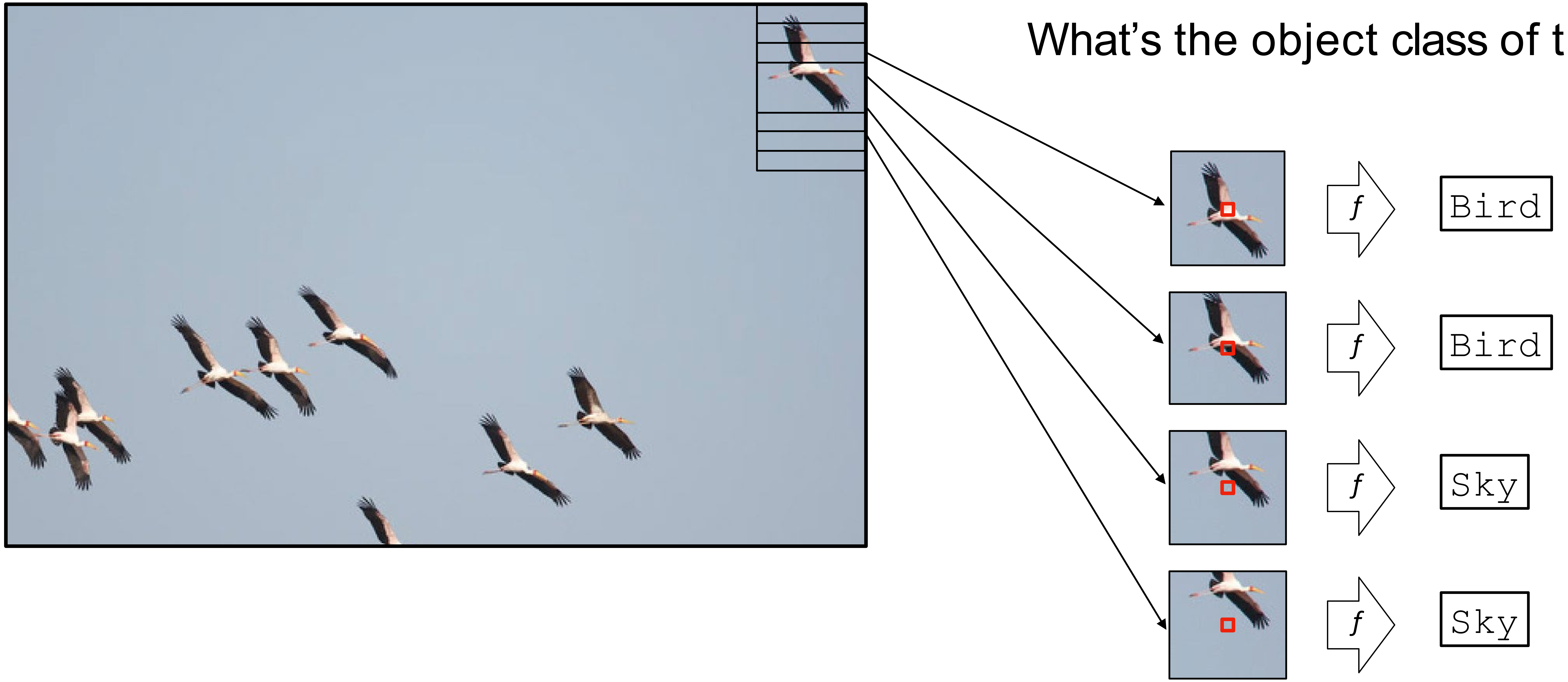

What's the object class of the center pixel?
Training data
Bird
Bird
Sky
⋮
f
Bird
f
Bird
f
Sky
f
Sky

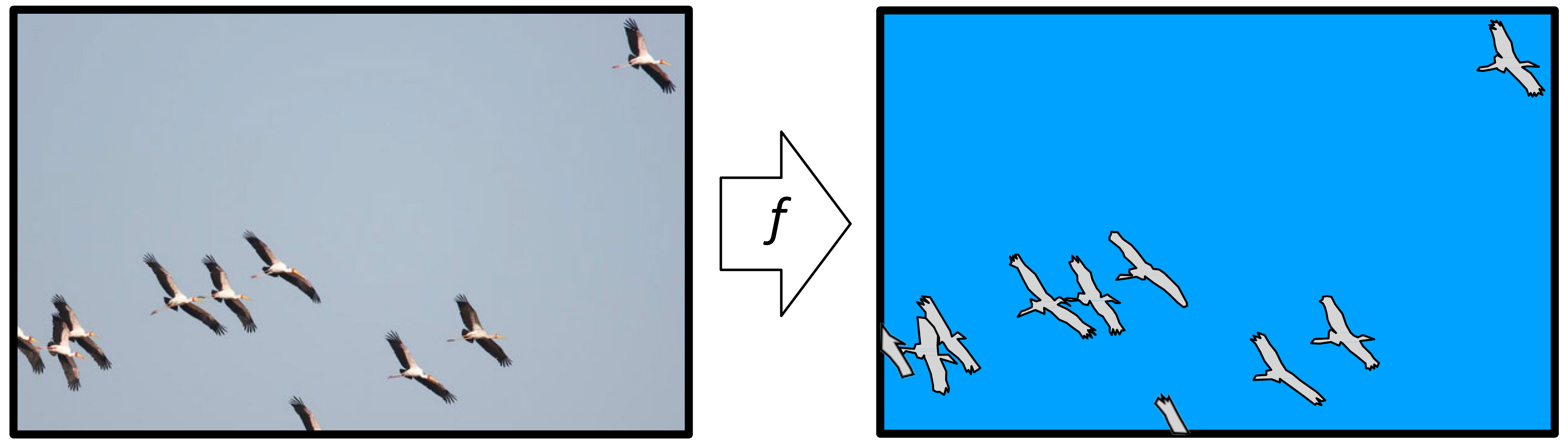


(Colors represent one-hot codes)

This problem is called **semantic segmentation**

What's the object class of the center pixel?



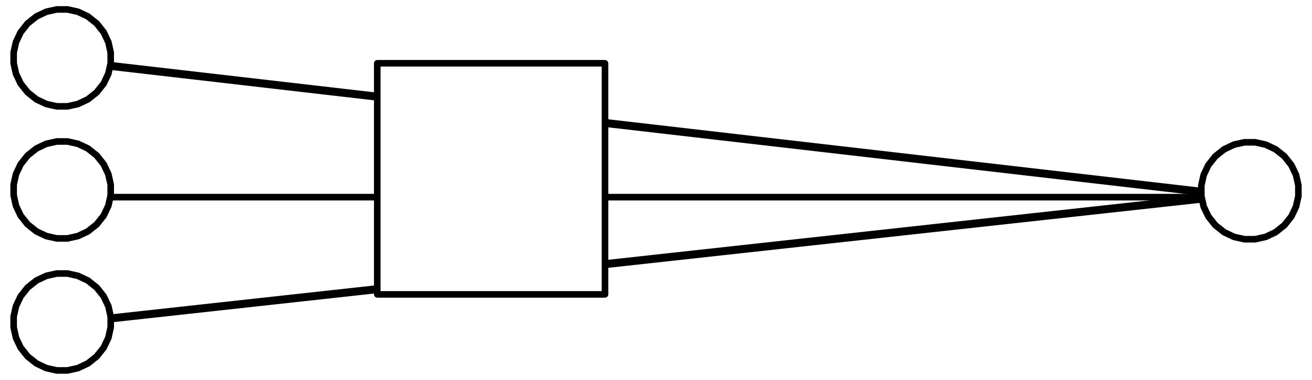


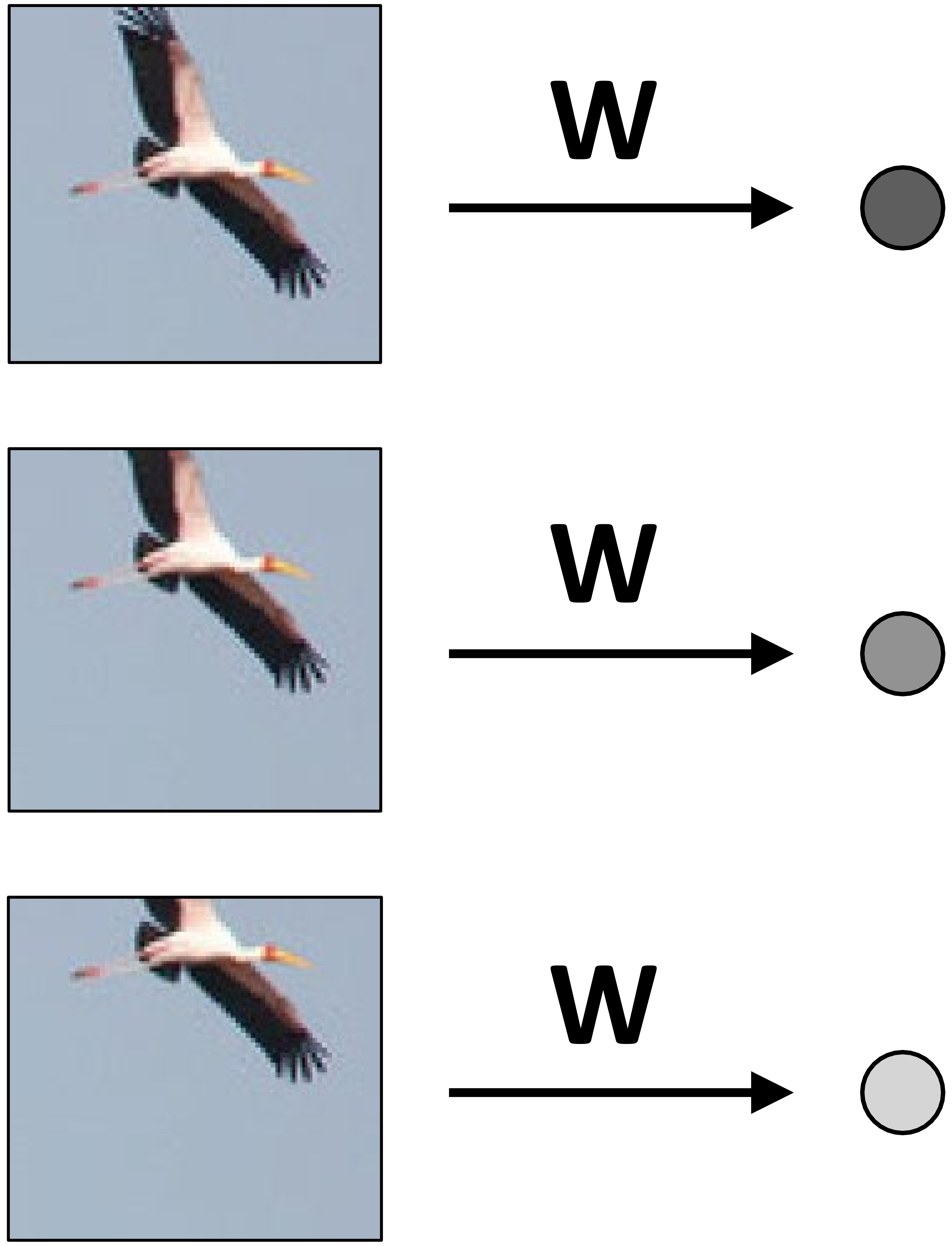
Translation invariance: process each patch in the same way.

An *equivariant* mapping:

$$f\,(\mathrm{translate}(x)) = \mathrm{translate}(f\,(x))$$

W computes a weighted sum of all pixels in the patch

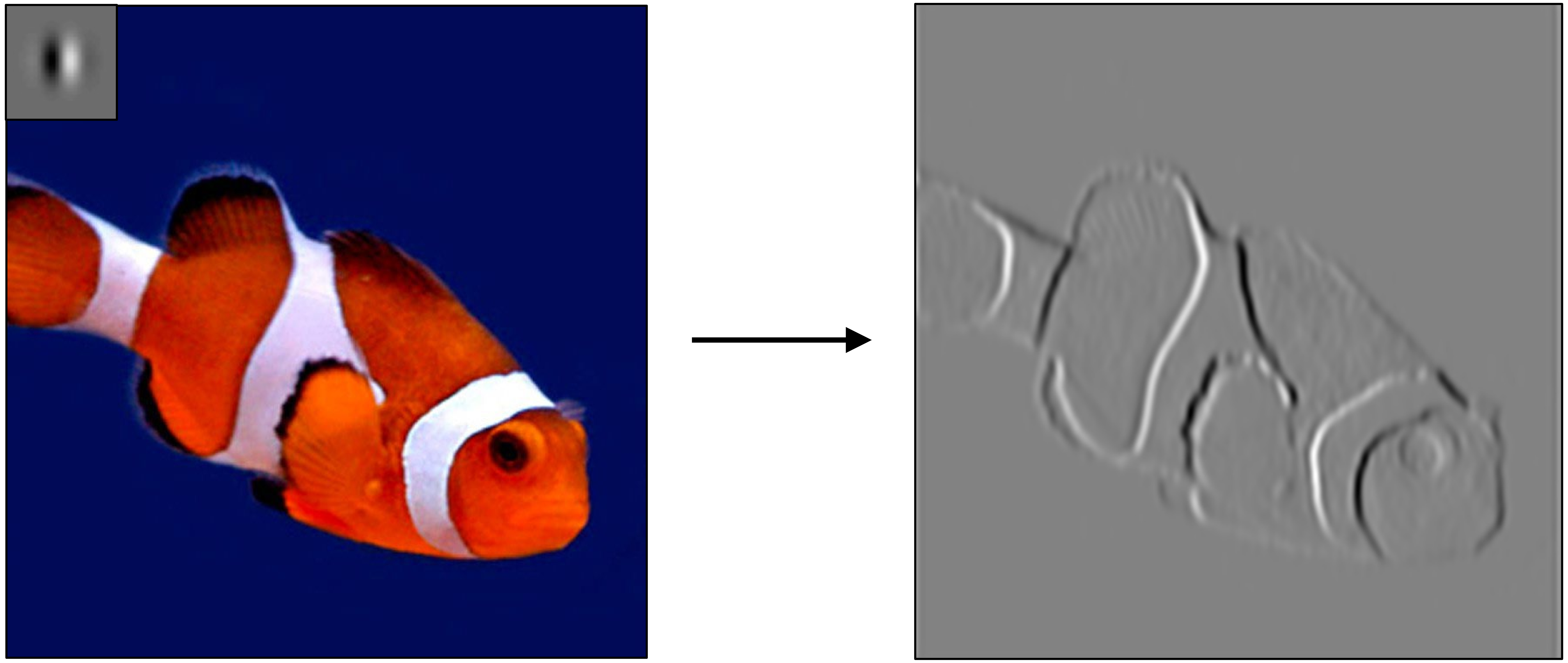


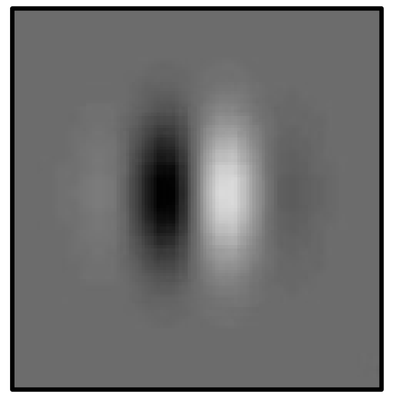


W is a **convolutional kernel** applied to the full image!

Convolution

Linear, shift-invariant transformation

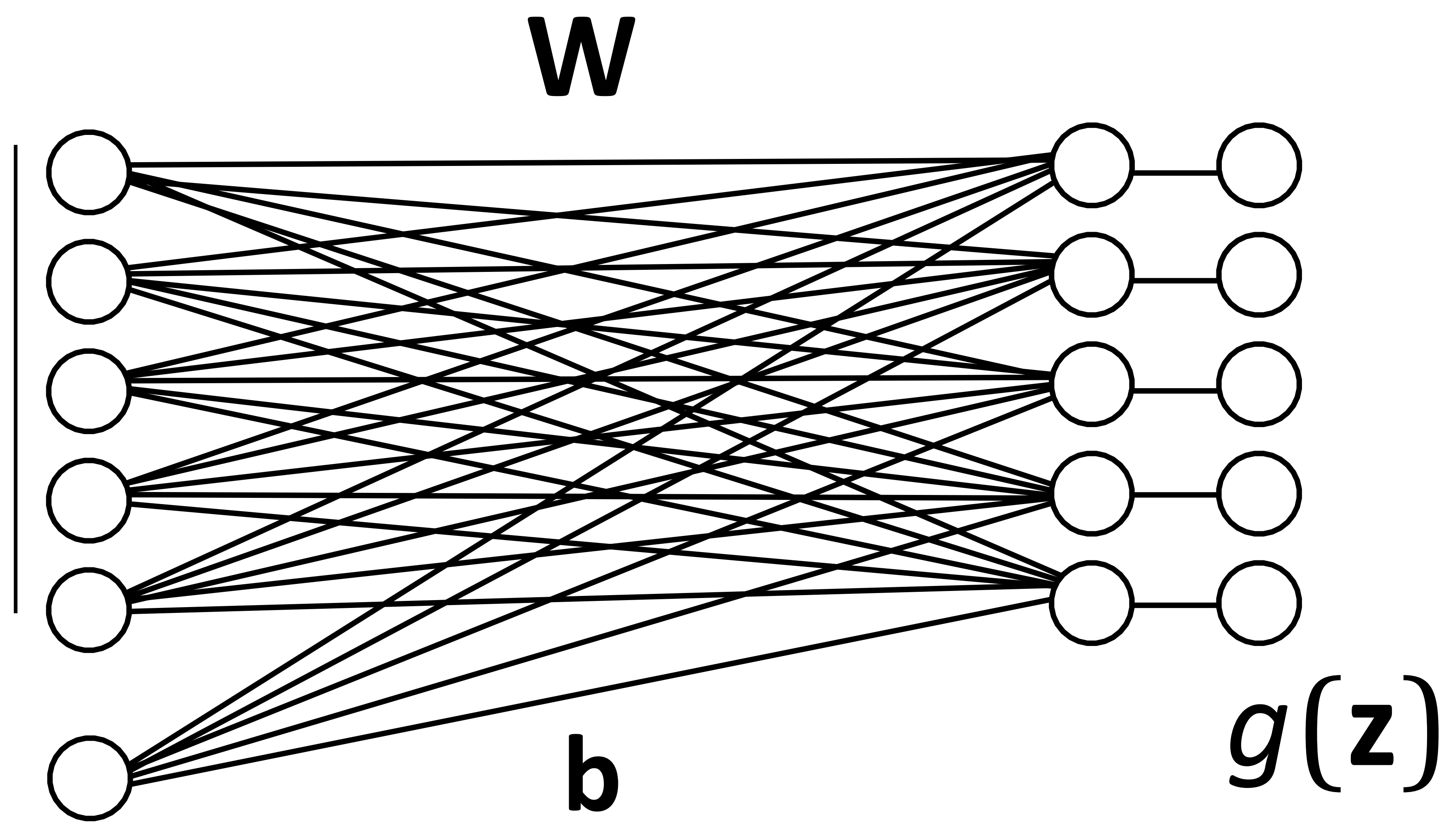


filter 

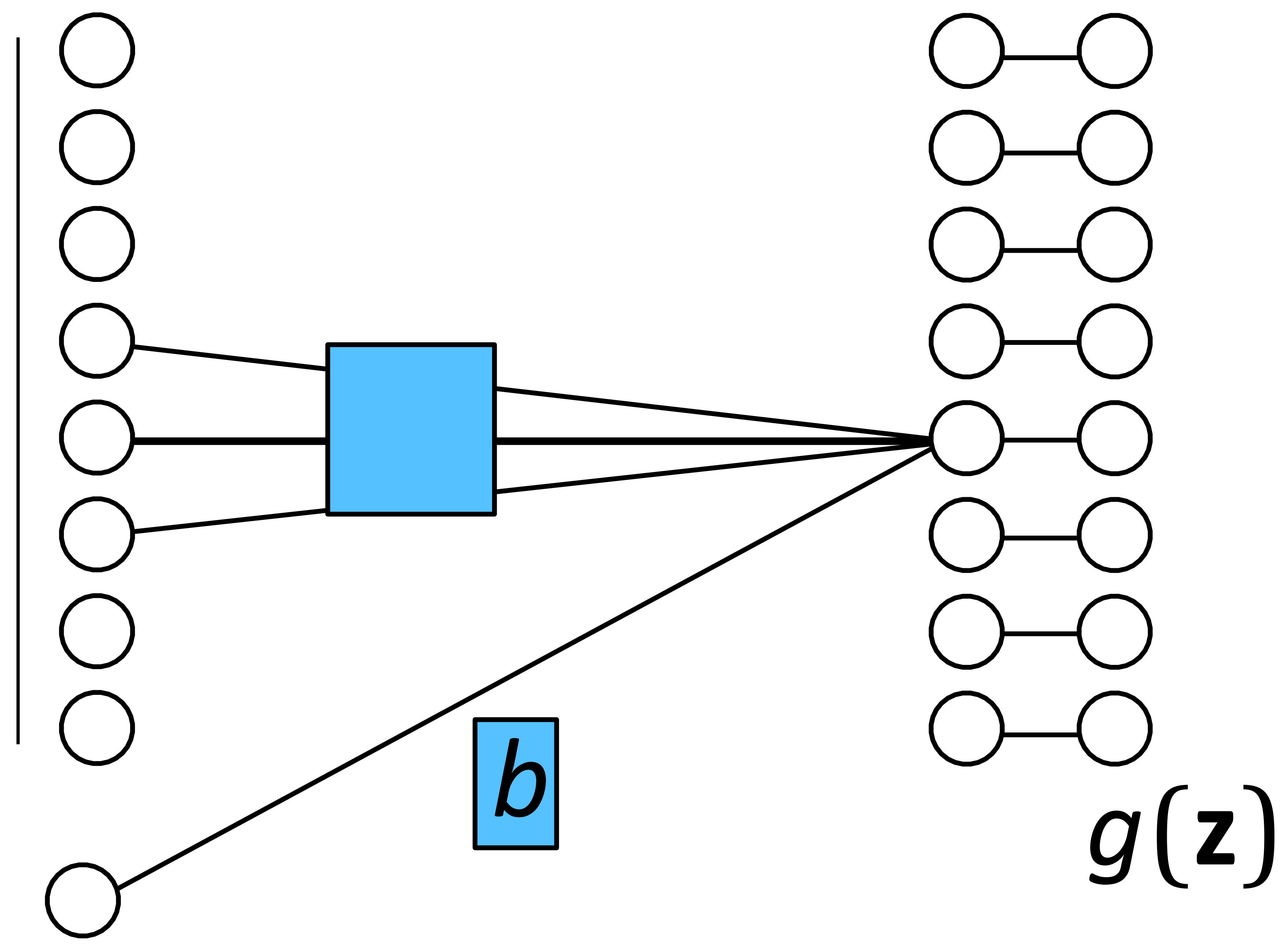
$$x_{\text{out}}[n,m] = b + \sum_{k_1,k_2=-K}^{K} w[k_1,k_2]\, x_{\text{in}}[n+k_1, m+k_2]$$

Fully-connected network

Fully-connected (fc) layer



Locally connected network

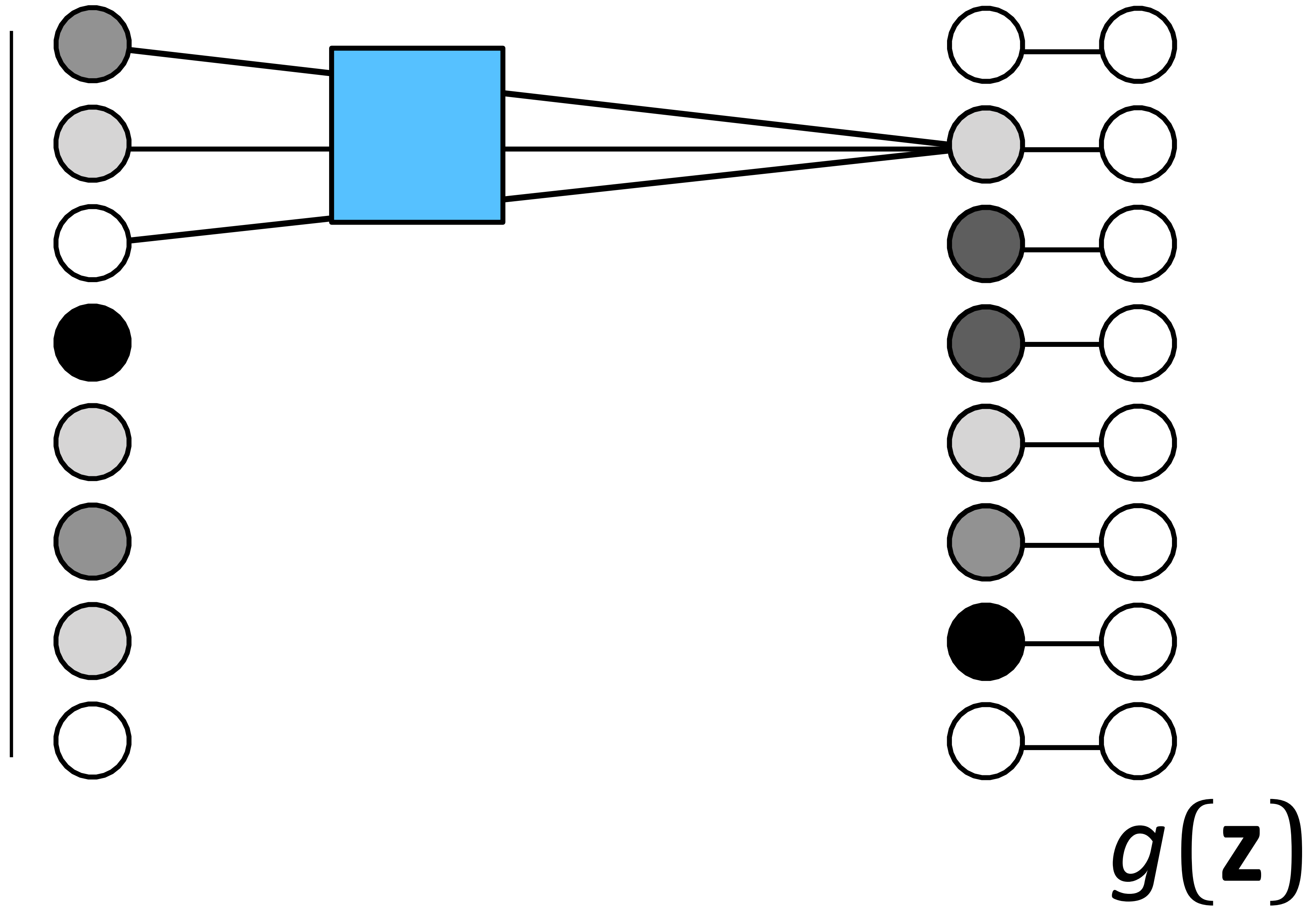


Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Convolutional neural network

Conv layer



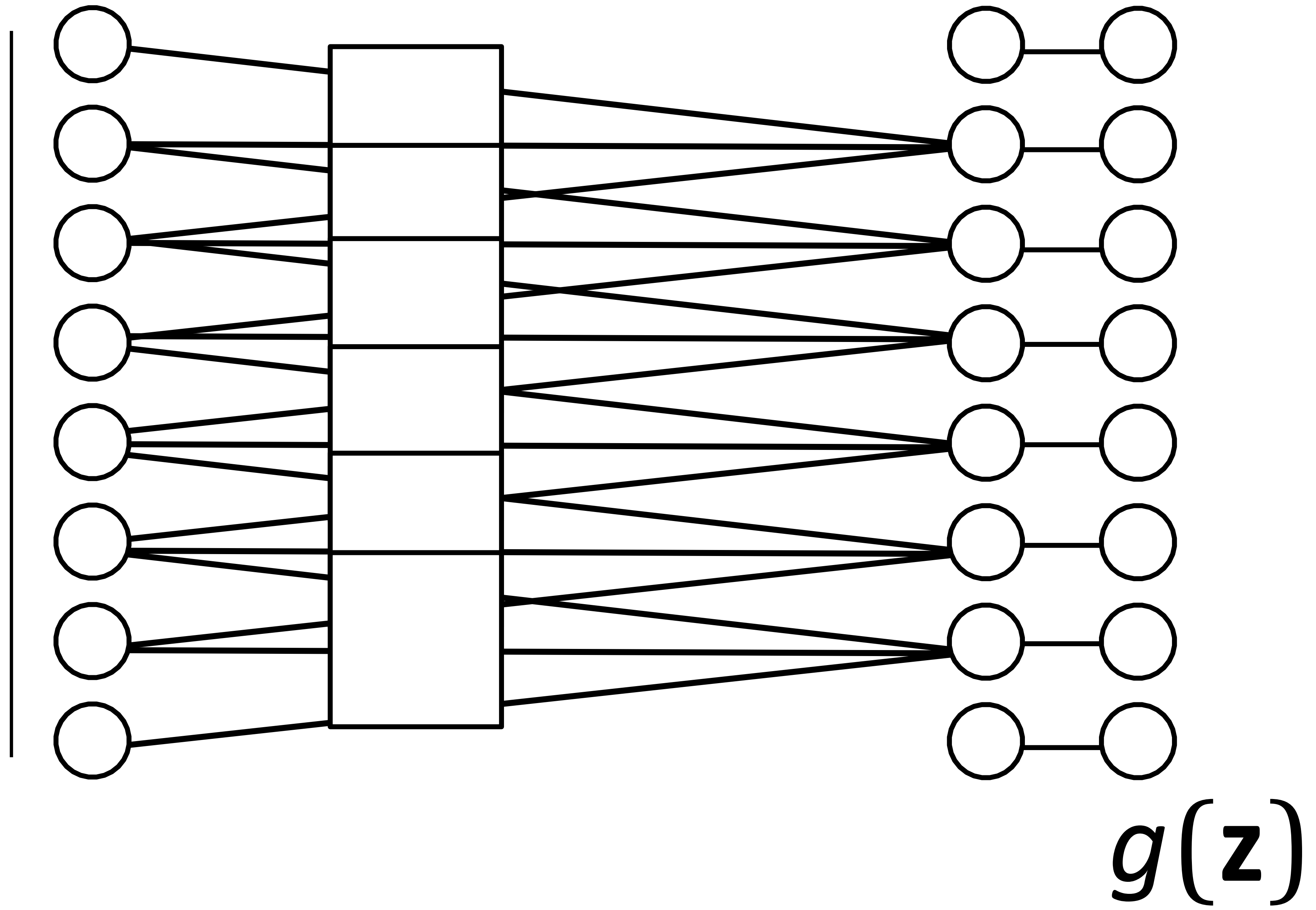
$$\mathbf{z} = \mathbf{w} \;\; \mathbf{x} + b$$

Often, we assume output is a **local** function of input.

If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

Weight sharing

Conv layer



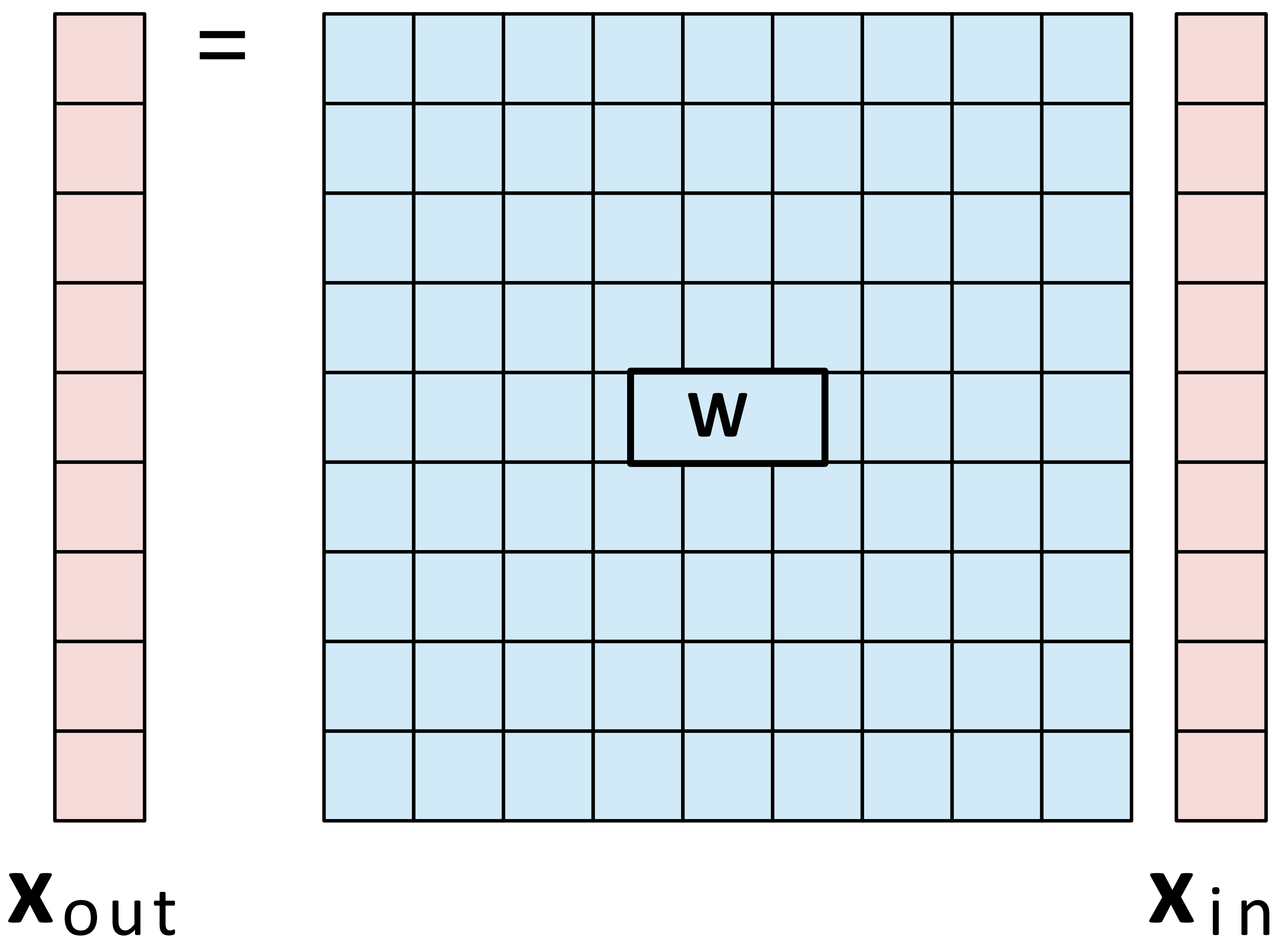
$$\mathbf{z} = \mathbf{w} \ \mathbf{x} + b$$

Often, we assume output is a **local** function of input.

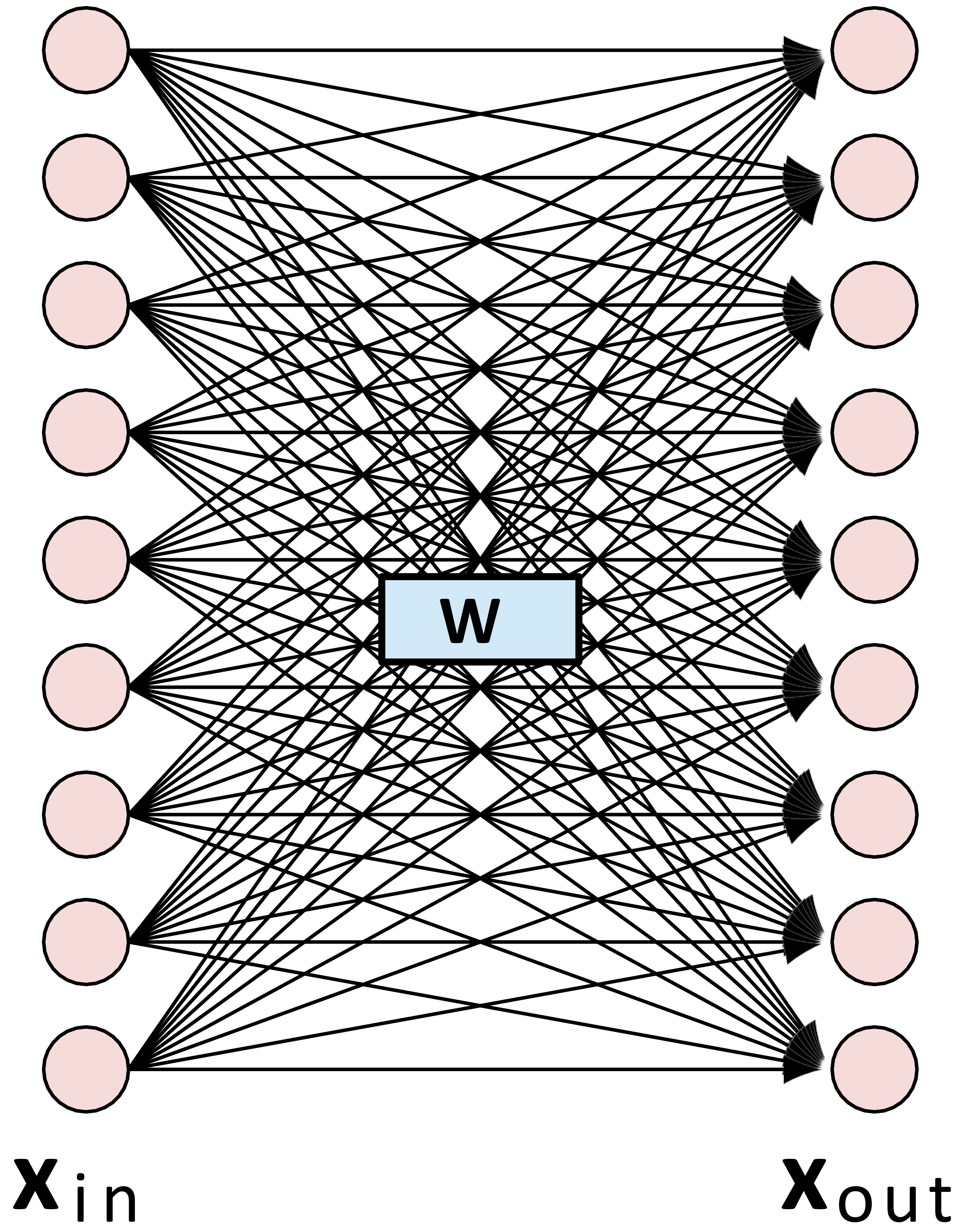
If we use the same weights (**weight sharing**) to compute each local function, we get a convolutional neural network.

(Fully-connected) linear layer

$$\mathbf{x}_{out} = \mathbf{W}\mathbf{x}_{in} + \mathbf{b}$$

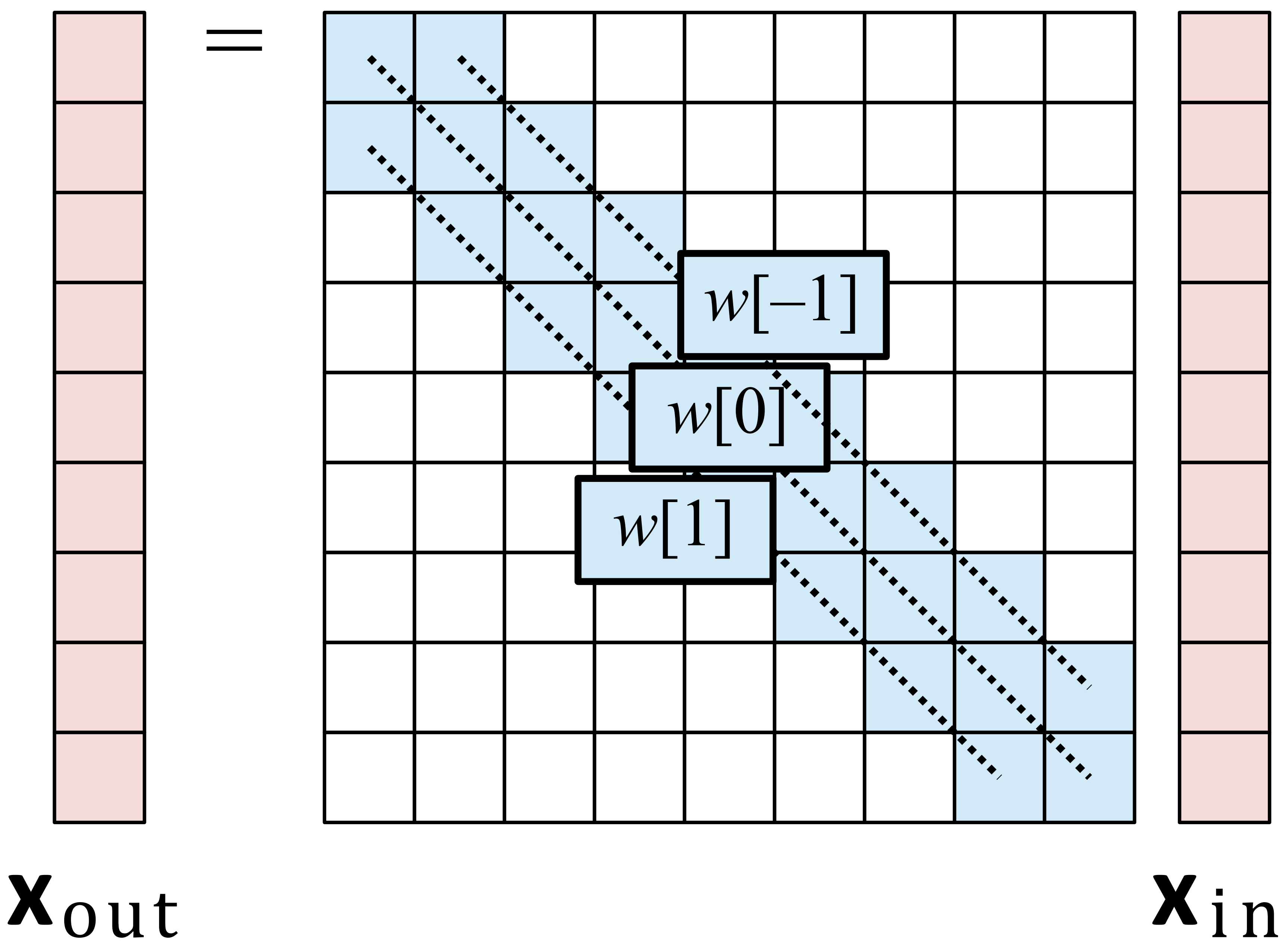


( )

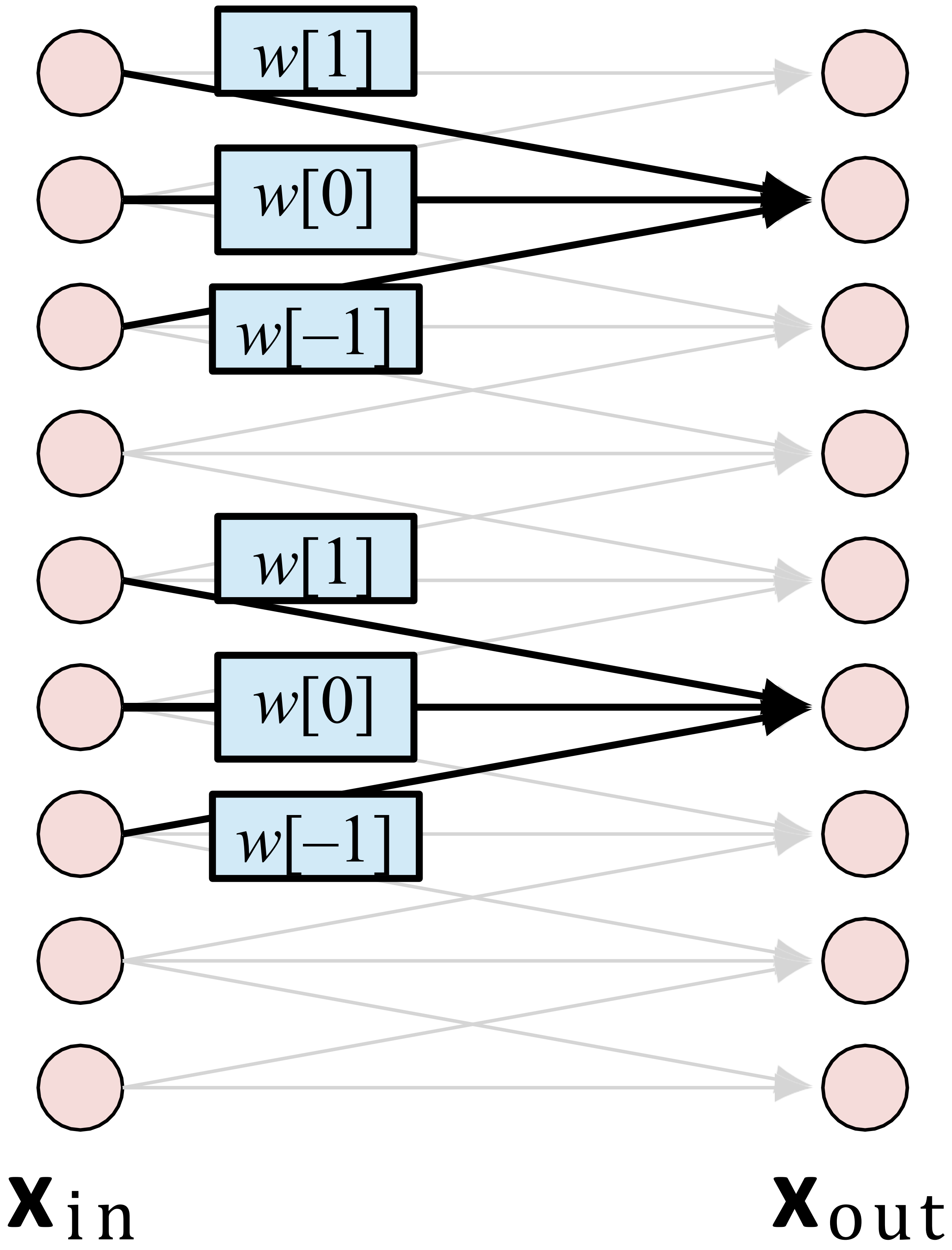


Convolutional layer

$$\mathbf{x}_{out} = \mathbf{w} \; \mathbf{x}_{in} \quad + b$$



( )



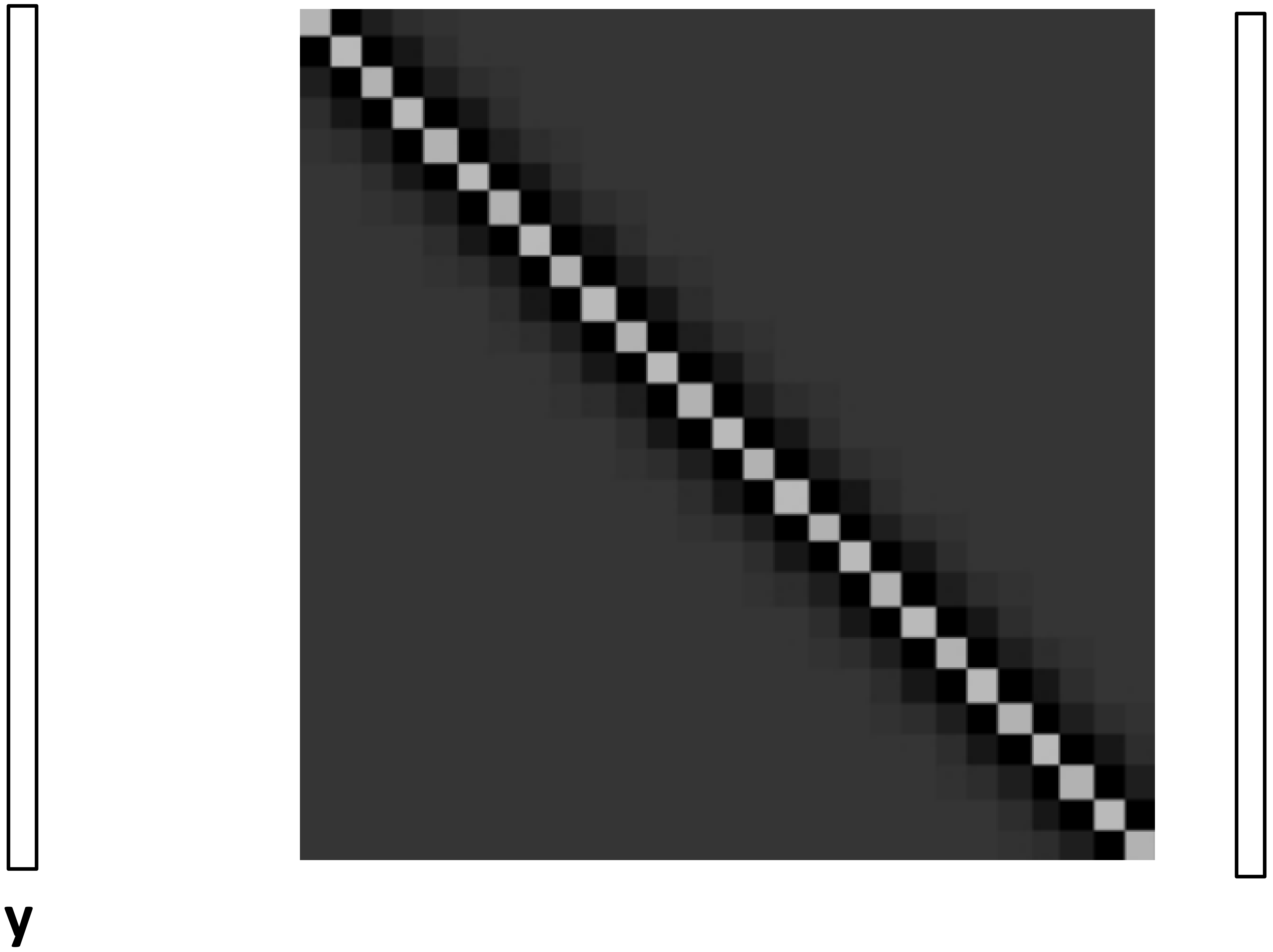
Toeplitz matrix

$$\begin{pmatrix} a & b & c & d & e \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ i & h & g & f & a \end{pmatrix}$$

e.g., pixel image

- Constrained linear layer
- Fewer parameters —> easier to learn, less overfitting

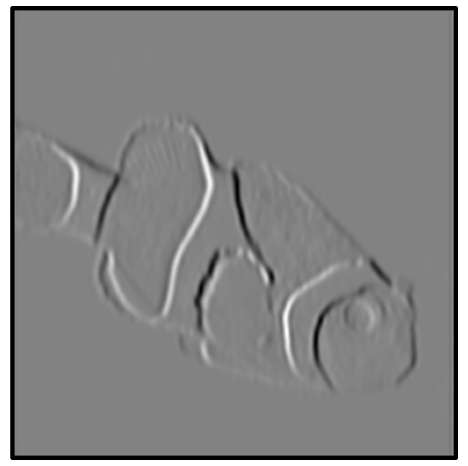
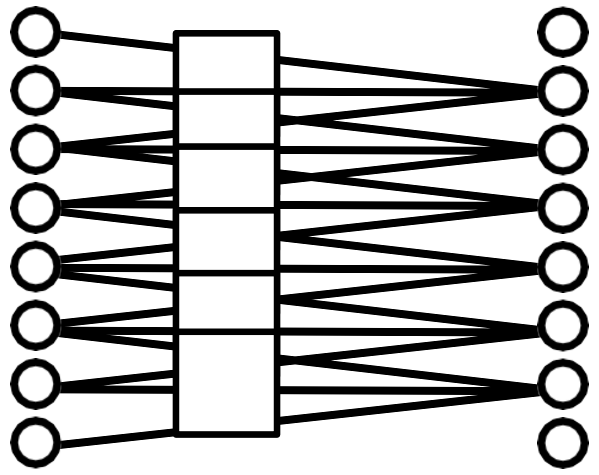

y



Conv layers can be applied to arbitrarily-sized inputs

(generalizes beyond the training data due to an architectural structure!)

Five views on convolutional layers

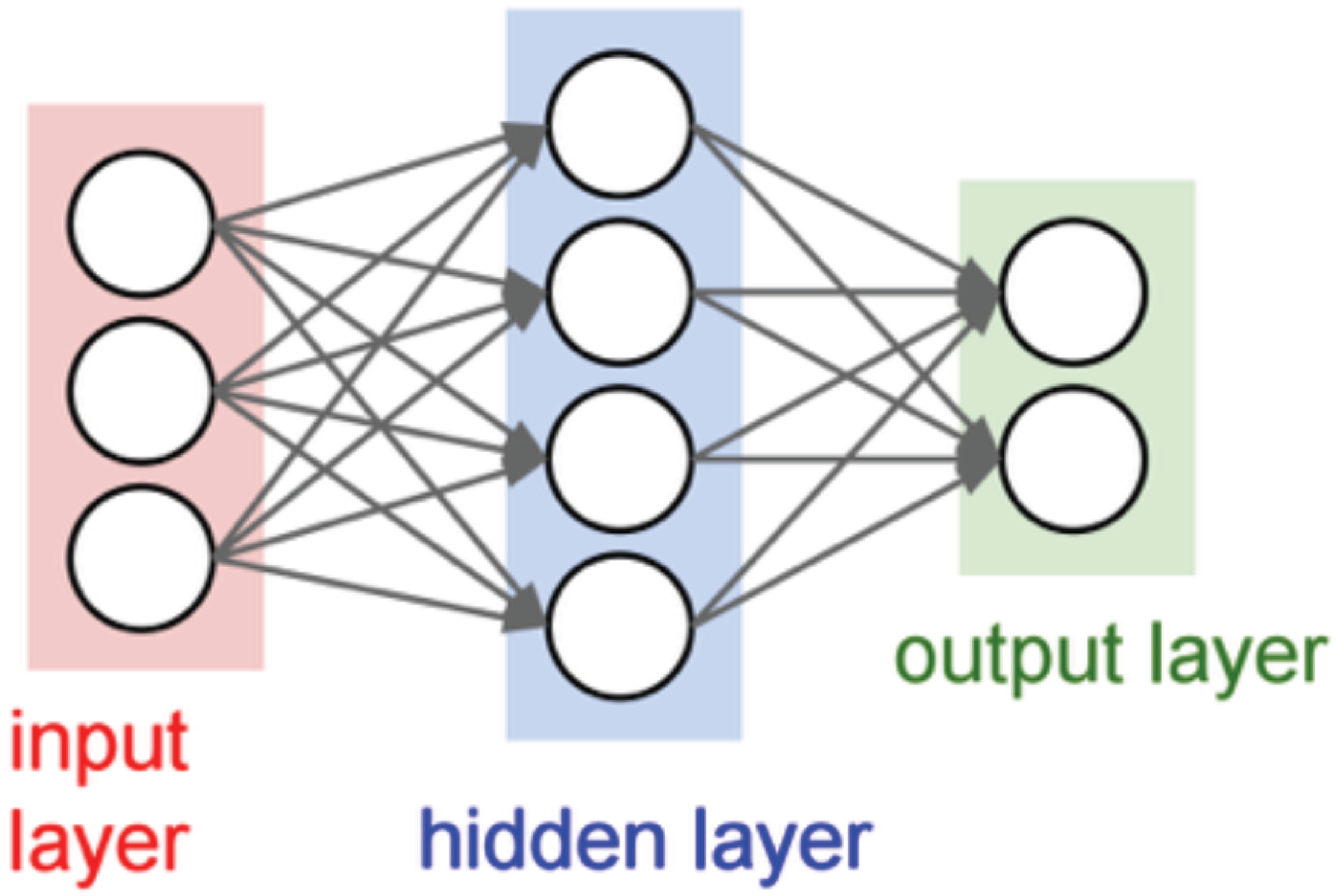
1. Equivariant with translation $f(\text{translate}(x)) = \text{translate}(f(x))$
2. Patch processing
3. Image filter 
4. Parameter sharing 
5. A way to process variable-sized tensors

ConvNets

They’re just neural networks with
3D activations and weight sharing

3D Activations

before:

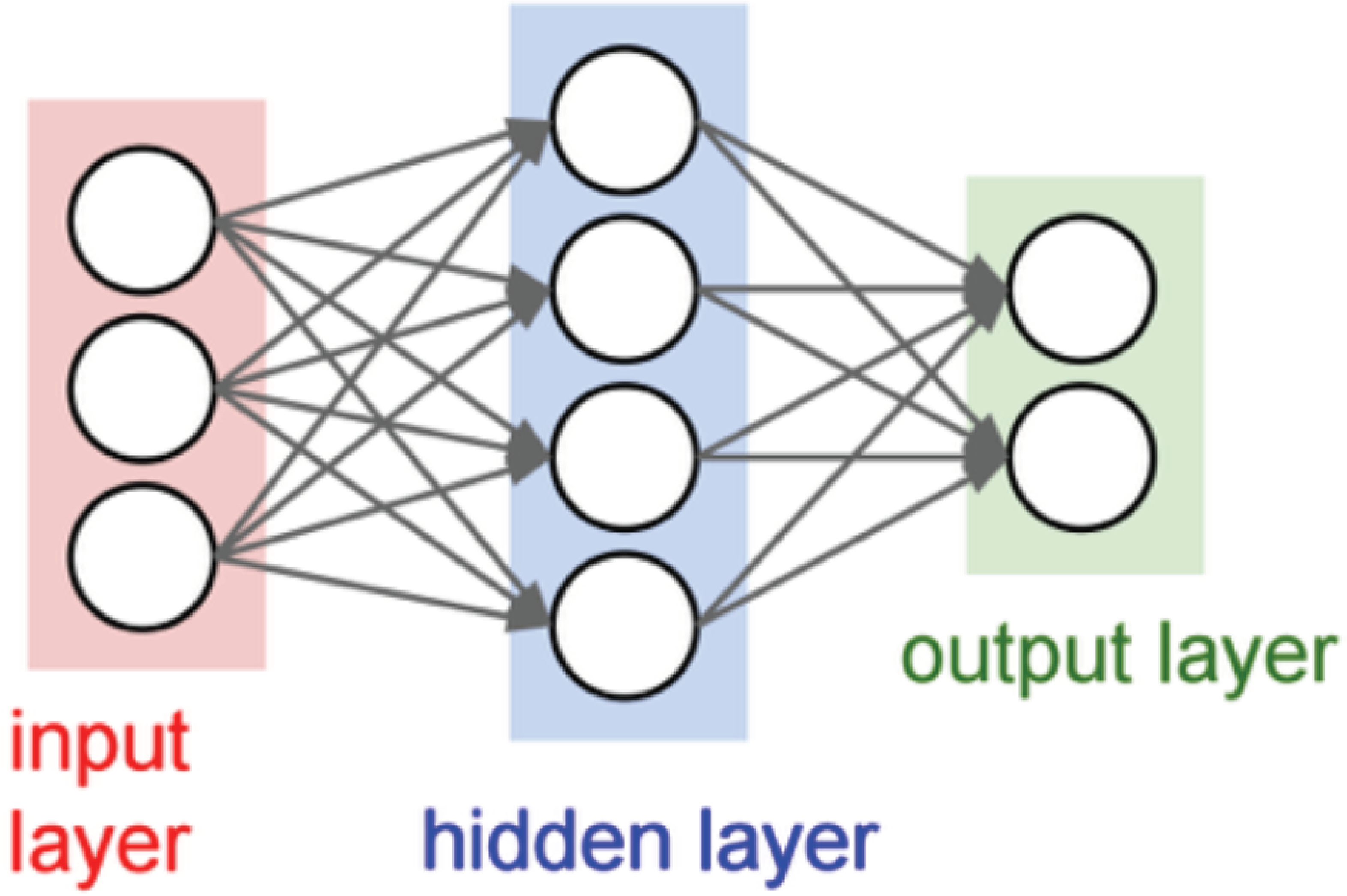


(1D vectors)

Figure: Andrej Karpathy

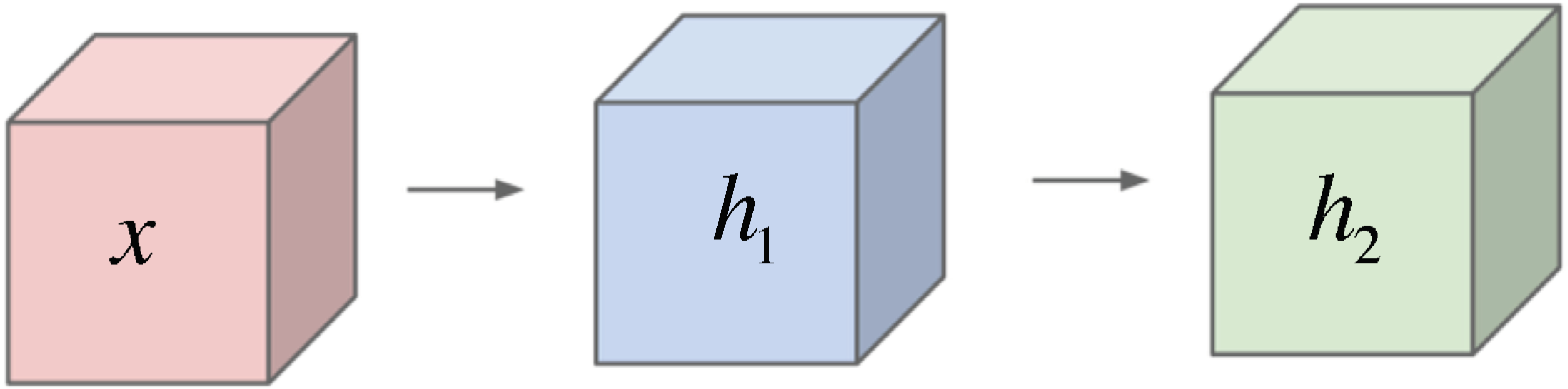
3D Activations

before:



(1D vectors)

now:



(3D arrays)

Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**

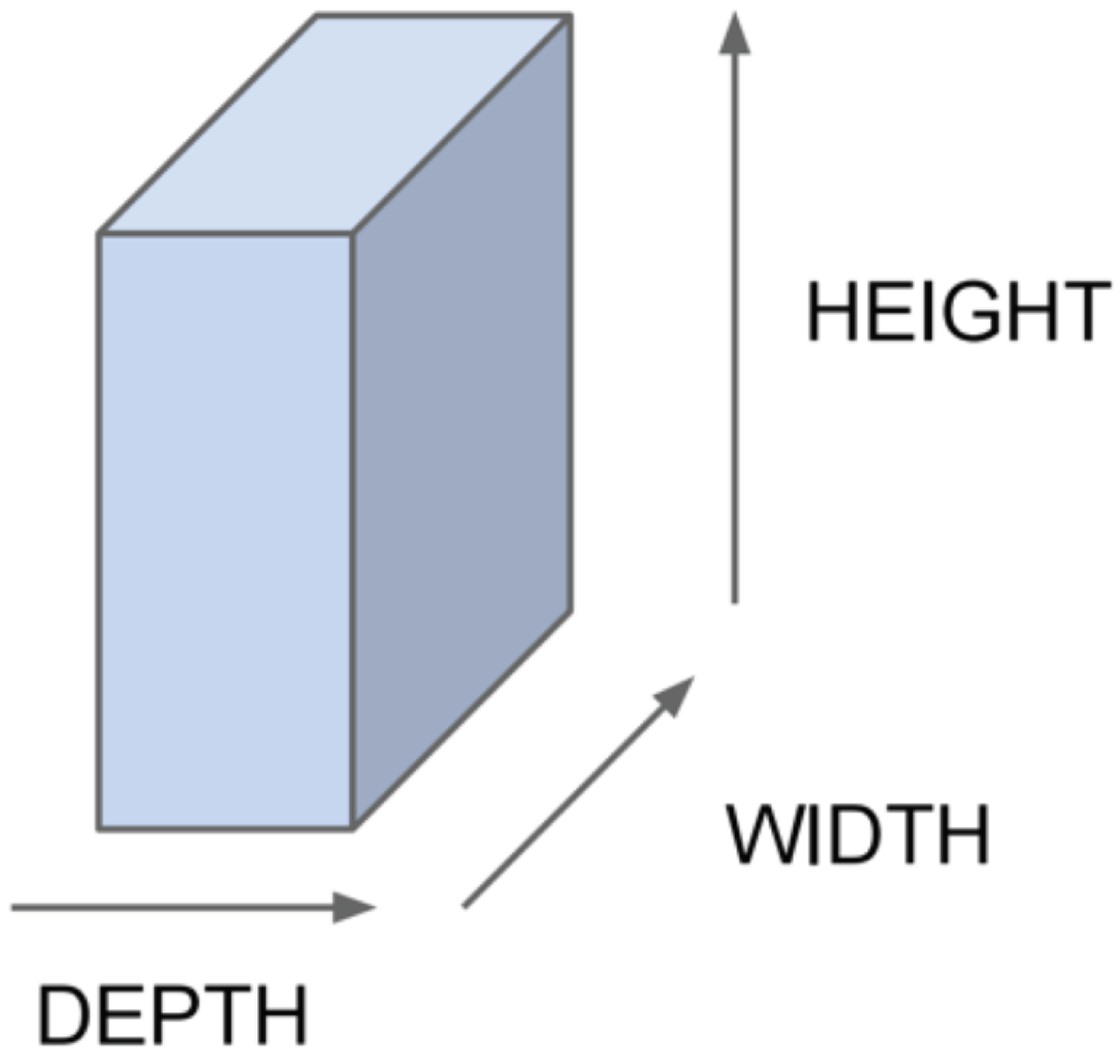
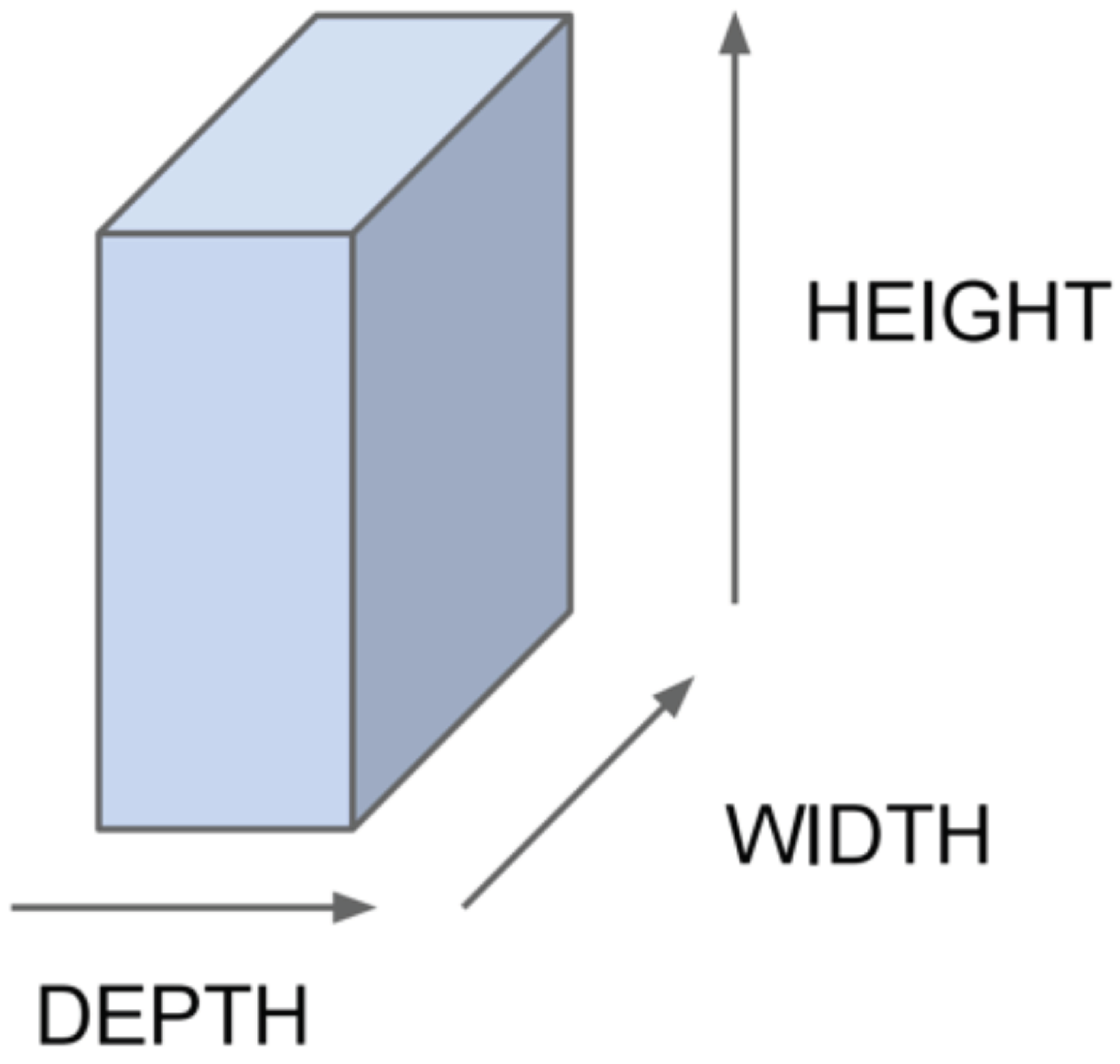


Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**



For example, a CIFAR-10 image is a 3x32x32 volume
(3 depth — RGB channels, 32 height, 32 width)

Figure: Andrej Karpathy

3D Activations

1D Activations:

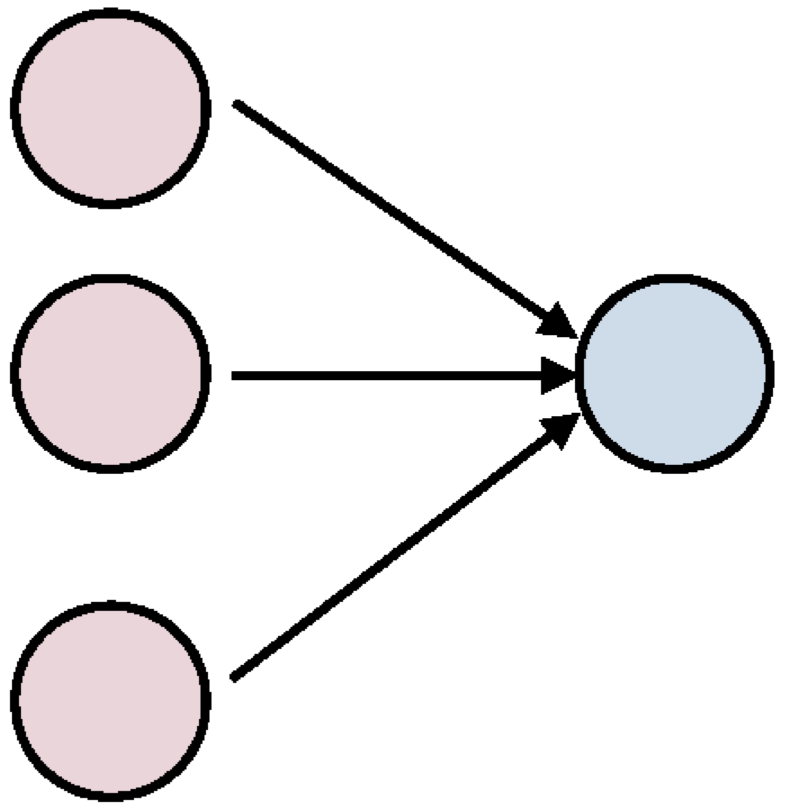
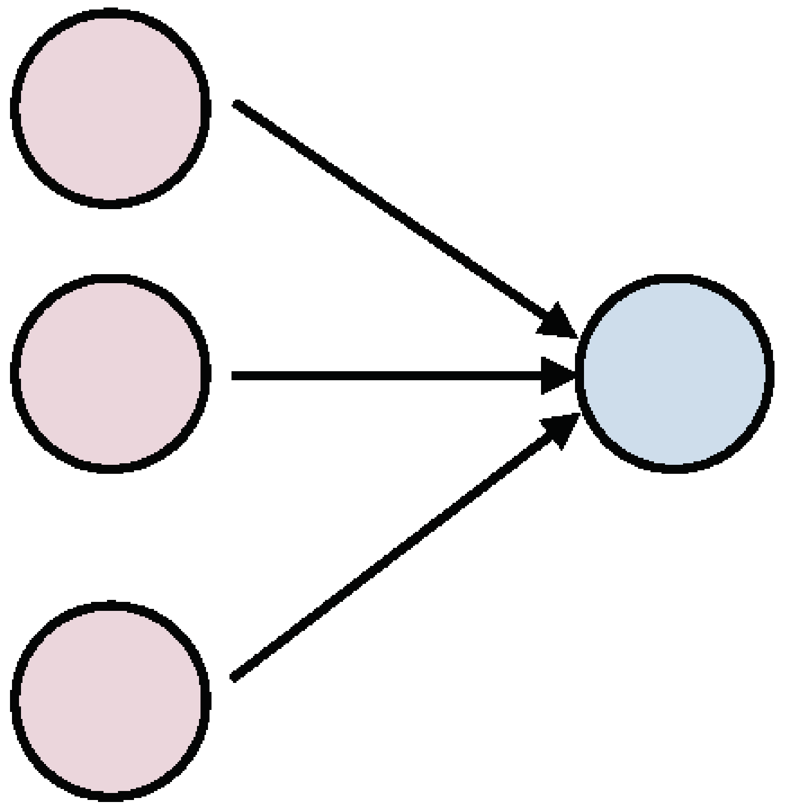


Figure: Andrej Karpathy

3D Activations

1D Activations:



3D Activations:

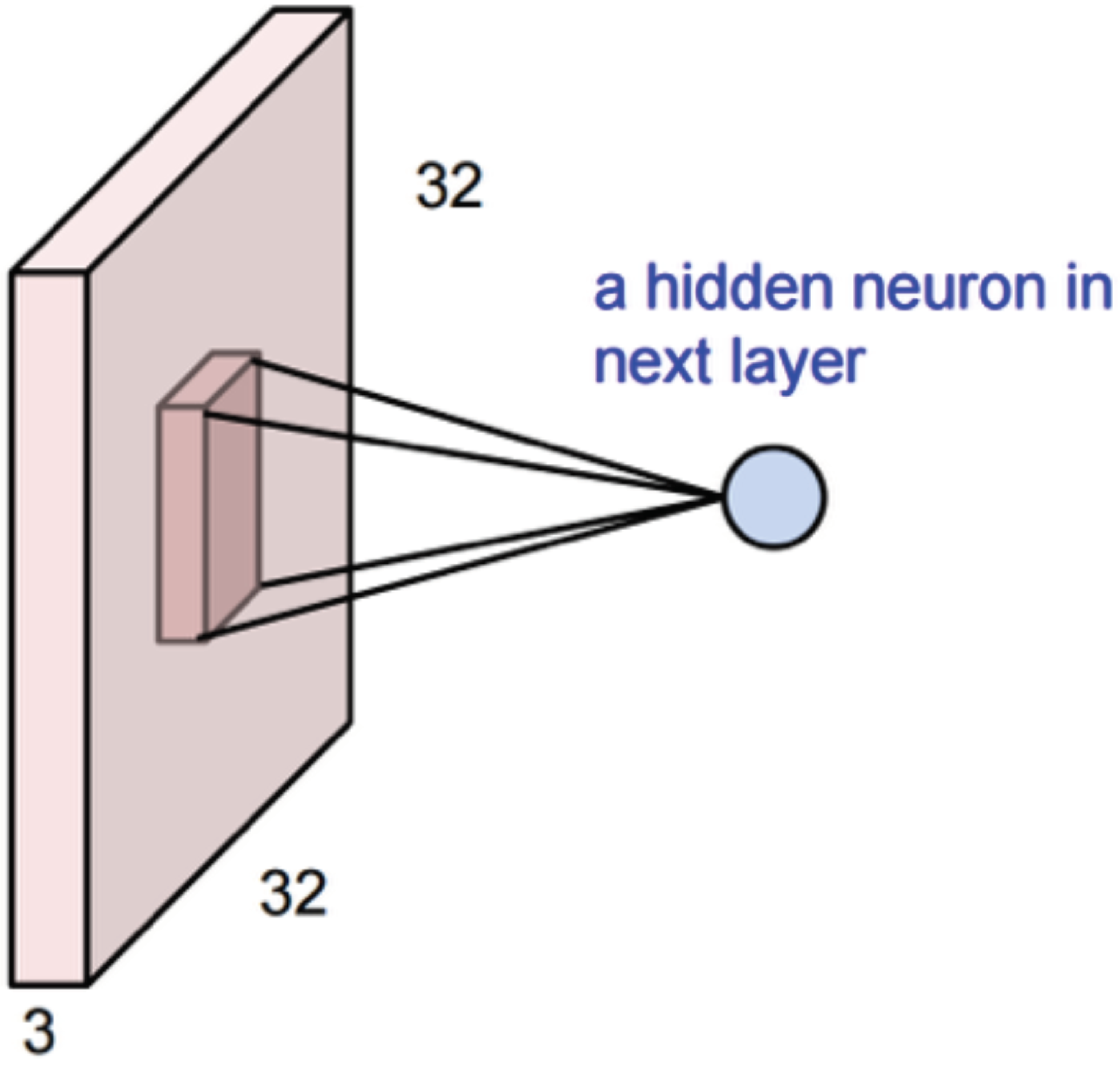
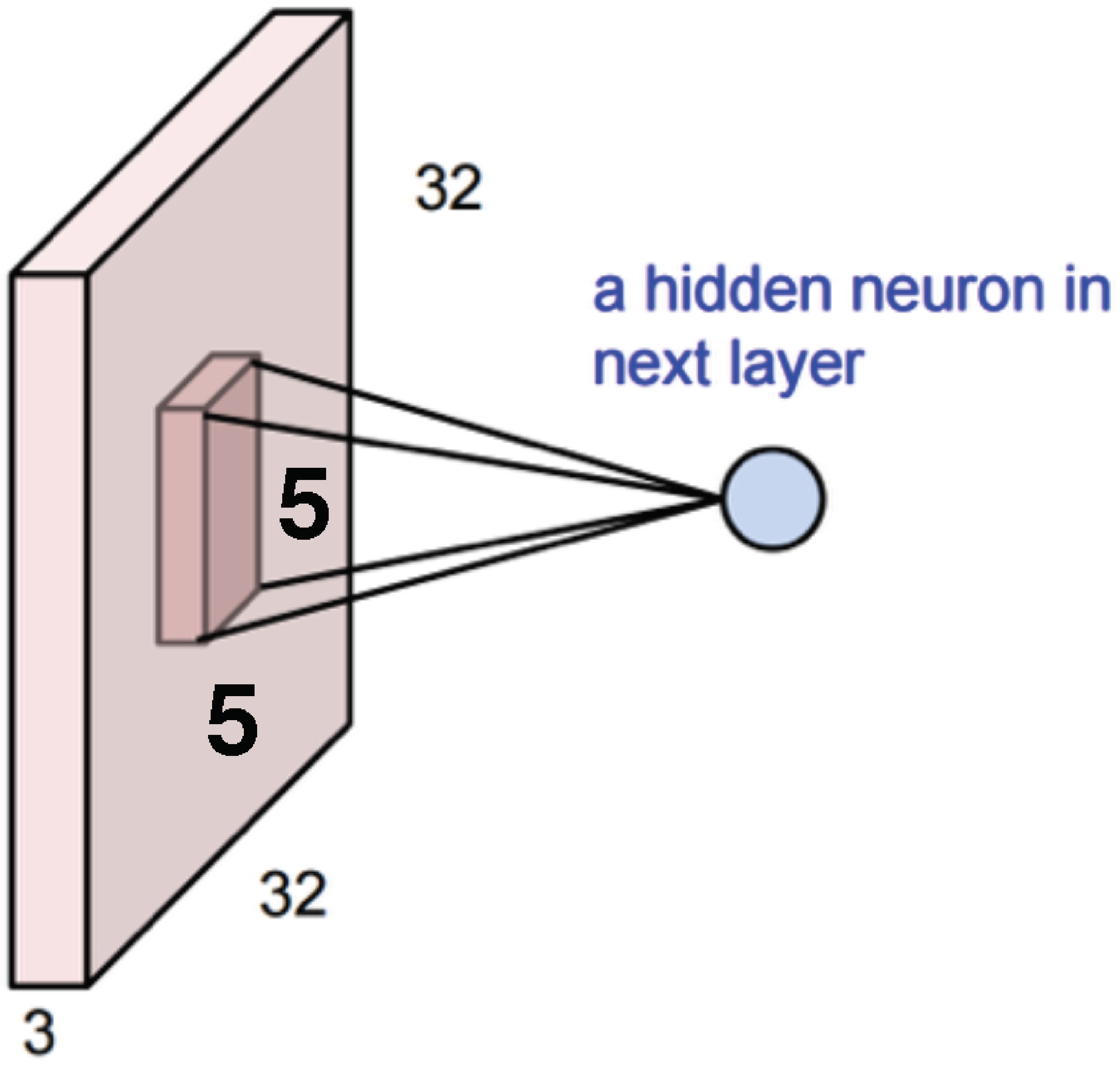


Figure: Andrej Karpathy

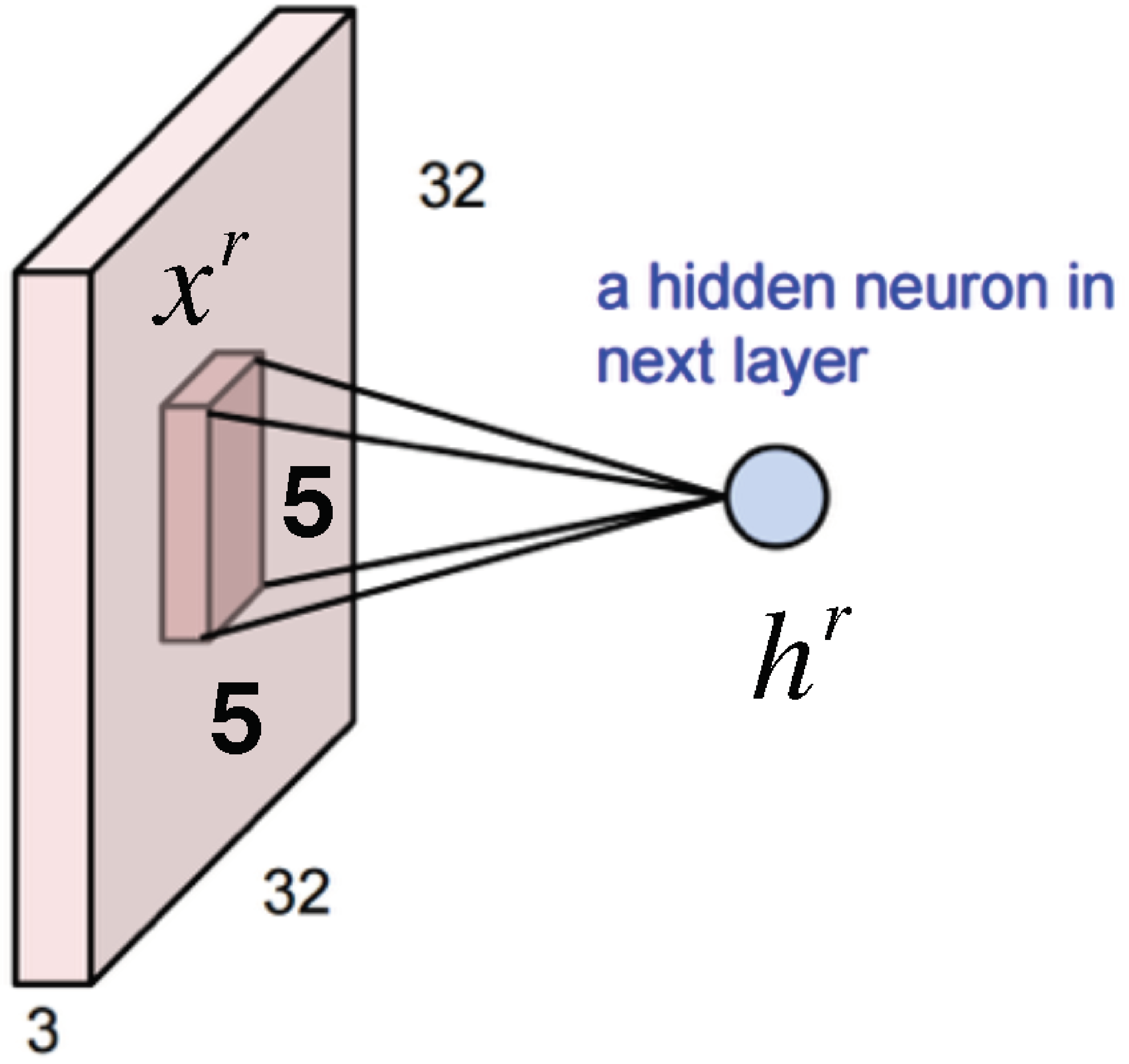
3D Activations



- The input is 3x32x32
- This neuron depends on a 3x5x5 chunk of the input
- The neuron also has a 3x5x5 set of weights and a bias (scalar)

Figure: Andrej Karpathy

3D Activations

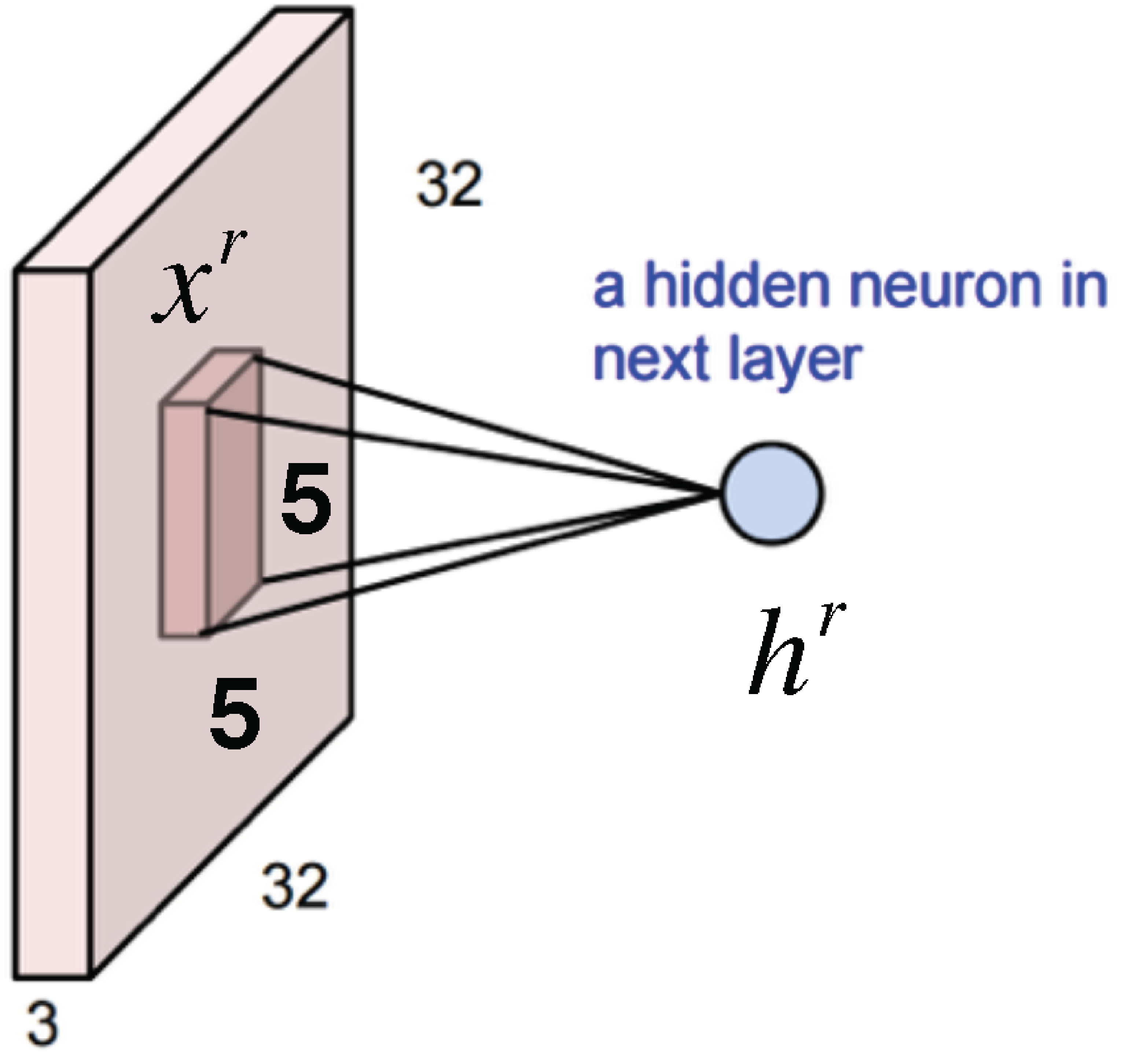


Example: consider the region of the input "x^r"

With output neuron h^r

Figure: Andrej Karpathy

3D Activations



Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Figure: Andrej Karpathy

Feb 12, 2025

3D Activations

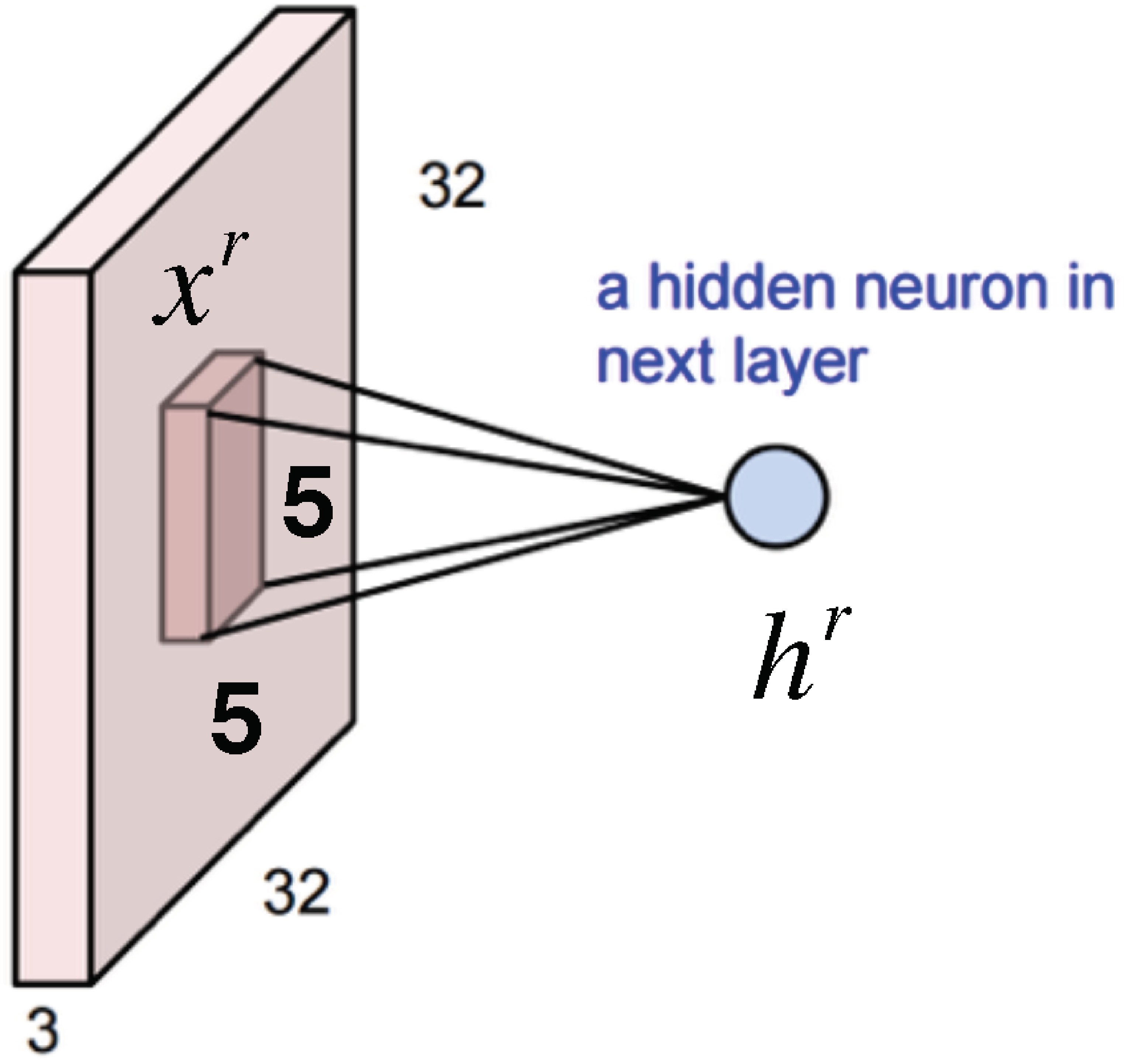


Figure: Andrej Karpathy

Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Sum over 3 axes

3D Activations

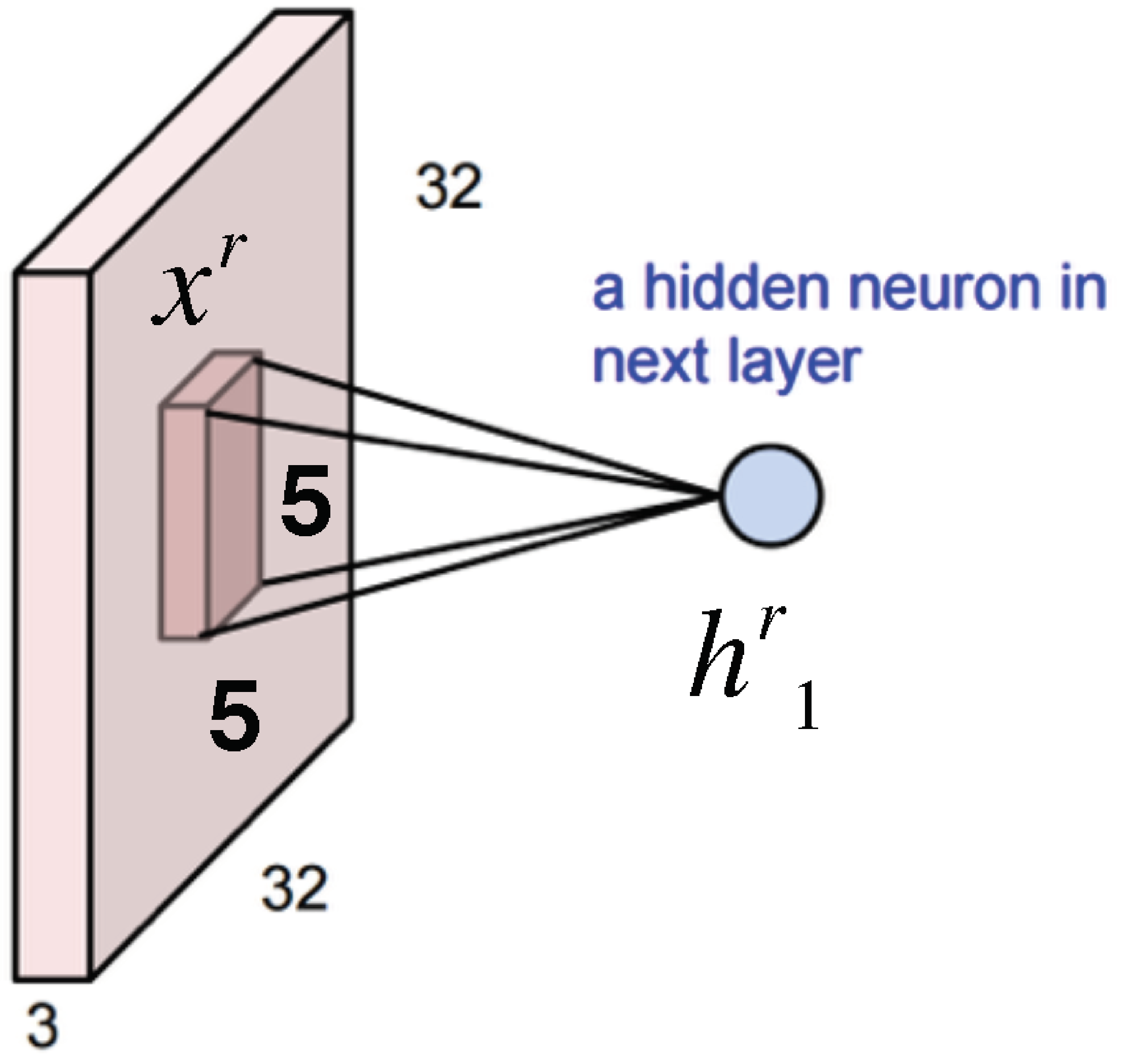


Figure: Andrej Karpathy

3D Activations

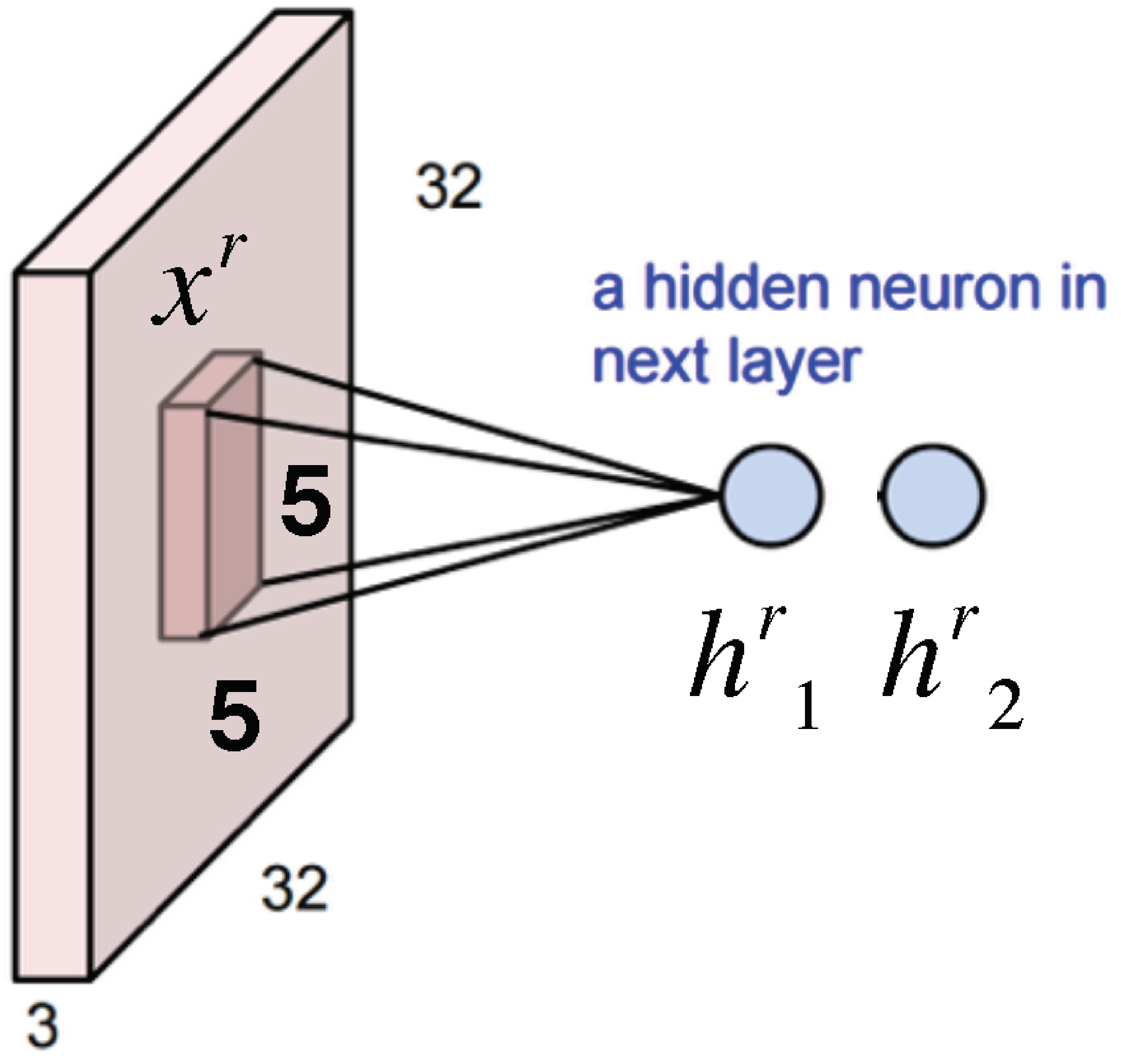
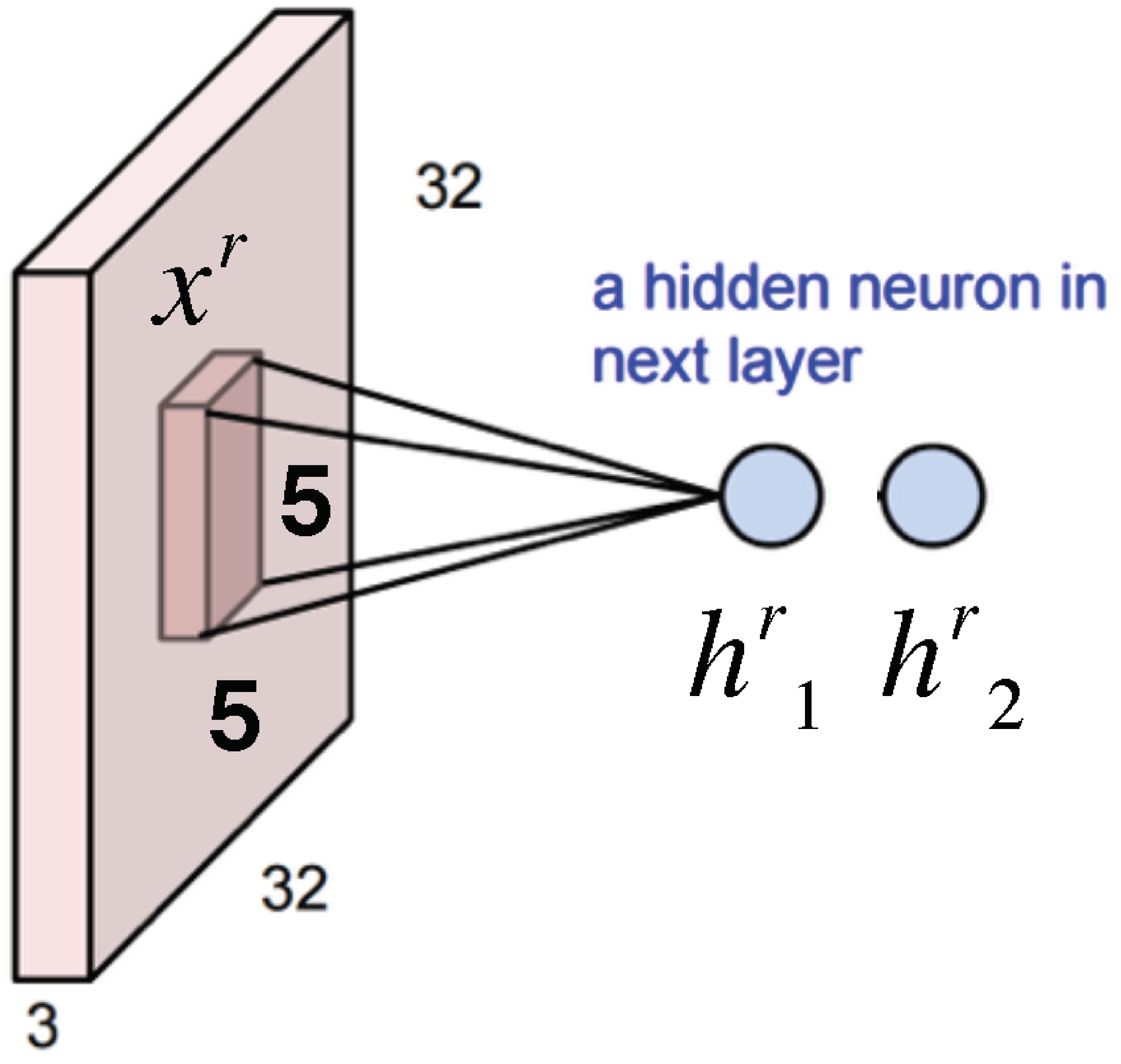


Figure: Andrej Karpathy

3D Activations



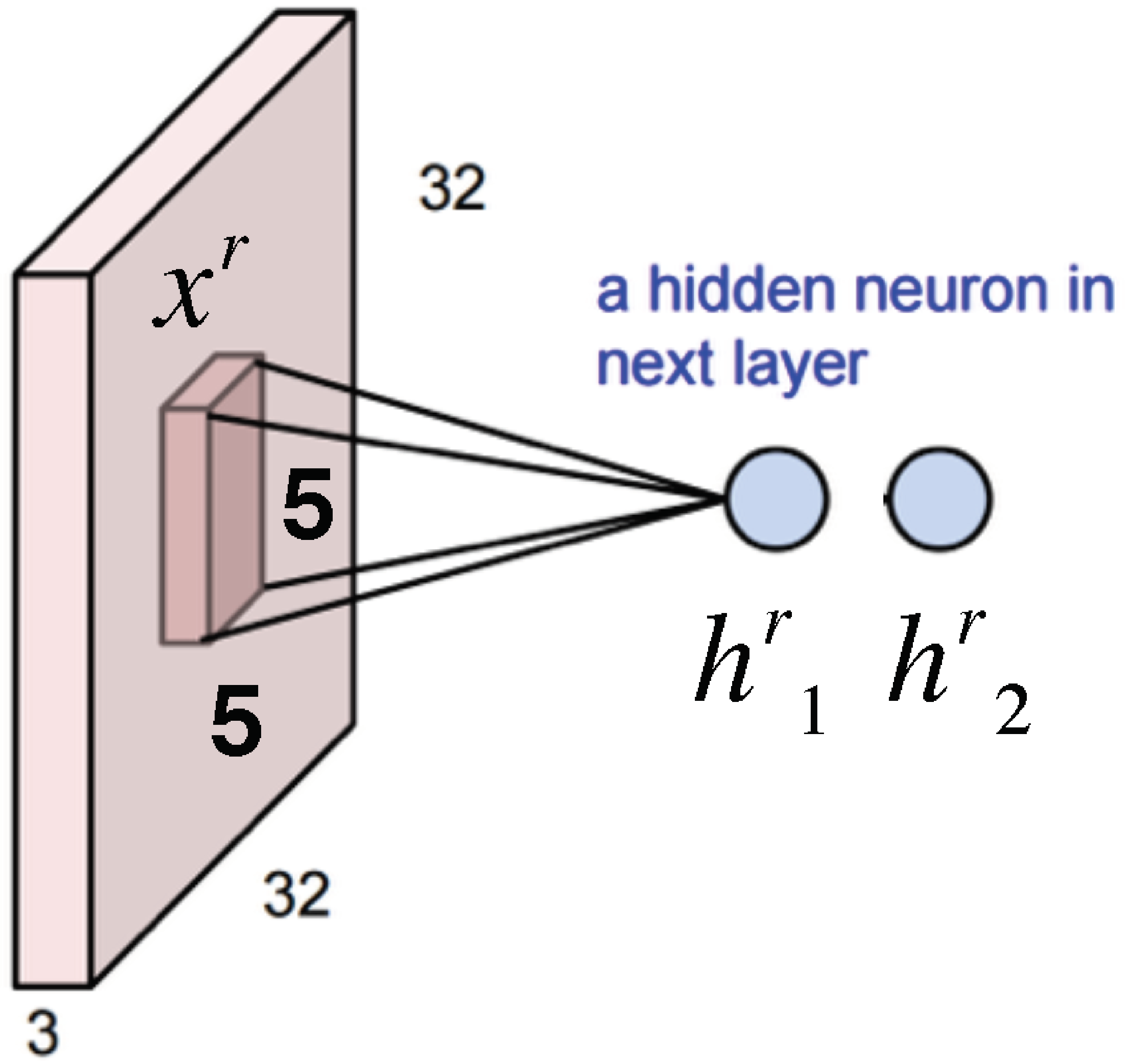
With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

3D Activations



With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

3D Activations

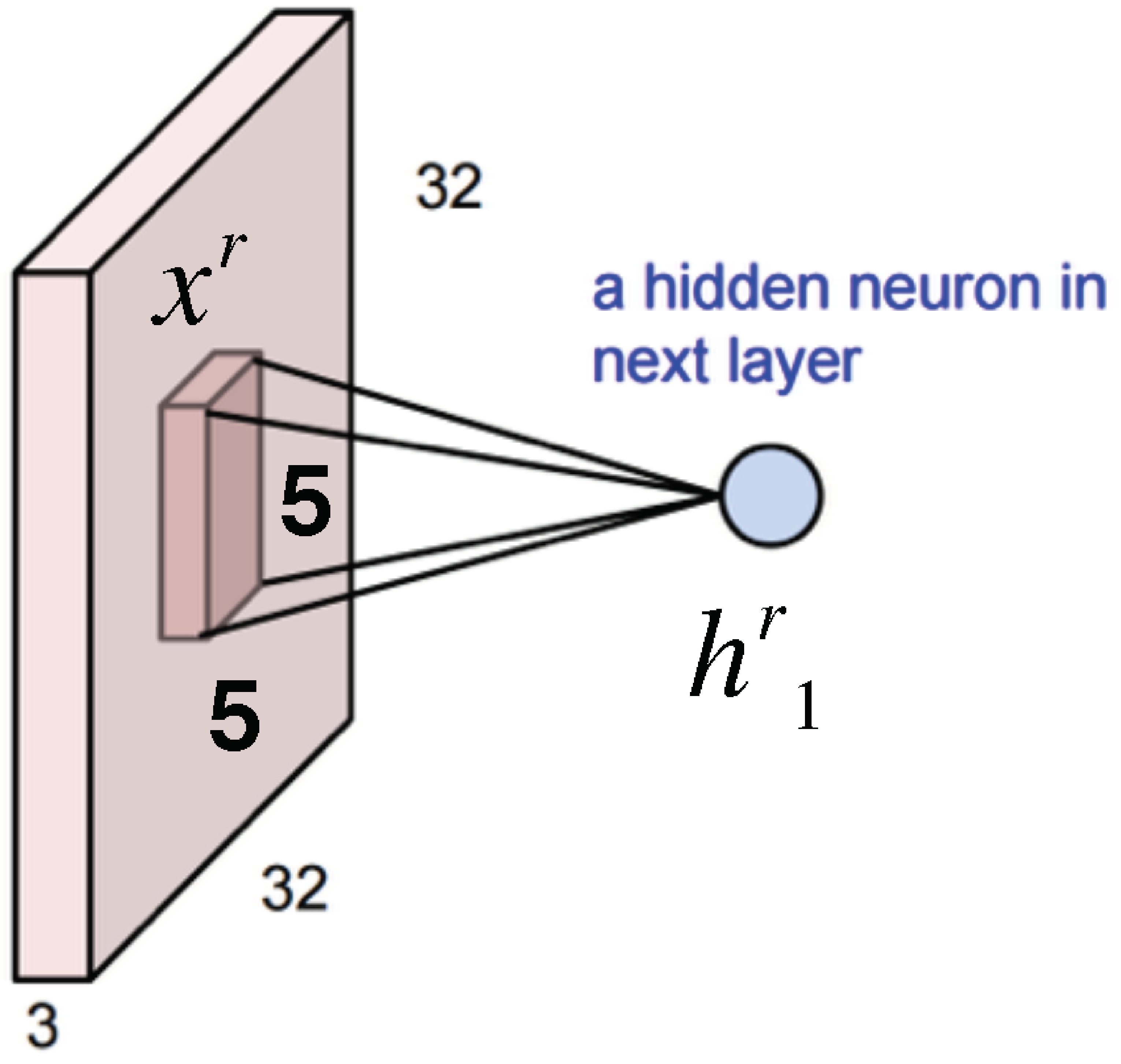


Figure: Andrej Karpathy

3D Activations

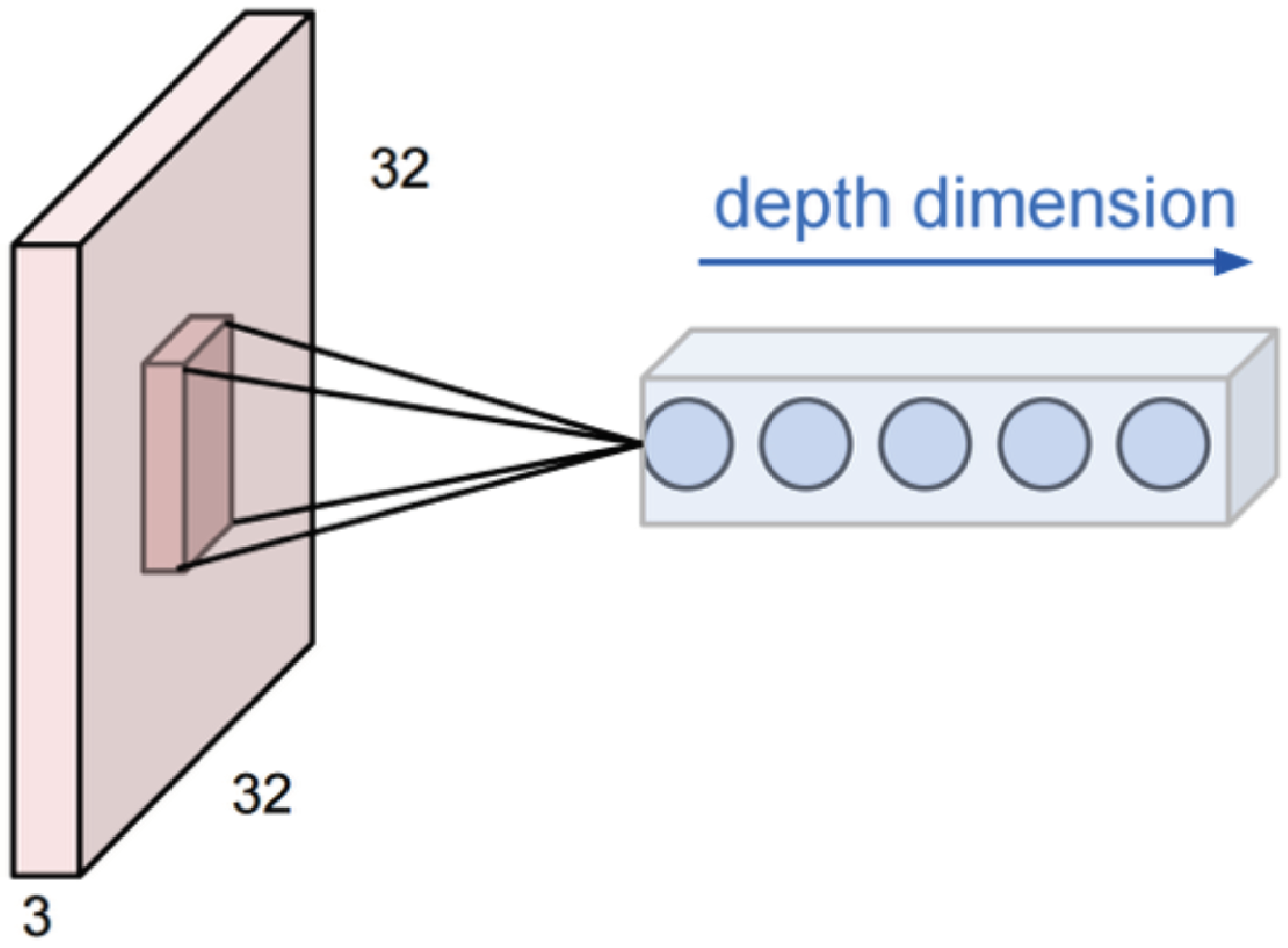
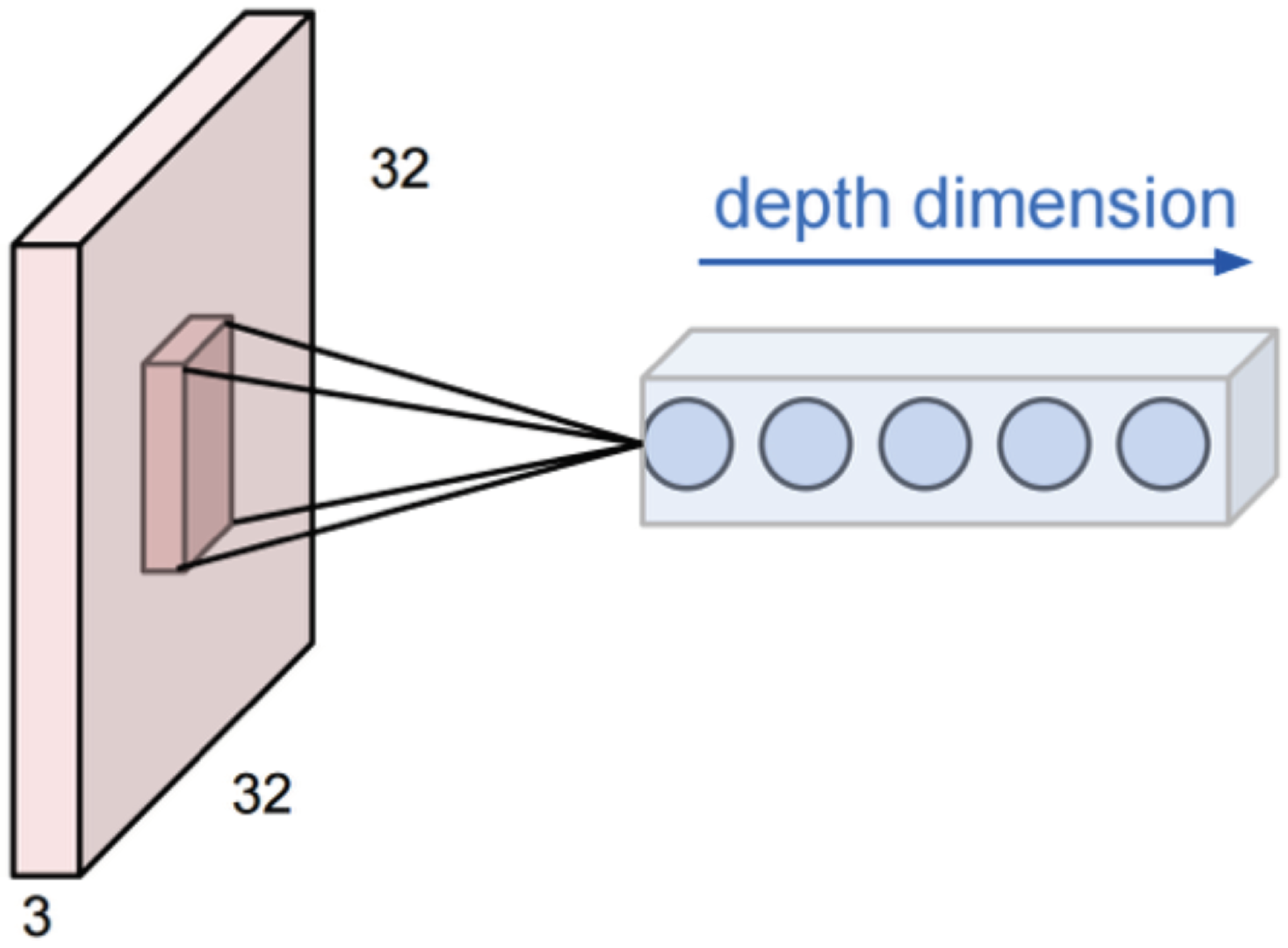


Figure: Andrej Karpathy

3D Activations

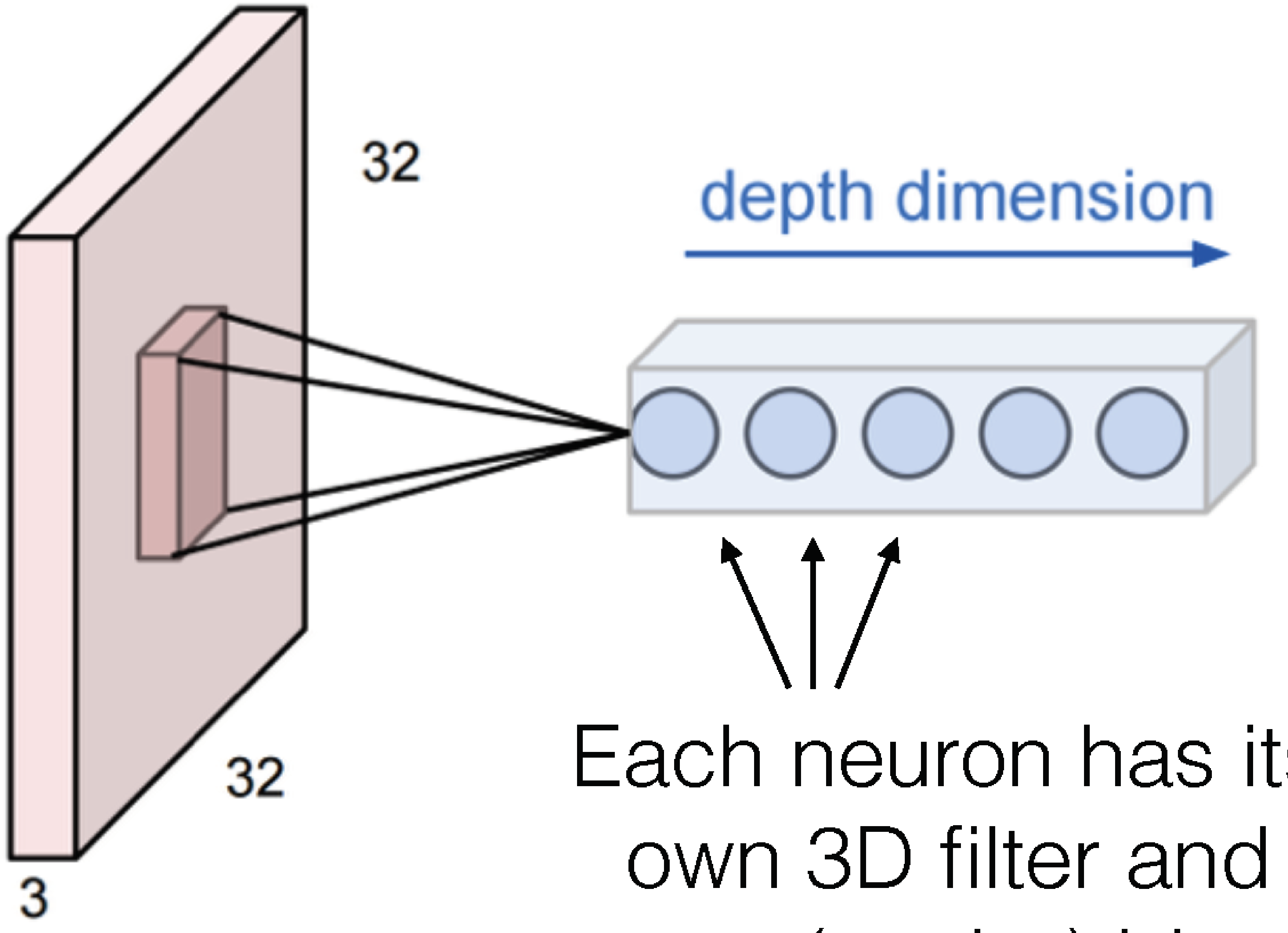


We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations



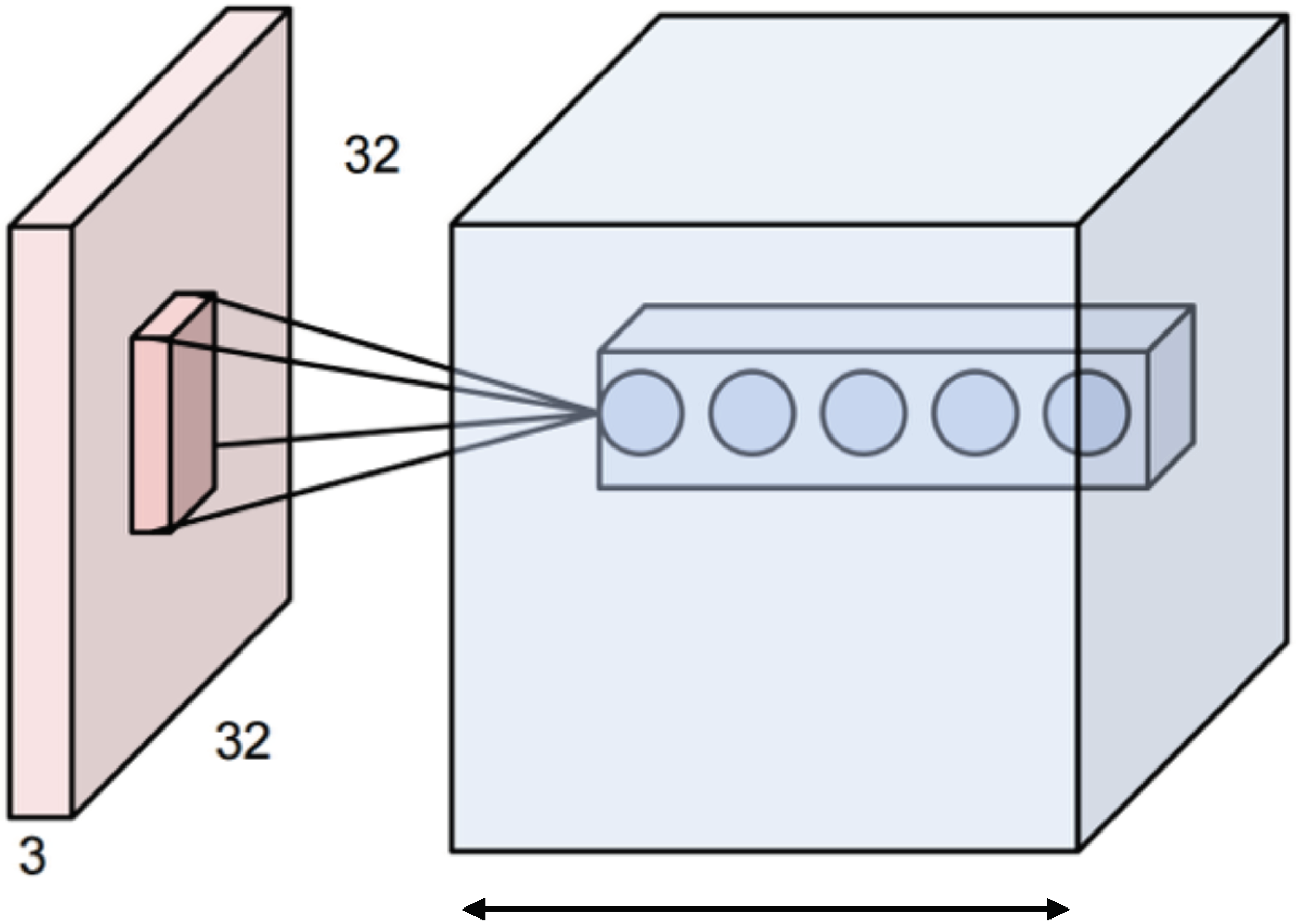
Each neuron has its own 3D filter and own (scalar) bias

We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations

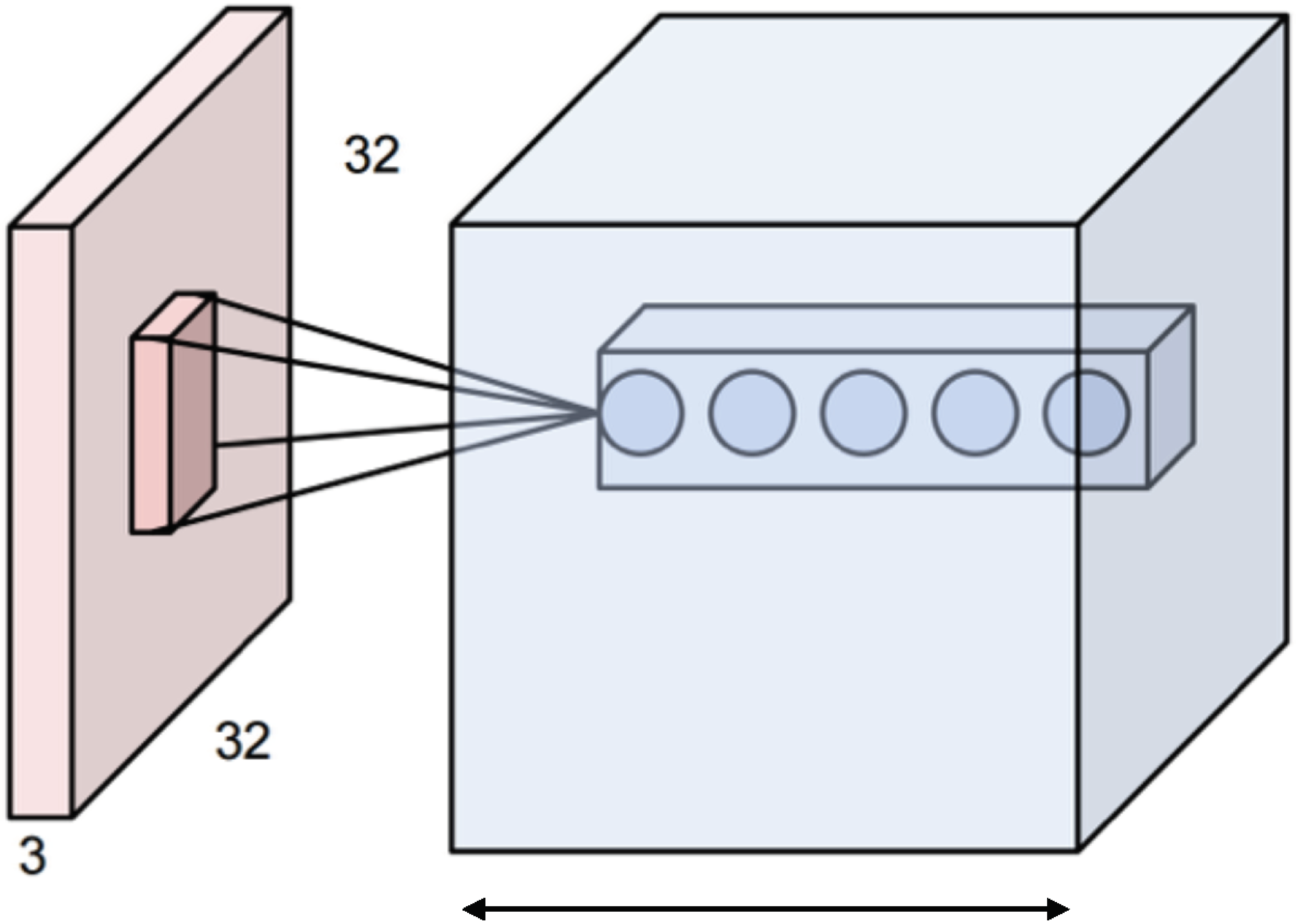


D sets of weights
(also called filters)

Now repeat this
across the input

Figure: Andrej Karpathy

3D Activations



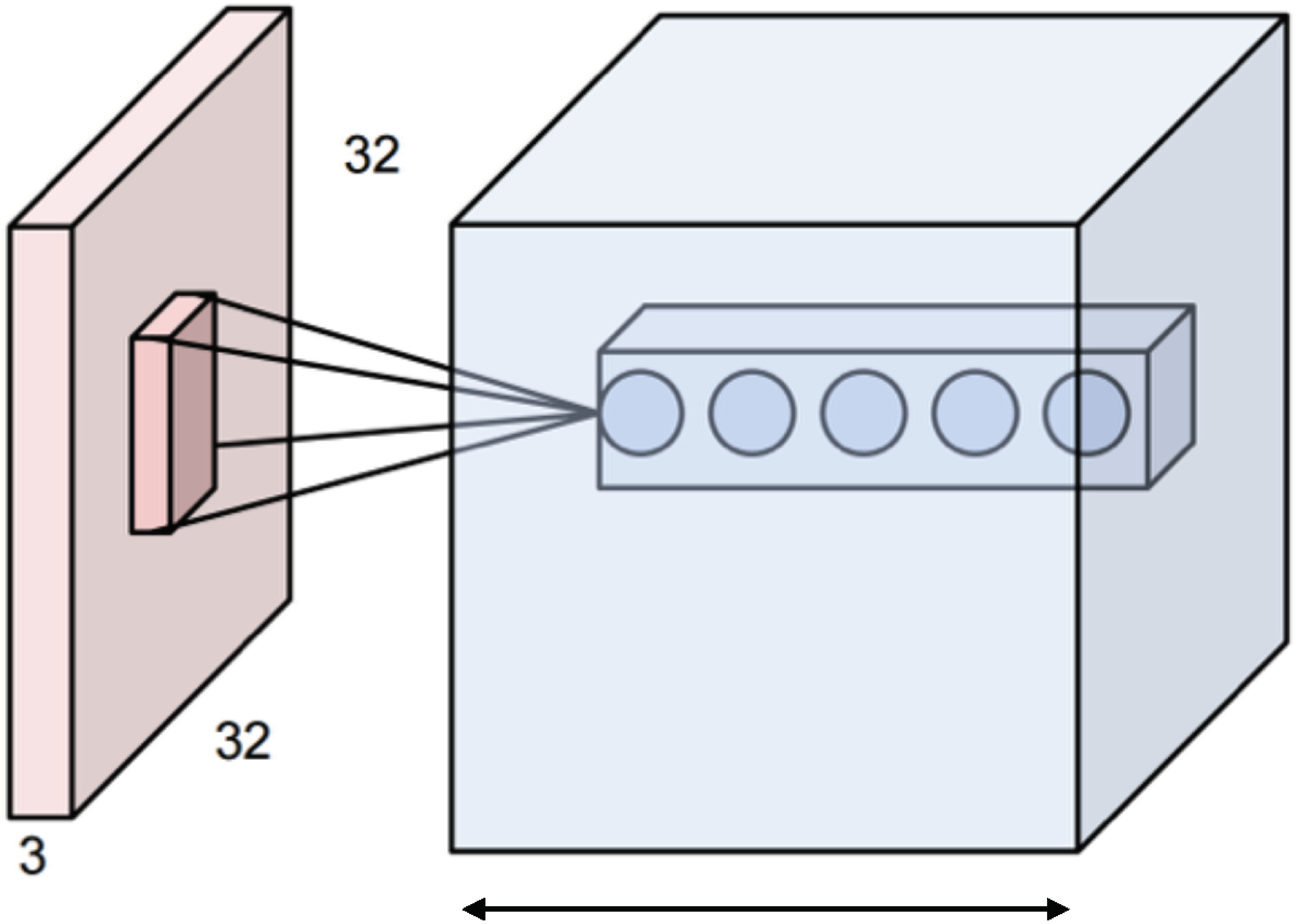
D sets of weights
(also called filters)

Now repeat this across the input

Weight sharing:
Each filter shares the same weights (but each depth index has its own set of weights)

Figure: Andrej Karpathy

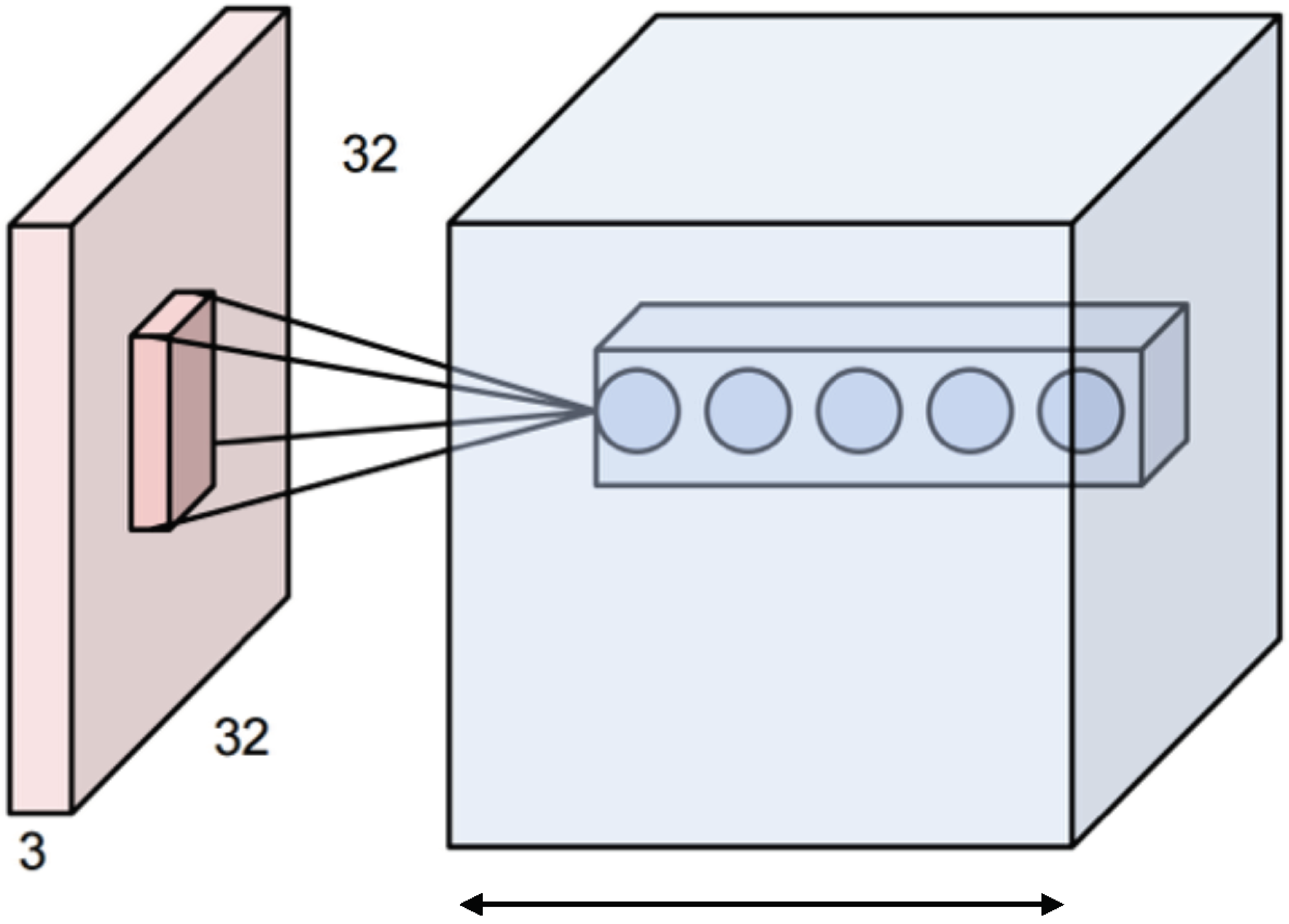
3D Activations



D sets of weights
(also called filters)

Figure: Andrej Karpathy

3D Activations

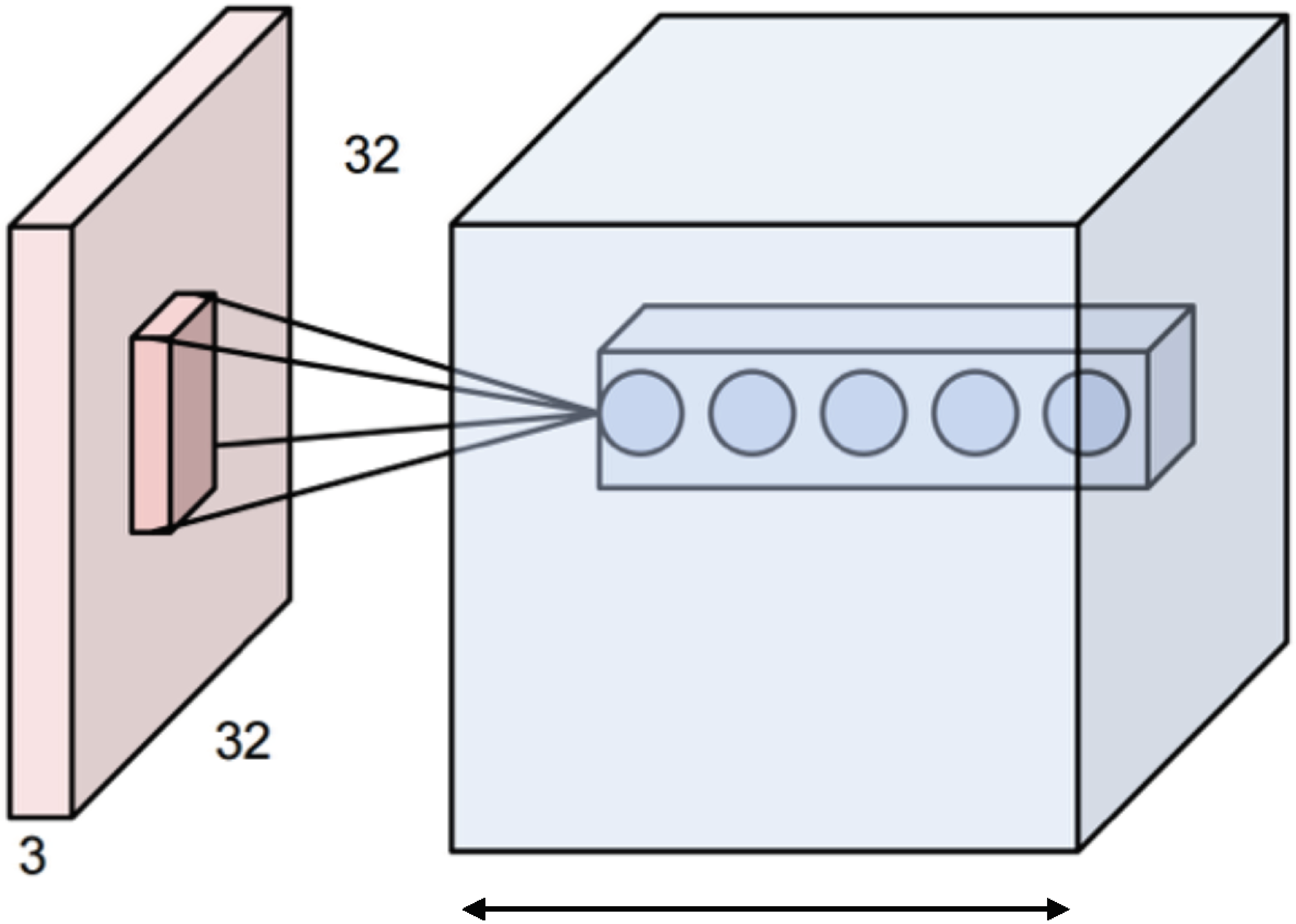


D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Figure: Andrej Karpathy

3D Activations



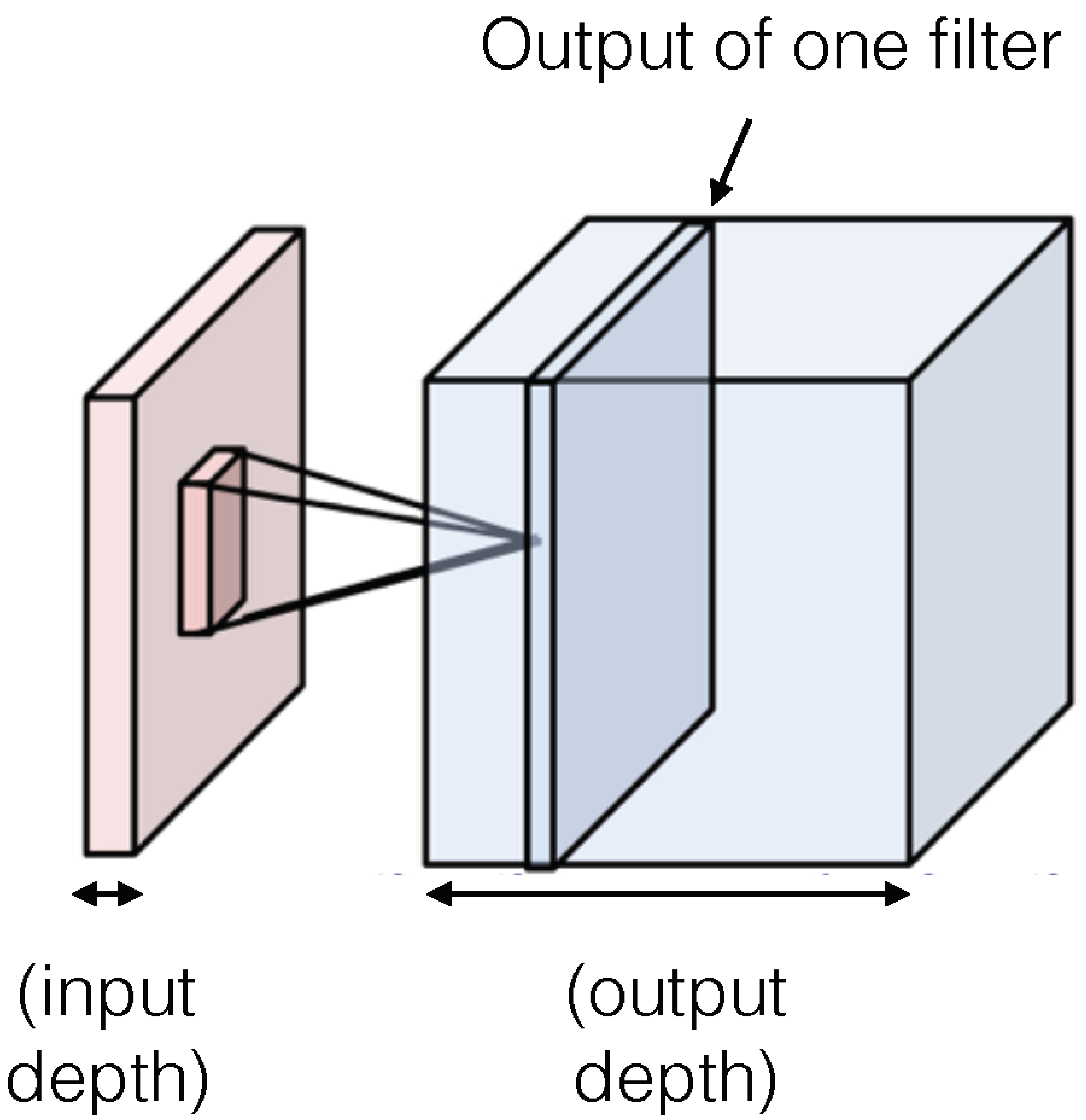
D sets of weights
(also called filters)

With weight sharing,
this is called **convolution**

Without weight sharing,
this is called a **locally connected layer**

Figure: Andrej Karpathy

3D Activations

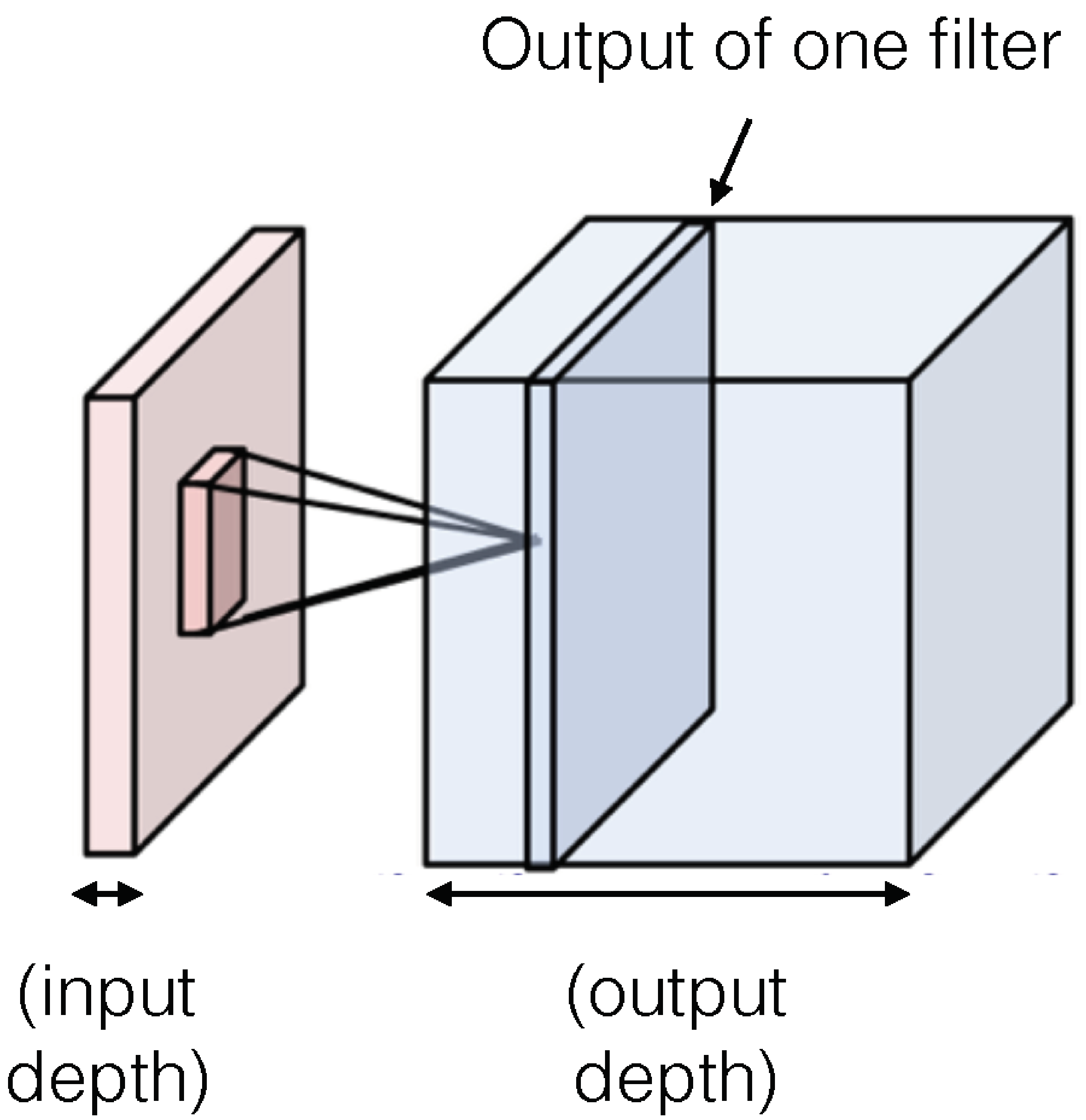


One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

3D Activations



One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

All together, the weights are **4** dimensional:
(output depth, input depth, kernel height, kernel width)

3D Activations

We can unravel the 3D cube and show each layer separately:

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

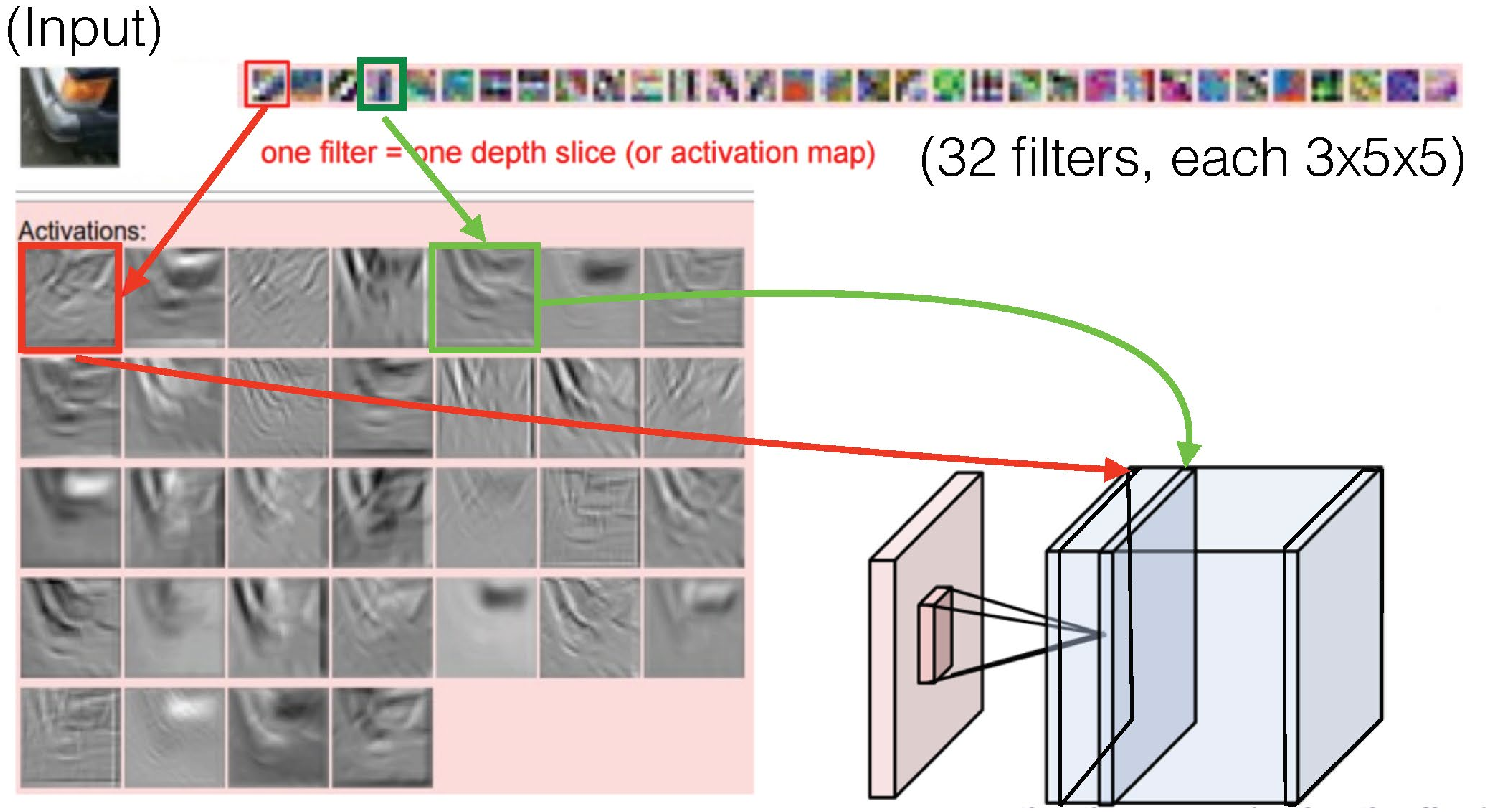


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

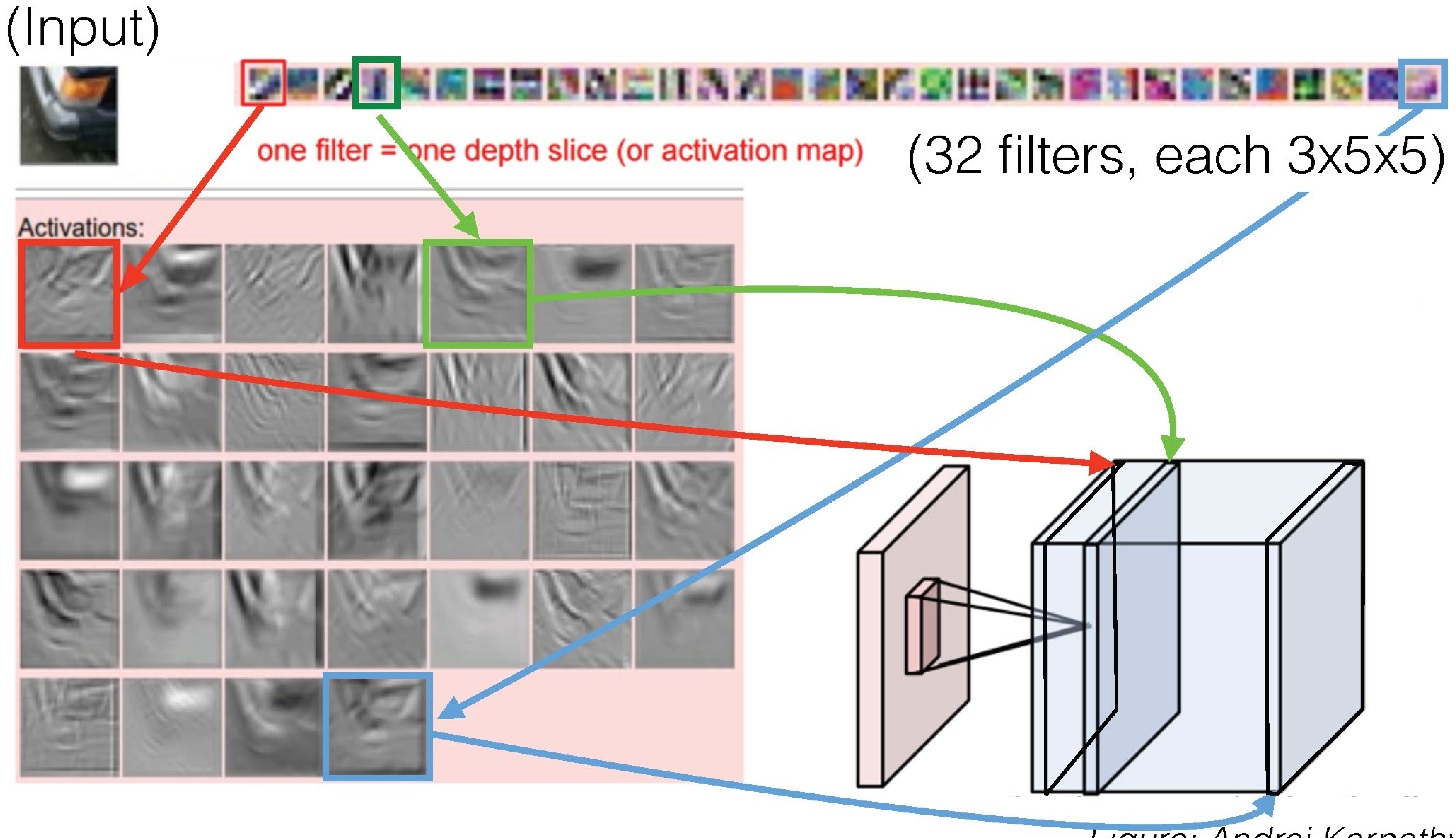
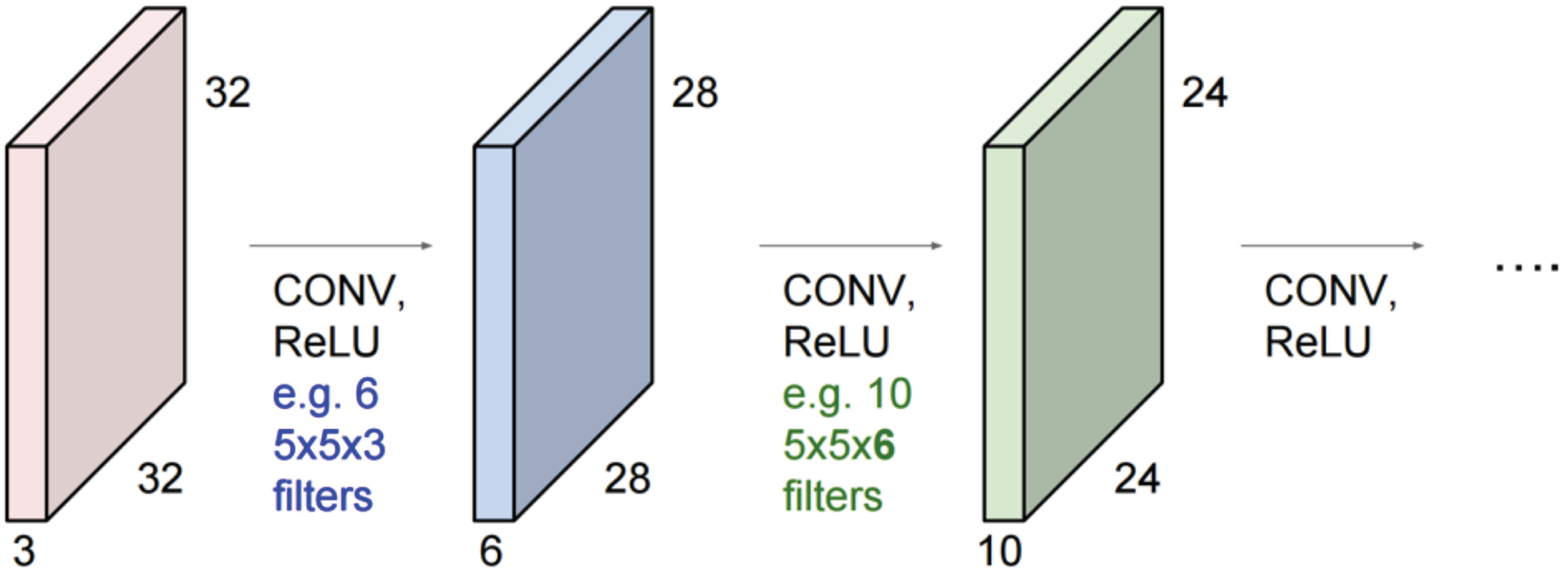


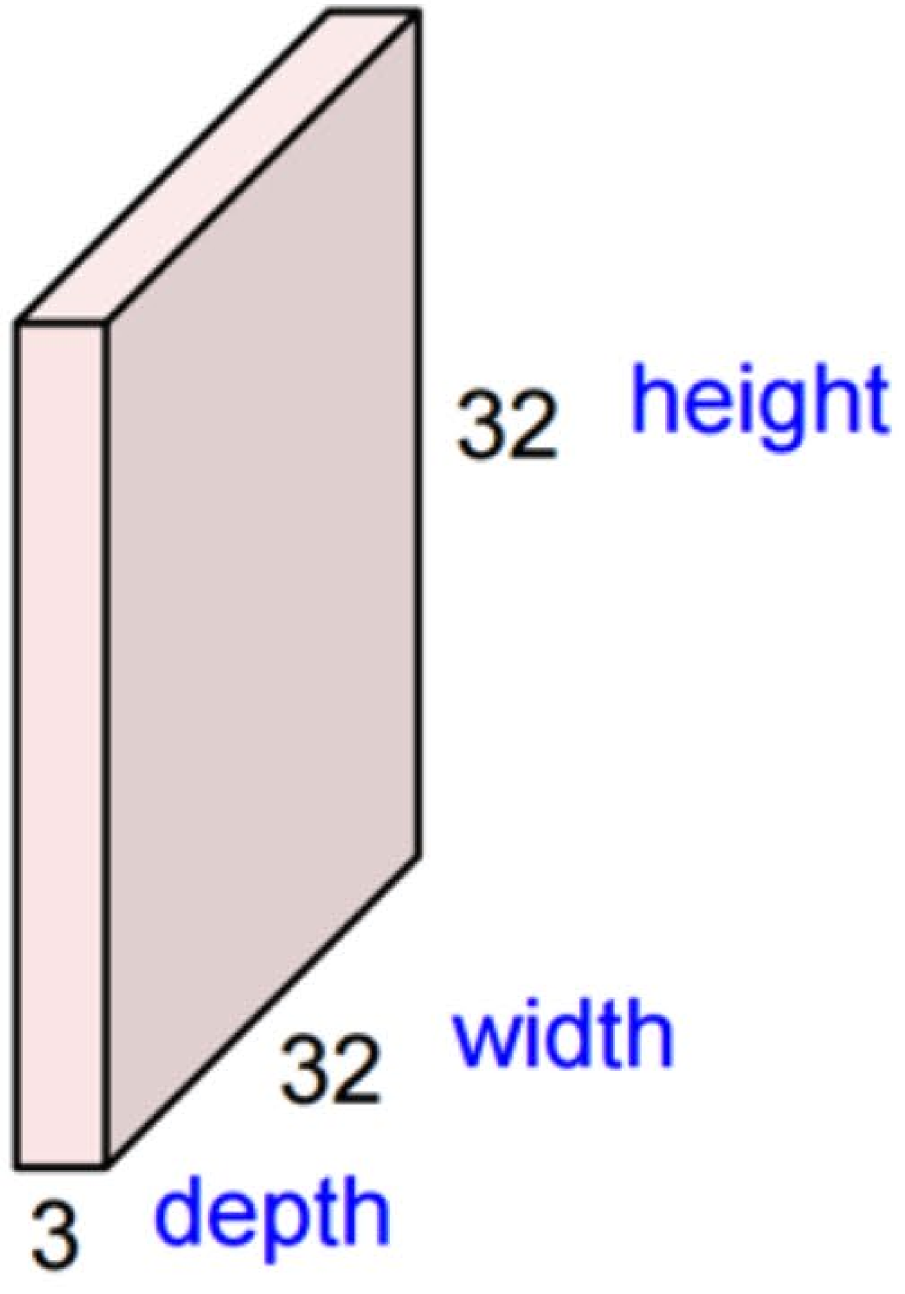
Figure: Andrej Karpathy

A **ConvNet** is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types)



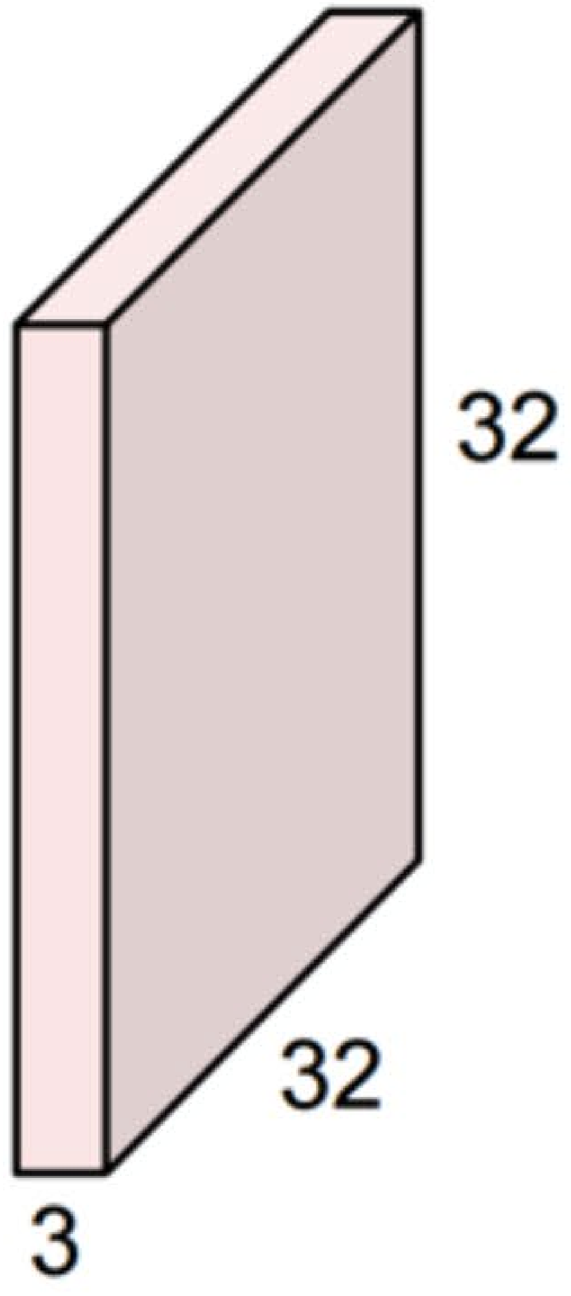
Convolution Layer

32x32x3 image

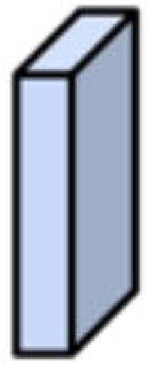


Convolution Layer

32x32x3 image



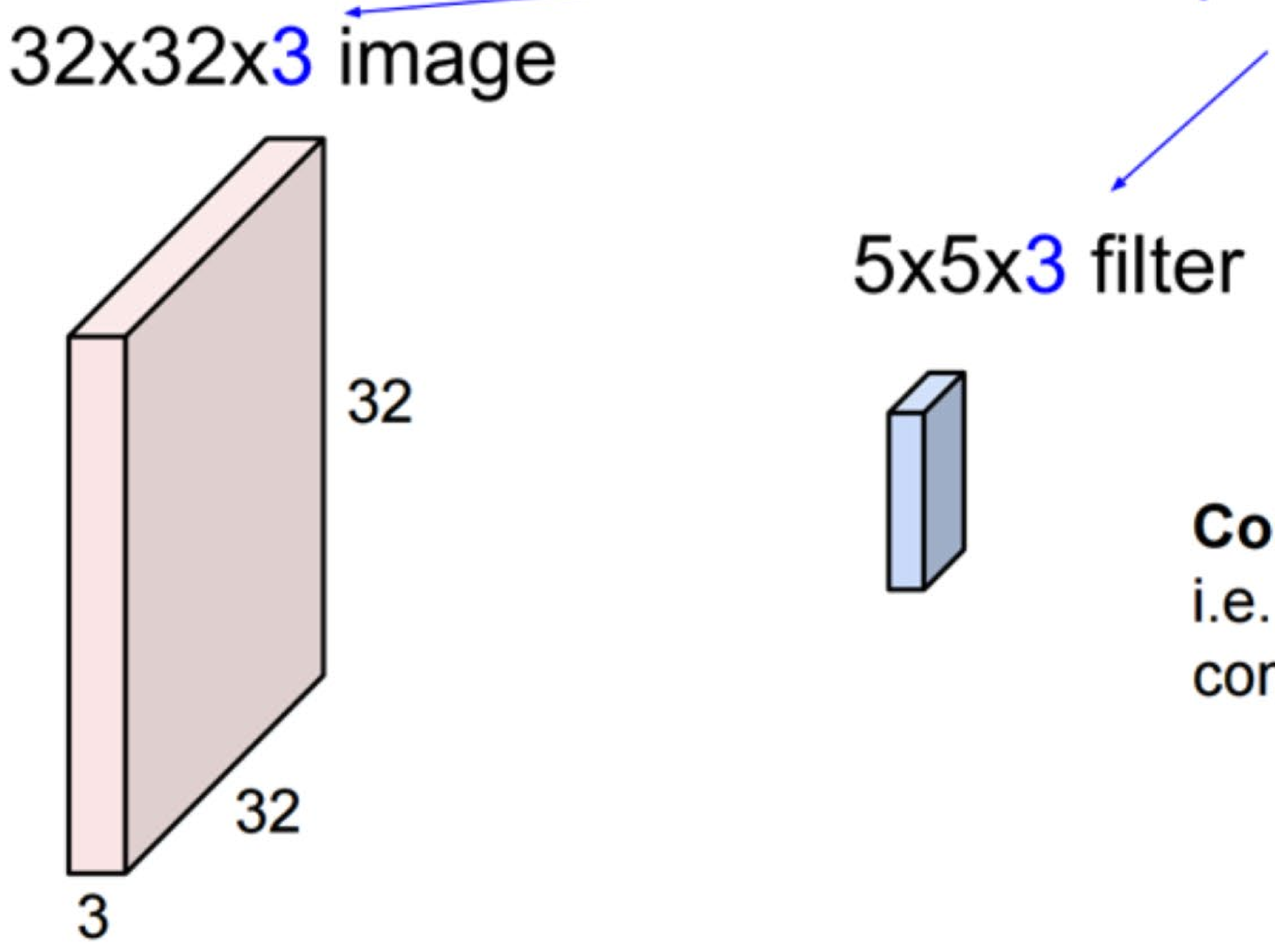
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

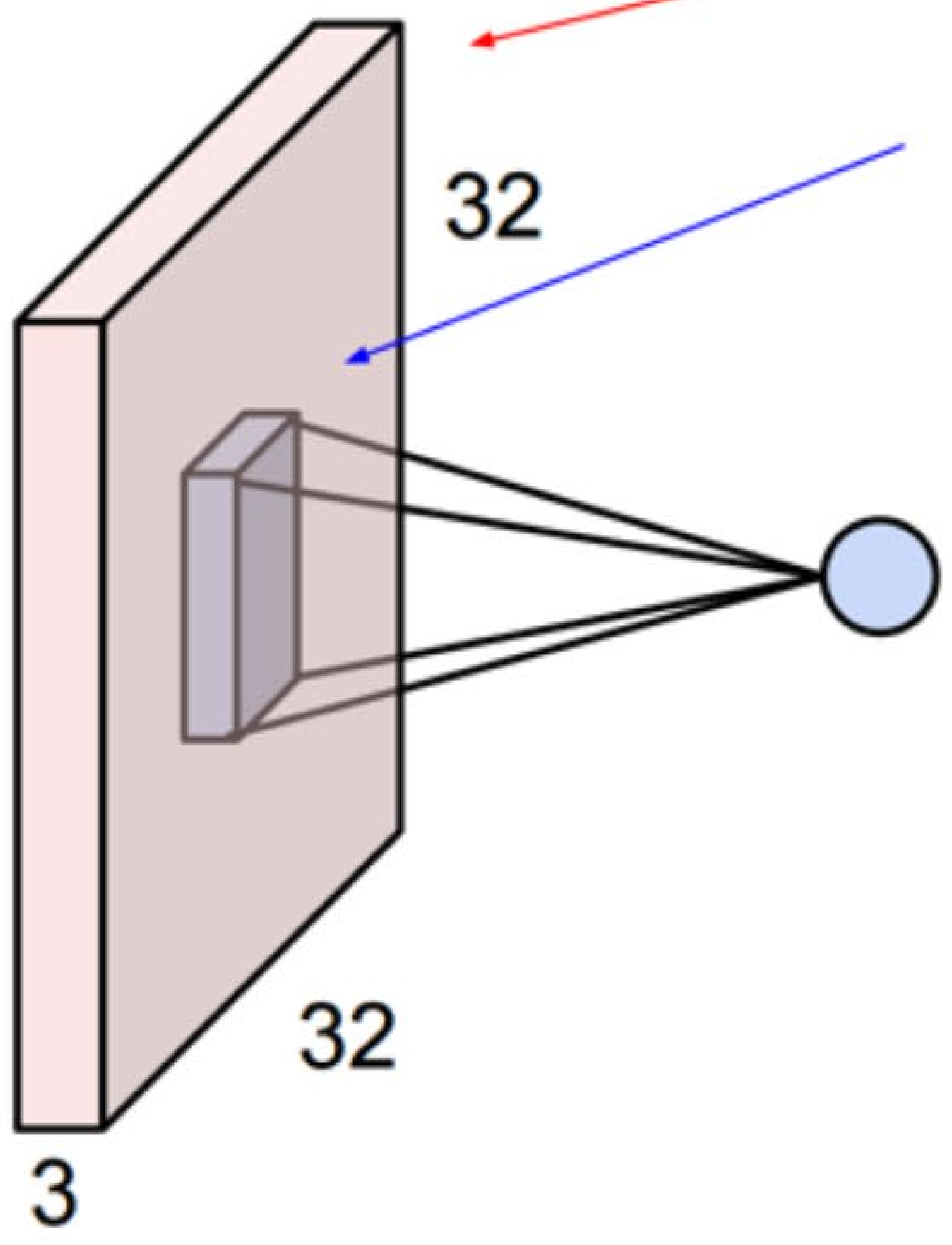
Convolution Layer

Filters always extend the full depth of the input volume



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

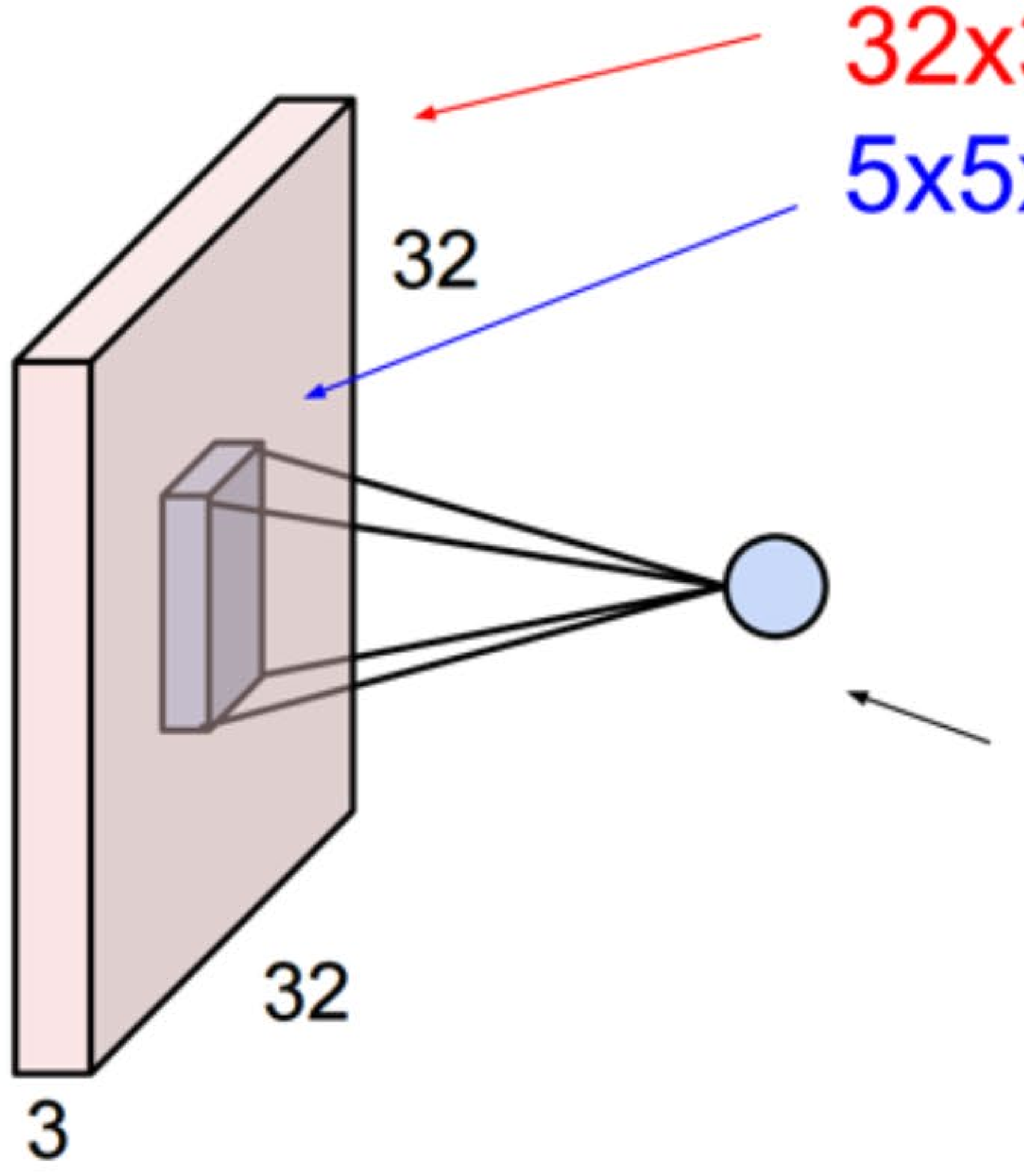
What will the output size be?

You will need to make some assumptions ...

Convolution Layer

32x32x3 image

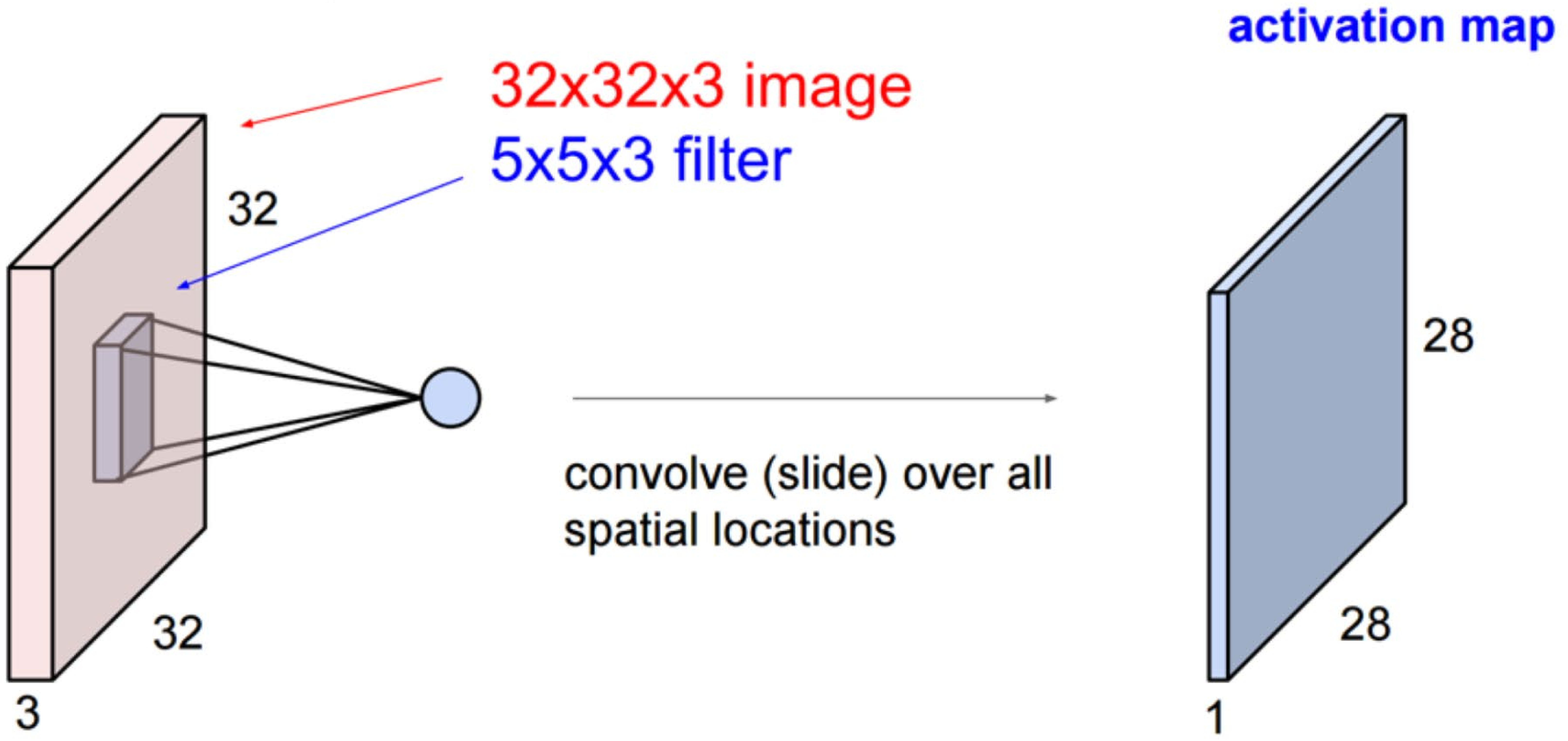
5x5x3 filter w



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

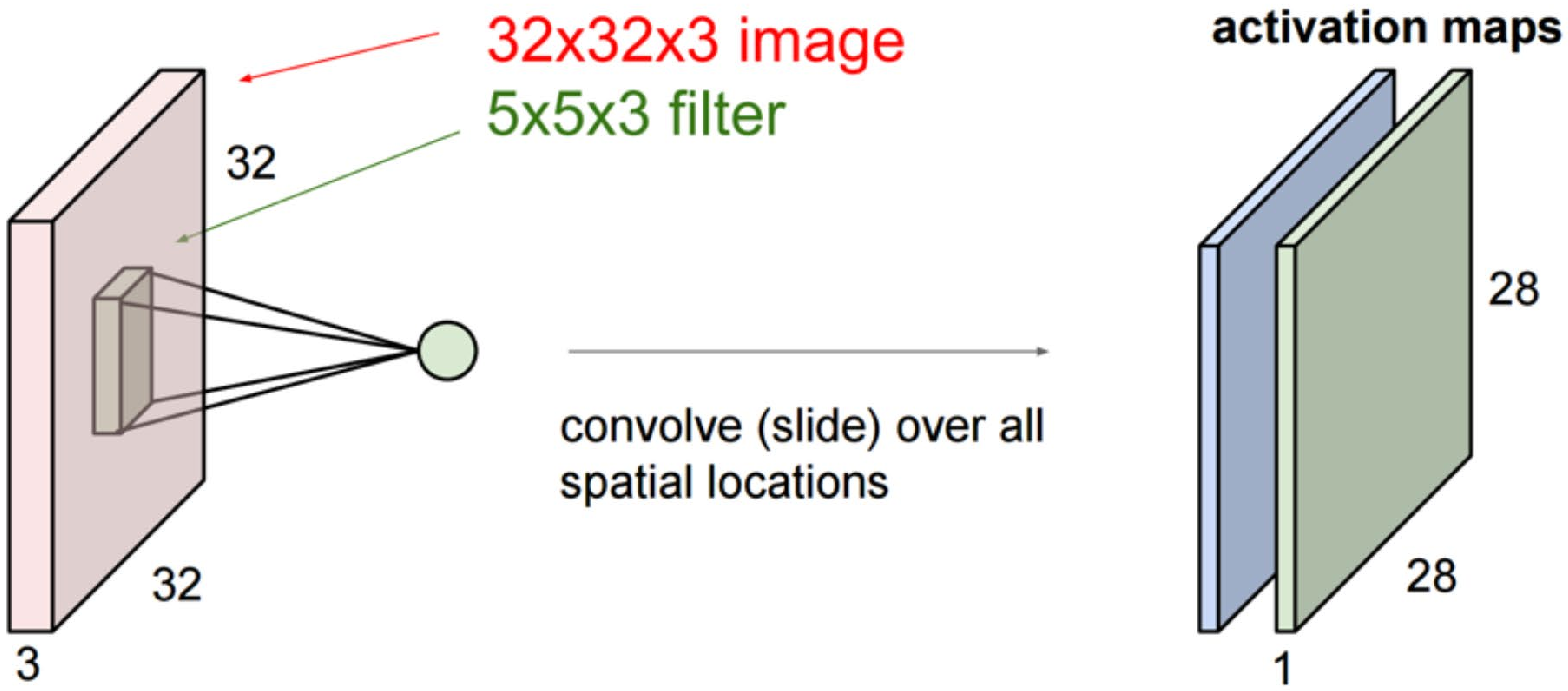
$$w^T x + b$$

Convolution Layer



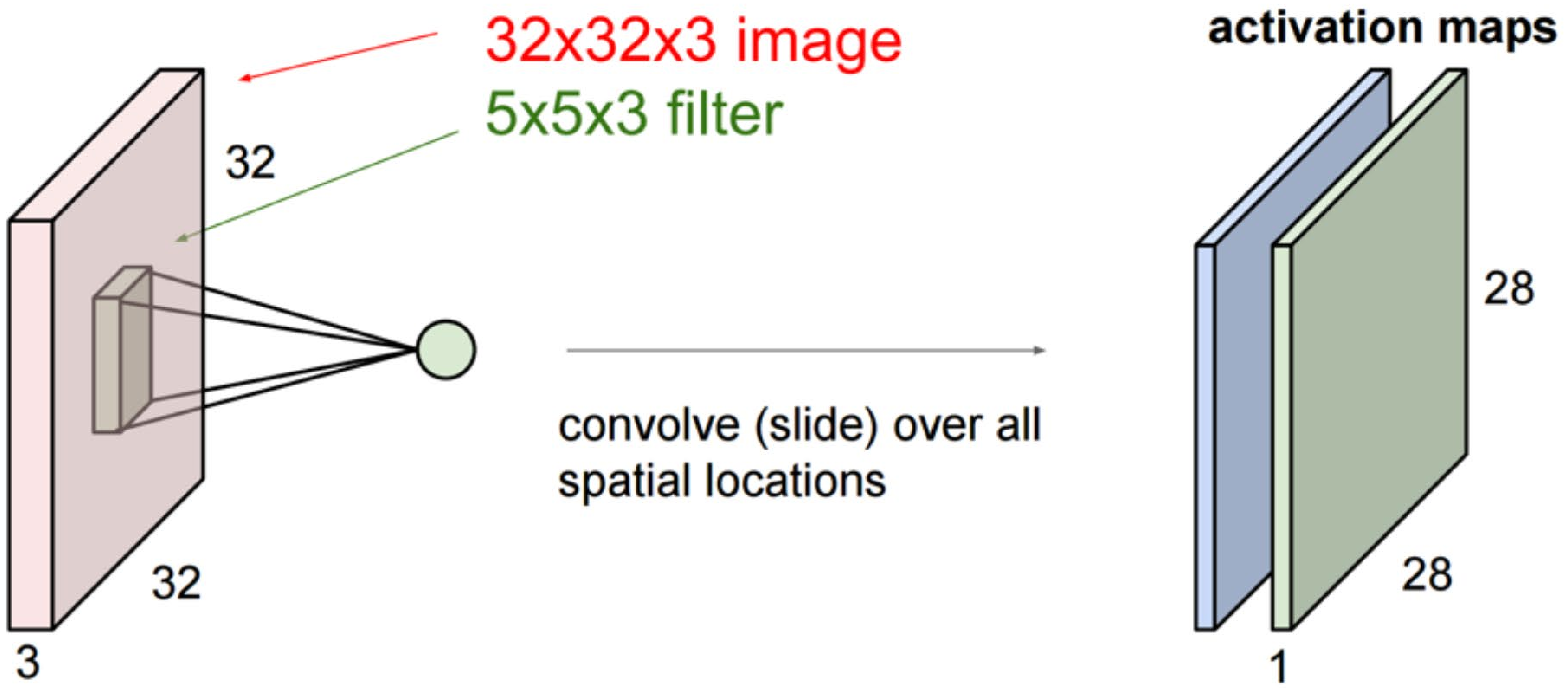
Convolution Layer

Consider a second filter ...

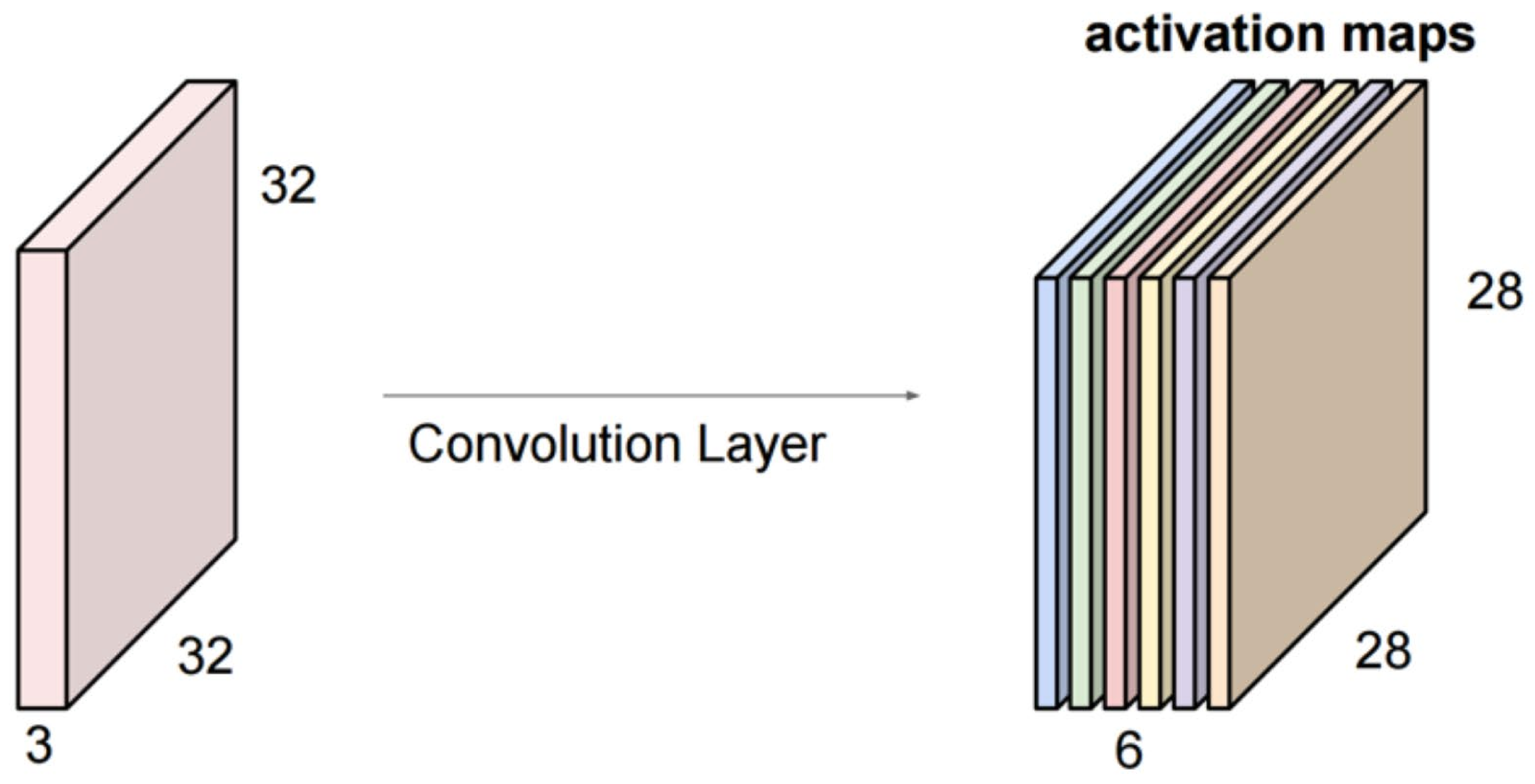


What will the output size be if we have 6 filters?

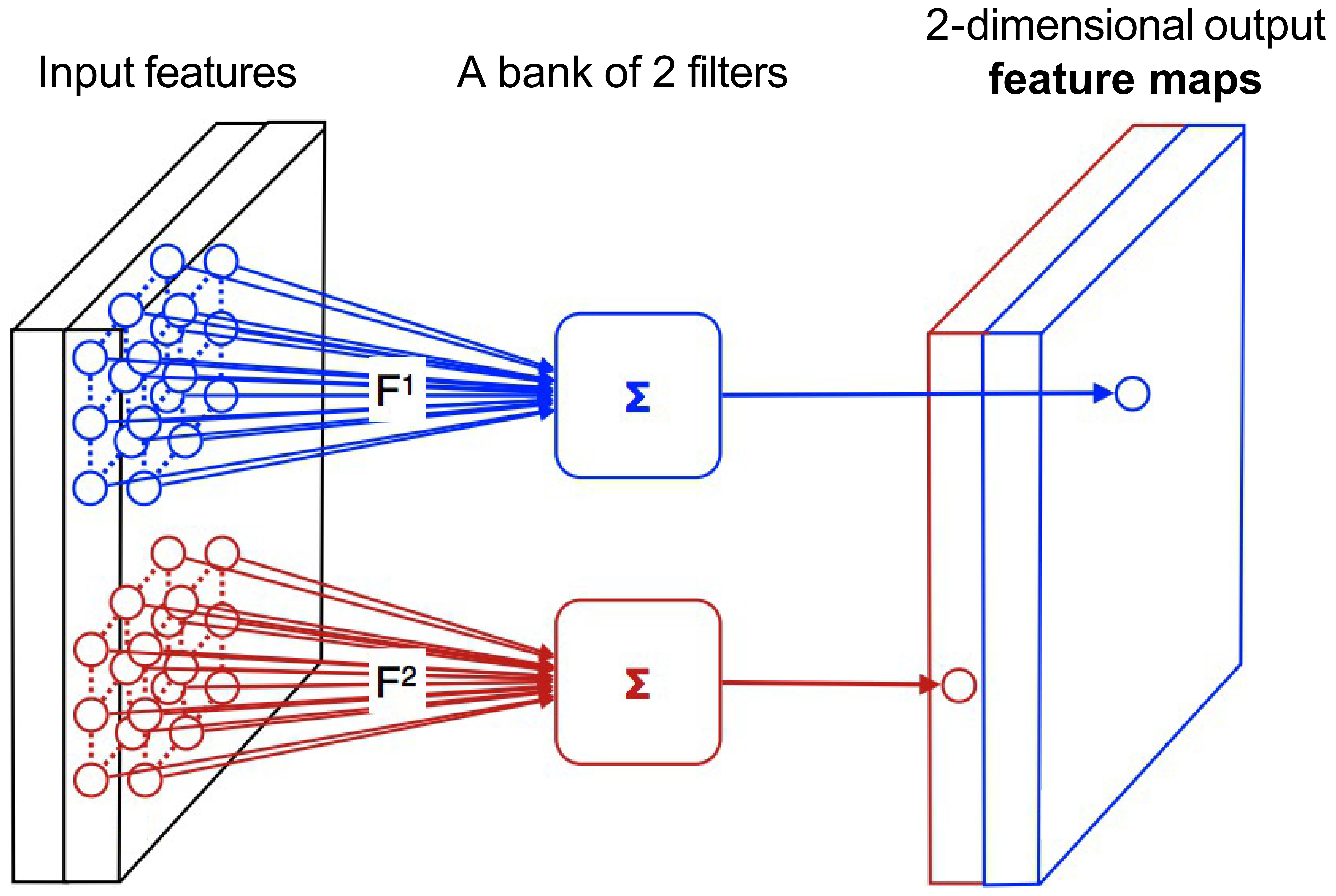
Convolution Layer



For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



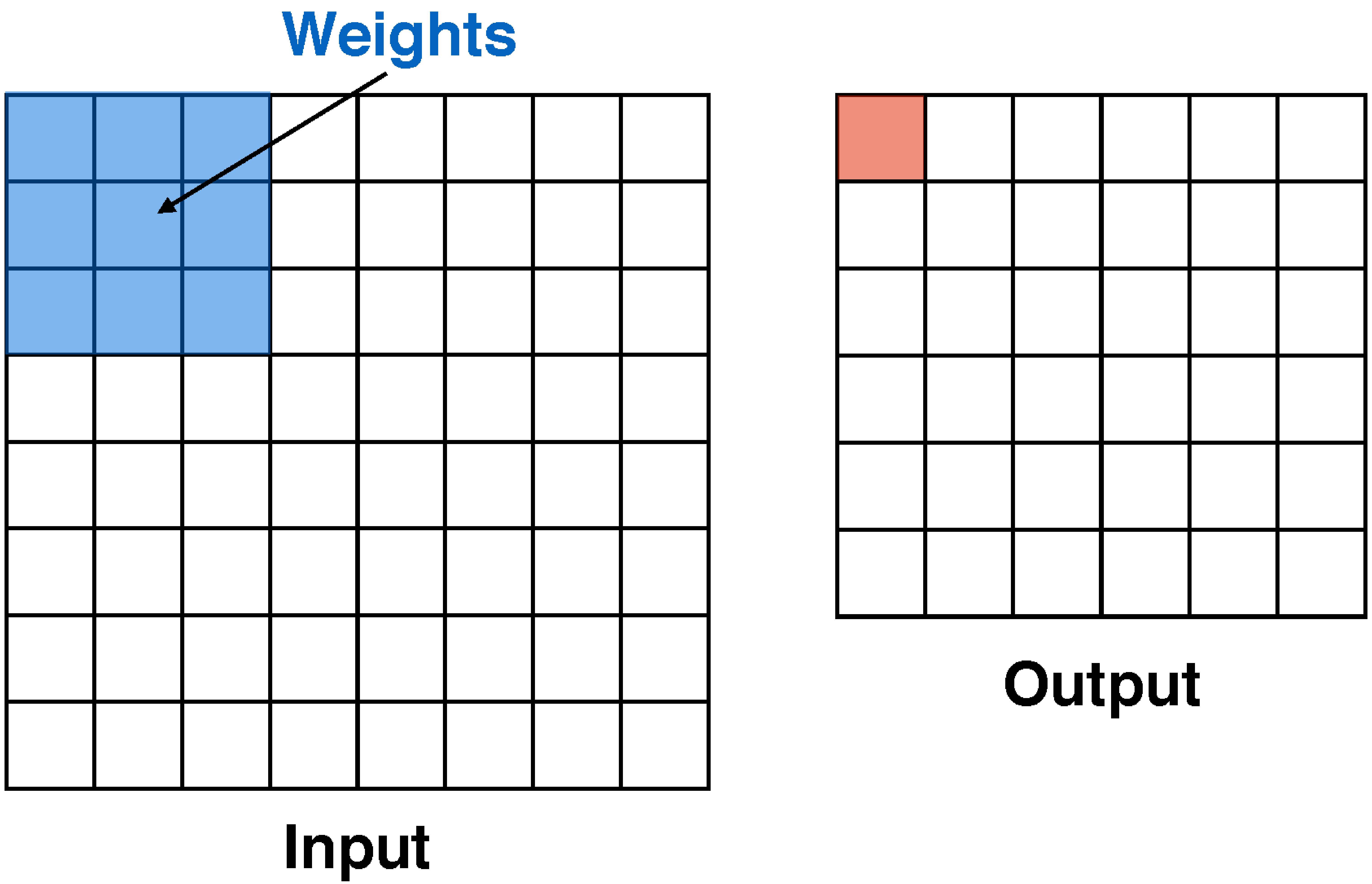
We stack these up to get a "new image" of size 28x28x6!



$$\mathbf{x}_{\mathtt{in}} \in \mathbb{R}^{C_{\mathtt{in}} \times H \times W} \rightarrow \mathbf{x}_{\mathtt{out}} \in \mathbb{R}^{C_{\mathtt{out}} \times H \times W}$$

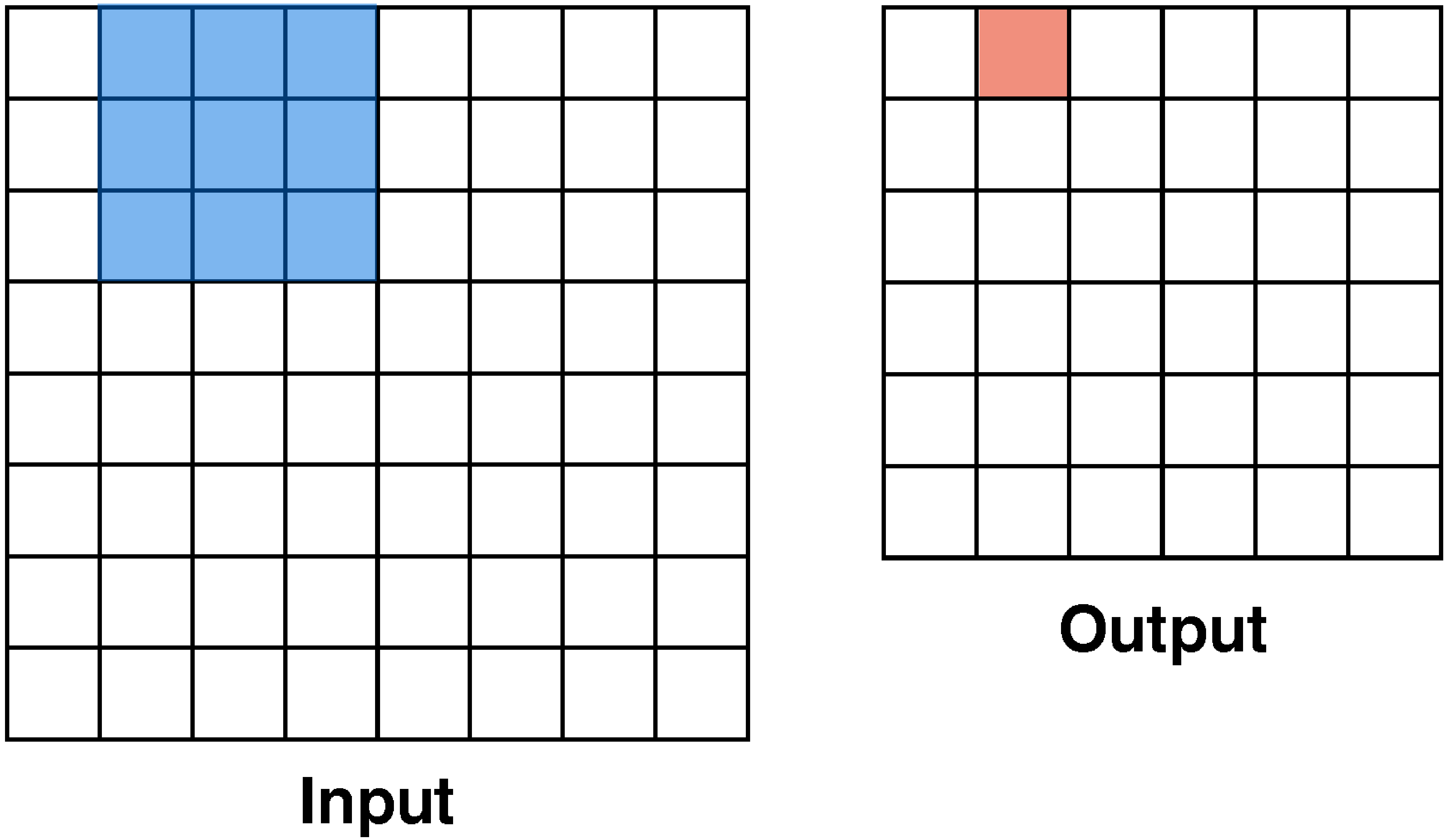
Convolution: Stride

During convolution, the weights "slide" along the input to generate each output



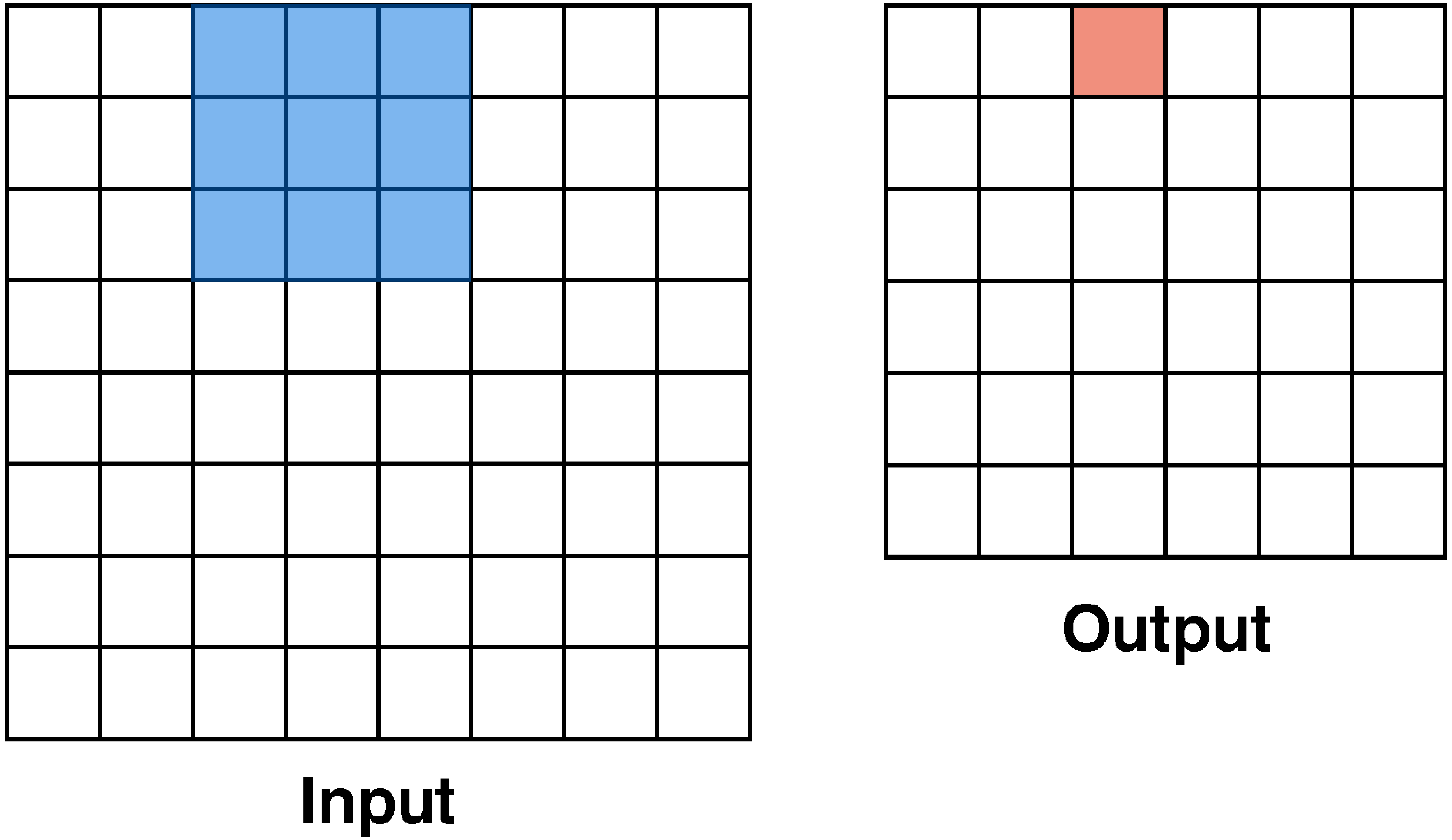
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



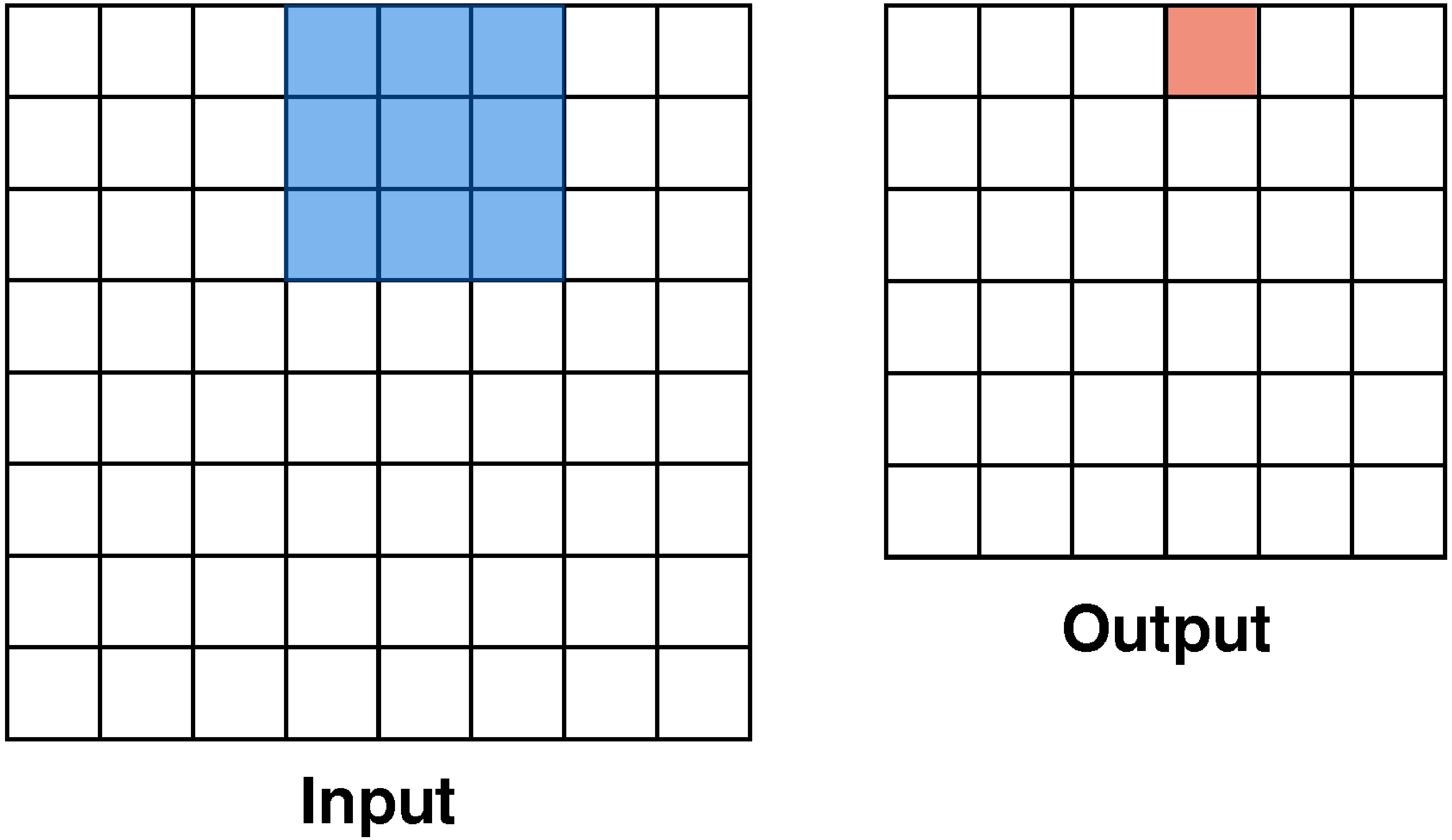
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



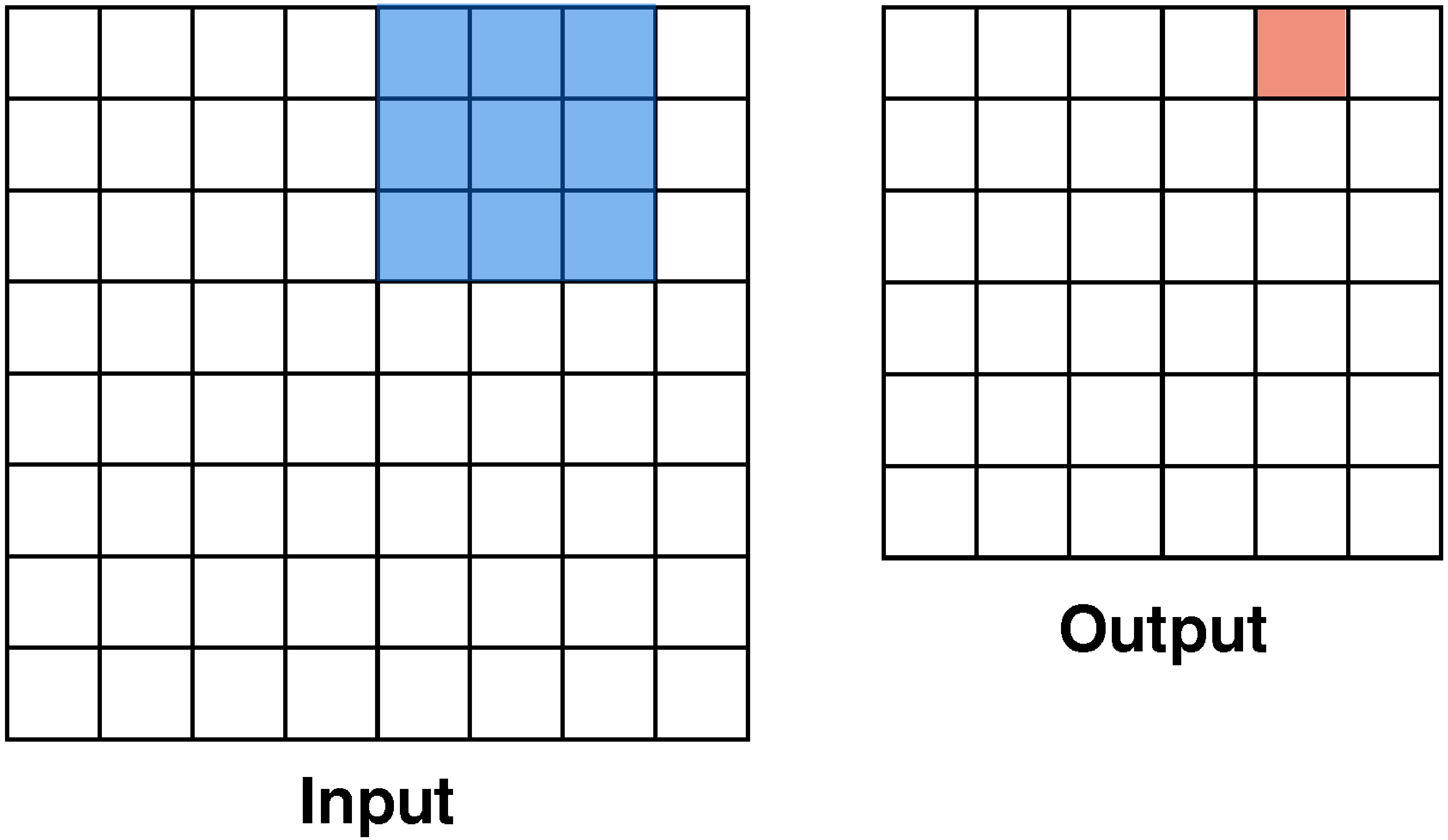
Convolution: Stride

During convolution, the weights "slide" along the input to generate each output



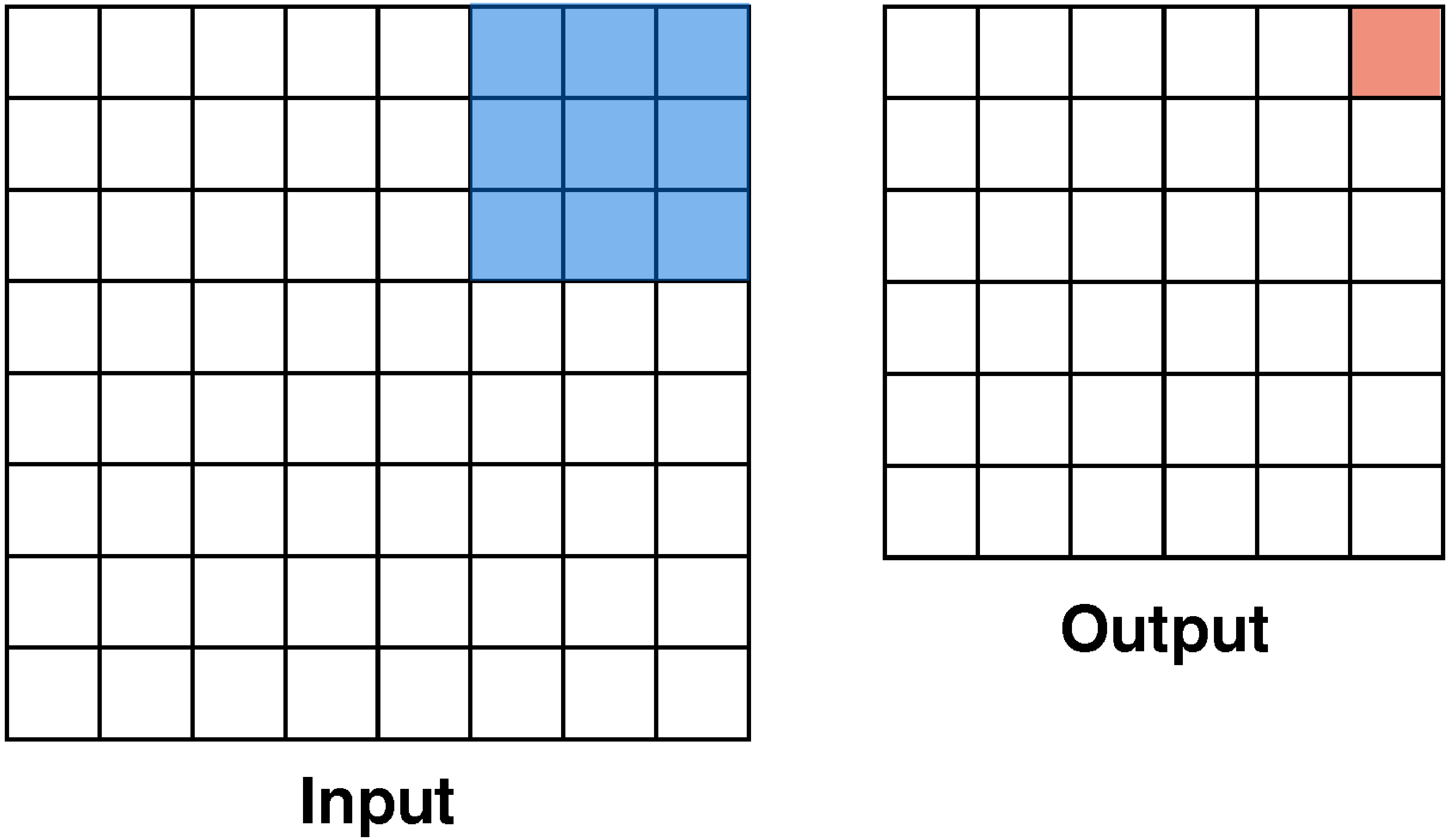
Convolution: Stride

During convolution, the weights “slide” along the input to generate each output



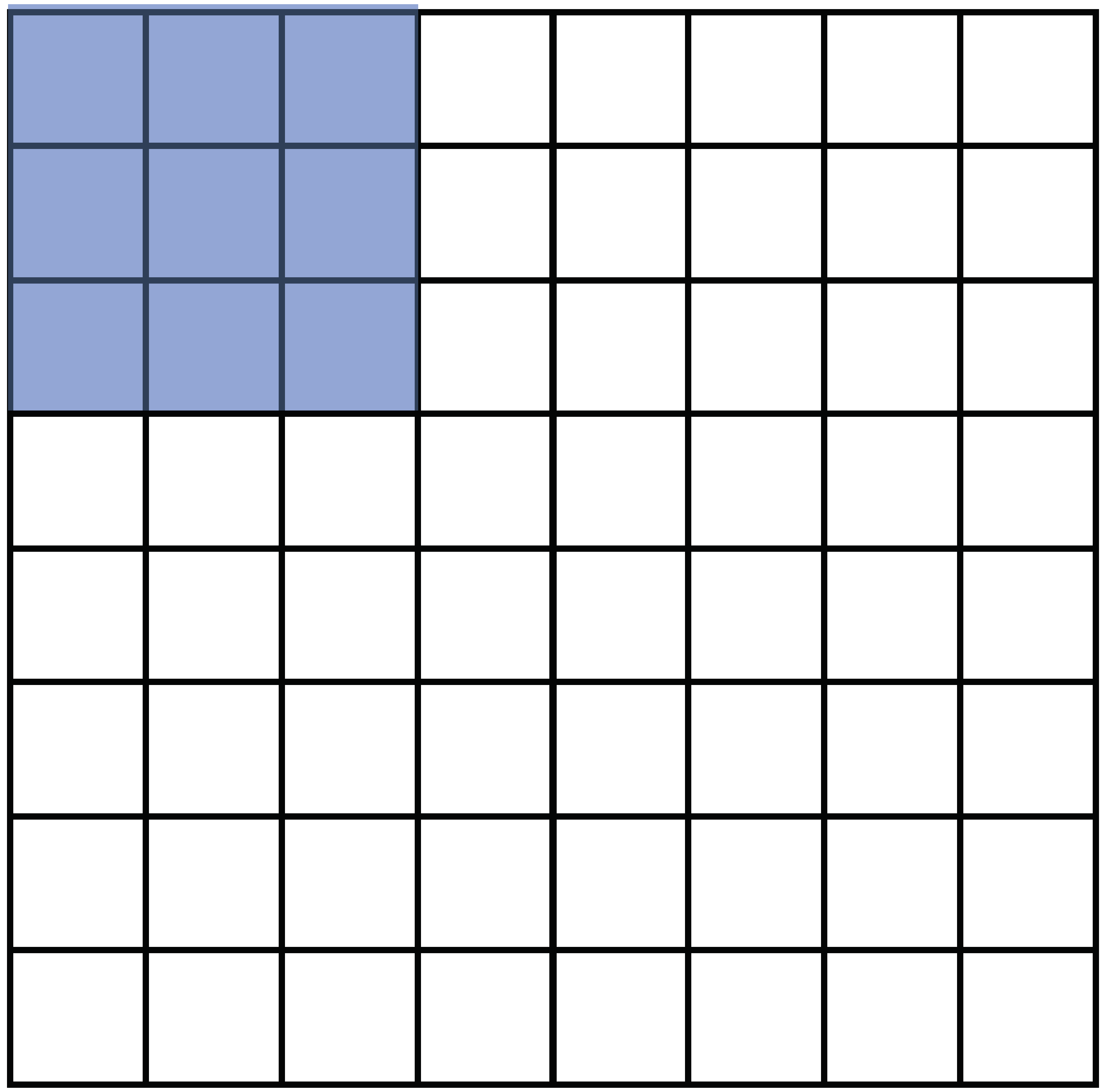
Convolution: Stride

During convolution, the weights "slide" along the input to generate each output



Convolution: Stride

During convolution, the weights "slide" along the input to generate each output



Input

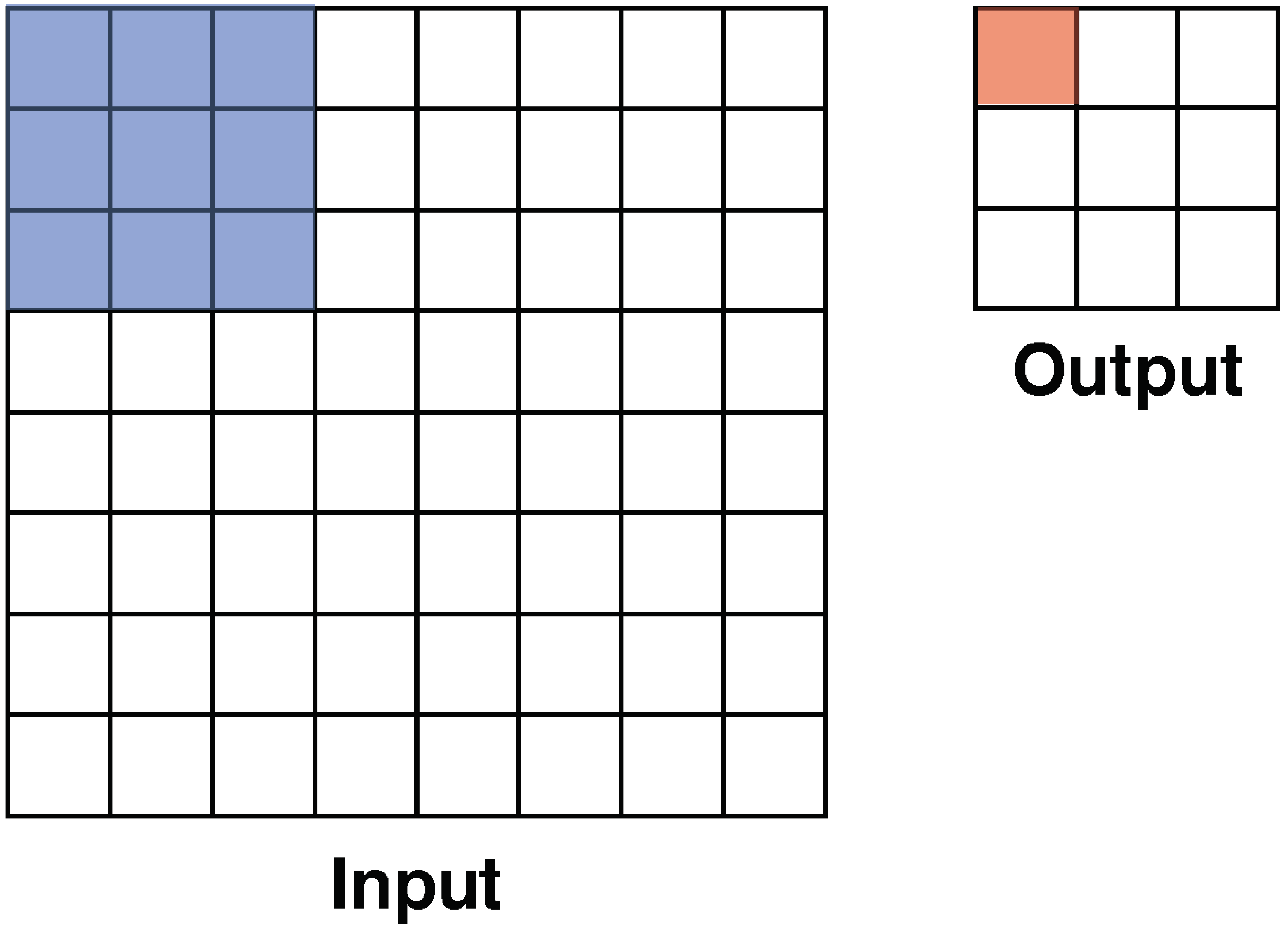
Recall that at each position, we are doing a **3D** sum:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

(channel, row, column)

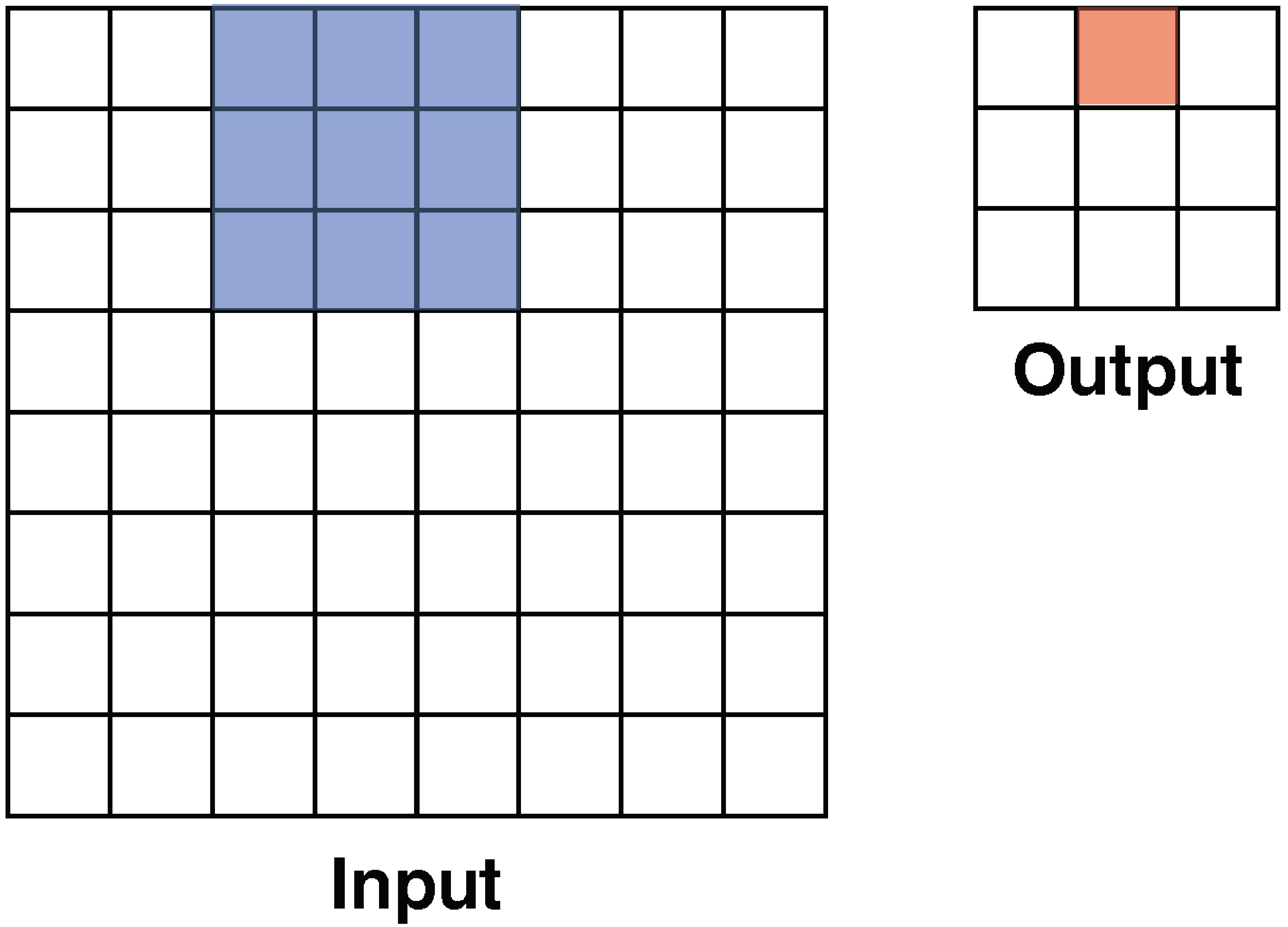
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



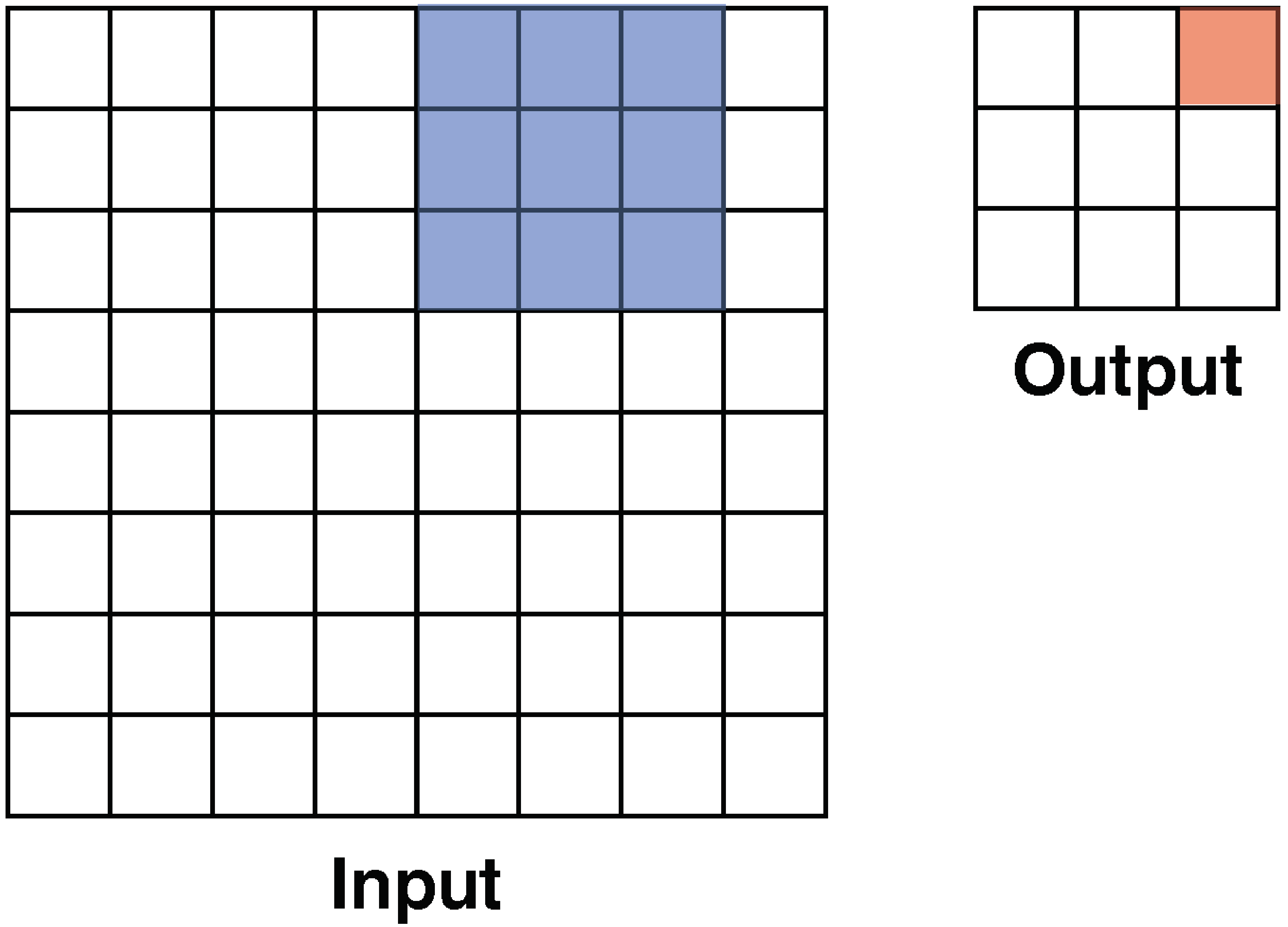
Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



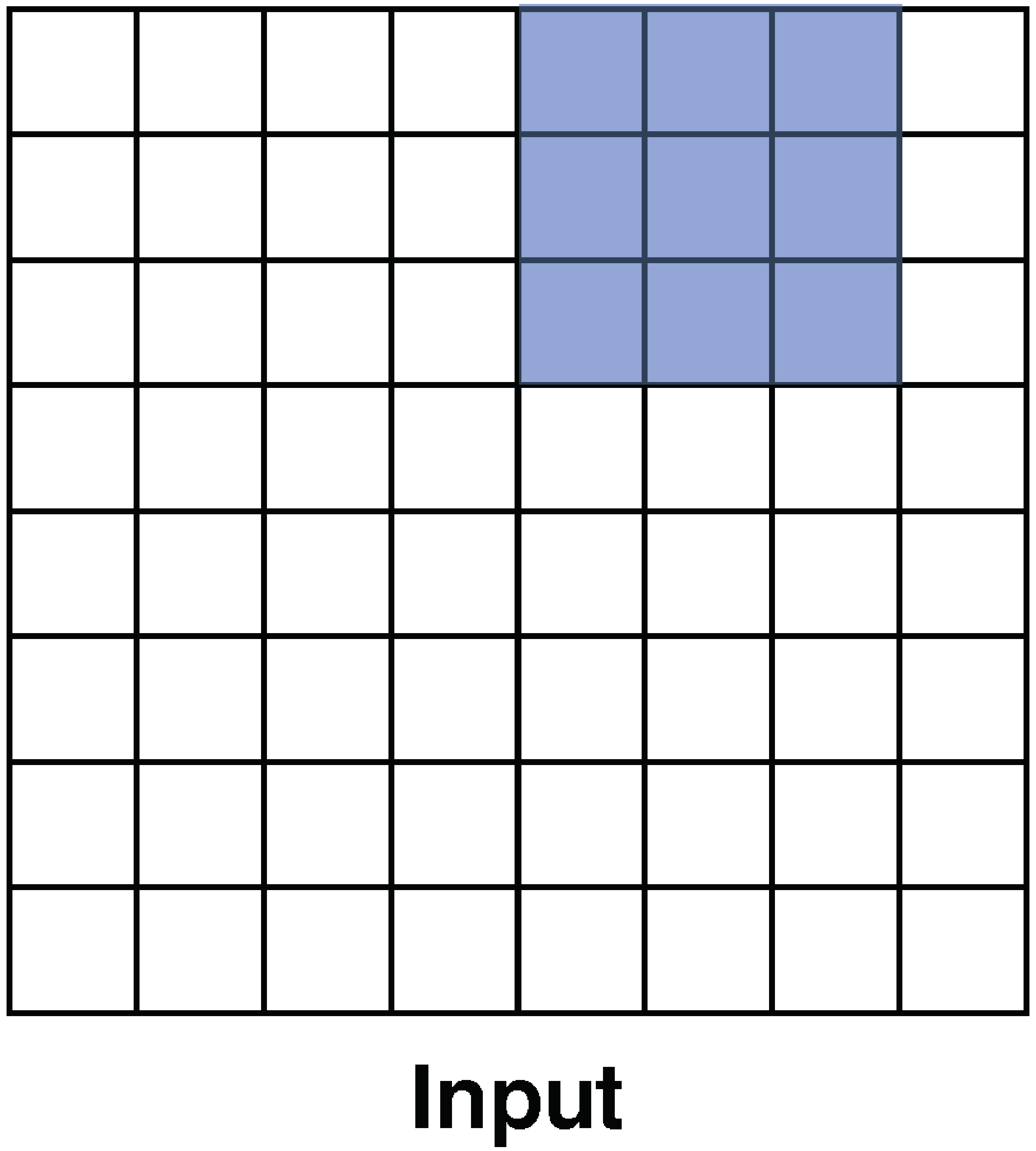
Convolution: Stride

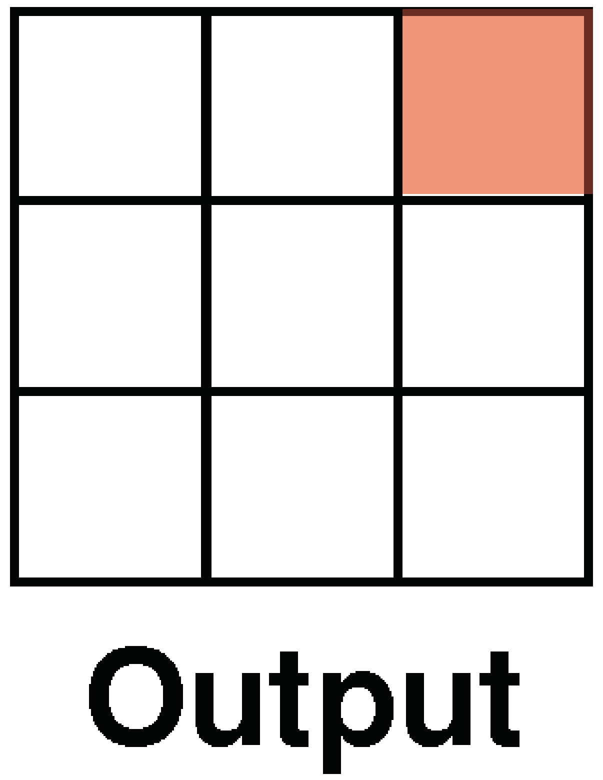
But we can also convolve with a **stride**, e.g. stride = 2



Convolution: Stride

But we can also convolve with a **stride**, e.g. stride = 2



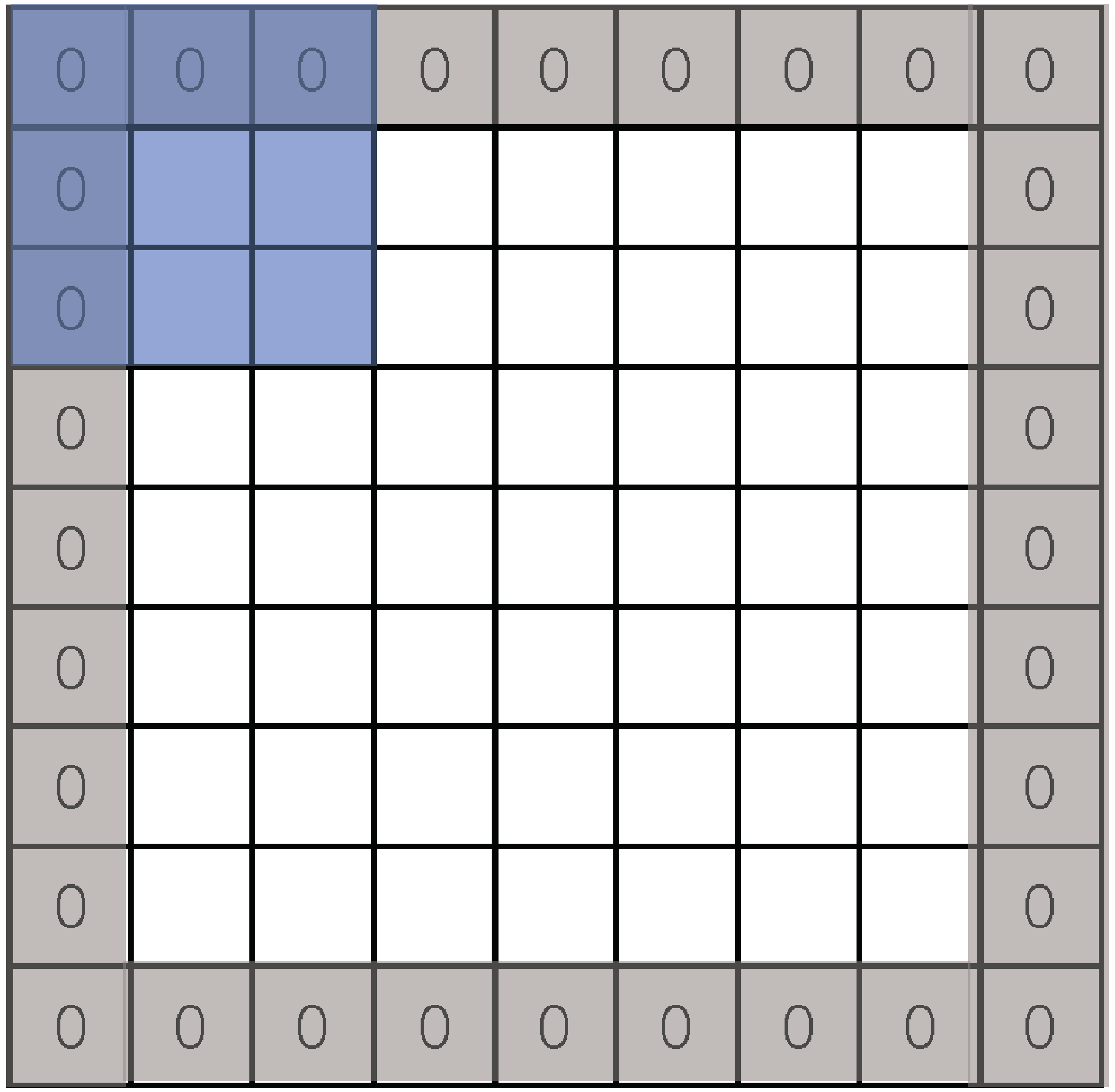


- Notice that with certain strides, we may not be able to cover all of the input

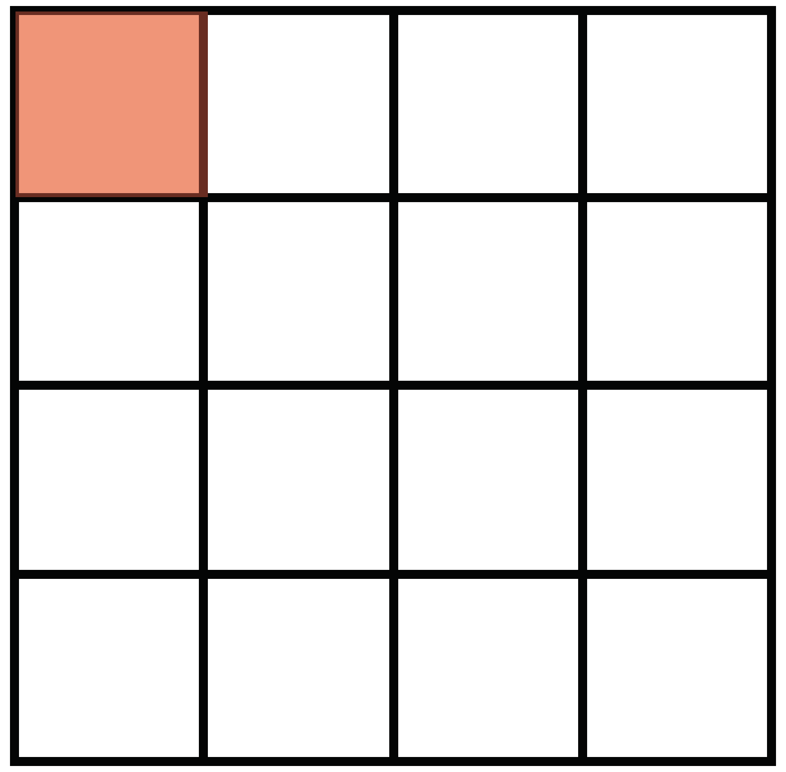
- The output is also half the size of the input

Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



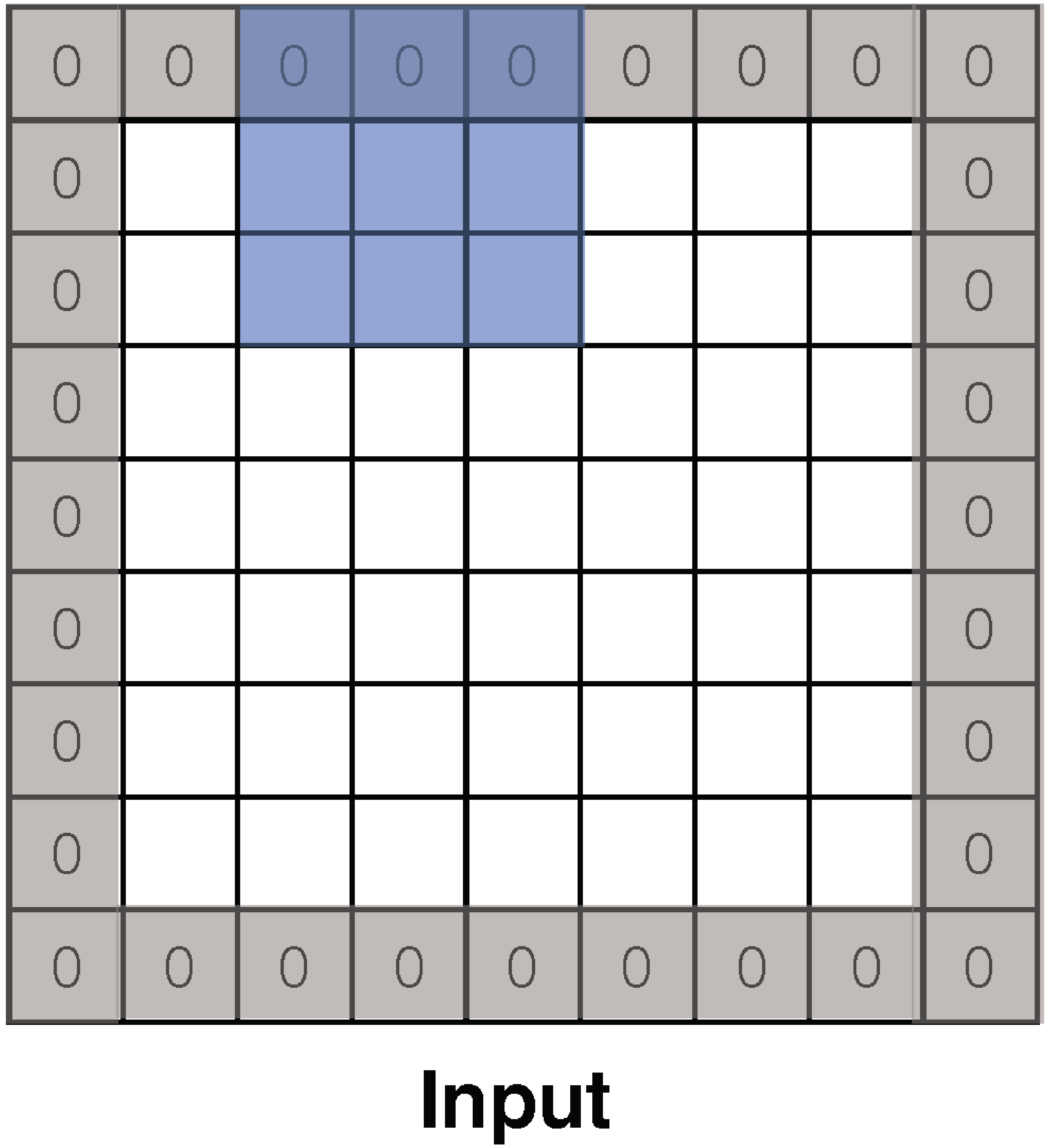
Input

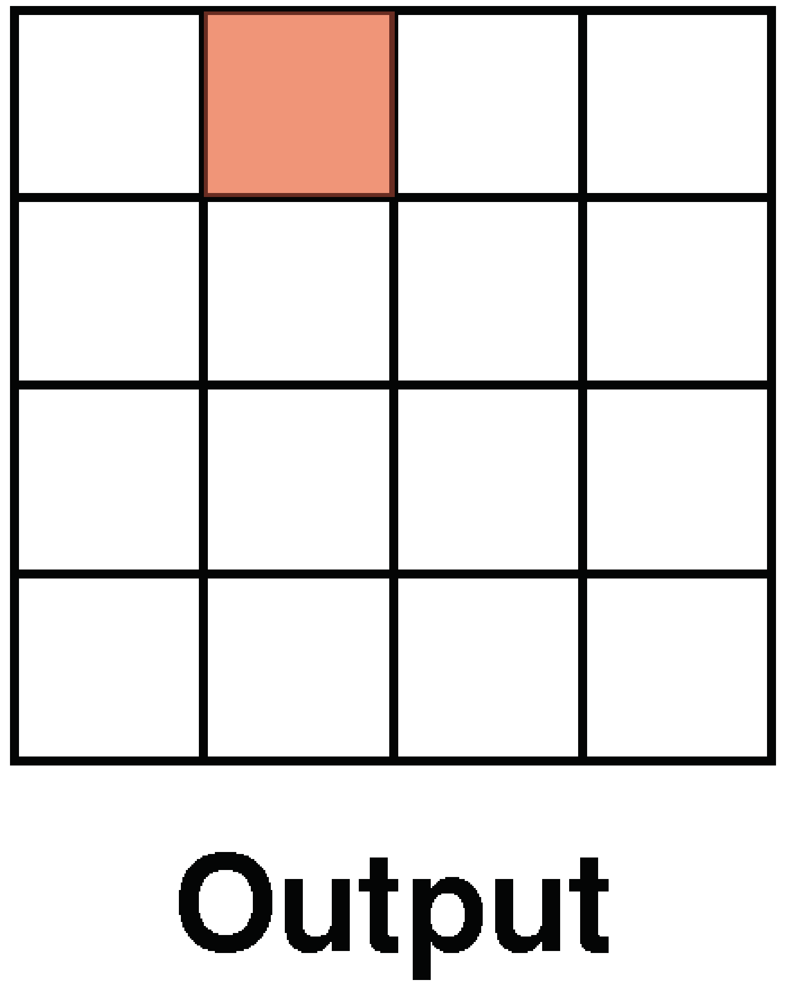


Output

Convolution: Padding

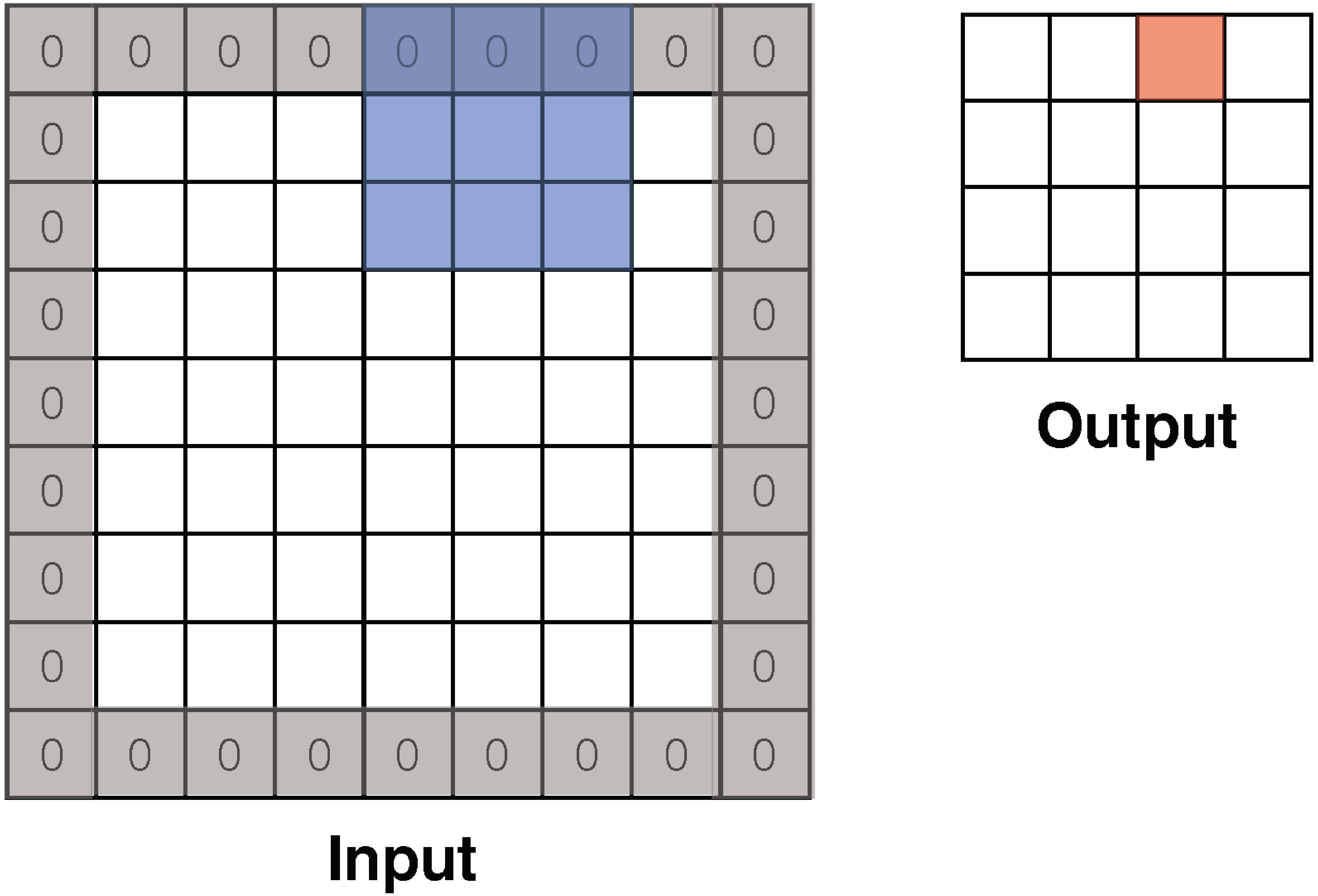
We can also pad the input with zeros.
Here, **pad = 1, stride = 2**





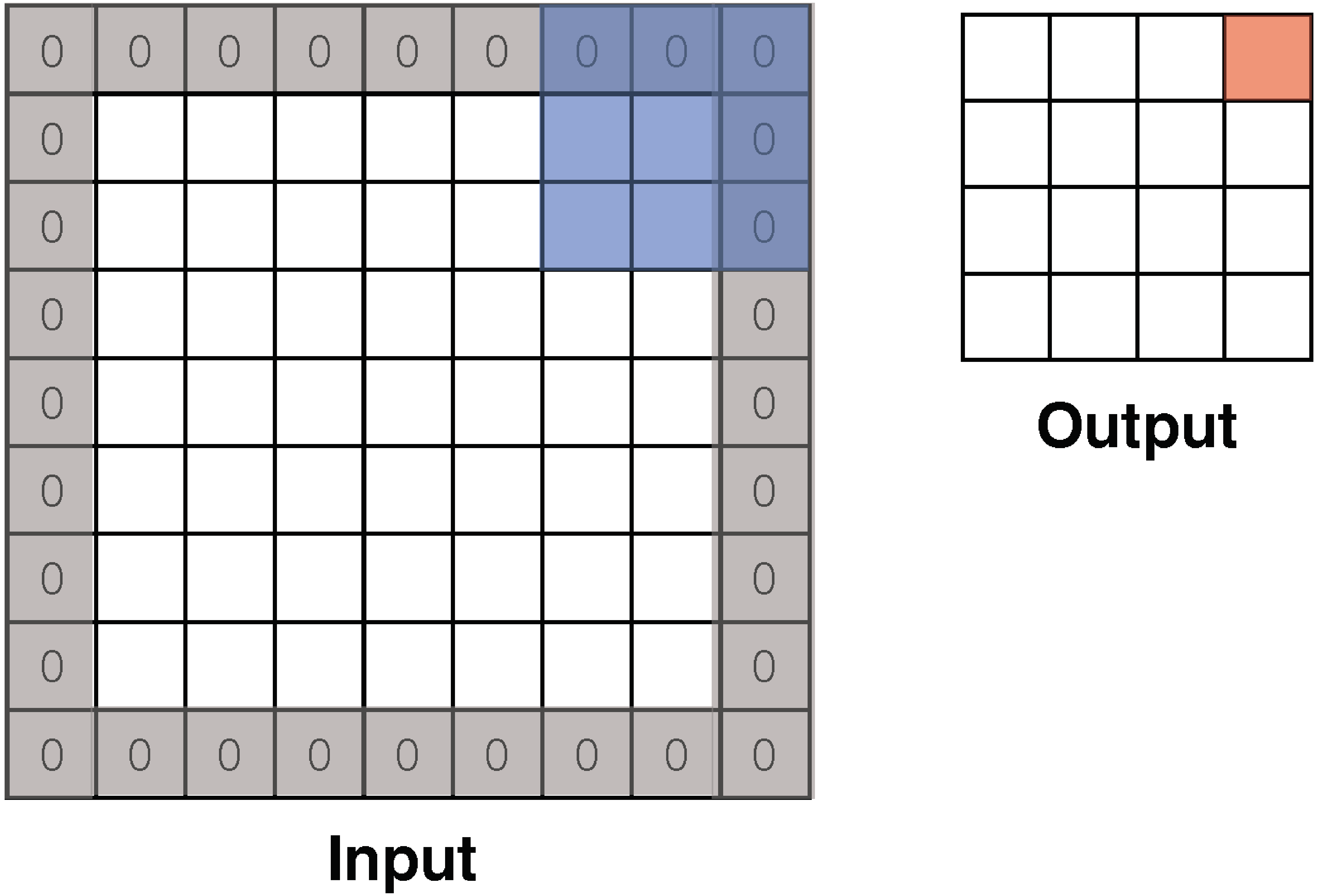
Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



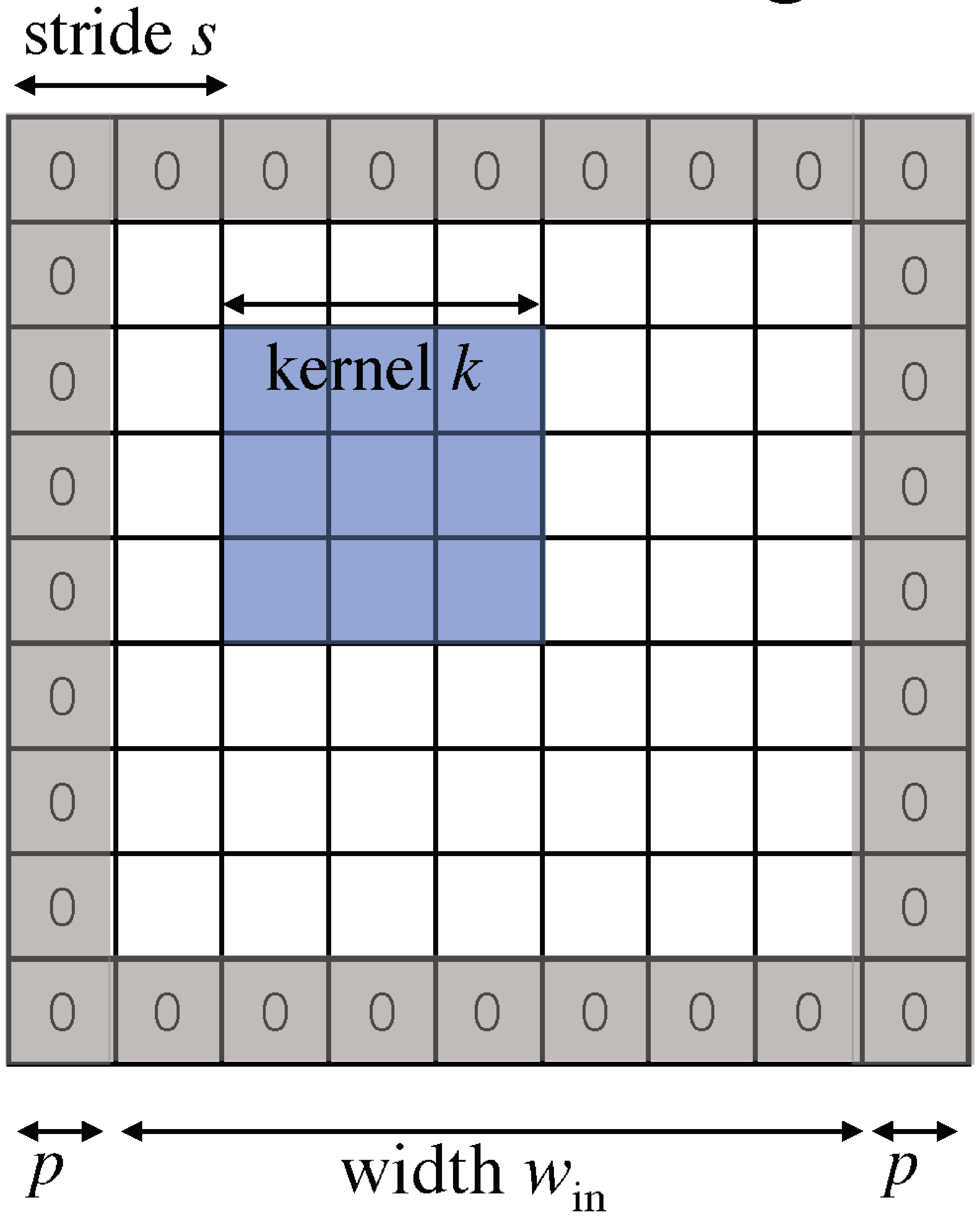
Convolution: Padding

We can also pad the input with zeros.
Here, **pad = 1, stride = 2**



Convolution:

How big is the output?

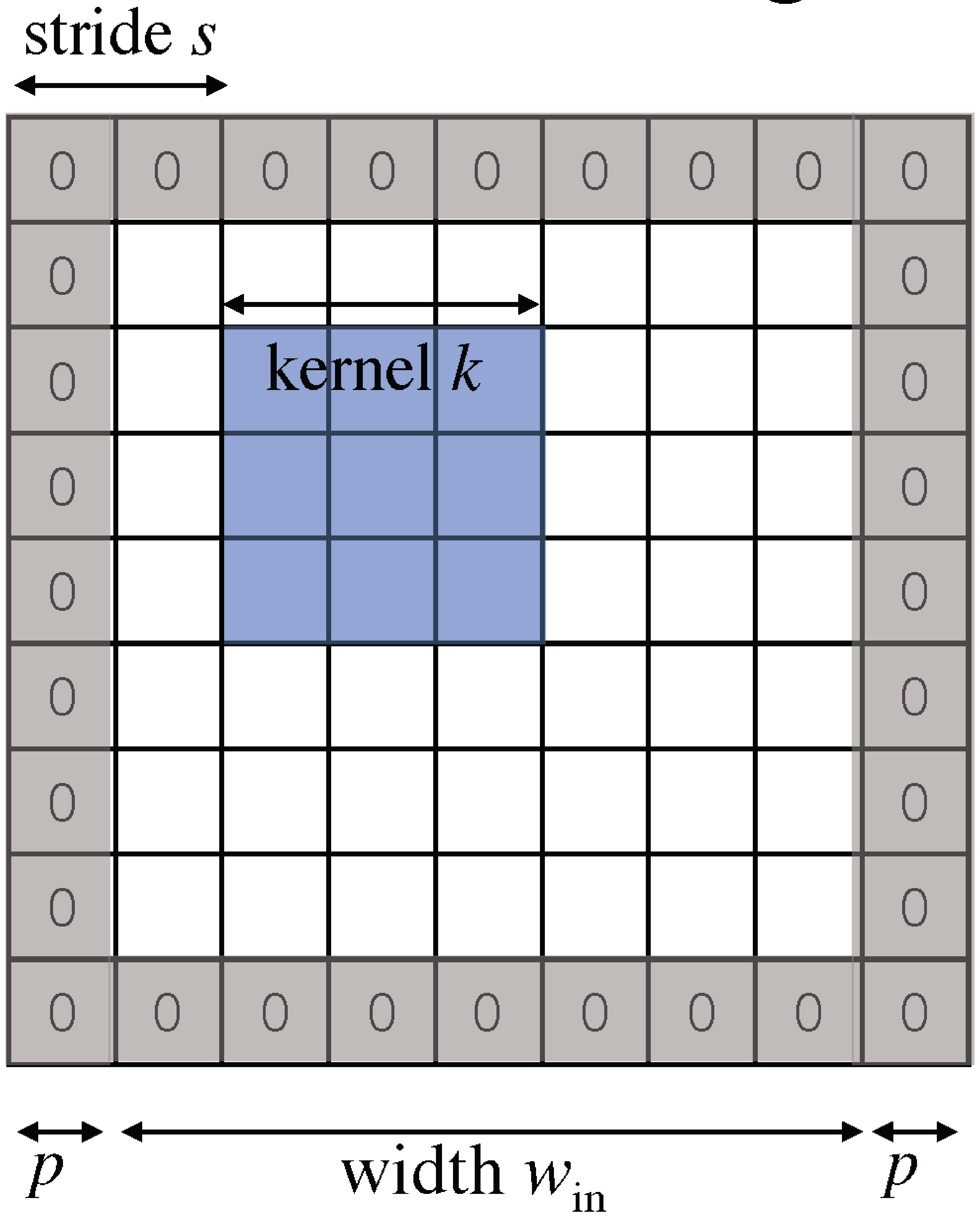


In general, the output has size:

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

Convolution:

How big is the output?



Example: k=3, s=1, p=1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$

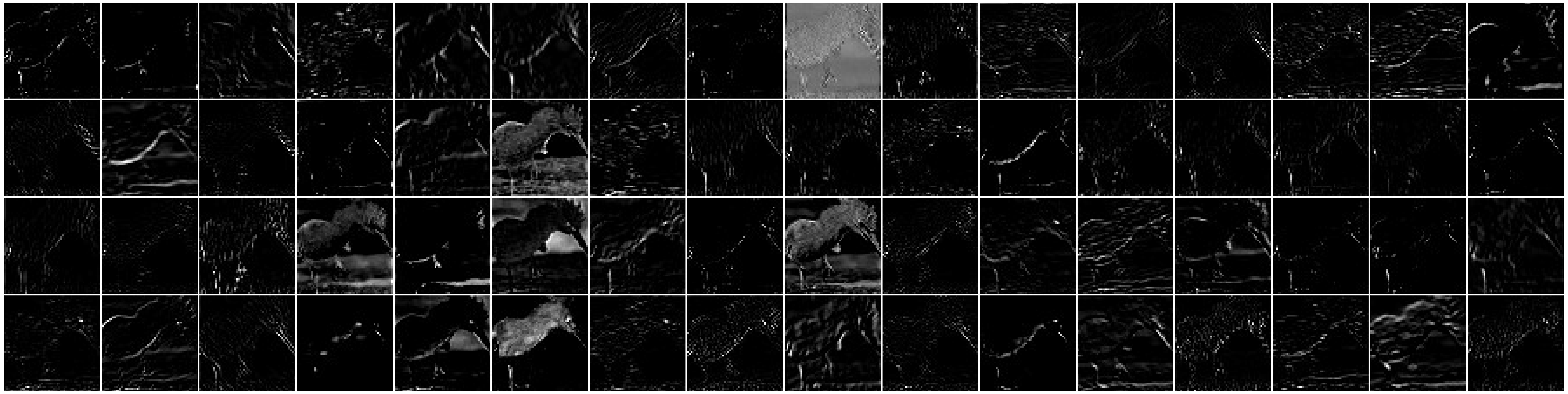
$$= w_{in}$$

VGGNet [Simonyan 2014] uses filters of this shape

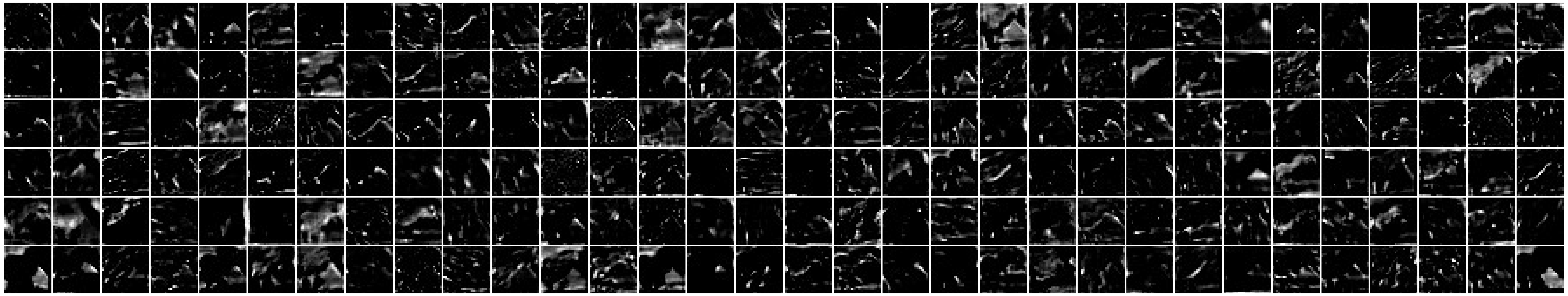
Feature maps

Input

conv1 (after first conv layer)

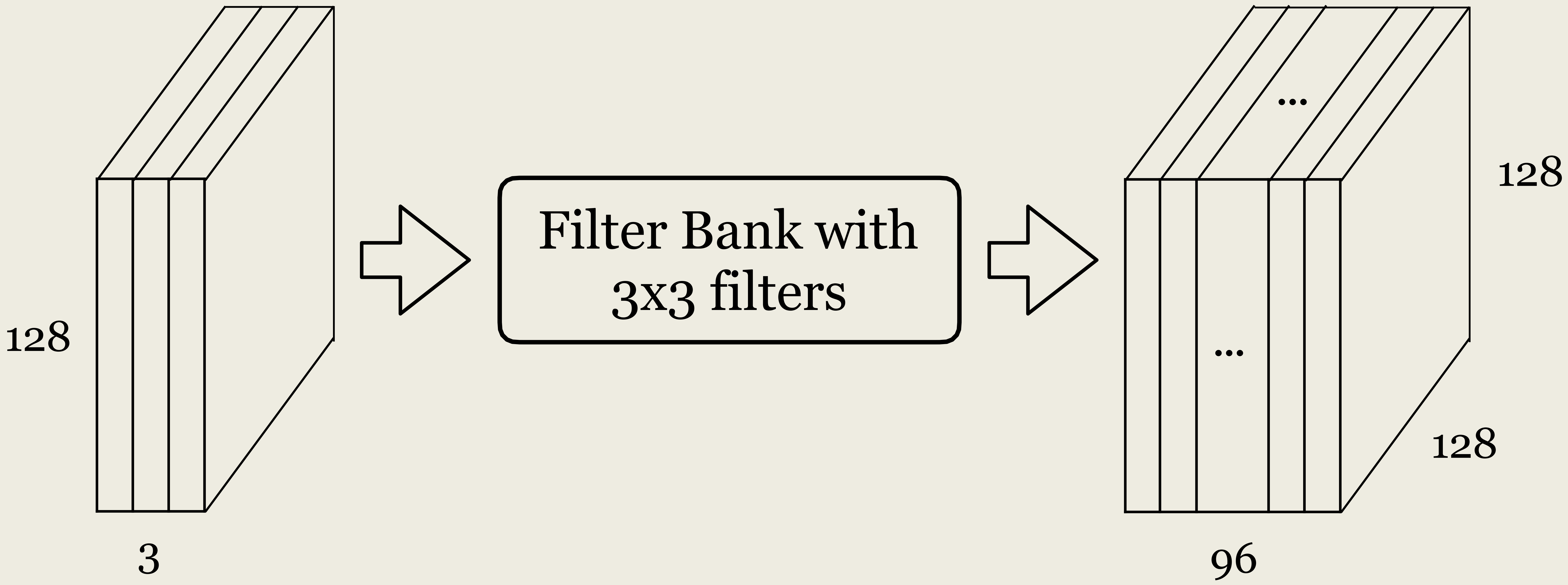


conv2 (after second conv layer)



- Each layer can be thought of as a set of C **feature maps** aka **channels**
- Each feature map is an NxM image

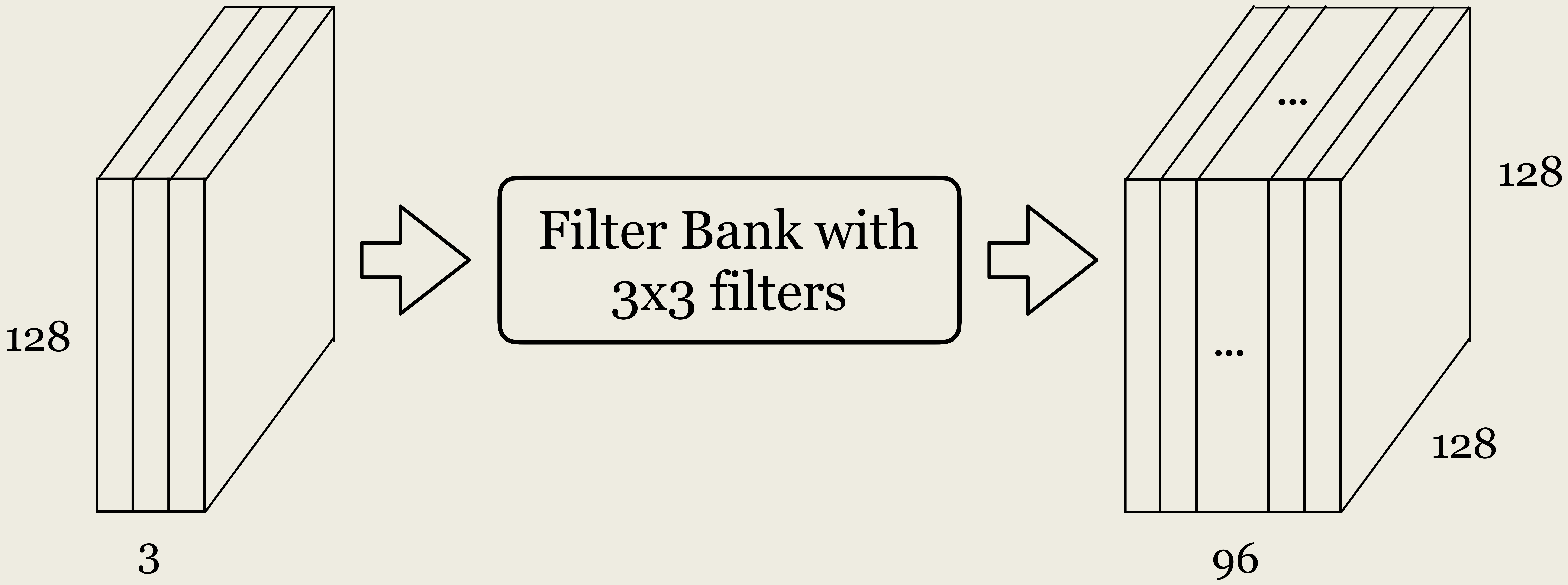
Knowledge Check ...



How many parameters does each *filter* have?

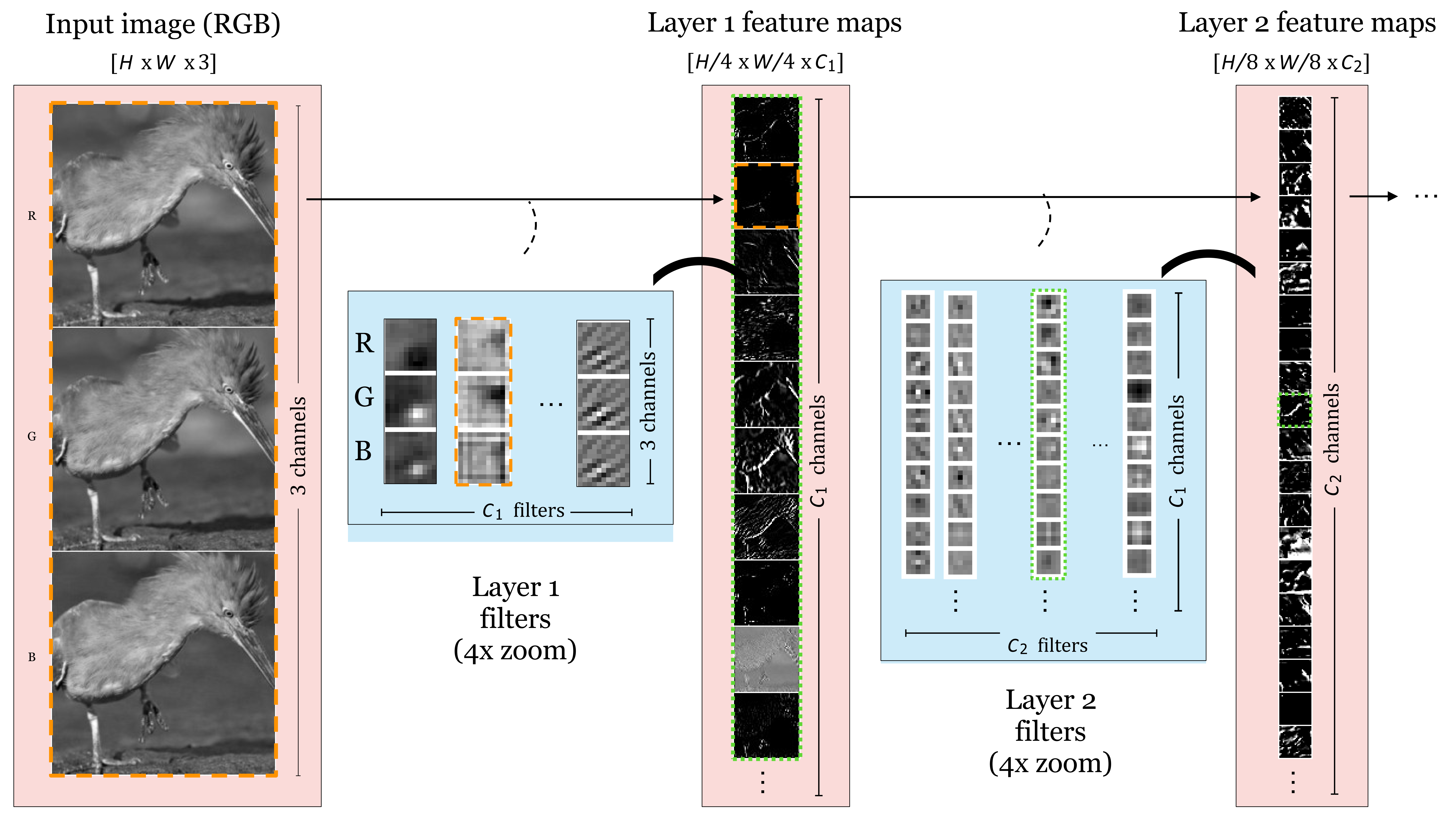
(a) 9 (b) 27 (c) 96 (d) 864

Knowledge Check ...



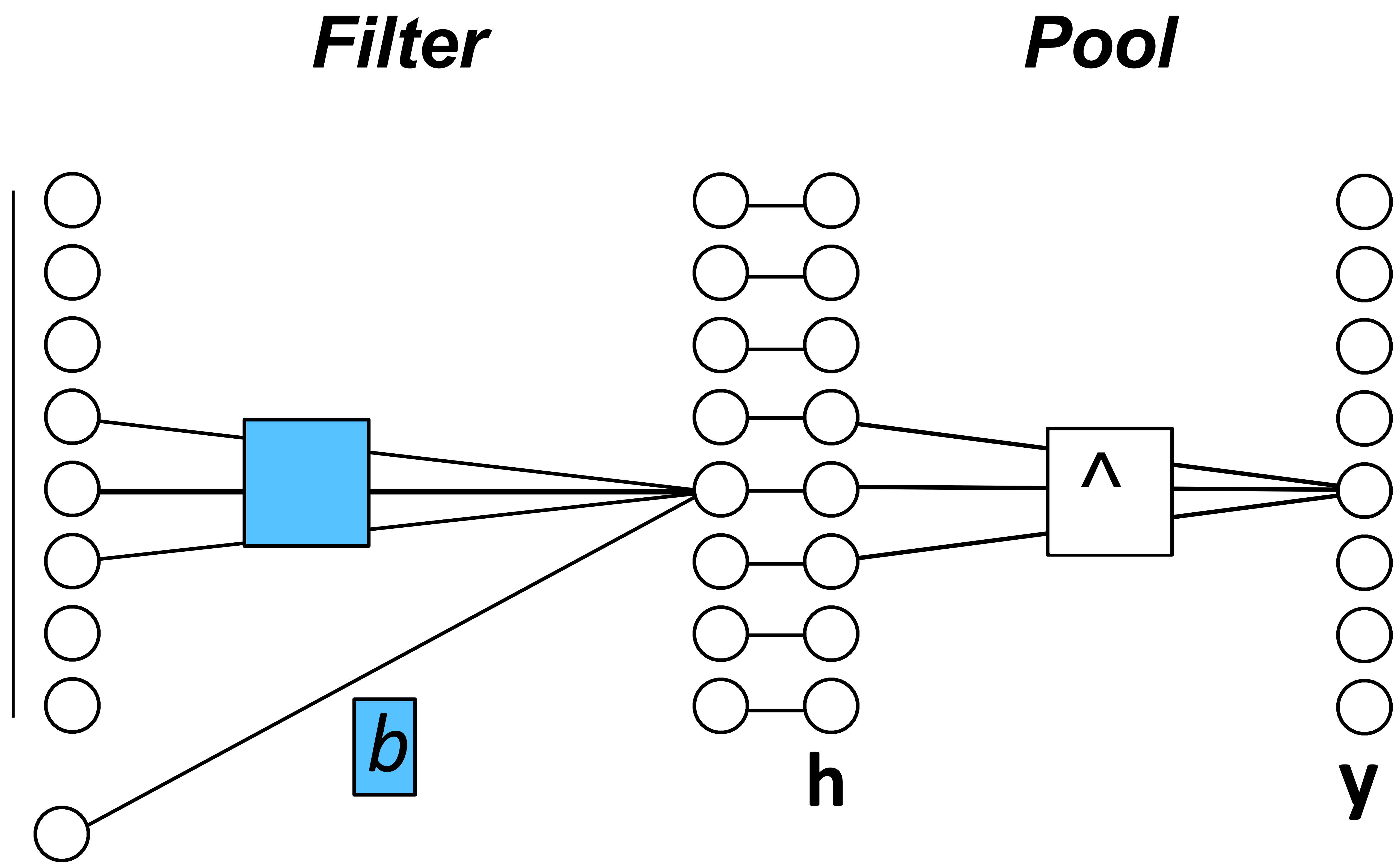
How many filters are in the bank?

(a) 3 (b) 27 (c) 96 (d) can't say



Input image (RGB)
[H x W x 3]
R
G
B
3 channels
Layer 1 feature maps
[$H/4$ x $W/4$ x C_1]
C_1 channels
Layer 2 feature maps
[$H/8$ x $W/8$ x C_2]
C_2 channels
R
G
B
3 channels
C_1 filters
Layer 1 filters (4x zoom)
C_1 channels
C_2 filters
Layer 2 filters (4x zoom)

Pooling



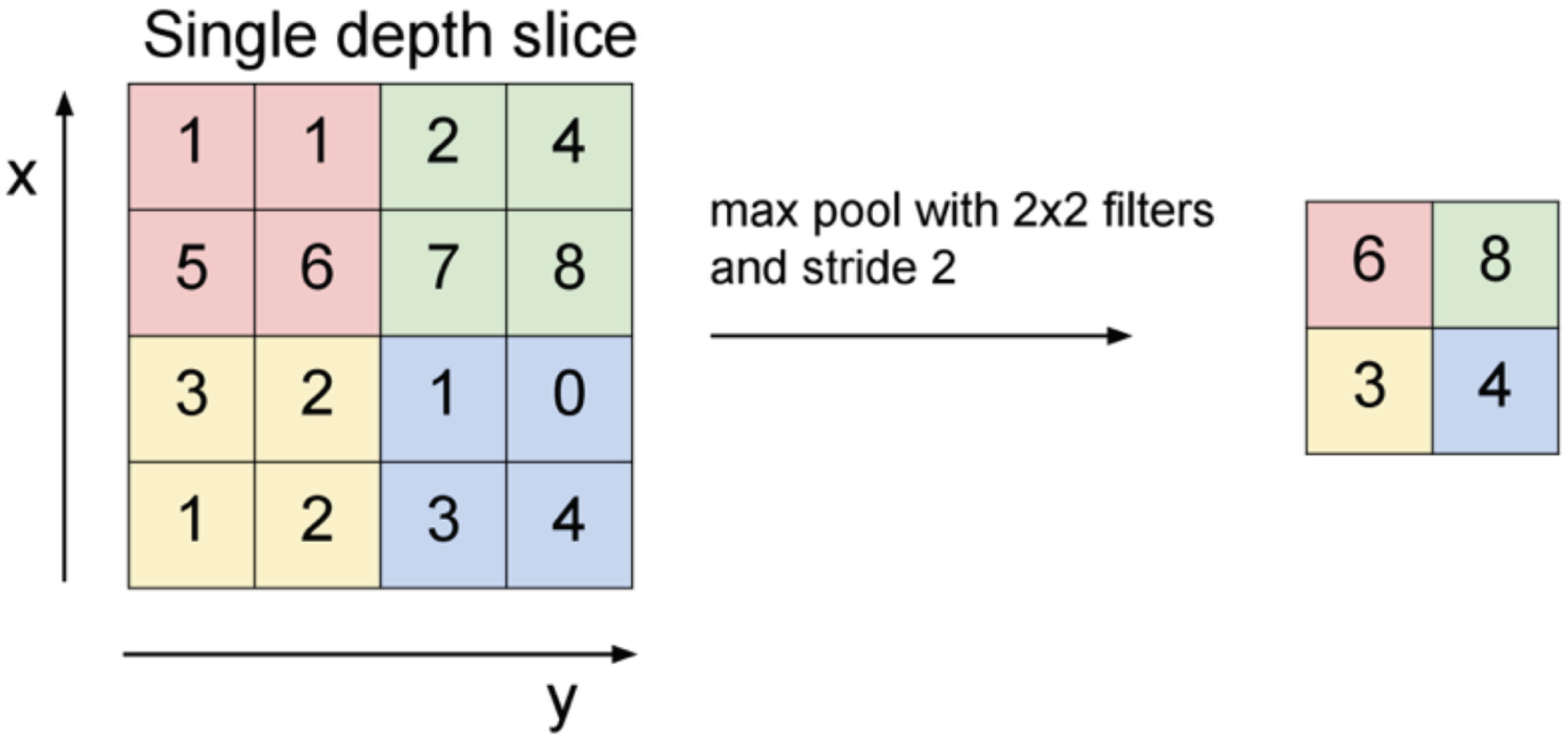
Max pooling

$$y_j = \max_{j \in \mathcal{N}(j)} h_j$$

Mean pooling

$$y_j = \frac{1}{|\mathcal{N}|} \sum_{j \in \mathcal{N}(j)} h_j$$

Max Pooling



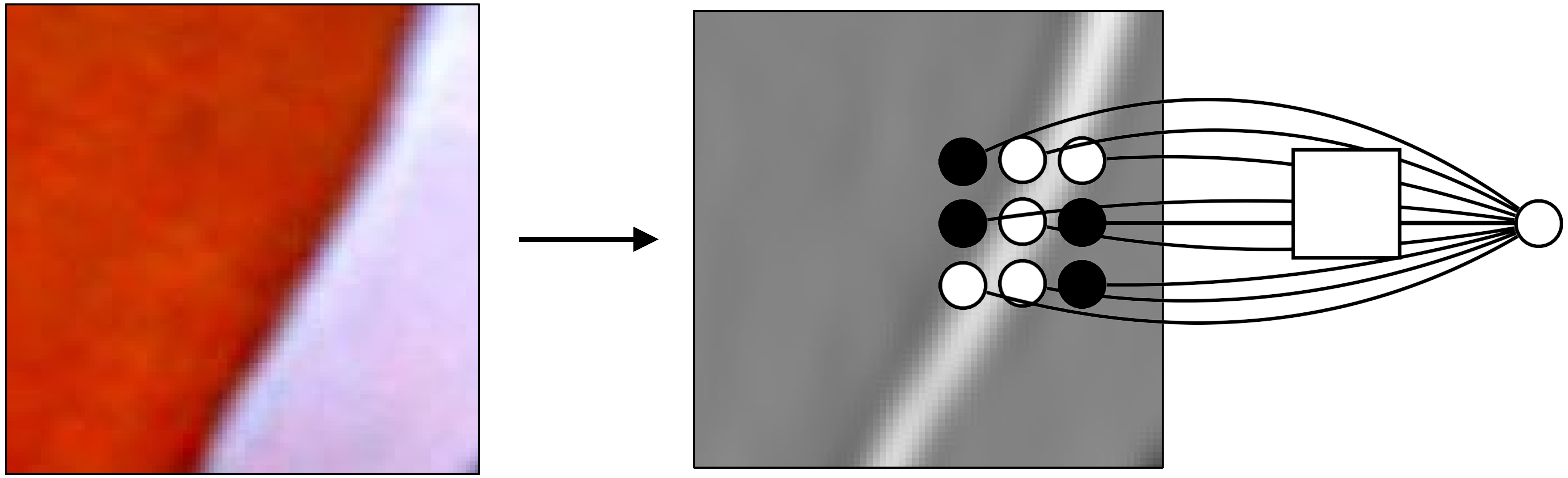
What's the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

Figure: Andrej Karpathy

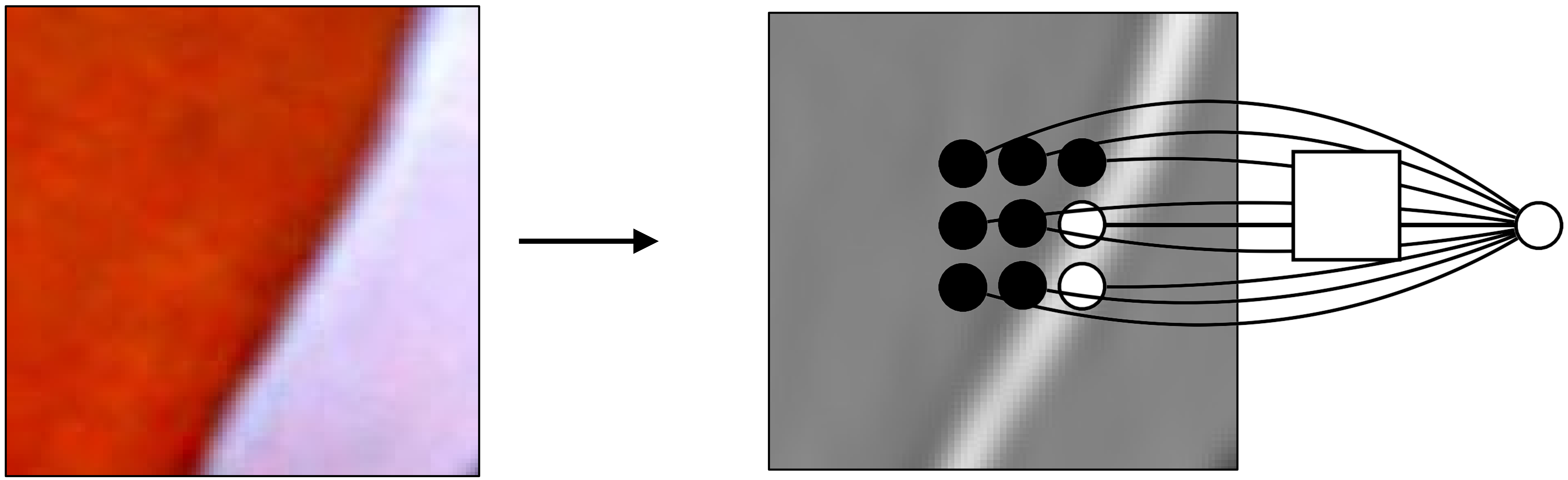
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



Pooling — Why?

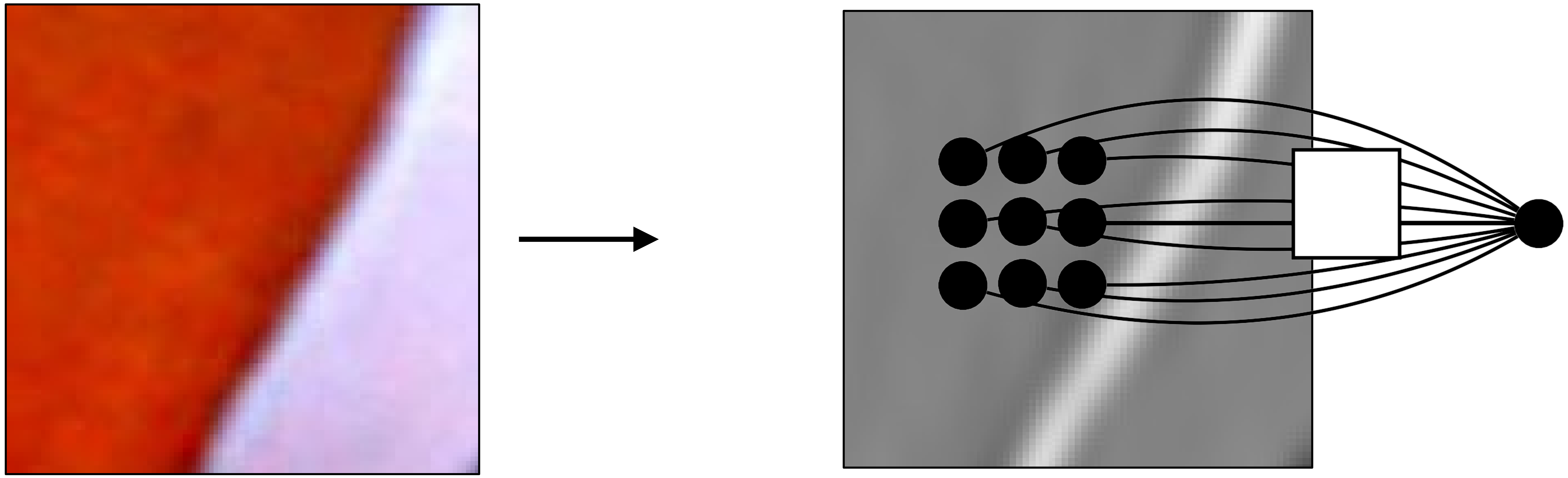
Pooling across spatial locations achieves stability w.r.t. small translations:



large response regardless of exact position of edge

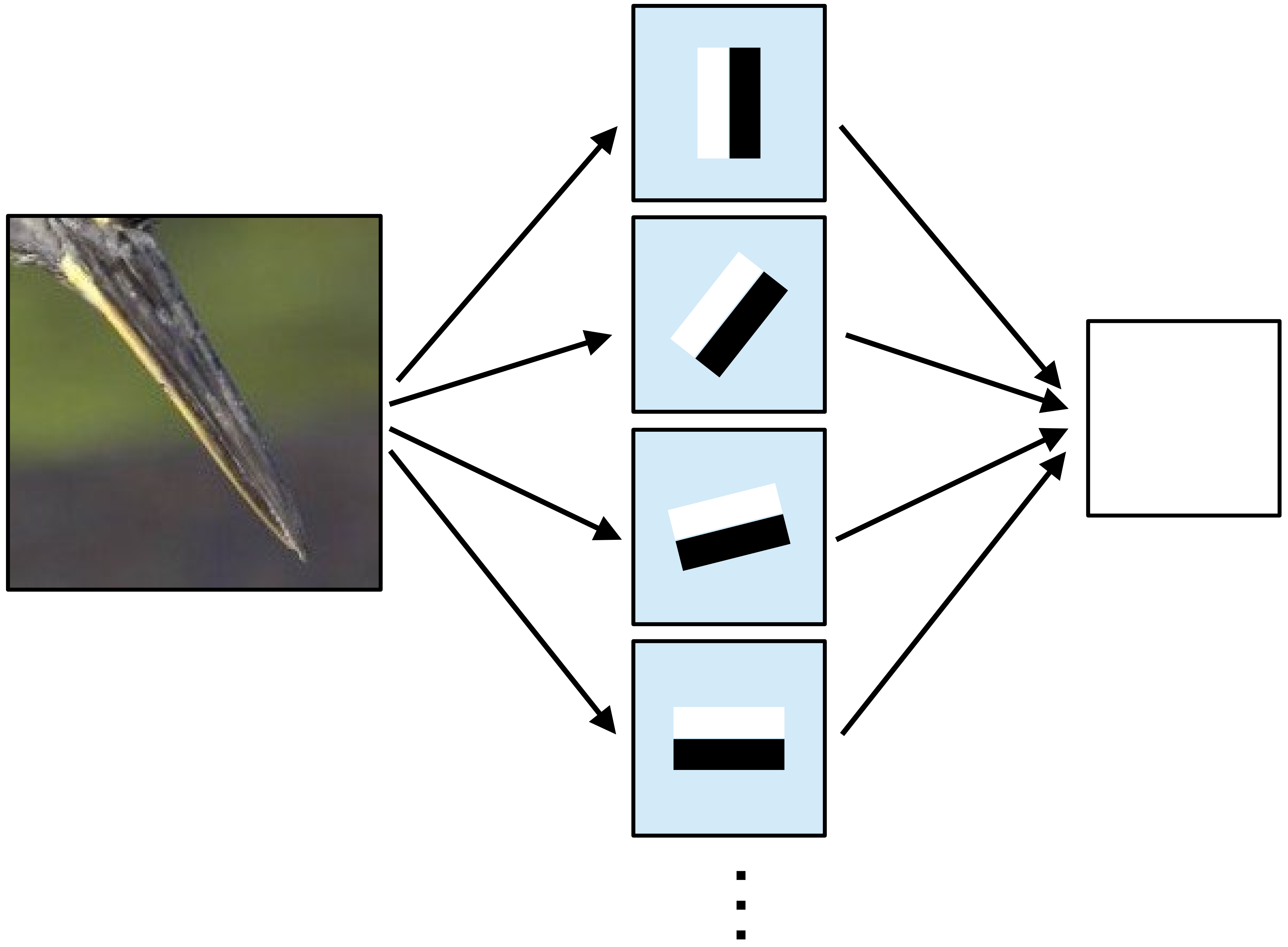
Pooling — Why?

Pooling across spatial locations achieves stability w.r.t. small translations:



Pooling *across channels* — Why?

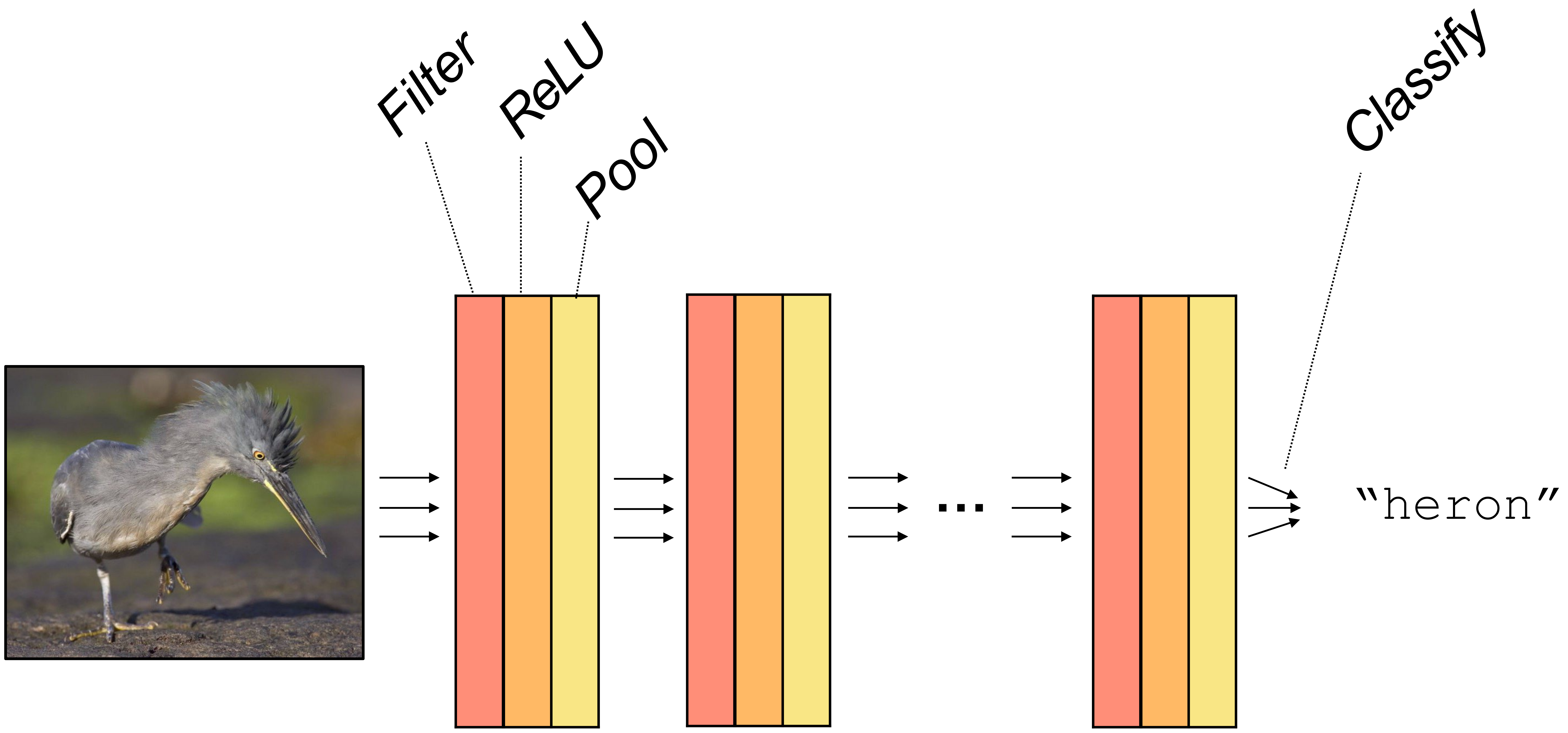
Pooling across feature channels (filter outputs)
can achieve other kinds of invariances:



large response
for any edge,
regardless of its
orientation

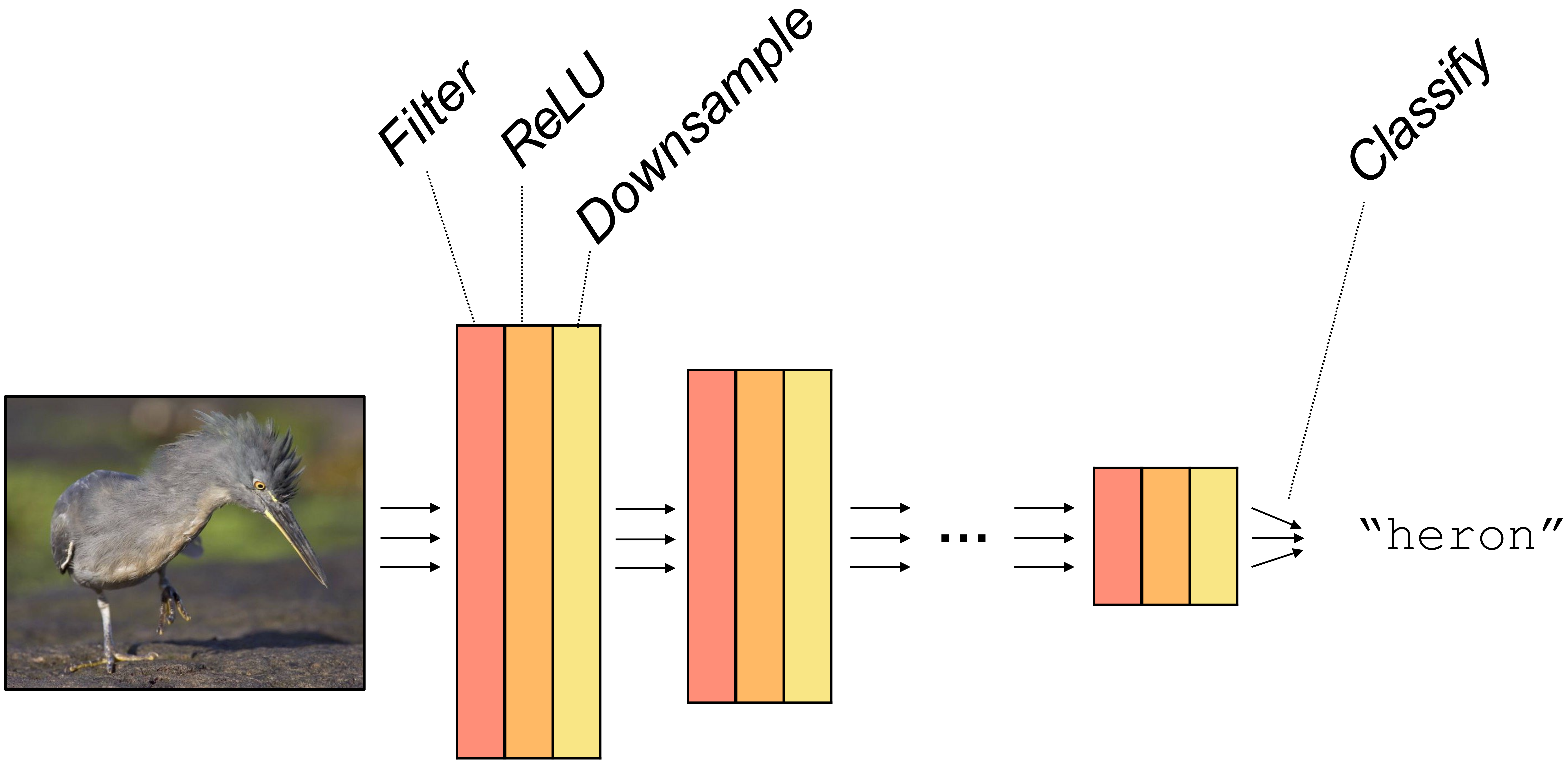
Pooling vs Downsampling

Computation in a neural net



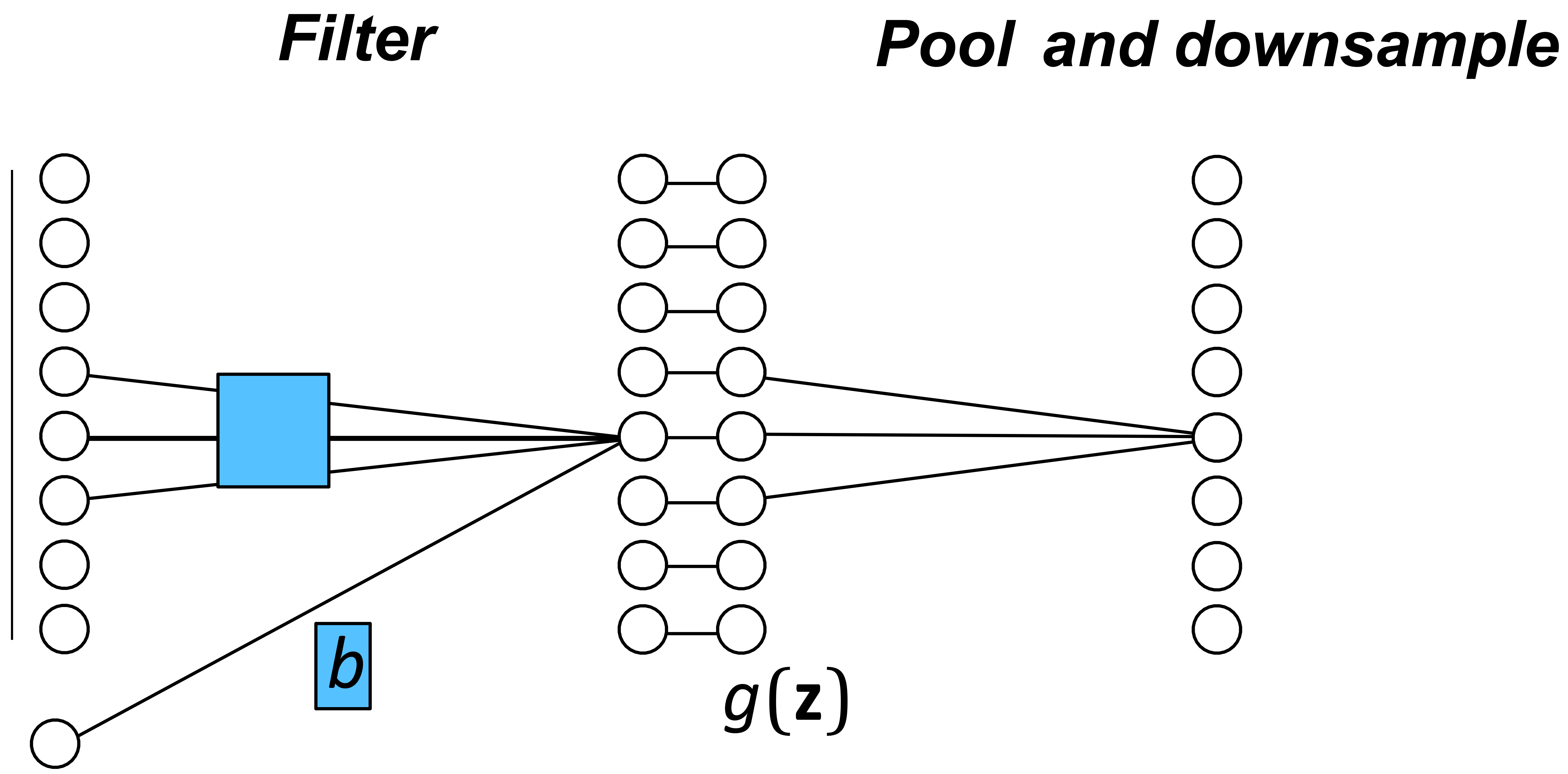
$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

Computation in a neural net

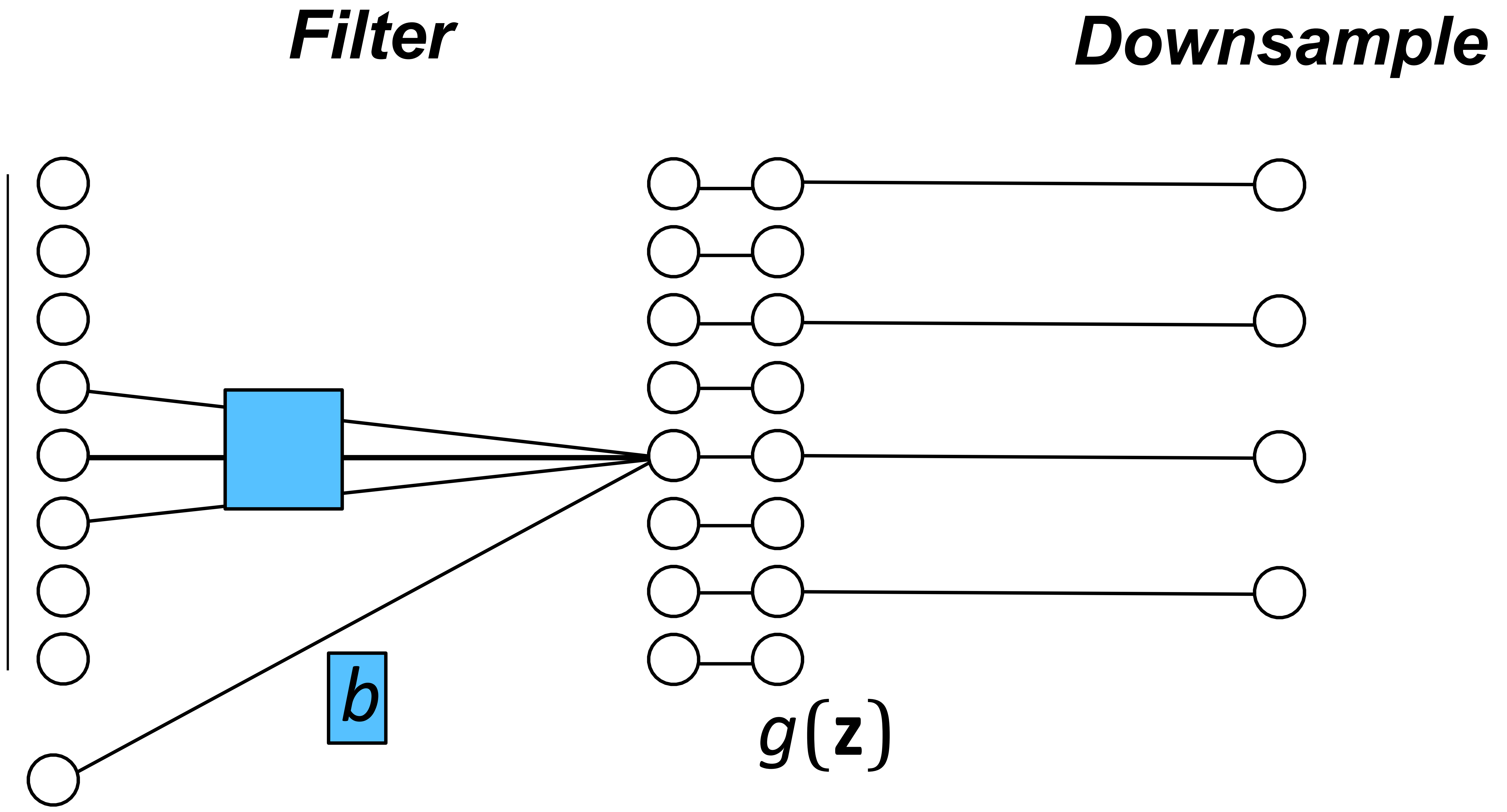


$$f(\mathbf{x}) = f_L(\ldots f_2(f_1(\mathbf{x})))$$

Downsampling



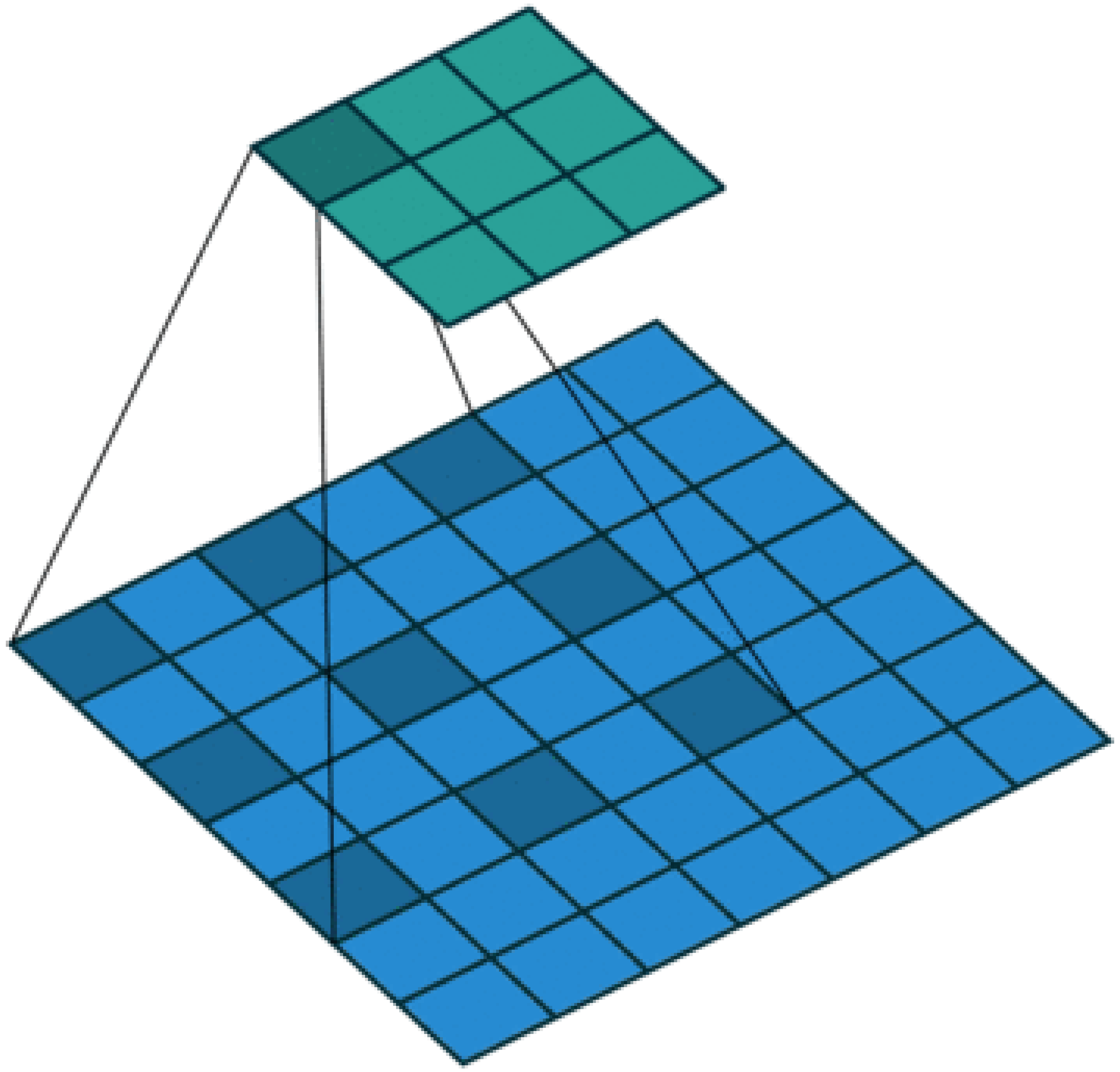
Downsampling



Dilated Convolutions

Allows increasing the receptive field of the convolutional layer

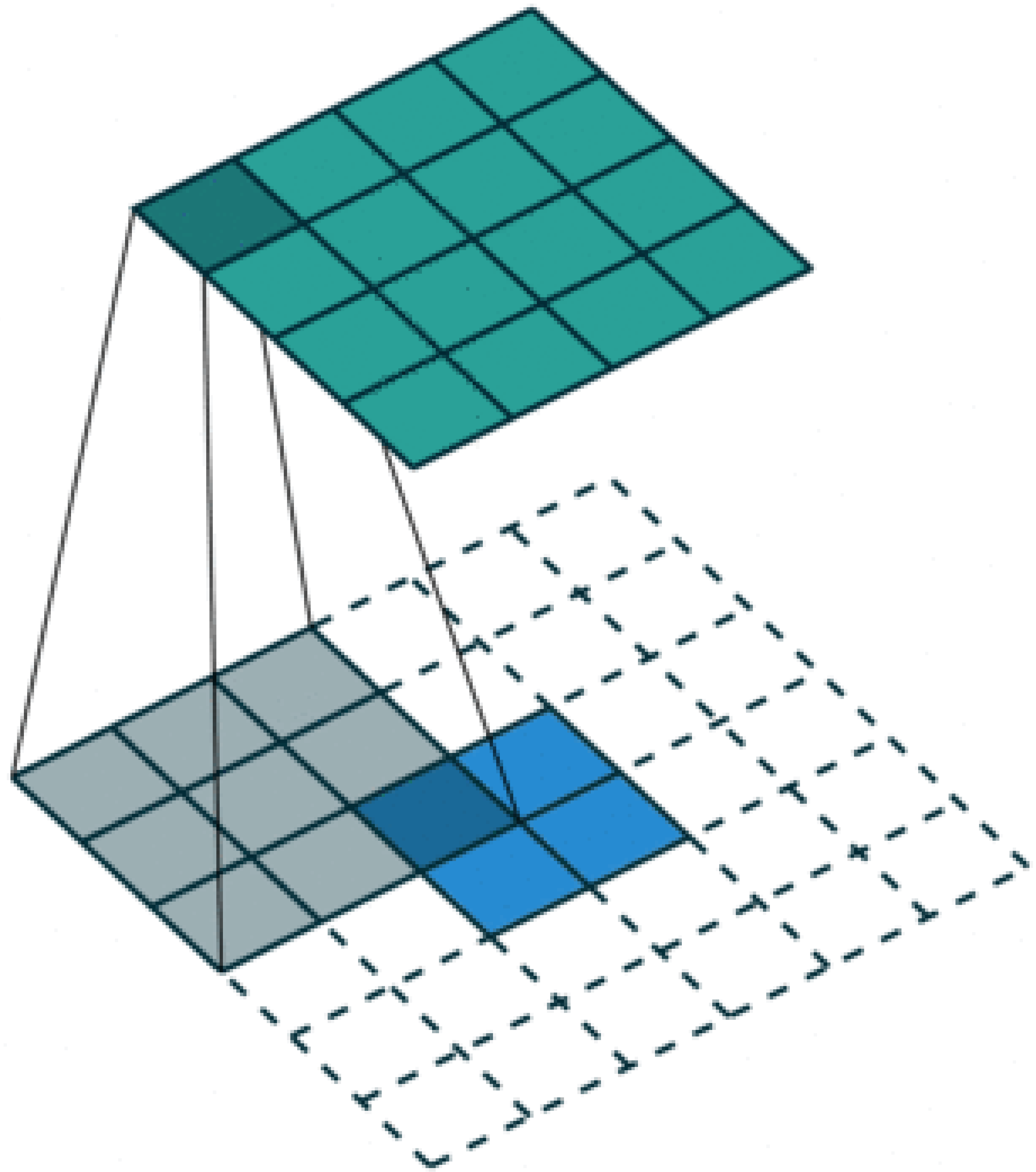
Useful for looking at larger spatial context without looking at every pixel



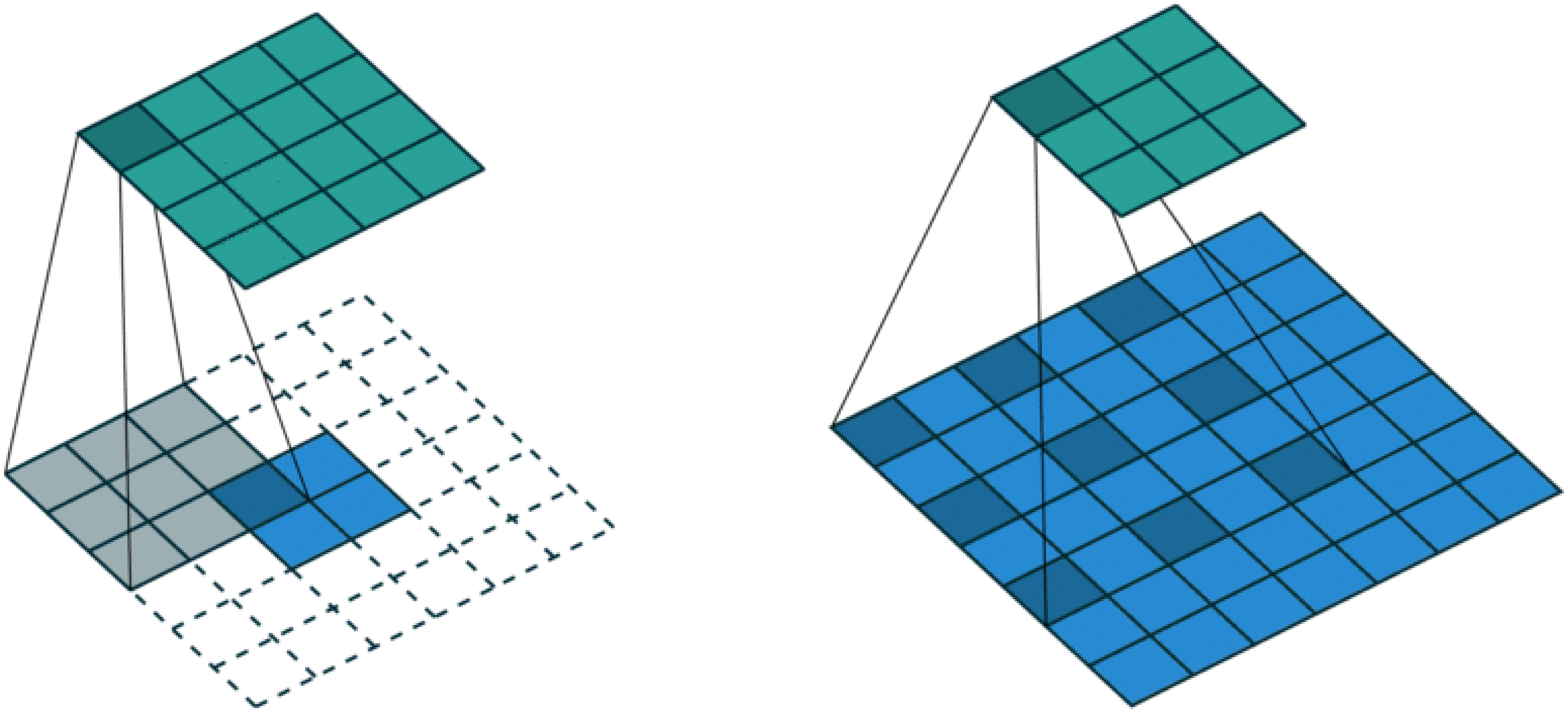
Transposed Convolution

The transposed convolution *a.k.a*

- deconvolution layer
- fractionally strided convolution



Transposed Conv vs. Dilated Conv

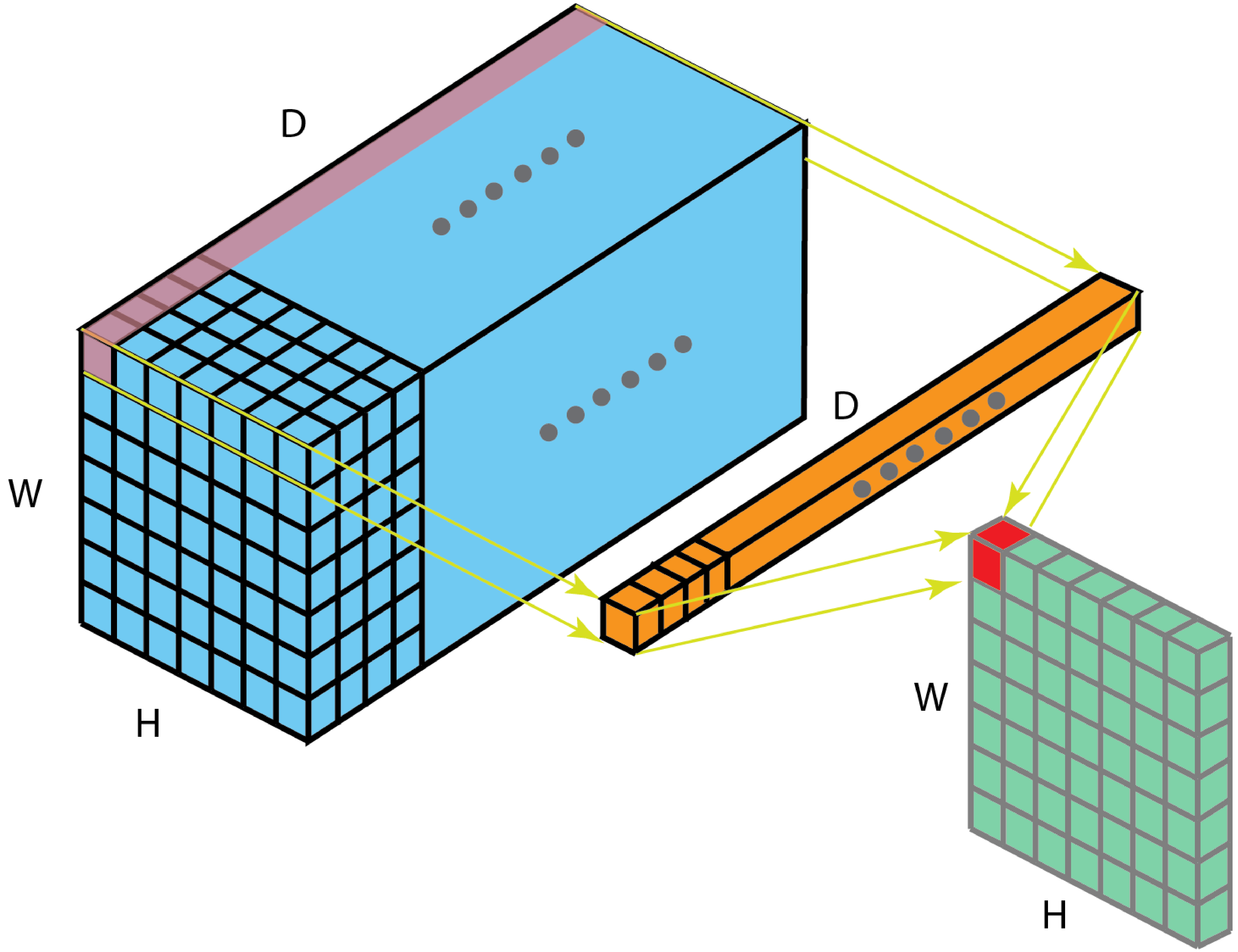


1x1 convolution

How is this not just multiplication?

Multiplications followed by a RELU activation

Good for dimensionality reduction, efficient storage

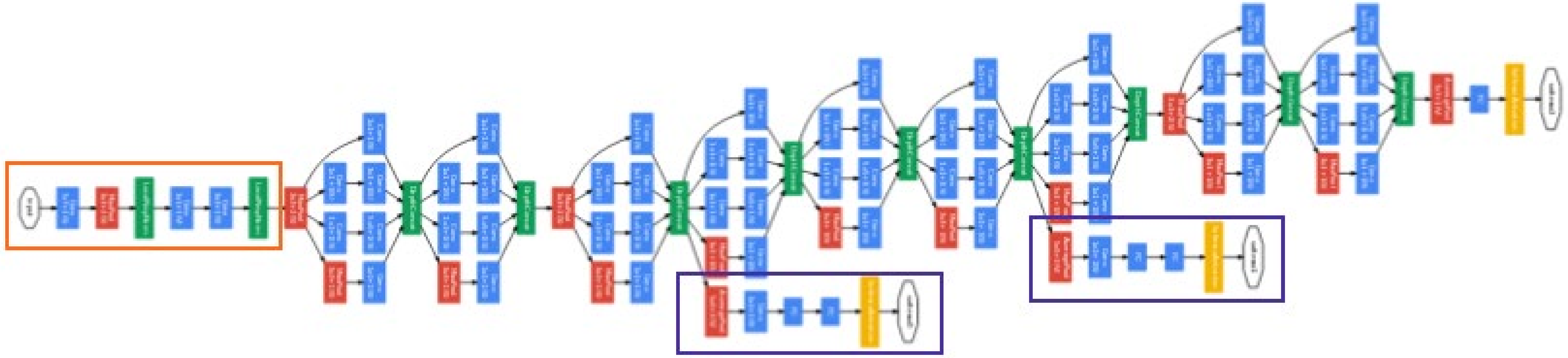


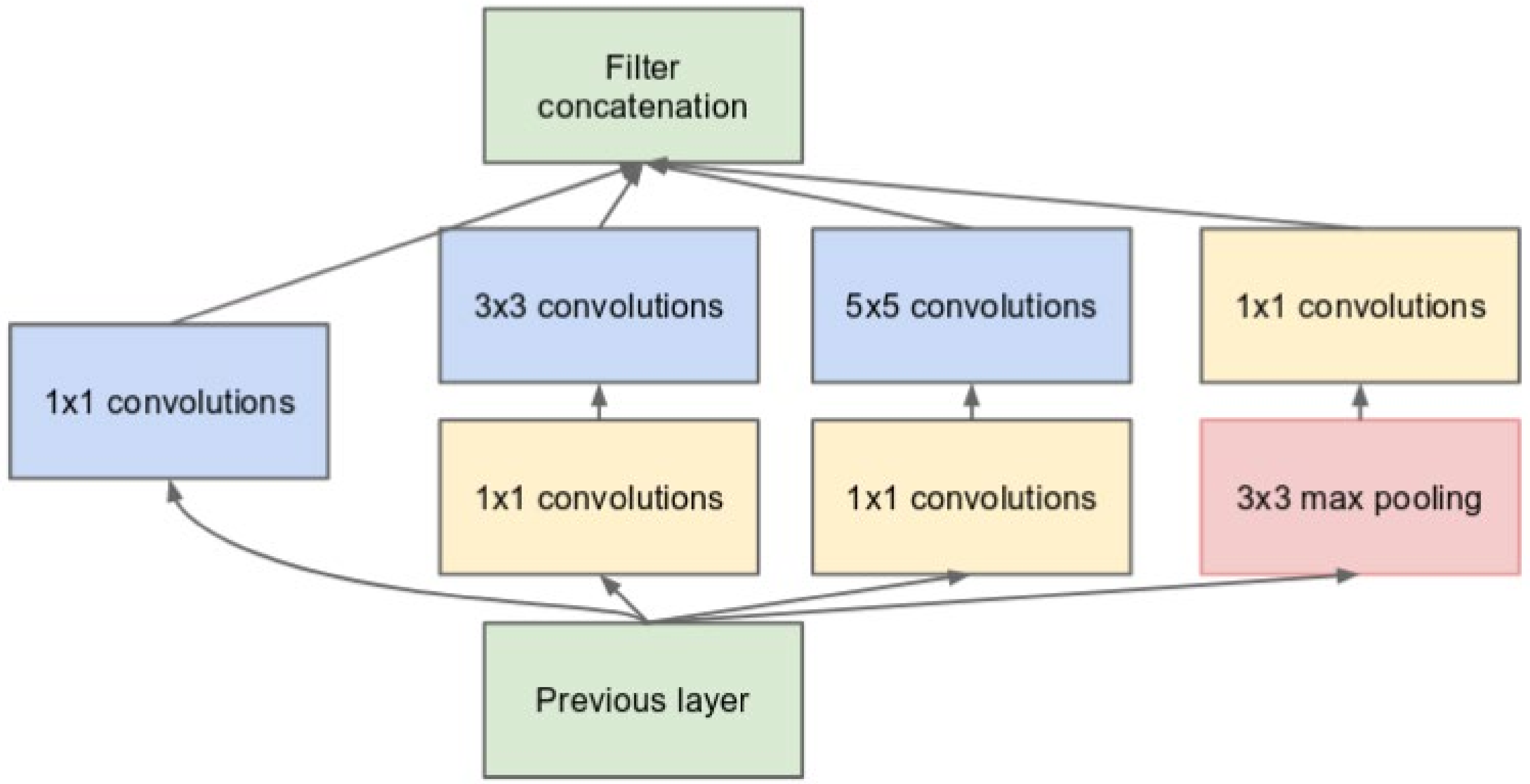
Used in GoogleNet as Inception Layers

Used in GoogleNet in the Inception architecture

Able to get large layer network by doing this

Task: Object classification





(b) Inception module with dimension reductions

Example ConvNet

Figure: Andrej Karpathy

Example ConvNet

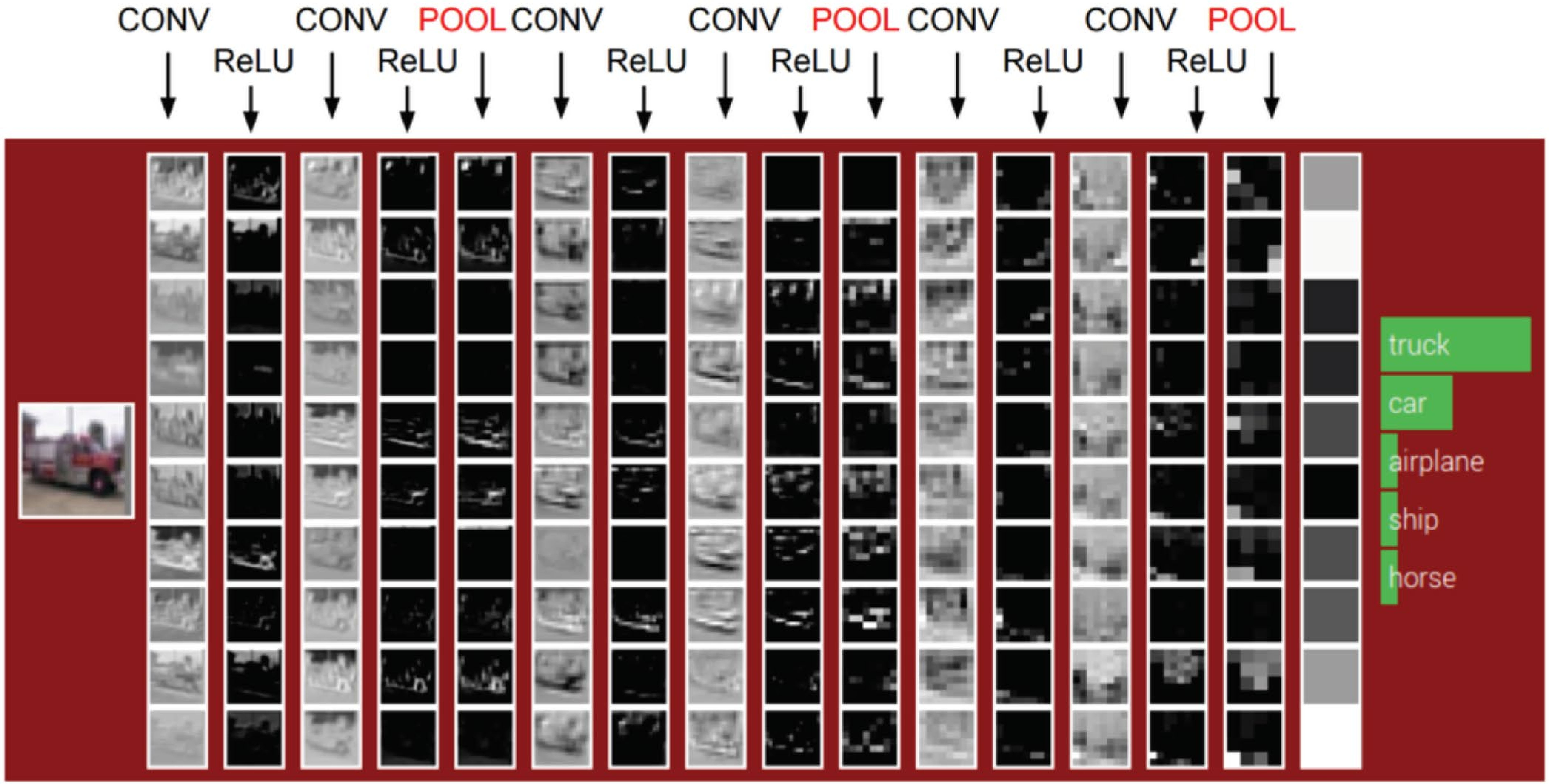


Figure: Andrej Karpathy

Example ConvNet

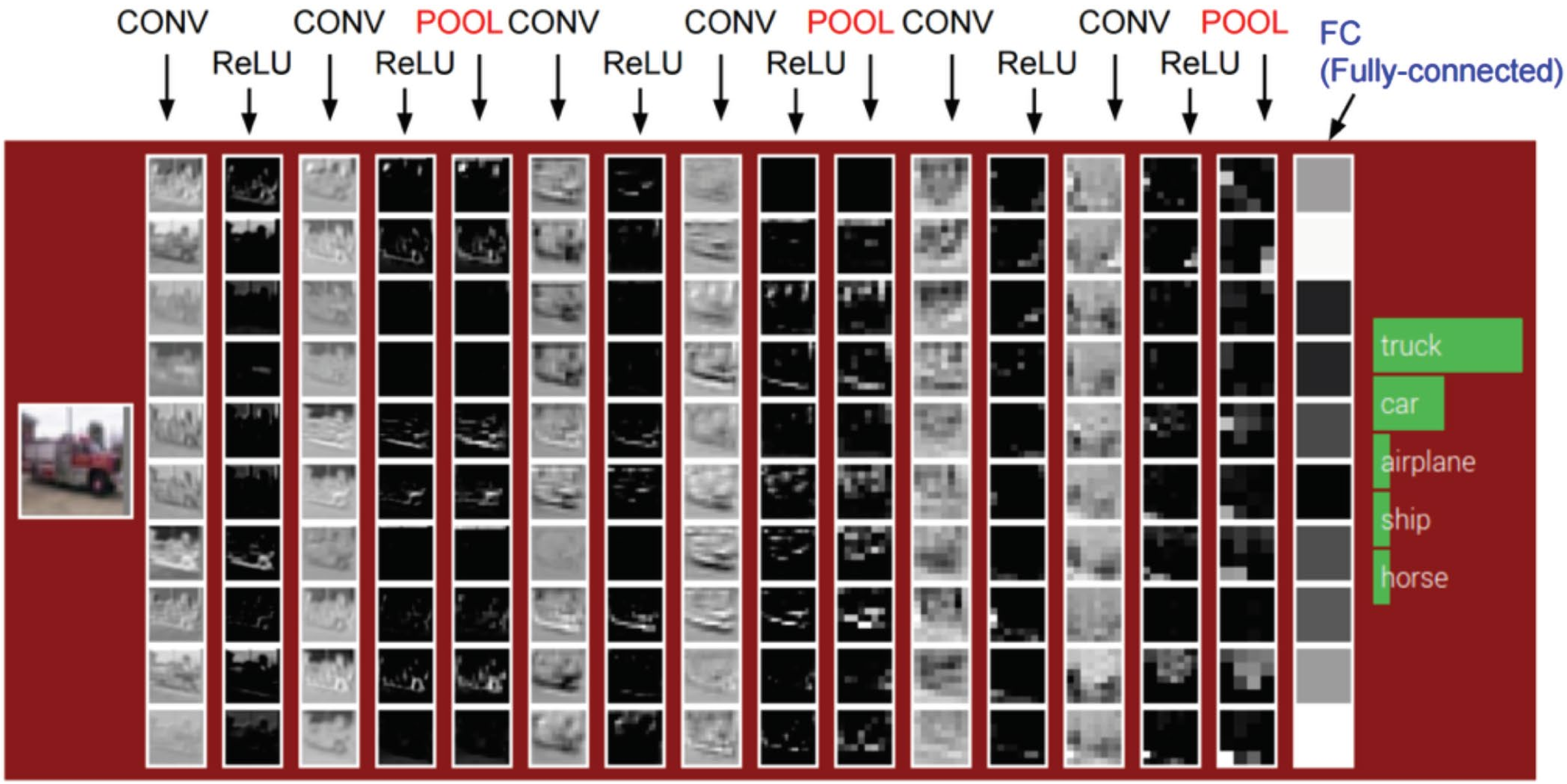
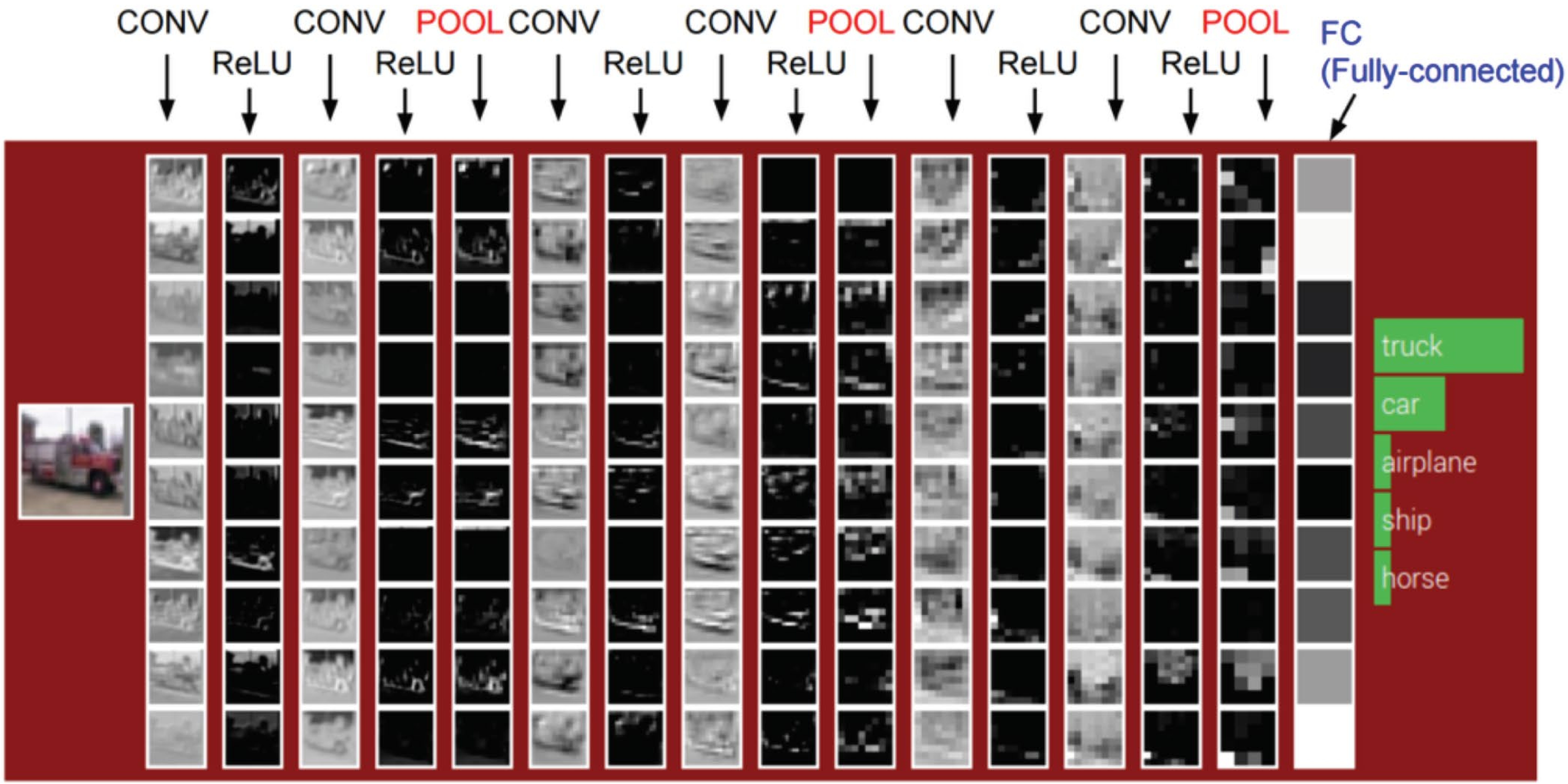


Figure: Andrej Karpathy

Example ConvNet



10x3x3 conv filters, stride 1, pad 1

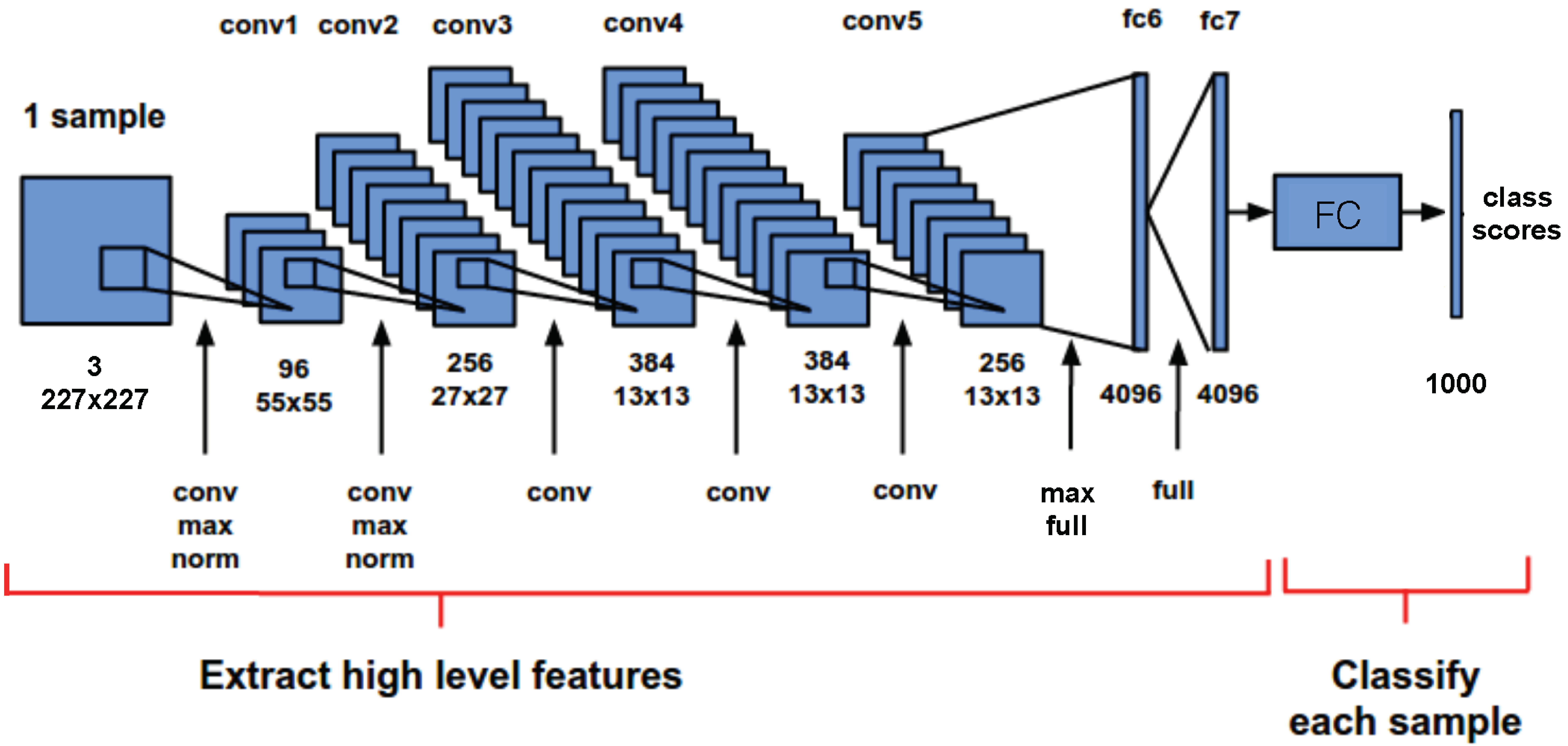
2x2 pool filters, stride 2

Figure: Andrej Karpathy


alexnet
vgg16
resnet18
input image
conv1
conv2
conv3
conv4
conv5
input image
conv1
conv4
conv7
conv10
conv13
input image
conv1
block2
block4
block6
block8

Layer Visualizations

Example: AlexNet [Krizhevsky 2012]



"max": max pooling

"norm": local response normalization

"full": fully connected

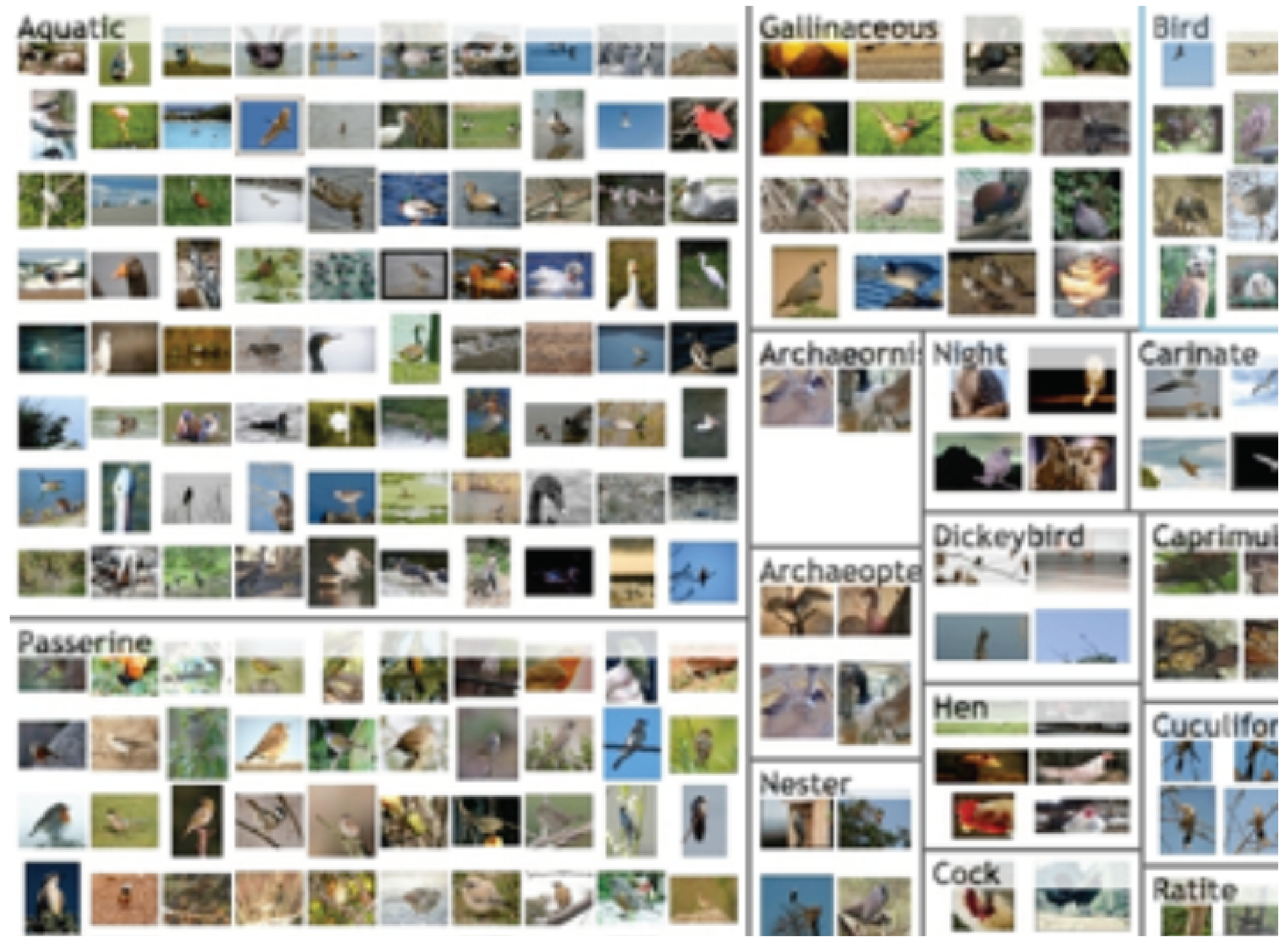
Figure: [Karnowski 2015] *(with corrections)*

Training ConvNets

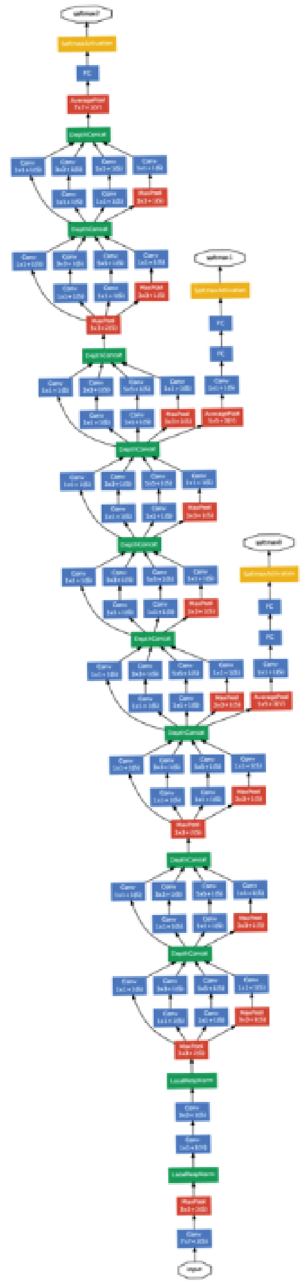
How do you actually train these things?

Roughly speaking:

Gather labeled data



Find a ConvNet architecture



Minimize the loss

Training a convolutional neural network

- Split and preprocess your data
- Choose your network architecture
- Initialize the weights
- Find a learning rate and regularization strength
- Minimize the loss and monitor progress
- Fiddle with knobs

Mini-batch Gradient Descent

Loop:

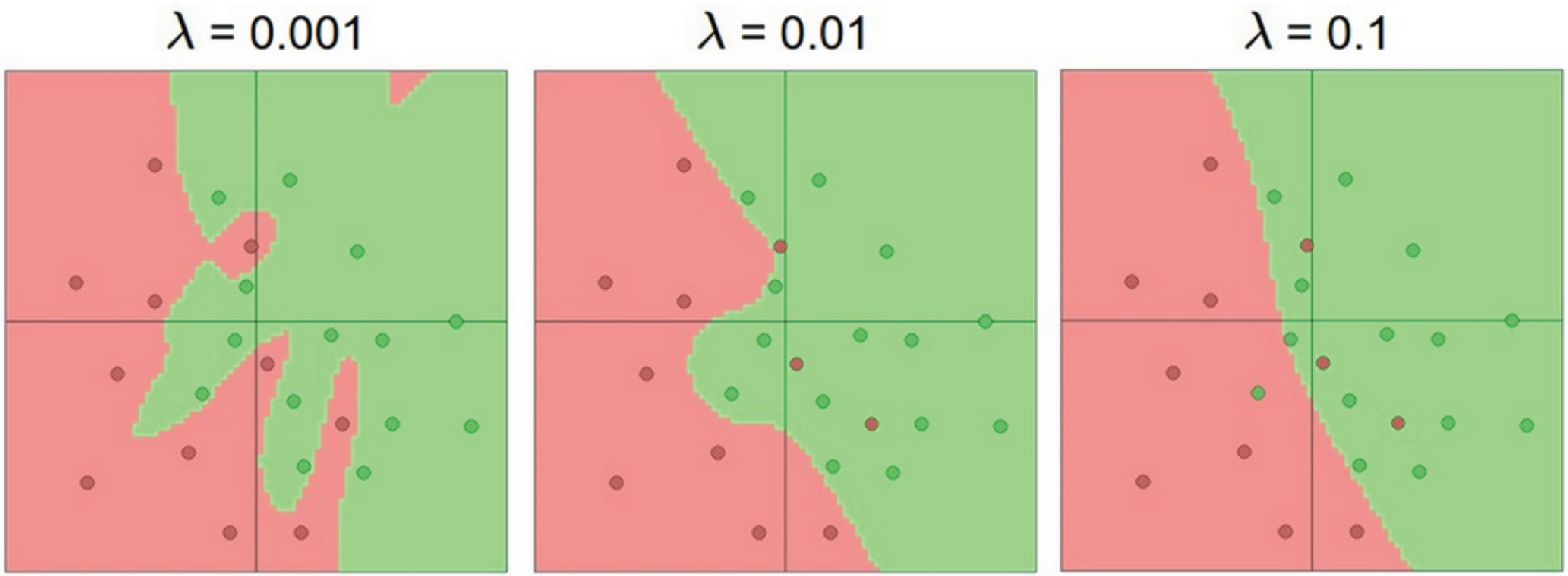
1. Sample a batch of training data (~100 images)
2. Forwards pass: compute loss (avg. over batch)
3. Backwards pass: compute gradient
4. Update all parameters

Note: usually called "stochastic gradient descent" even though SGD has a batch size of 1

Regularization

Regularization reduces overfitting:

$$L = L_{\text{data}} + L_{\text{reg}} \qquad L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$



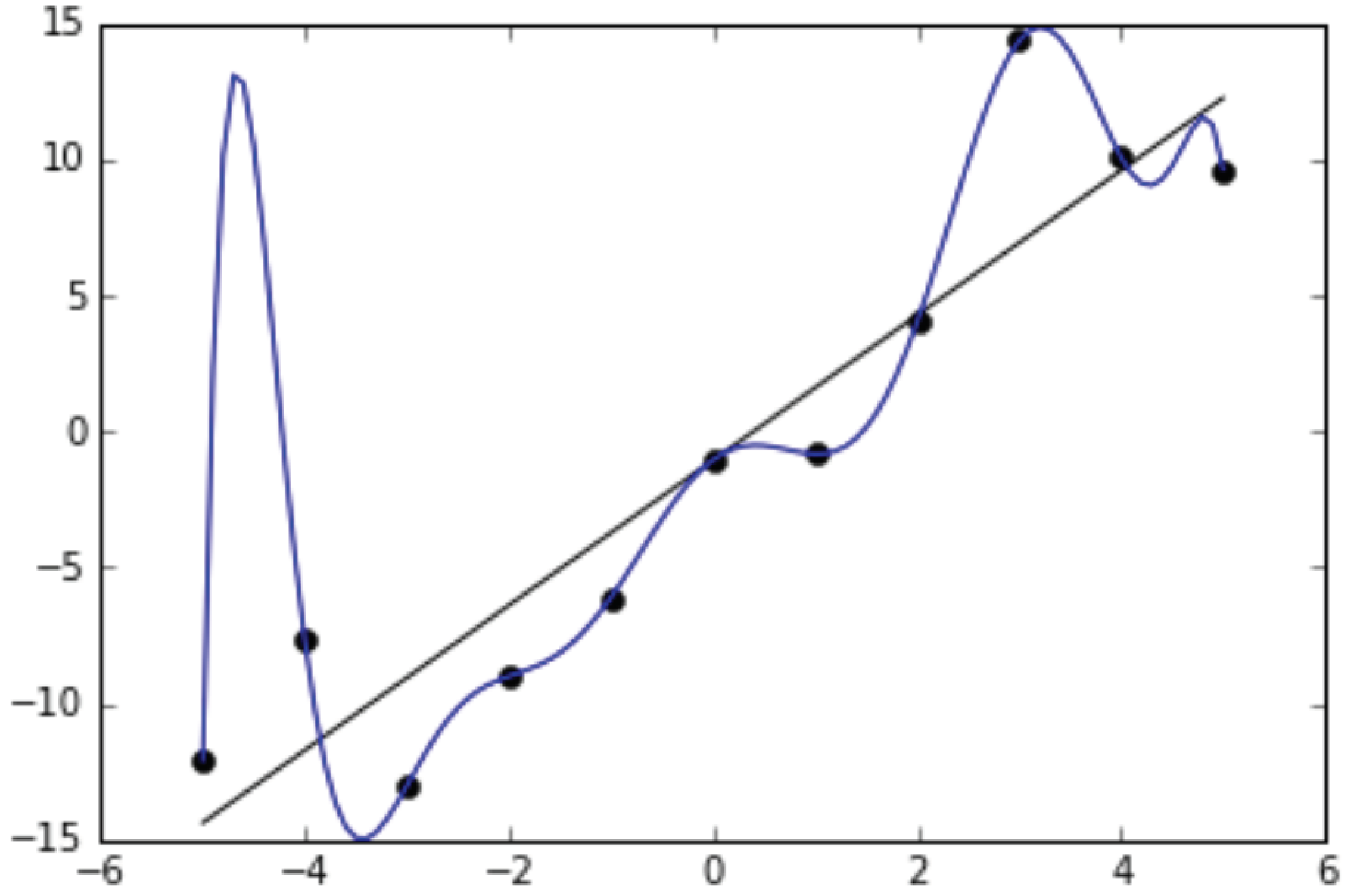
[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Overfitting

Overfitting: modeling noise in the training set instead of the "true" underlying relationship

Underfitting: insufficiently modeling the relationship in the training set

General rule: models that are "bigger" or have more capacity are more likely to overfit



[Image: https://en.wikipedia.org/wiki/File:Overfitted_Data.png]

Summary of things to fiddle

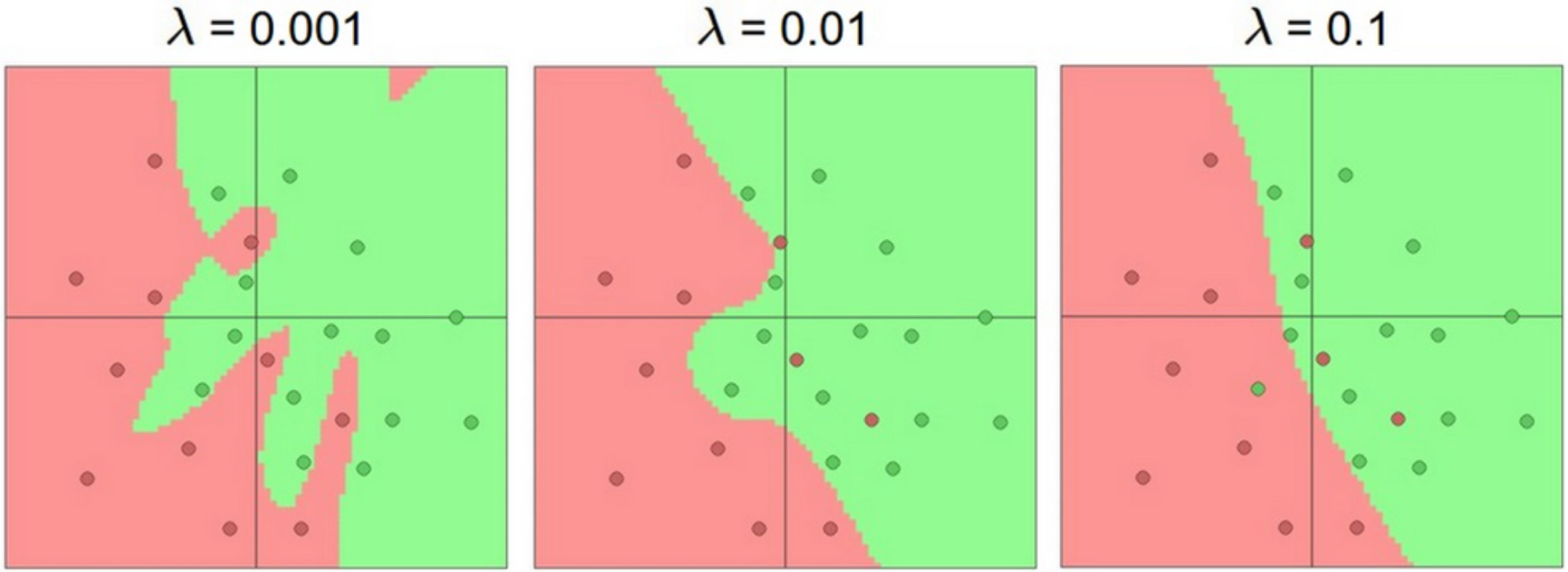
- Network architecture
- Learning rate, decay schedule, update type
- Regularization (L2, L1, maxnorm, dropout, …)
- Loss function (softmax, SVM, …)
- Weight initialization

Neural network parameters

(Recall) Regularization reduces overfitting

$$L = L_{\text{data}} + L_{\text{reg}}$$

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$



[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Example Regularizers

L2 regularization

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2$$

(L2 regularization encourages small weights)

L1 regularization

$$L_{\text{reg}} = \lambda \|W\|_1 = \lambda \sum_{ij} |W_{ij}|$$

(L1 regularization encourages sparse weights: weights are encouraged to reduce to exactly zero)

"Elastic net"

$$L_{\text{reg}} = \lambda_1 \|W\|_1 + \lambda_2 \|W\|_2^2$$

(combine L1 and L2 regularization)

Max norm

Clamp weights to some max norm

$$\|W\|_2^2 \leq c$$

"Weight decay"

Regularization is also called "weight decay" because the weights "decay" each iteration:

$$L_{\text{reg}} = \lambda \frac{1}{2} \|W\|_2^2 \longrightarrow \frac{\partial L}{\partial W} = \lambda W$$

Gradient descent step:

$$W \leftarrow W - \alpha\lambda W - \frac{\partial L_{\text{data}}}{\partial W}$$

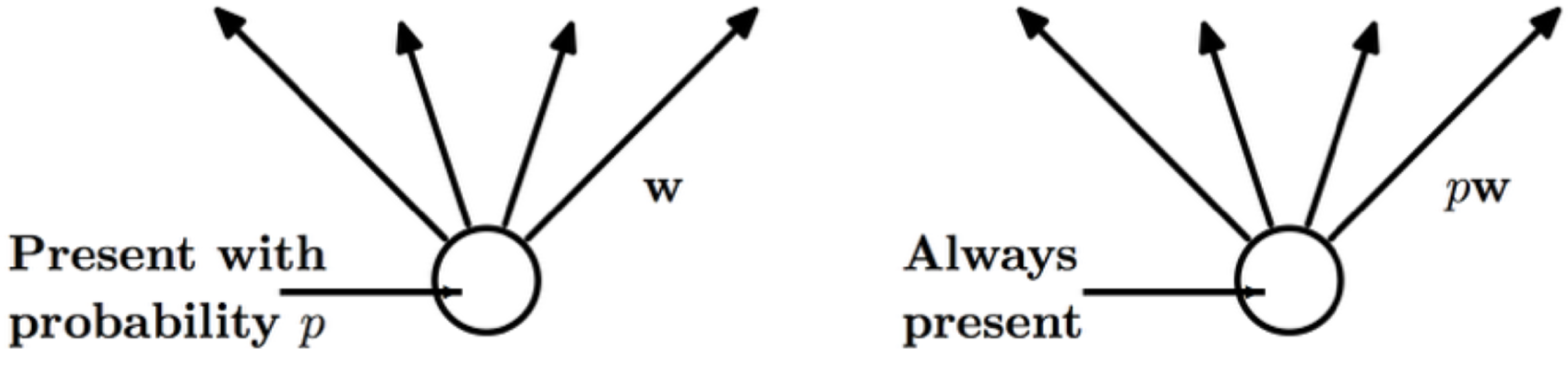
Weight decay: $\alpha\lambda$ (weights always decay by this amount)

Note: biases are sometimes excluded from regularization

[Andrej Karpathy http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html]

Dropout

Simple but powerful technique to reduce overfitting:

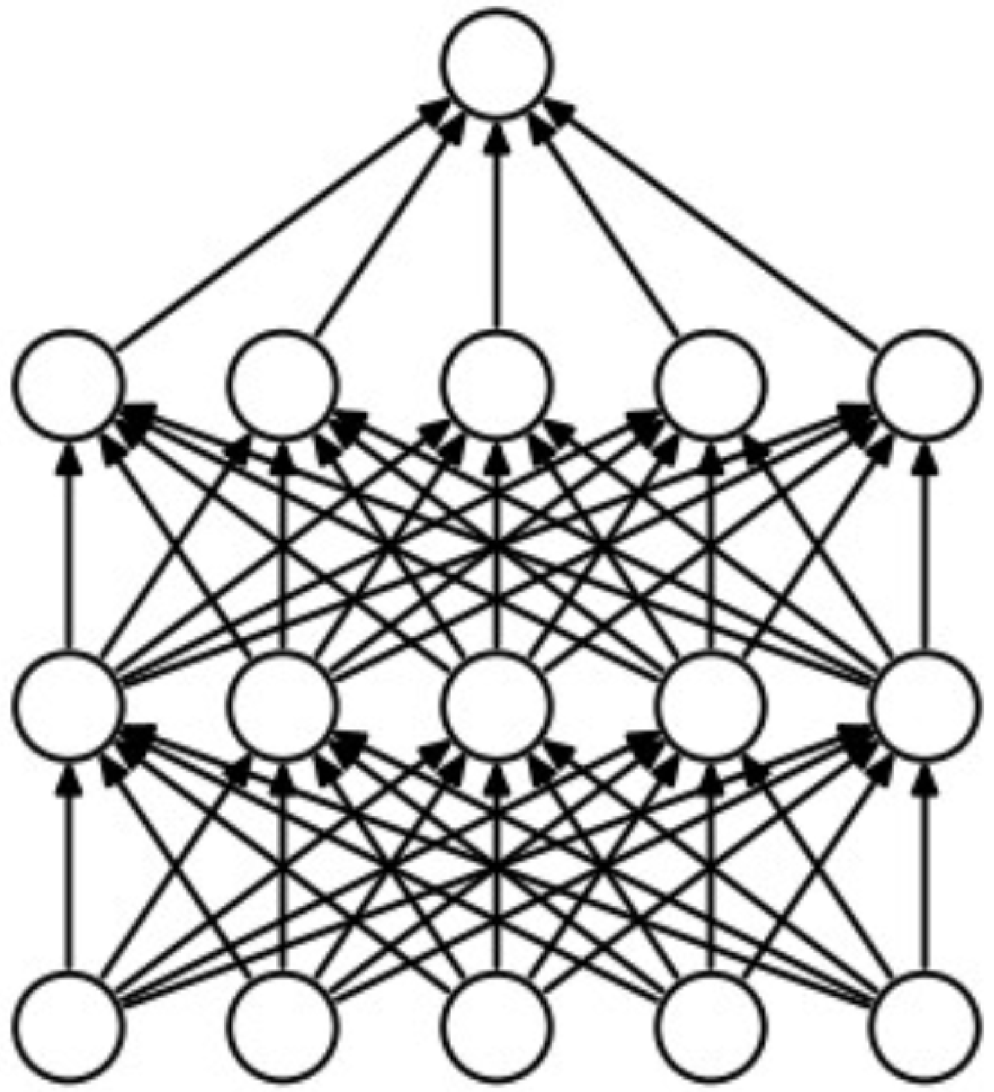


(a) At training time (b) At test time

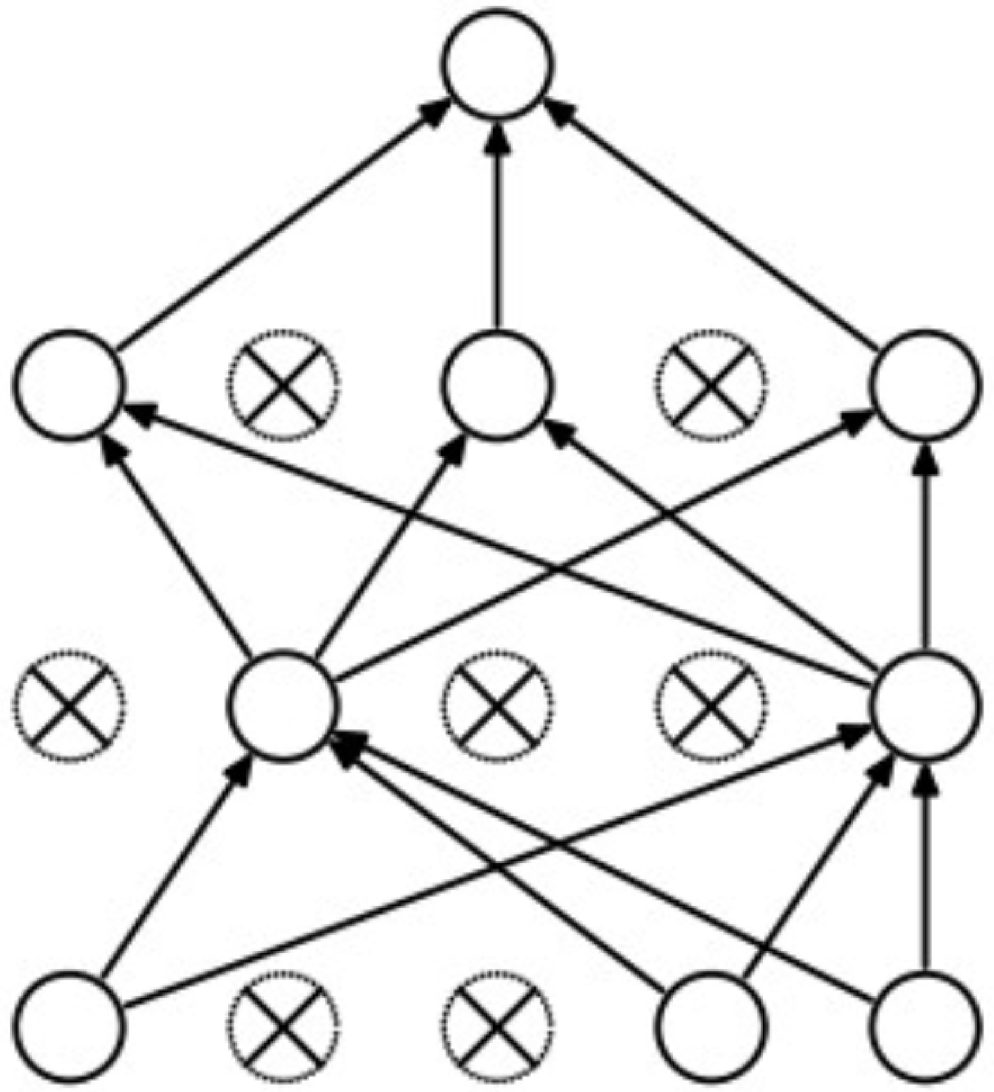
[Srivasta et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, JMLR 2014]

Dropout

Simple but powerful technique to reduce overfitting:



(a) Standard Neural Net



(b) After applying dropout.

Note: Dropout can be interpreted as an approximation to taking the geometric mean of an ensemble of exponentially many models

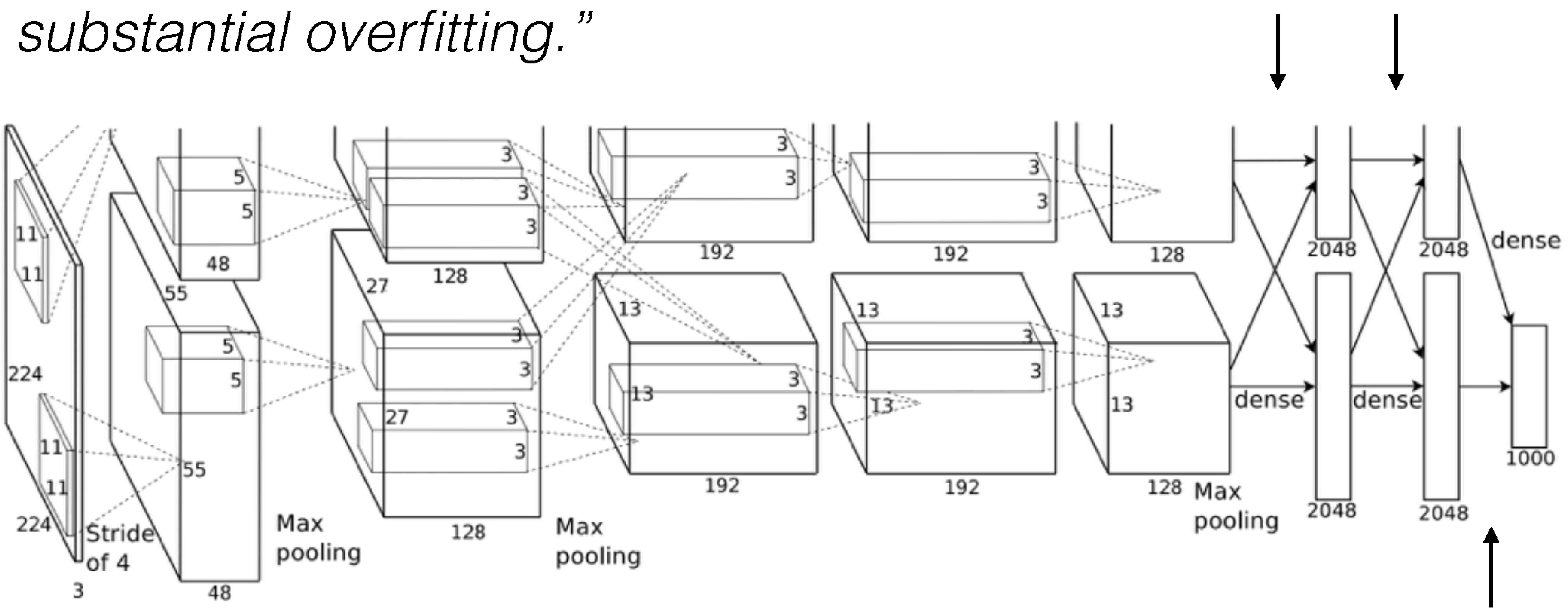
[Srivasta et al, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", JMLR 2014]

Dropout

Case study: [Krizhevsky 2012]

"Without dropout, our network exhibits substantial overfitting."

Dropout here



But not here — why?

[Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012]

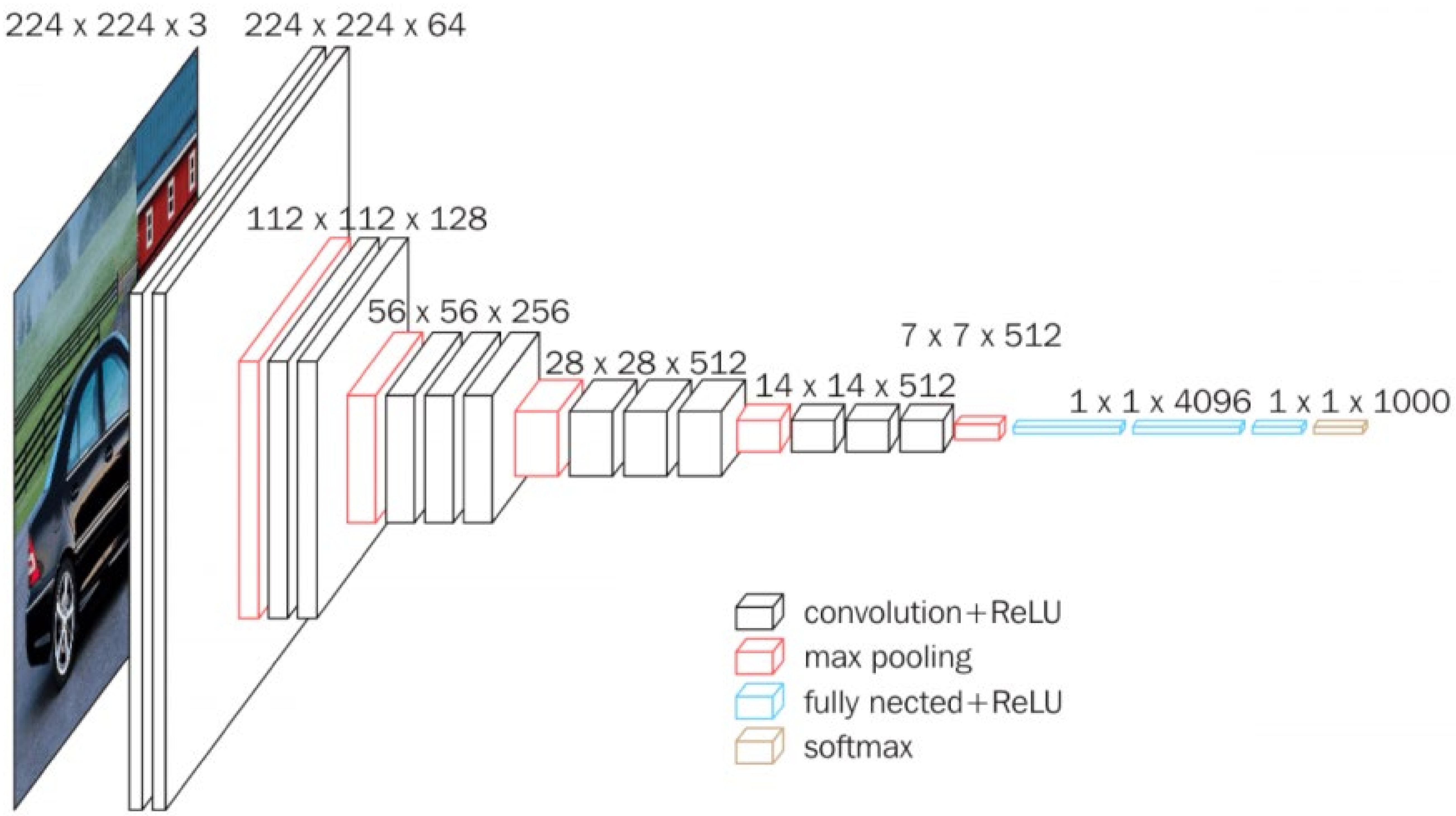
Summary

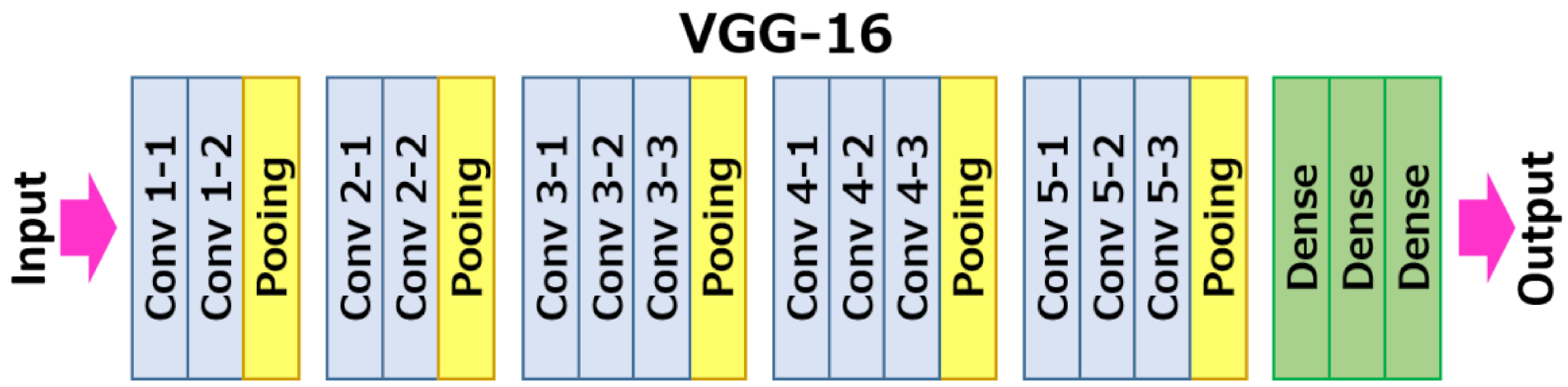
- Preprocess the data (subtract mean, sub-crops)
- Initialize weights carefully
- Use Dropout
- Use SGD + Momentum
- Fine-tune from ImageNet
- Babysit the network as it trains

Common Architectures

VGG

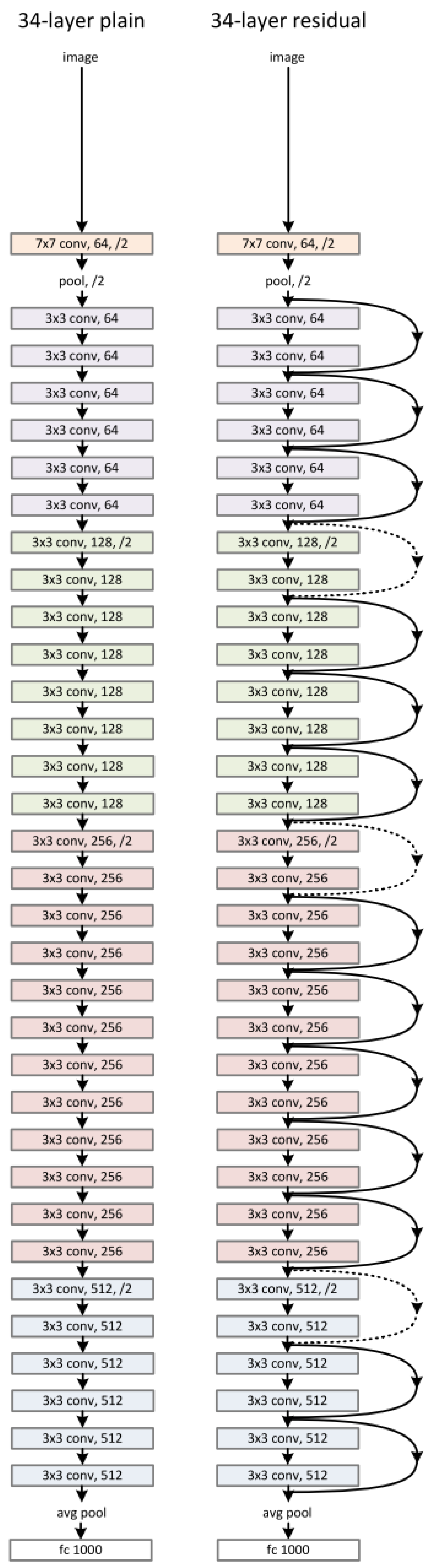
- Simonyan and Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition"
- Used to be very common (before ResNets)

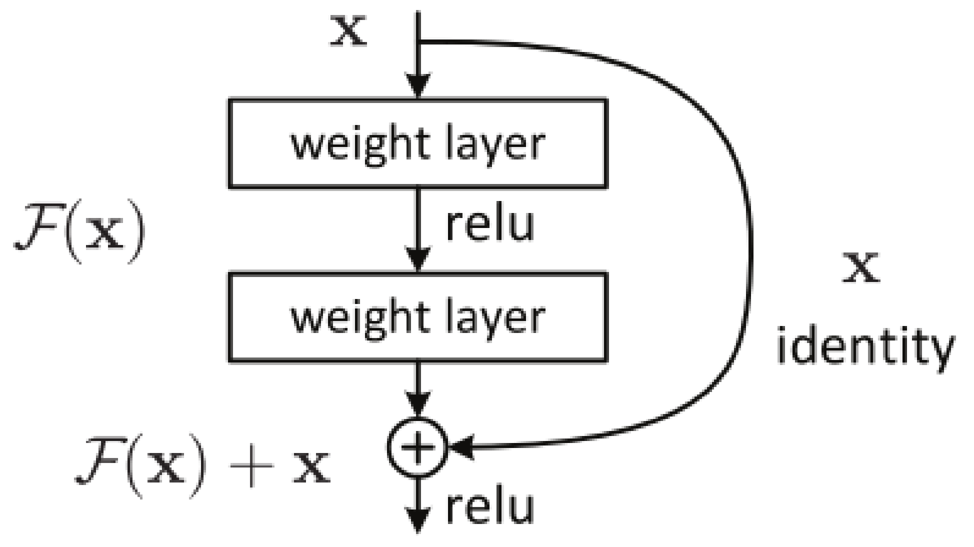




ResNet

- He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016). "Deep Residual Learning for Image Recognition" (PDF). Proc. Computer Vision and Pattern Recognition (CVPR), IEEE.
- Deep networks with more layers does not always mean better performance (vanishing gradient problem)
- Residual blocks = has skip connections
- Skipped layers train faster at the beginning, then later are filled out



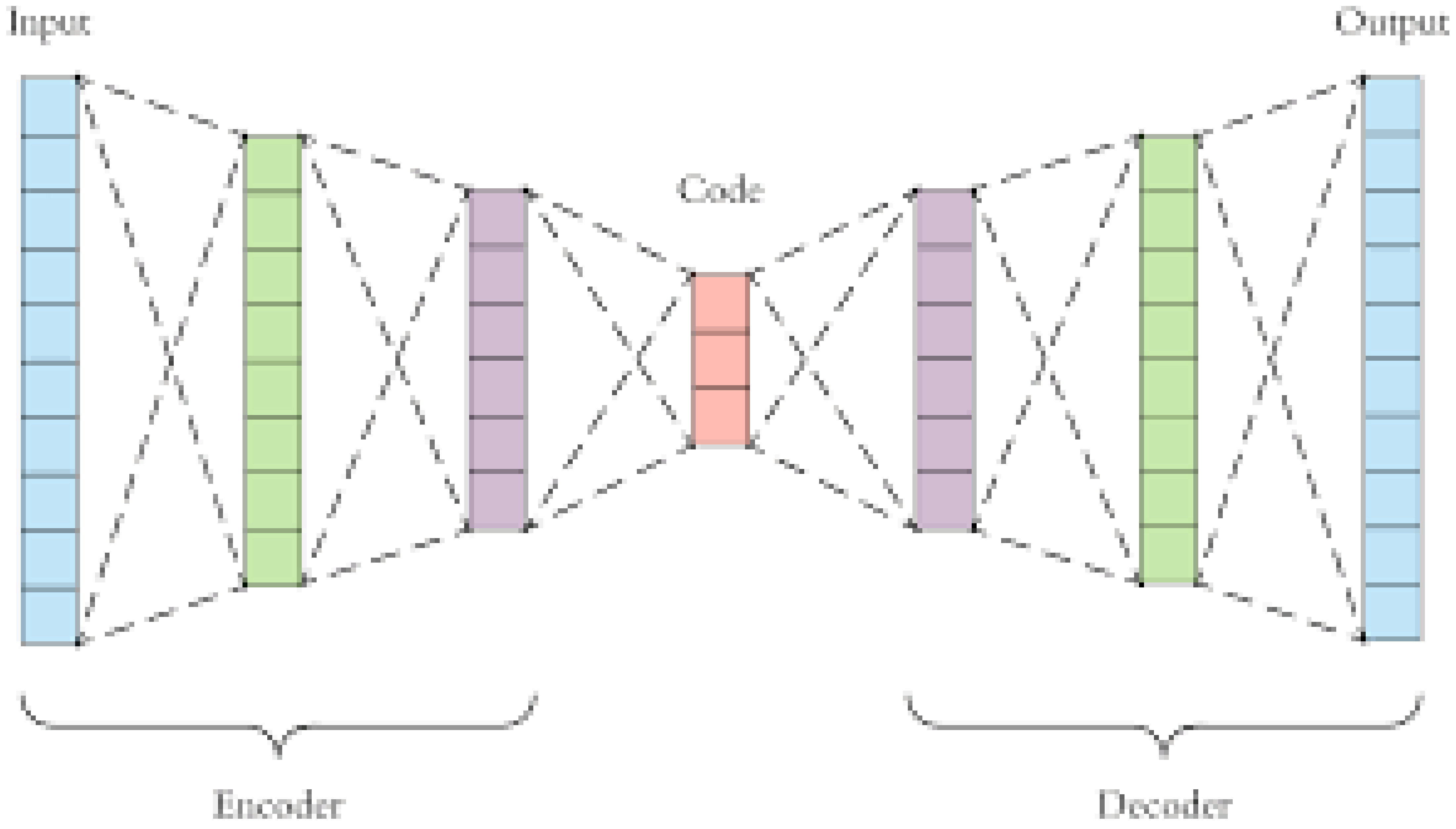


Autoencoder

- Can be done with either fully connected or convolutional layers

- Idea is to reduce the input to a bottleneck or latent code, then reconstruct it again

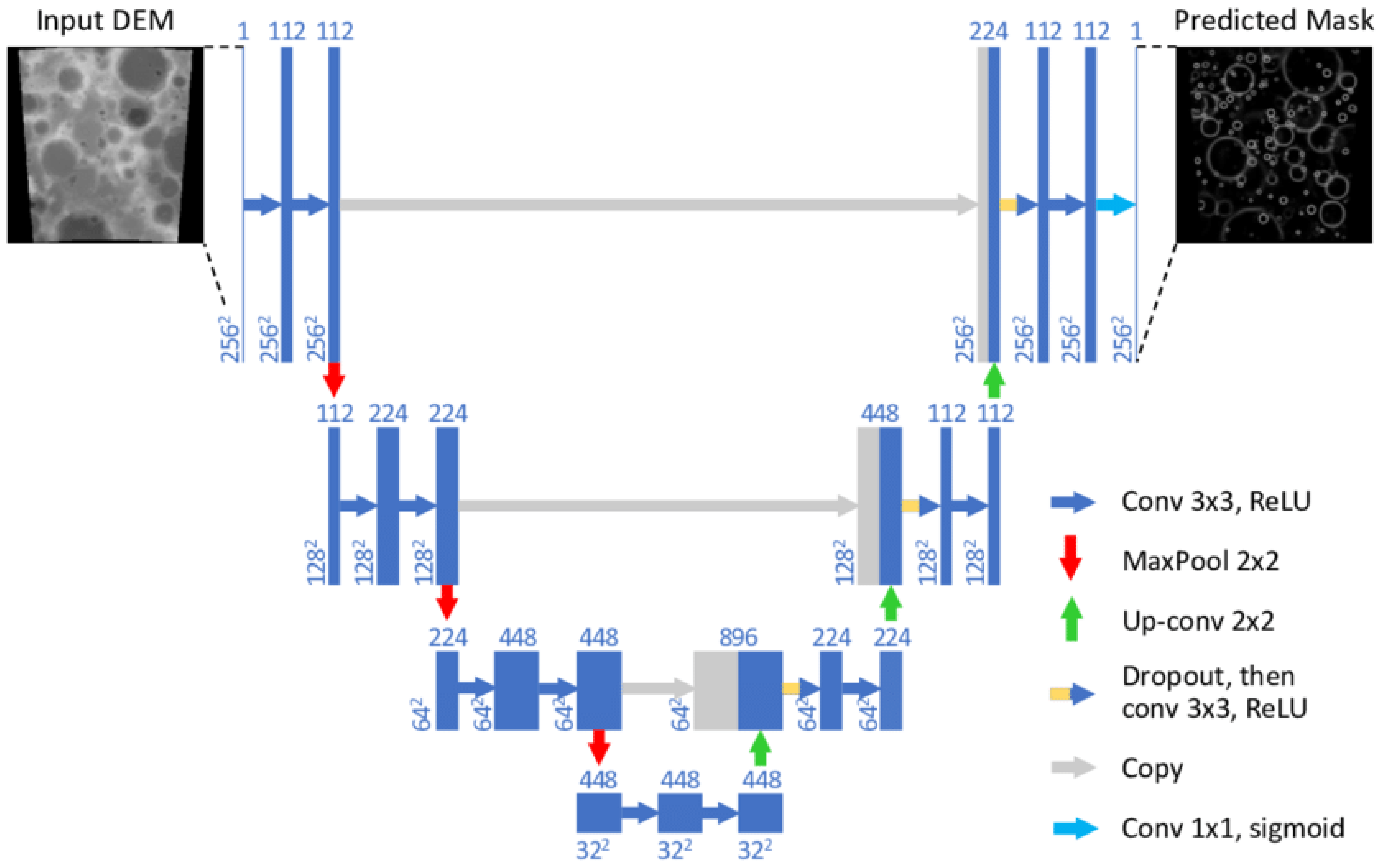
- Sometimes can be used to train a feature extractor by enforcing the output = input, and then use the first part of the network as a feature extractor



U-Net

- Common architecture for image reconstruction tasks

- Features skip connections and transposed convolutions (up-conv)



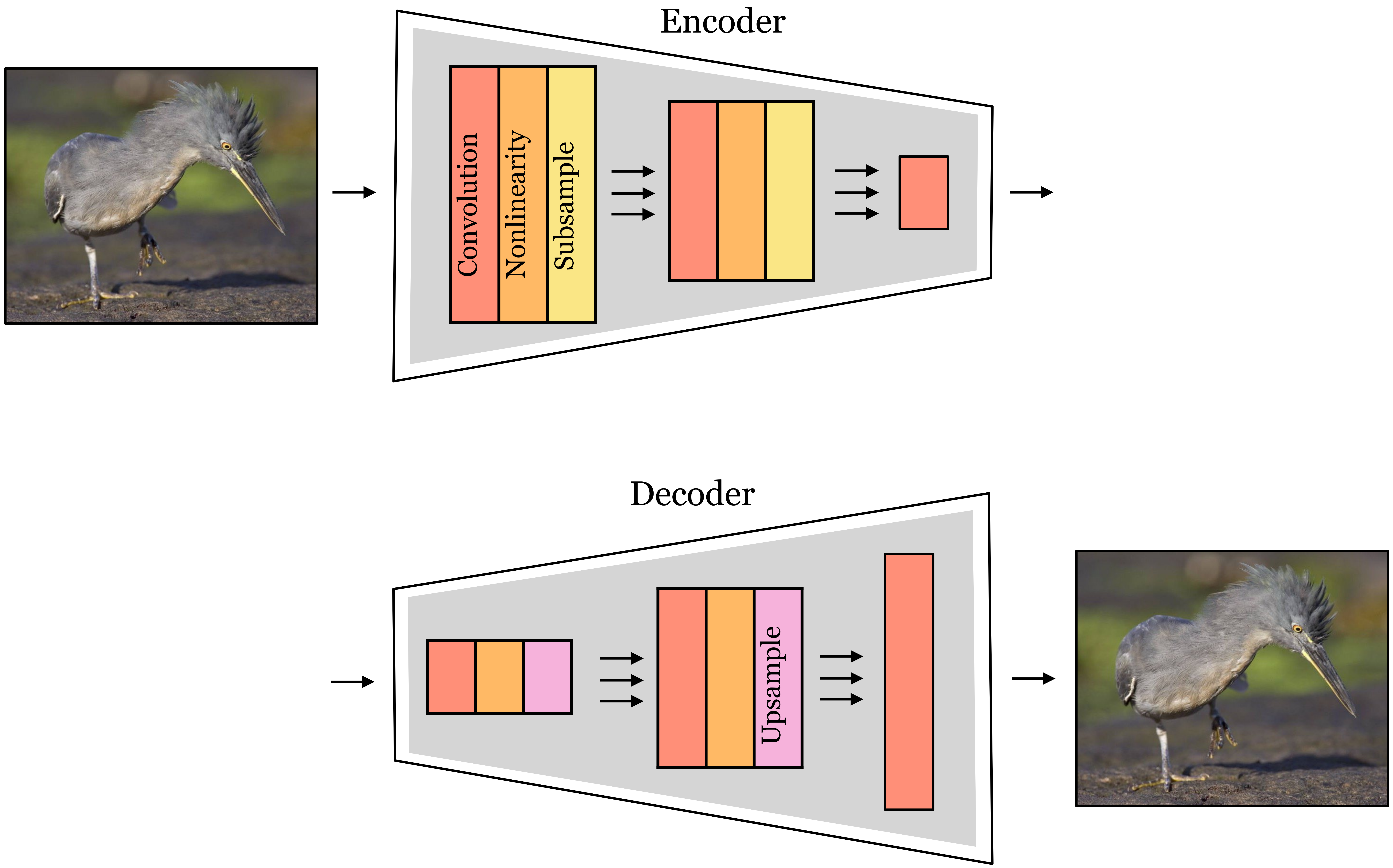

Encoder
Convolution
Nonlinearity
Subsample
Decoder
Upsample

Image-to-image

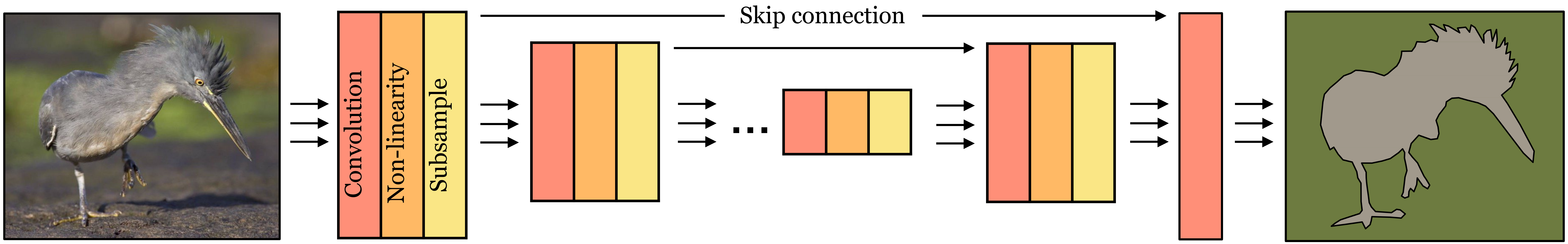
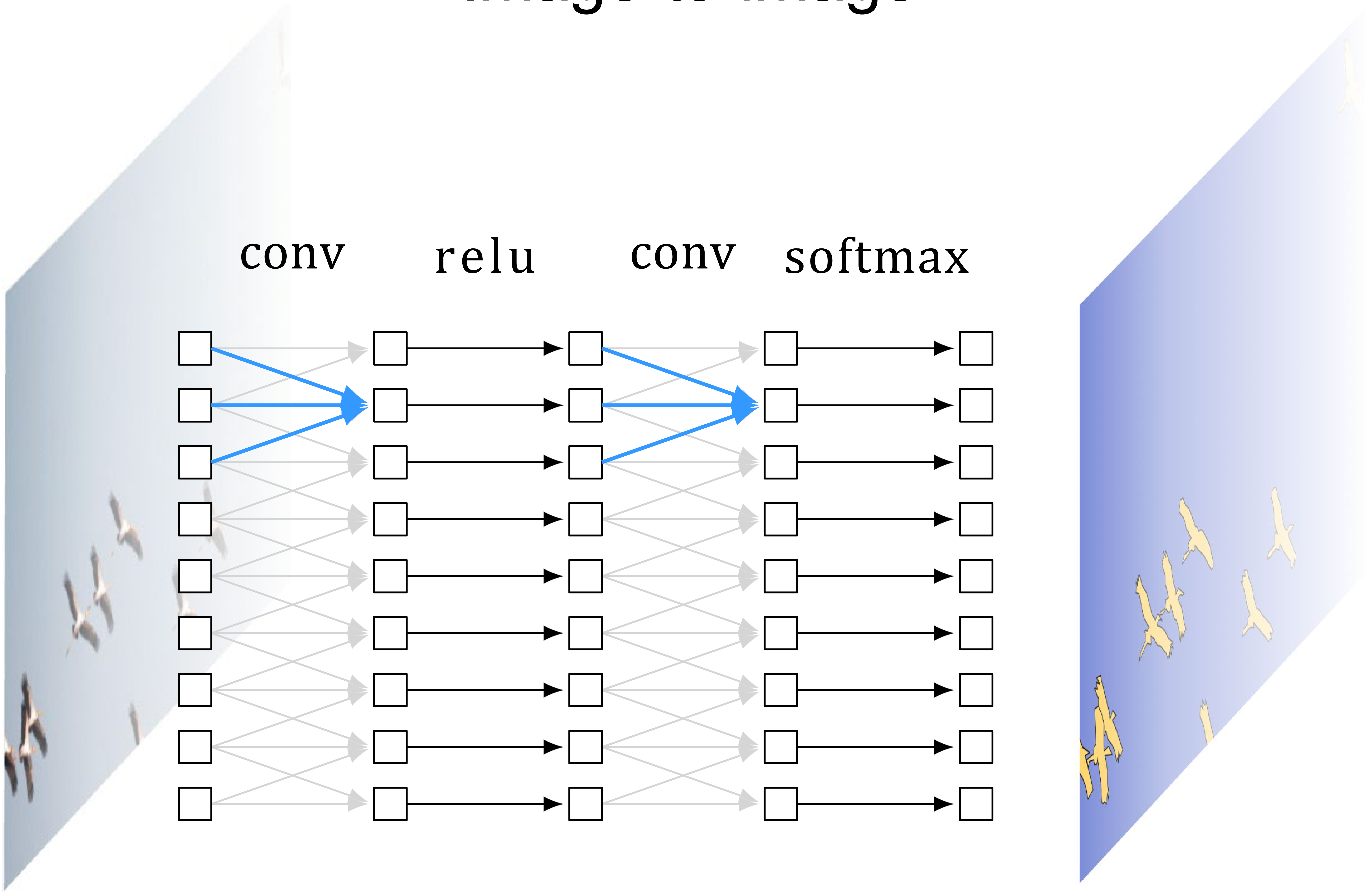
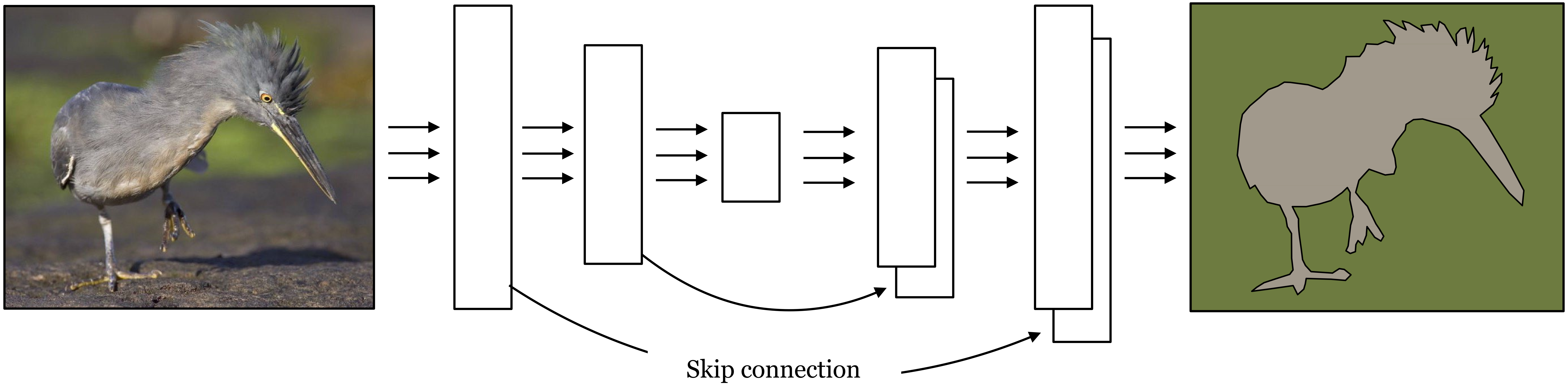


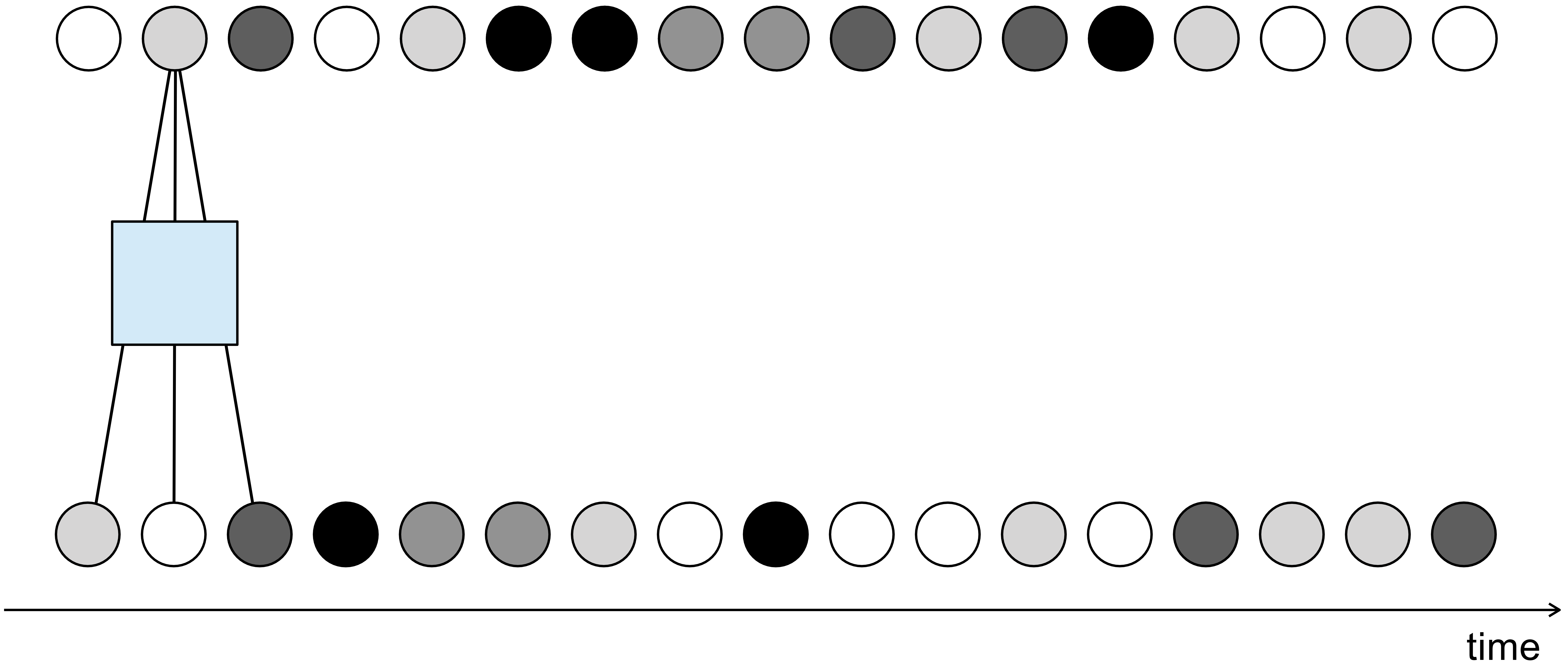
Image-to-image



U-net



Convolutions in time




m
t
n