Carnegie Mellon University
Heinz College

Deep Learning for Analyzing Images and Time Series

nearly all slides by George Chen (CMU)

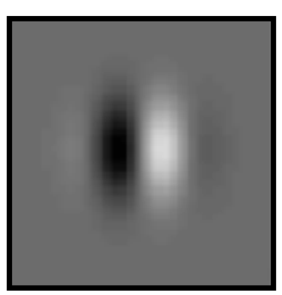
1 slide by Phillip Isola (OpenAI, UC Berkeley)

CMU 95-865 Fall 2017

Image Analysis with Convolutional Neural Nets (CNNs, also called convnets)

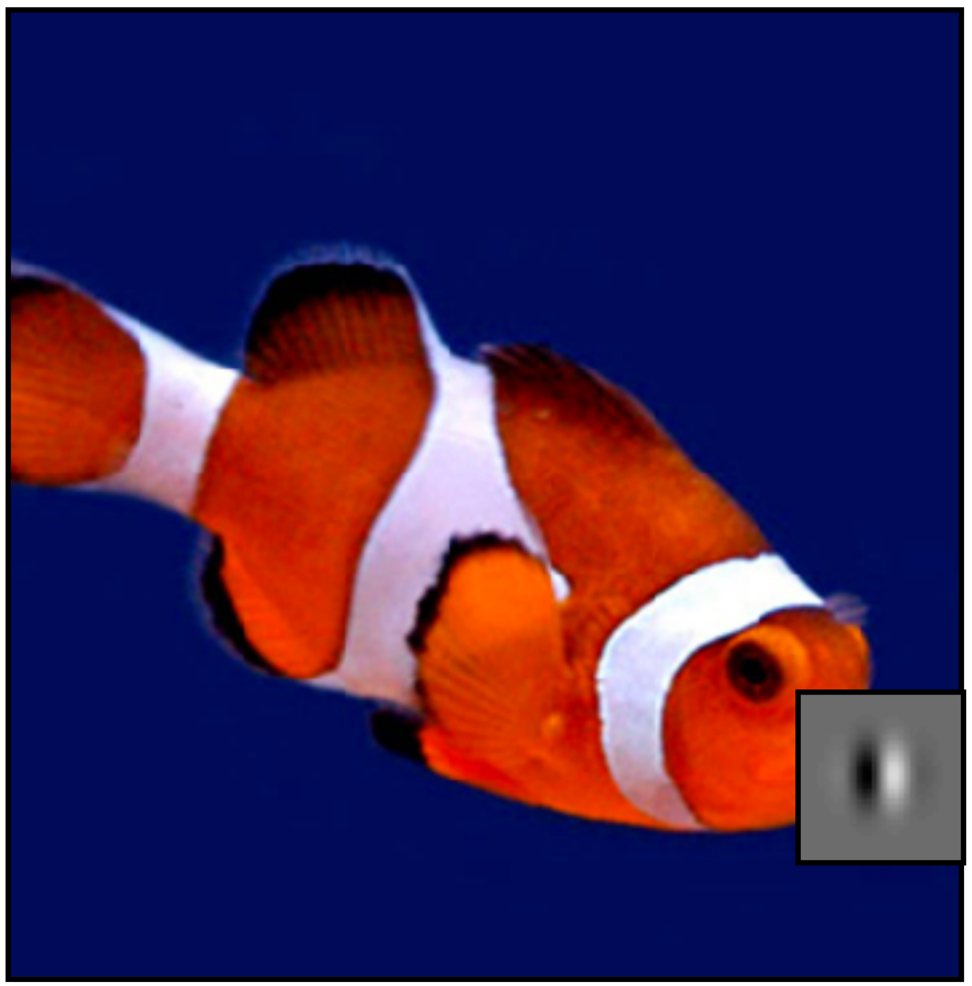
Convolution

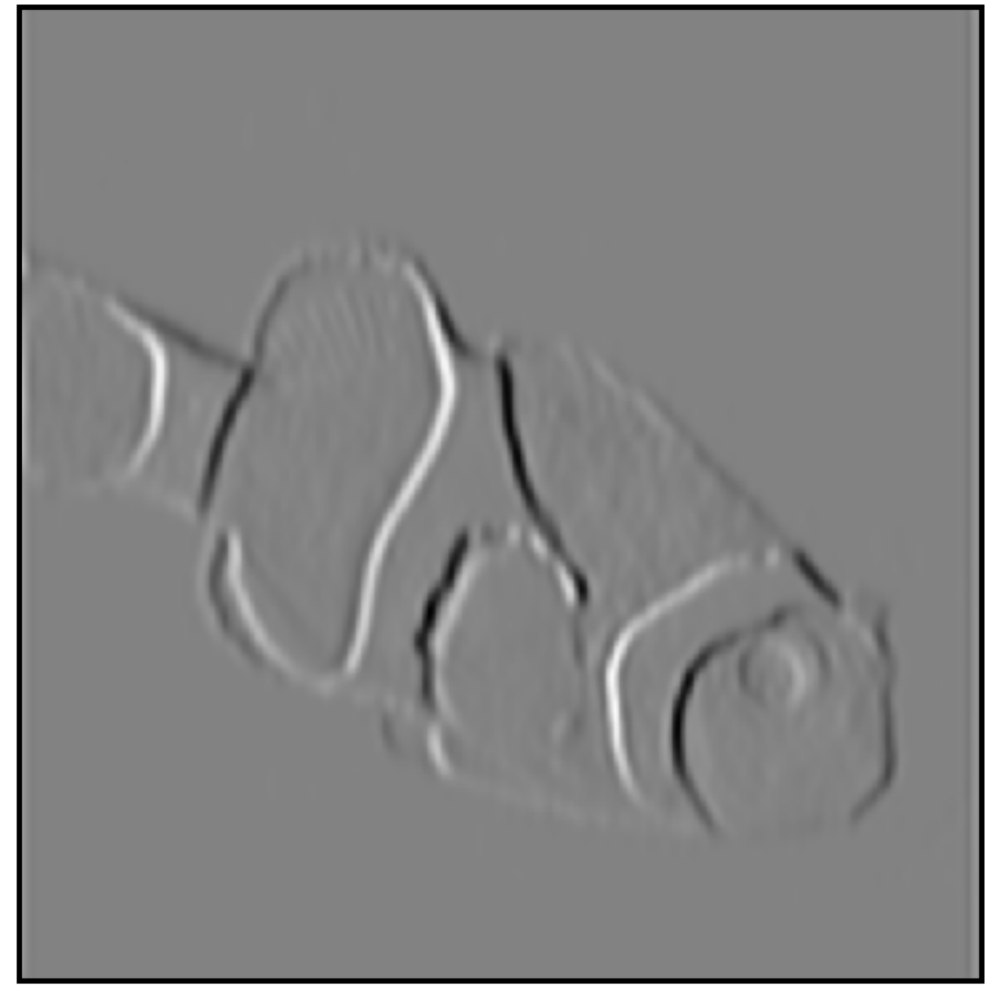
filter



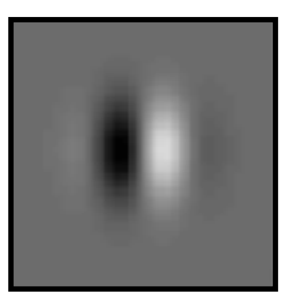
Slide by Phillip Isola

Convolution





filter



Slide by Phillip Isola

Convolution

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

Input image

0	0	0
0	1	0
0	0	0

Filter
(also called “kernel”)

Convolution

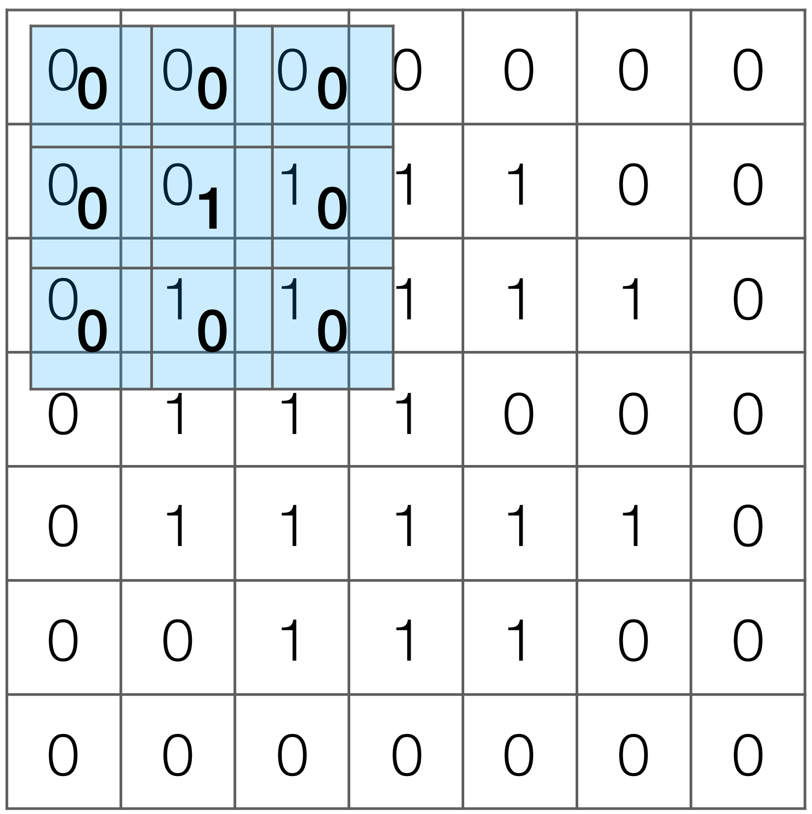
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0
0	1	1	1	1	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input image

0	0	0
0	1	0
0	0	0

Filter
(also called “kernel”)

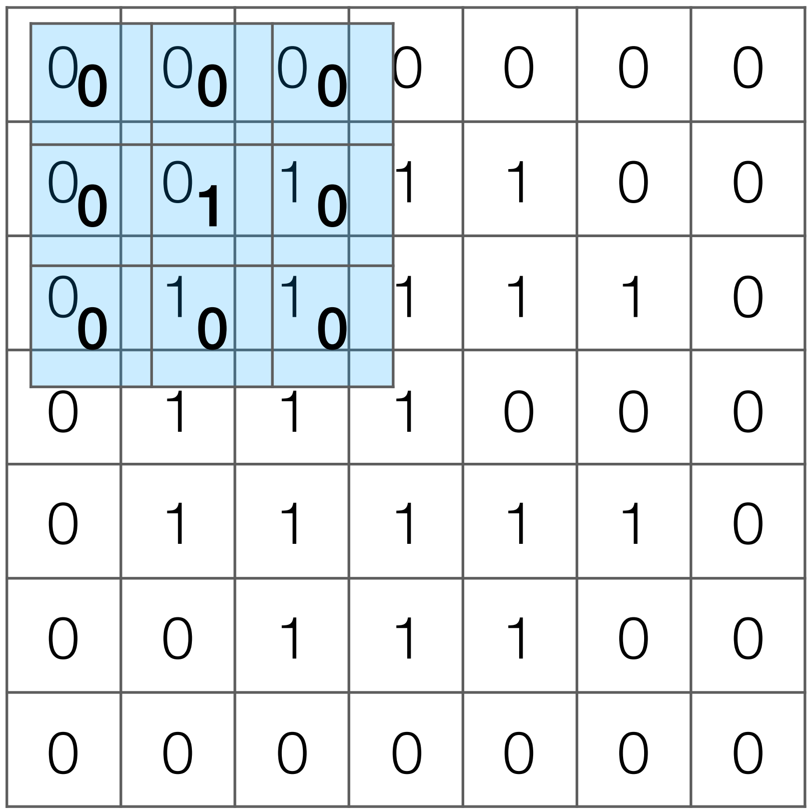
Convolution



Input image

Convolution

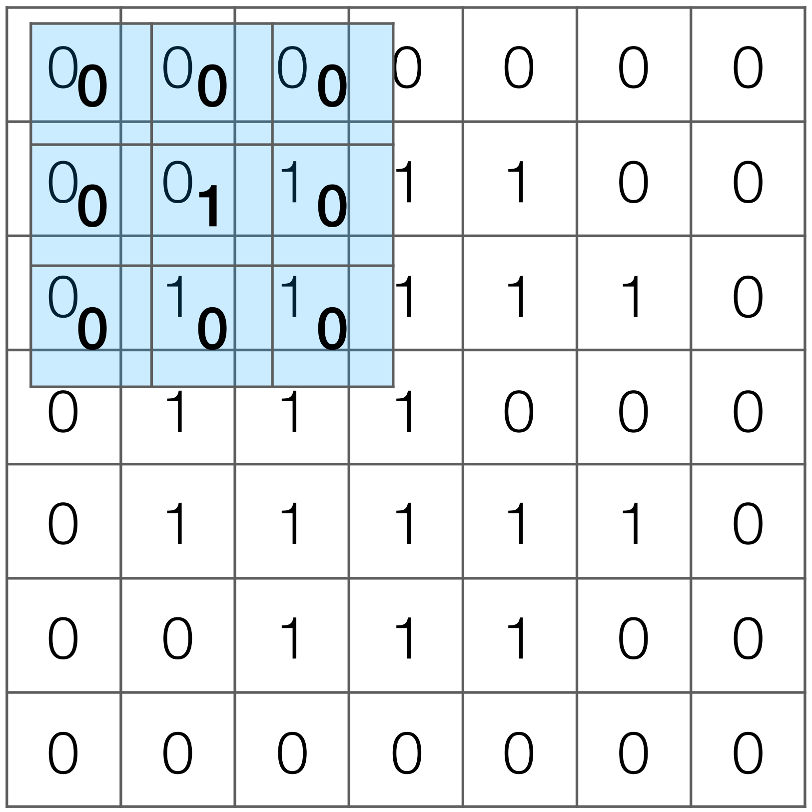
Take dot product!



Input image

Convolution

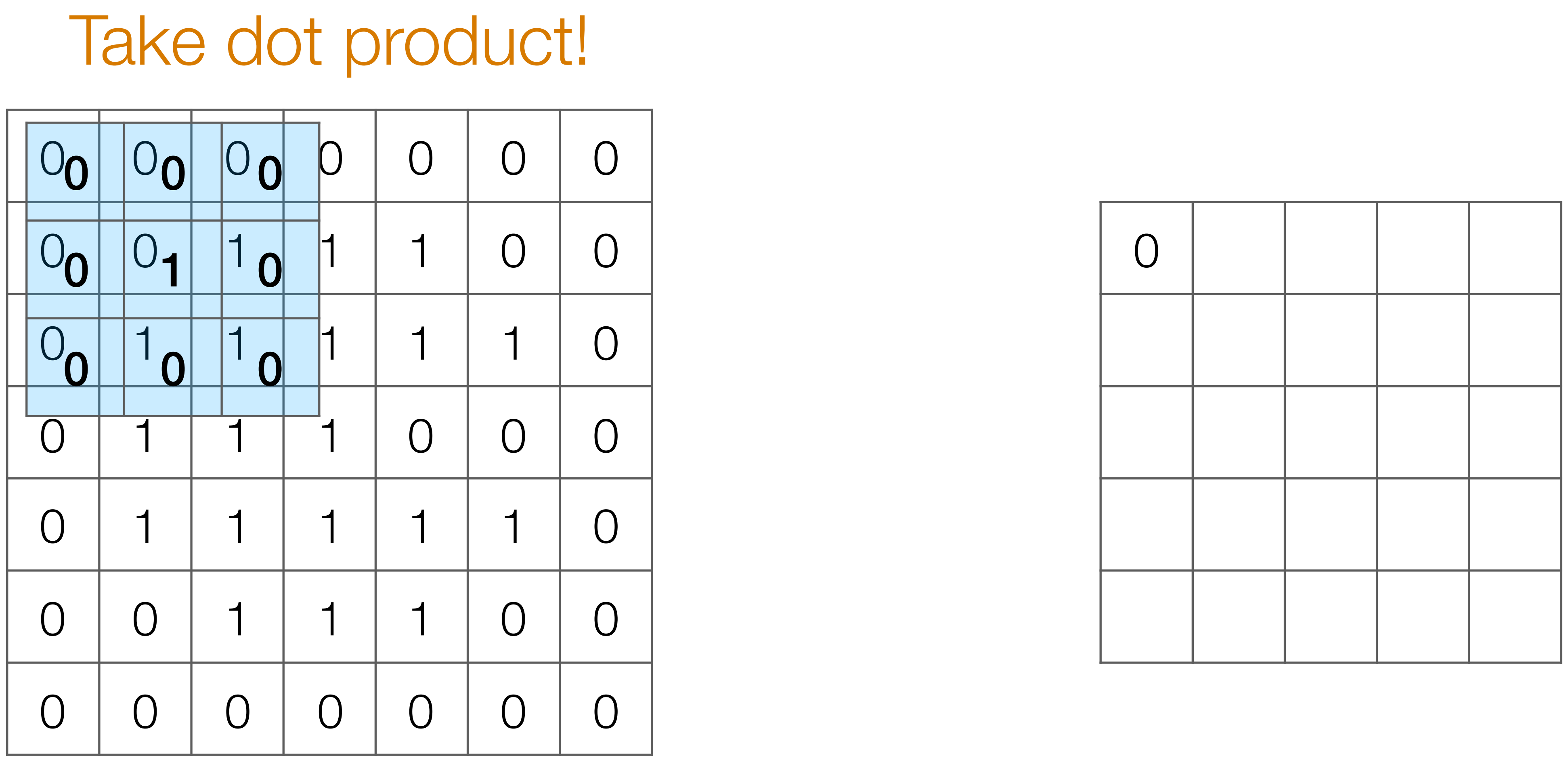
Take dot product!



Input image

Output image

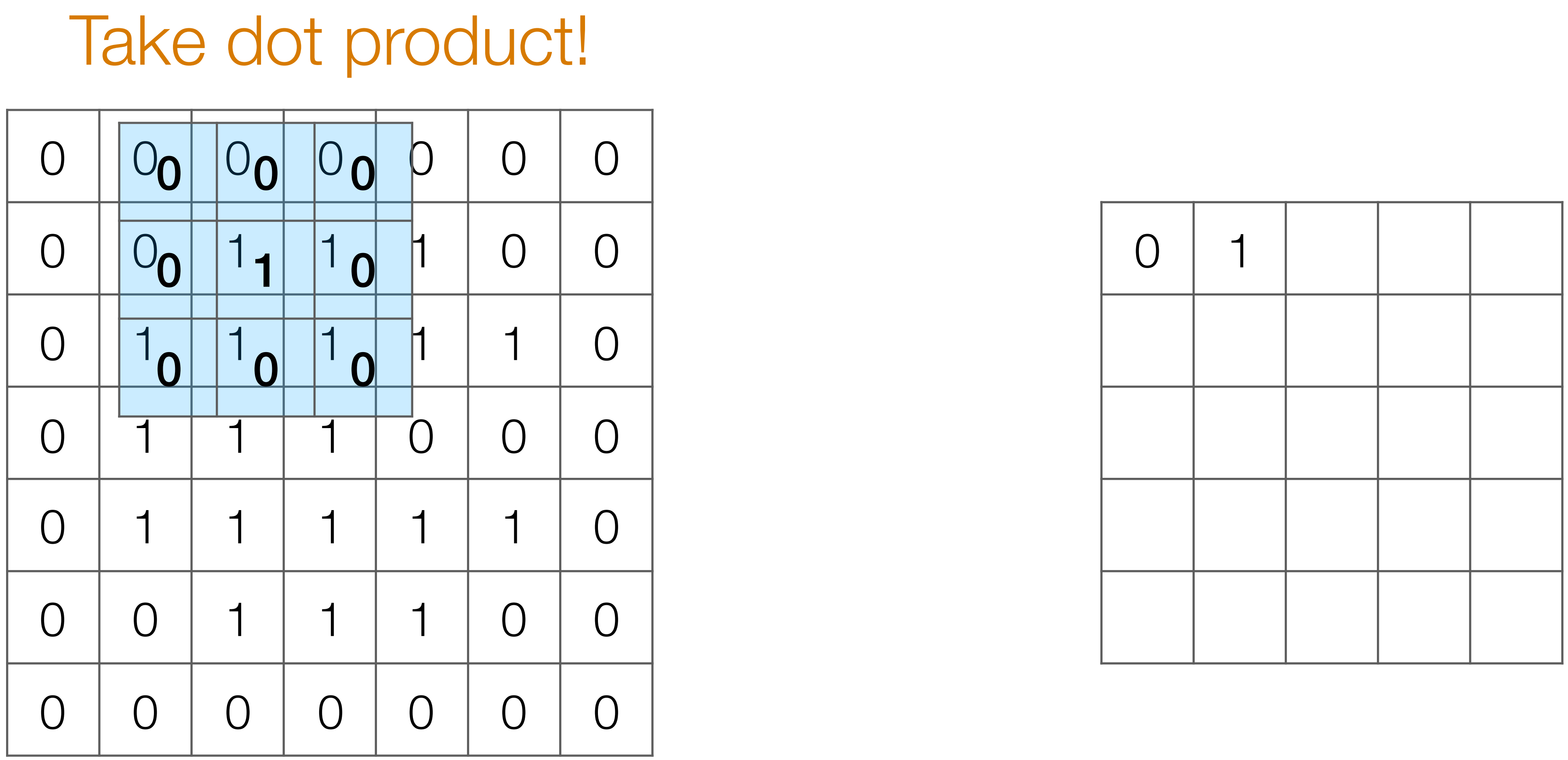
Convolution



Input image

Output image

Convolution

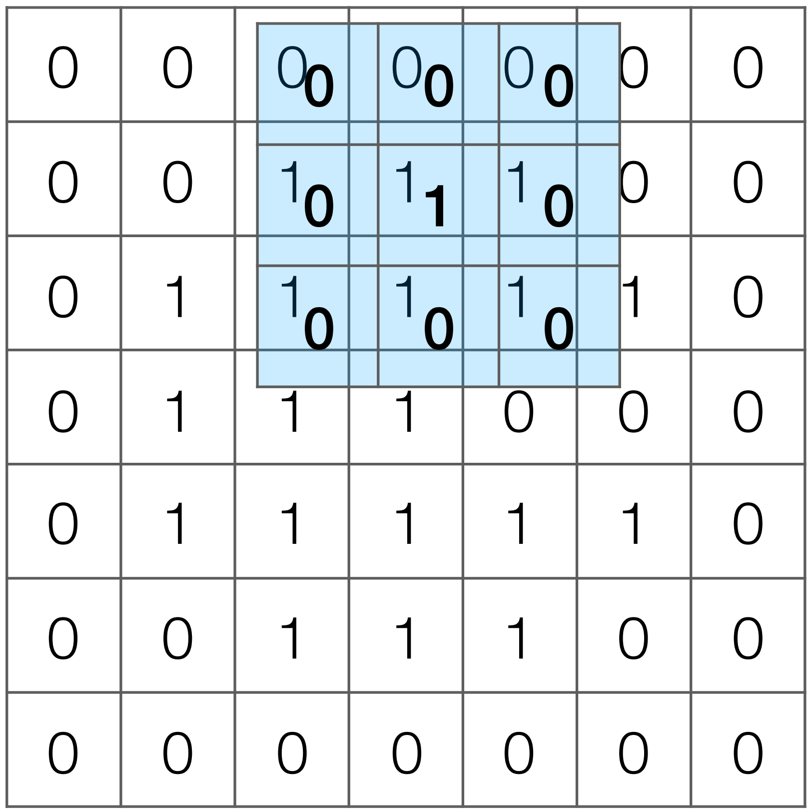


Input image

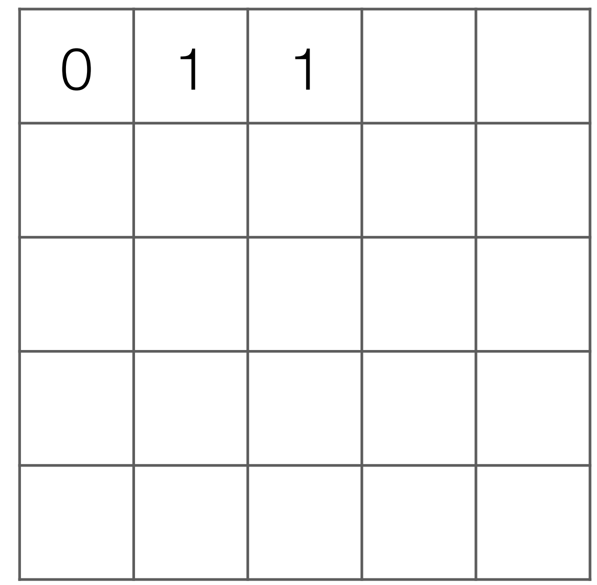
Output image

Convolution

Take dot product!



Input image



Output image

Convolution

Take dot product!

0	0	0	$0_{\mathbf{0}}$	$0_{\mathbf{0}}$	$0_{\mathbf{0}}$	0
0	0	1	$1_{\mathbf{0}}$	$1_{\mathbf{1}}$	$0_{\mathbf{0}}$	0
0	1	1	$1_{\mathbf{0}}$	$1_{\mathbf{0}}$	$1_{\mathbf{0}}$	0
0	1	1	1	0	0	0
0	1	1	1	1	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

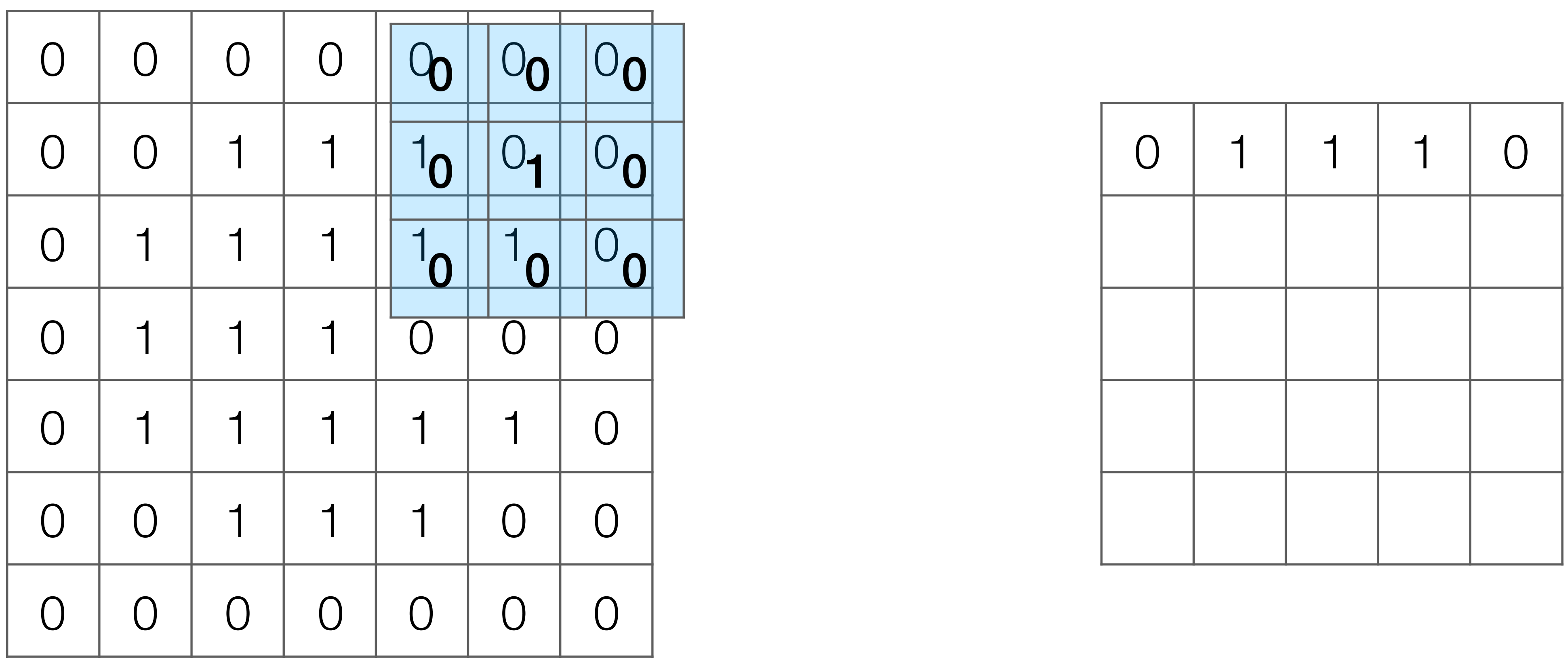
Input image

0	1	1	1	

Output image

Convolution

Take dot product!

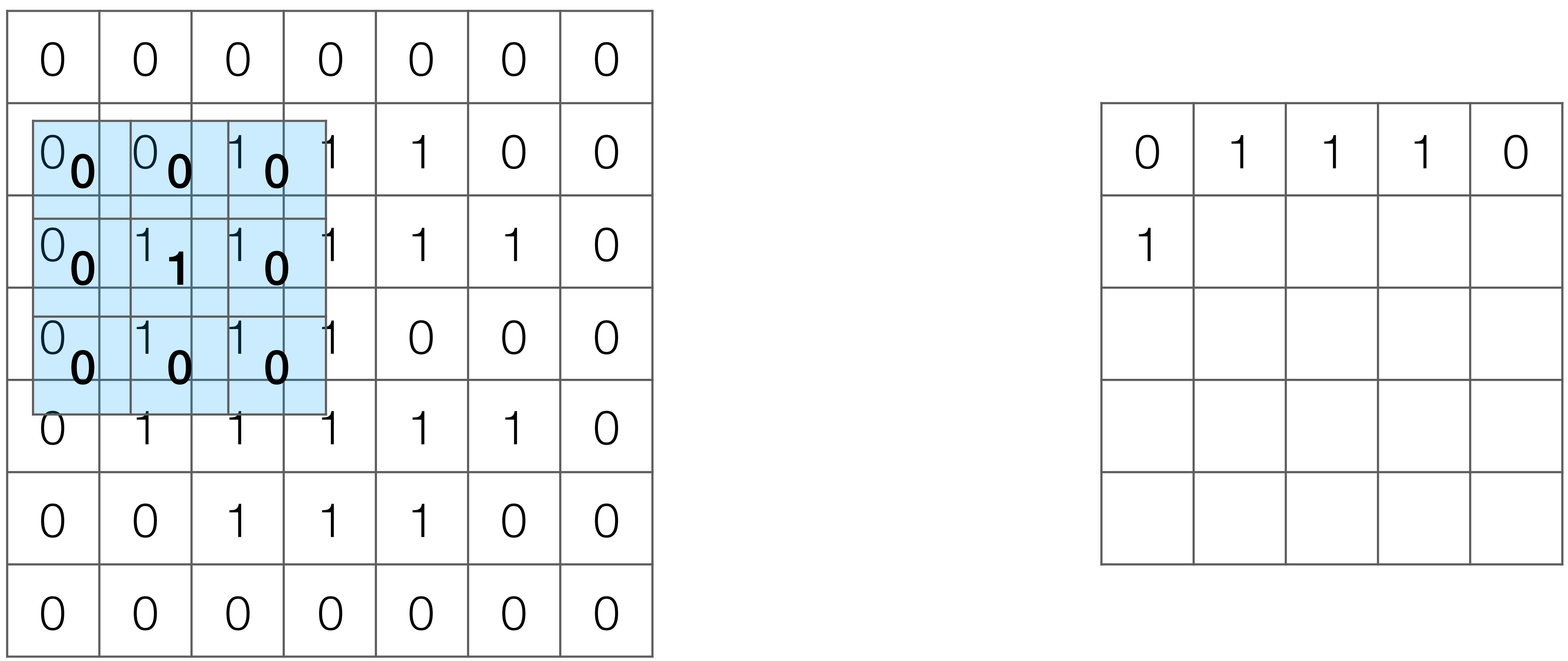


Input image

Output image

Convolution

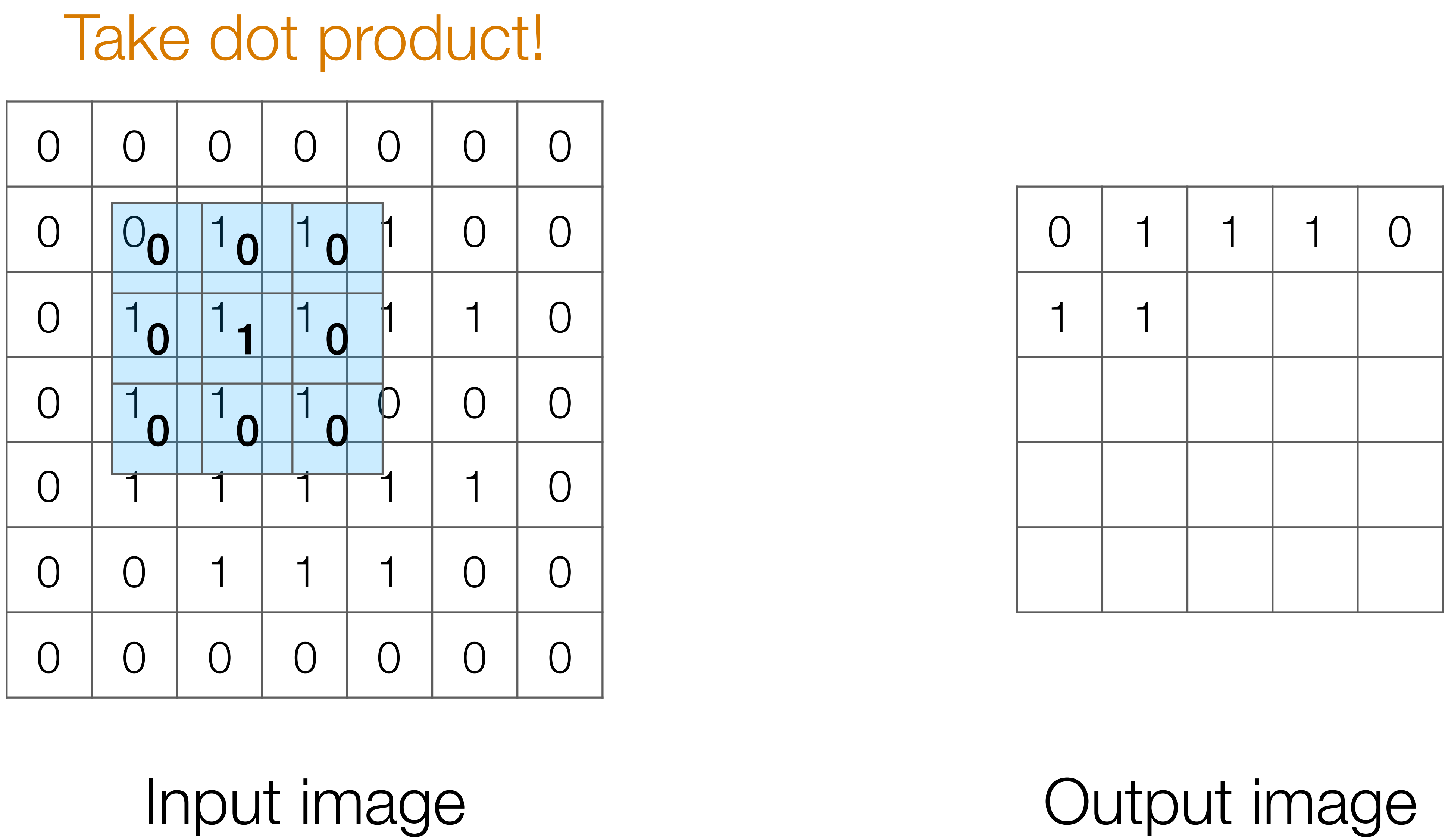
Take dot product!



Input image

Output image

Convolution



Convolution

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
\mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0}
\end{bmatrix}
=
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0
\end{bmatrix}
$$

Input image

Output image

Convolution

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

Input image

*

0	**0**	**0**
0	**1**	**0**
0	**0**	**0**

=

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

Output image

Note: output image is smaller than input image

Convolution

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

Input image

$*$

0	**0**	**0**
0	**1**	**0**
0	**0**	**0**

$=$

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

Output image

Note: output image is smaller than input image

If you want output size to be same as input, pad 0's to input

Convolution

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
\mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{1} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0}
\end{bmatrix}
=
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0
\end{bmatrix}
$$

Input image

Output image

Note: output image is smaller than input image

If you want output size to be same as input, pad 0’s to input

Convolution

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
*
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
=
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0
\end{bmatrix}
$$

Input image

Output image

Note: output image is smaller than input image

If you want output size to be same as input, pad 0’s to input

Convolution

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	1	1	1	1	1	0	**0**
0	0	1	1	1	0	0	0	**0**
0	0	1	1	1	1	1	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**

Input image

*

0	**0**	**0**
0	**1**	**0**
0	**0**	**0**

=

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

Output image

Note: output image is smaller than input image

If you want output size to be same as input, pad 0's to input

Convolution

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

*

0	**0**	**0**
0	**1**	**0**
0	**0**	**0**

=

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

Input image

Output image

Convolution

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

*

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

=

3	5	6	5	3
5	8	8	6	3
6	9	8	7	4
5	8	8	6	3
3	5	6	5	3

Input image

Output image

Convolution

$$\begin{bmatrix} 0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0\\ 0&1&1&1&1&1&0\\ 0&1&1&1&0&0&0\\ 0&1&1&1&1&1&0\\ 0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0 \end{bmatrix} * \frac{1}{9}\begin{bmatrix} 1&1&1\\ 1&1&1\\ 1&1&1 \end{bmatrix} = \frac{1}{9}\begin{bmatrix} 3&5&6&5&3\\ 5&8&8&6&3\\ 6&9&8&7&4\\ 5&8&8&6&3\\ 3&5&6&5&3 \end{bmatrix}$$

Input image

Output image

Convolution

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

Output image

Convolution

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution

Very commonly used for:

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution

Very commonly used for:

- Blurring an image

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution

Very commonly used for:

- Blurring an image

$$* \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} =$$

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution

Very commonly used for:

- Blurring an image

*

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

- Finding edges

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution

Very commonly used for:

- Blurring an image

*

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

- Finding edges

*

-1	-1	-1
2	2	2
-1	-1	-1

=

(this example finds horizontal edges)

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

-1	-1	-1
2	2	2
-1	-1	-1

0	-1	0
-1	4	-1
0	-1	0

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

convolve with
each filter

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

-1	-1	-1
2	2	2
-1	-1	-1

0	-1	0
-1	4	-1
0	-1	0

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

-1	-1	-1
2	2	2
-1	-1	-1

convolve with
each filter

0	-1	0
-1	4	-1
0	-1	0

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

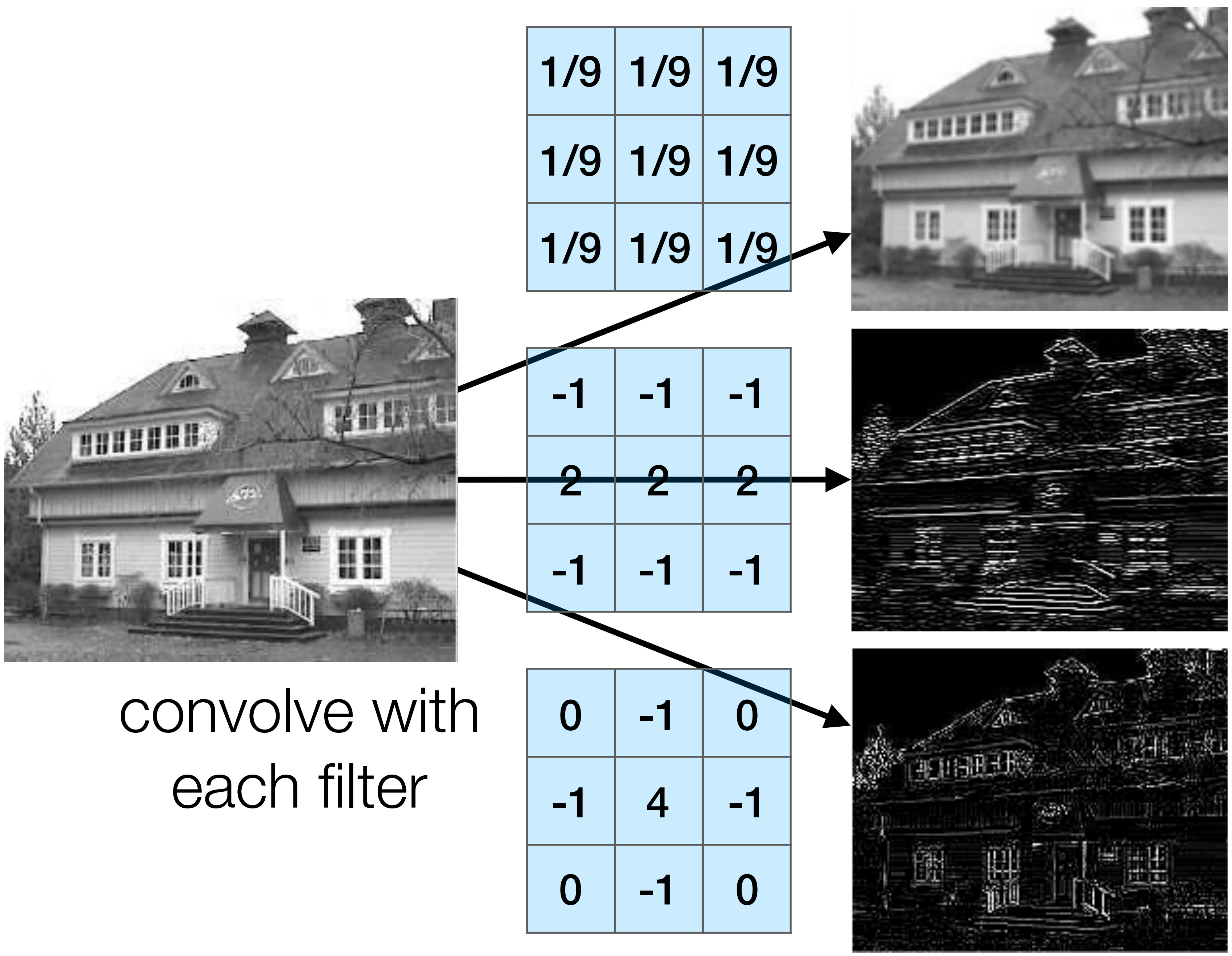
-1	-1	-1
2	2	2
-1	-1	-1

0	-1	0
-1	4	-1
0	-1	0

convolve with each filter

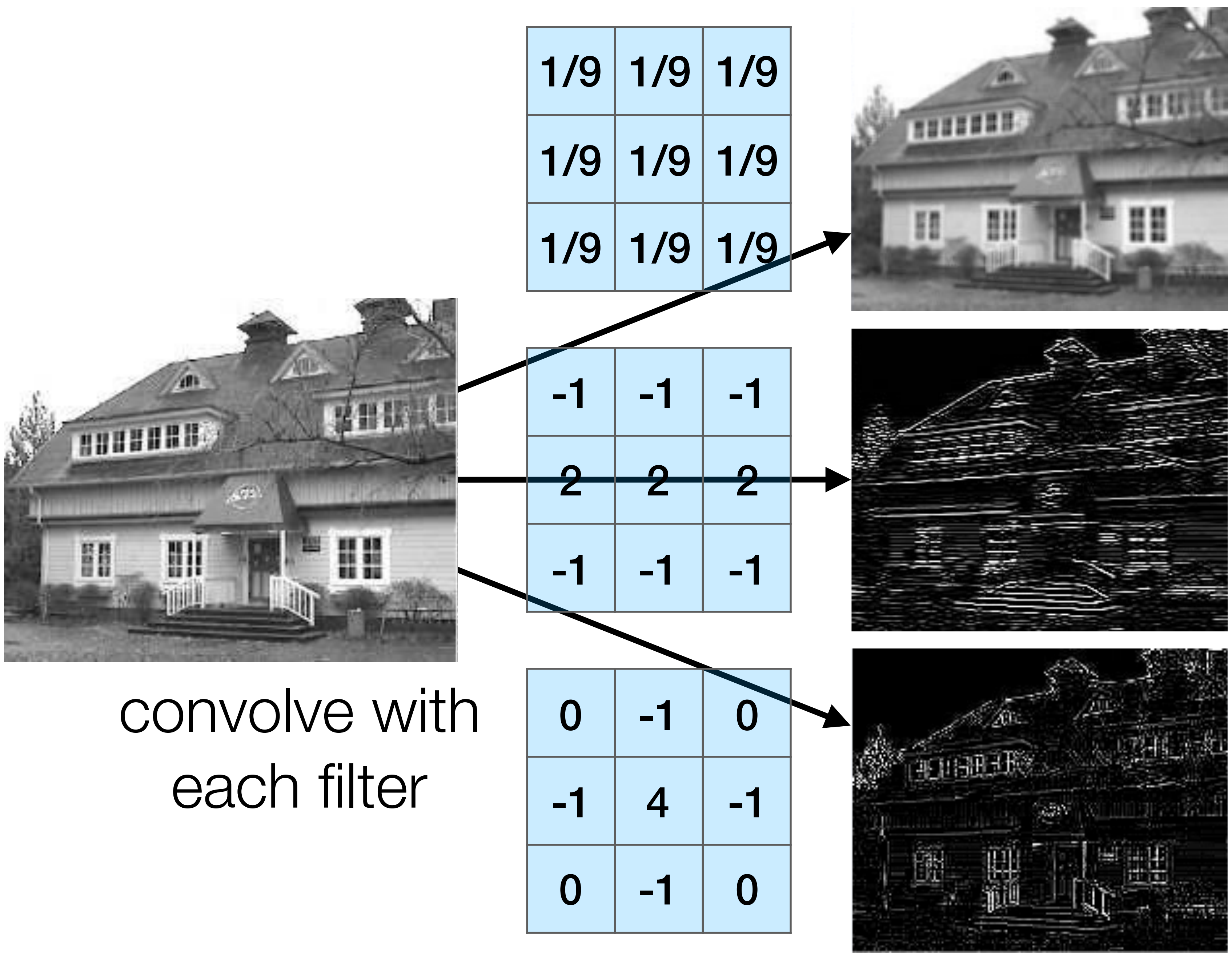
Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer



Images from: http://aishack.in/tutorials/image-convolution-examples/

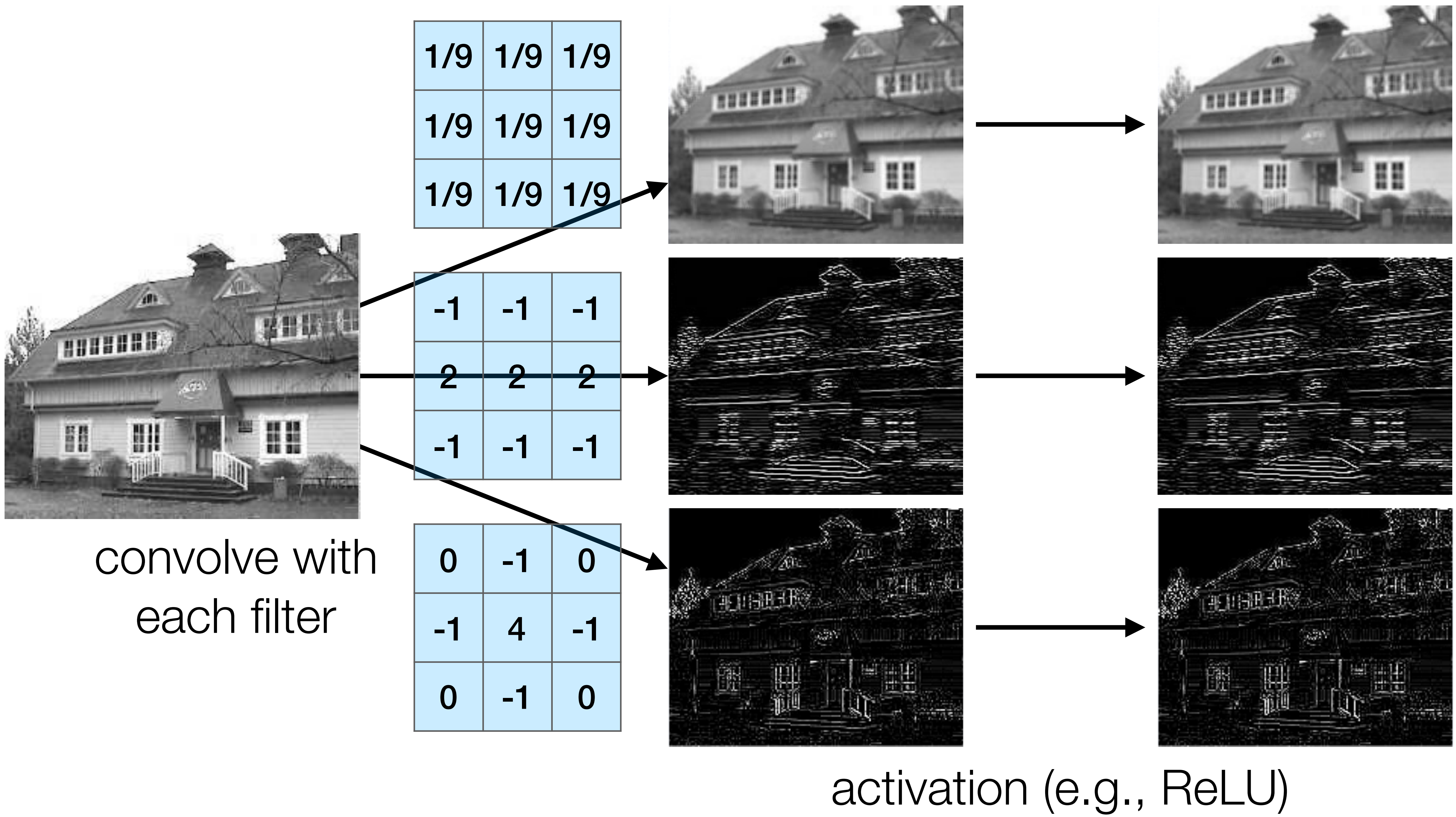
Convolution Layer



activation (e.g., ReLU)

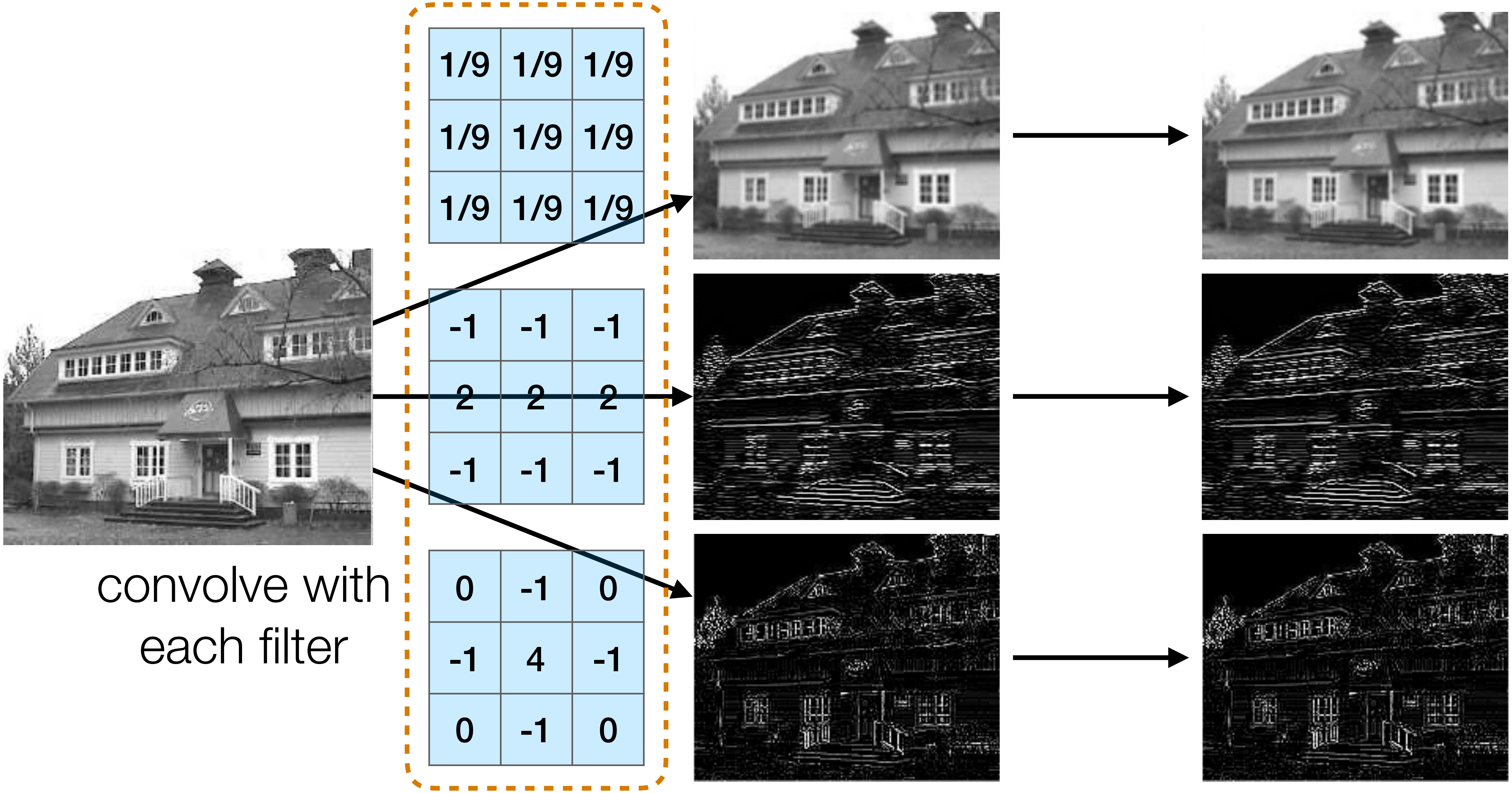
Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer



Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

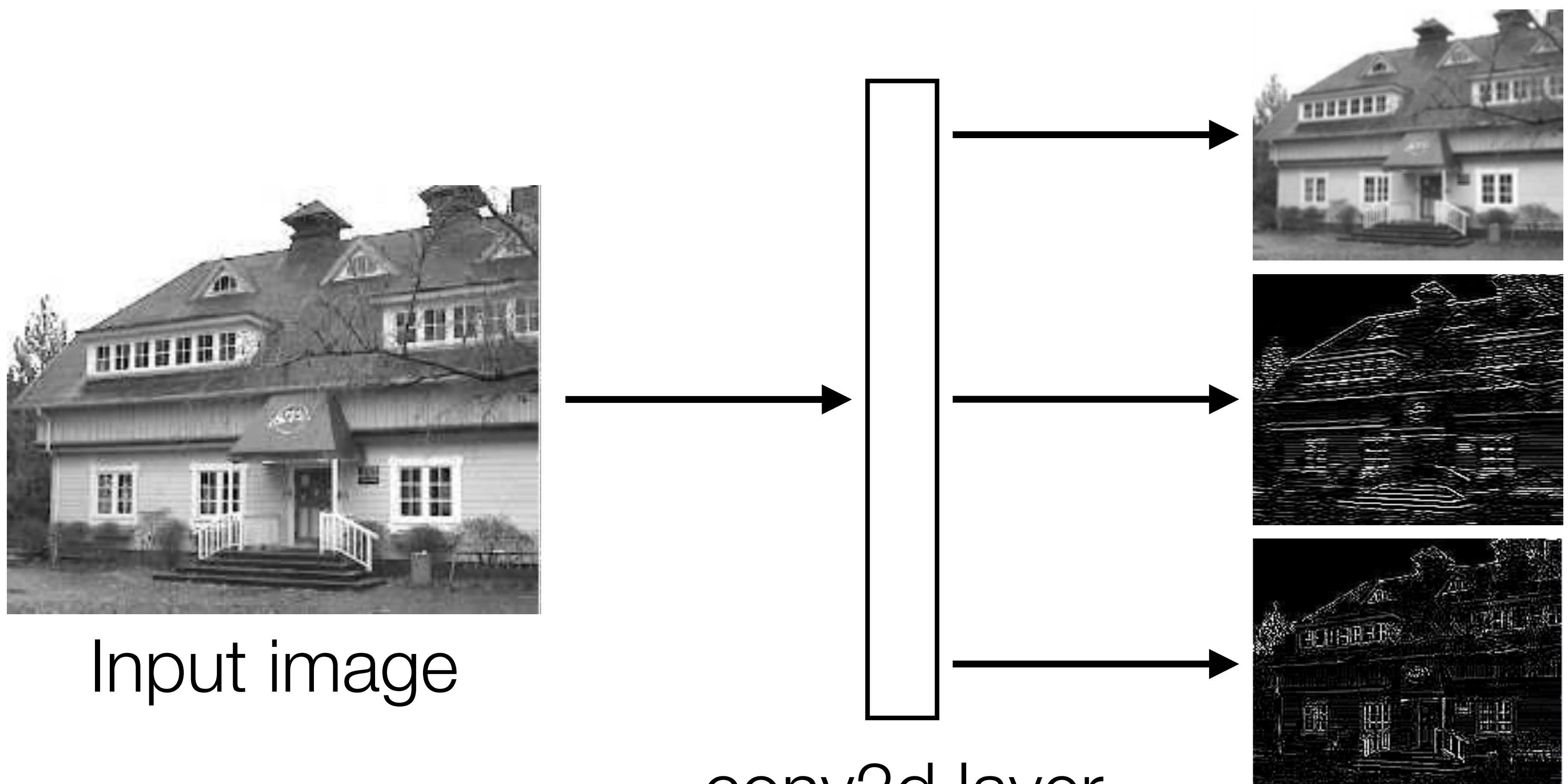


filters are actually unknown
and are learned!

activation (e.g., ReLU)

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer



Input image

Output images

conv2d layer
with ReLu activation
and a three 3x3 kernels

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input image

conv2d layer
with ReLu activation
and a three 3x3 kernels

Stack output
images into a
single “output
feature map”

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input image

dimensions:
height,
width

conv2d layer
with ReLu activation
and a three 3x3 kernels

Stack output images into a single “output feature map”

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input image

dimensions:
height,
width

conv2d layer
with ReLu activation
and a three 3x3 kernels

Stack output images into a single "output feature map"

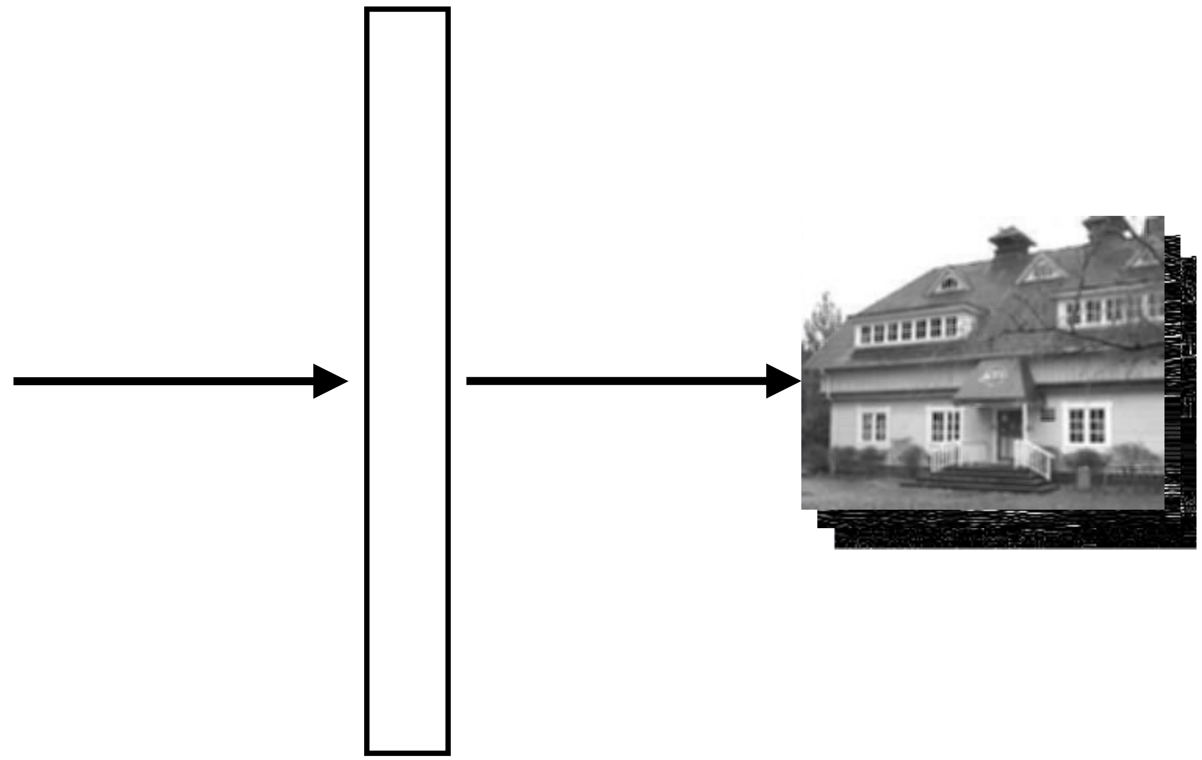
dimensions:
height-2,
width-2,
number of kernels
(3 in this case)

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input image

dimensions:
height,
width



conv2d layer
with ReLu activation
and *k* 3x3 kernels

Stack output images into a single "output feature map"

dimensions:
height-2,
width-2,
k

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input image

dimensions:
height,
width,
depth *d* (# channels)

conv2d layer
with ReLu activation
and *k* 3x3x*d* kernels

Stack output images into a single “output feature map”

dimensions:
height-2,
width-2,
k

Images from: http://aishack.in/tutorials/image-convolution-examples/

Convolution Layer

Input image

dimensions:
height,
width,
depth *d* (# channels)

conv2d layer
with ReLu activation
and *k* 3x3x*d* kernels

technical detail: there's also a bias vector

Stack output images into a single "output feature map"

dimensions:
height-2,
width-2,
k

Images from: http://aishack.in/tutorials/image-convolution-examples/

Pooling

Pooling

- Aggregate local information

Pooling

- Aggregate local information

- Produces a smaller image
  (each resulting pixel captures some “global” information)

Max Pooling

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

Input image

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image
after ReLU

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

Output after max pooling

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

Output after max pooling

Max Pooling

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

Input image

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image
after ReLU

1	

Output after
max pooling

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	3

Output after max pooling

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	3
1	

Output after max pooling

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	3
1	3

Output after max pooling

Max Pooling

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

Input image

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

1	3
1	3

Output after max pooling

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

What numbers were involved in computing this 1?

1	3
1	3

Output after max pooling

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

Output image after ReLU

What numbers were involved in computing this 1?

1	3
1	3

Output after max pooling

Max Pooling

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

Input image

$$\begin{bmatrix} 0&1&3&1&0\\ 1&1&1&3&3\\ 0&0&0&0&0\\ 1&1&1&3&3\\ 0&1&3&1&0 \end{bmatrix}$$

Output image after ReLU

What numbers were involved in computing this 1?

$$\begin{bmatrix} 1&3\\ 1&3 \end{bmatrix}$$

Output after max pooling

Max Pooling

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

Input image

*

-1	**-1**	**-1**
2	**2**	**2**
-1	**-1**	**-1**

=

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

0	1	3	1	0
1	1	1	3	3
0	0	0	0	0
1	1	1	3	3
0	1	3	1	0

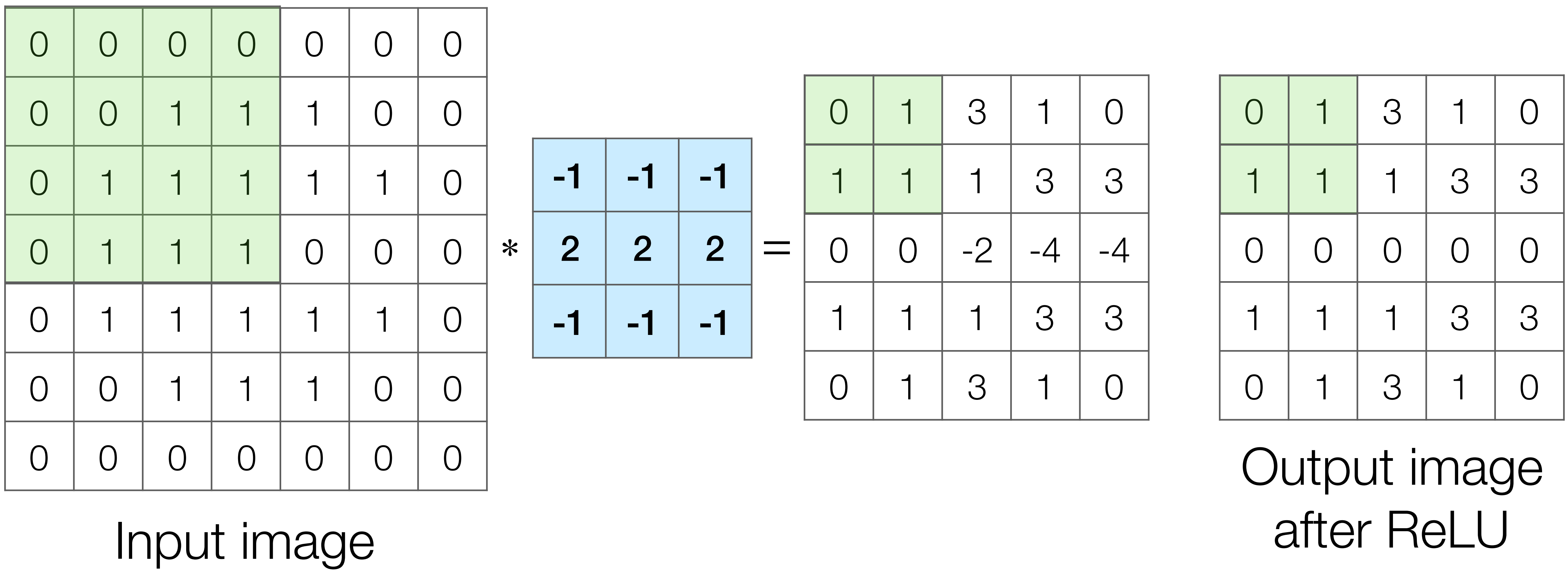
Output image after ReLU

What numbers were involved in computing this 1?

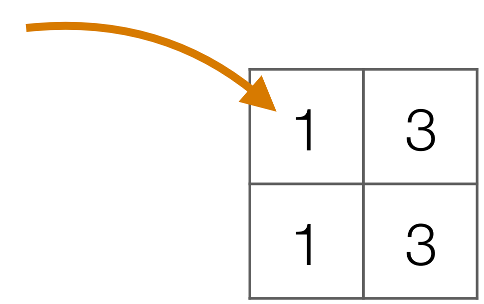
1	3
1	3

Output after max pooling

Max Pooling



What numbers were involved in computing this 1?



In this example: 1 pixel in max pooling output captures information from 16 input pixels!

Output after
max pooling

Max Pooling

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

Input image

$$\begin{bmatrix} 0&1&3&1&0\\ 1&1&1&3&3\\ 0&0&0&0&0\\ 1&1&1&3&3\\ 0&1&3&1&0 \end{bmatrix}$$

Output image after ReLU

What numbers were involved in computing this 1?

$$\begin{bmatrix} 1&3\\ 1&3 \end{bmatrix}$$

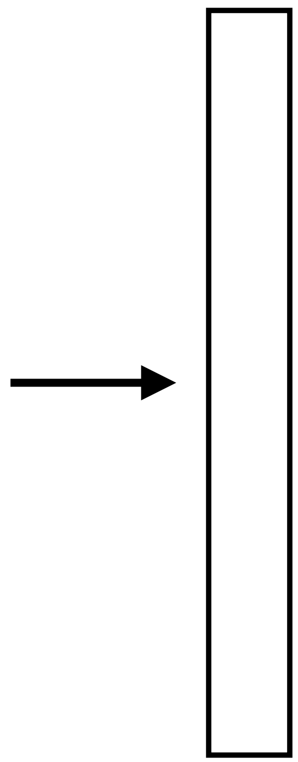
Output after max pooling

In this example: 1 pixel in max pooling output captures information from 16 input pixels!

Example: applying max pooling again results in a single pixel that captures info from entire input image!

Basic Building Block of CNN's

Input image

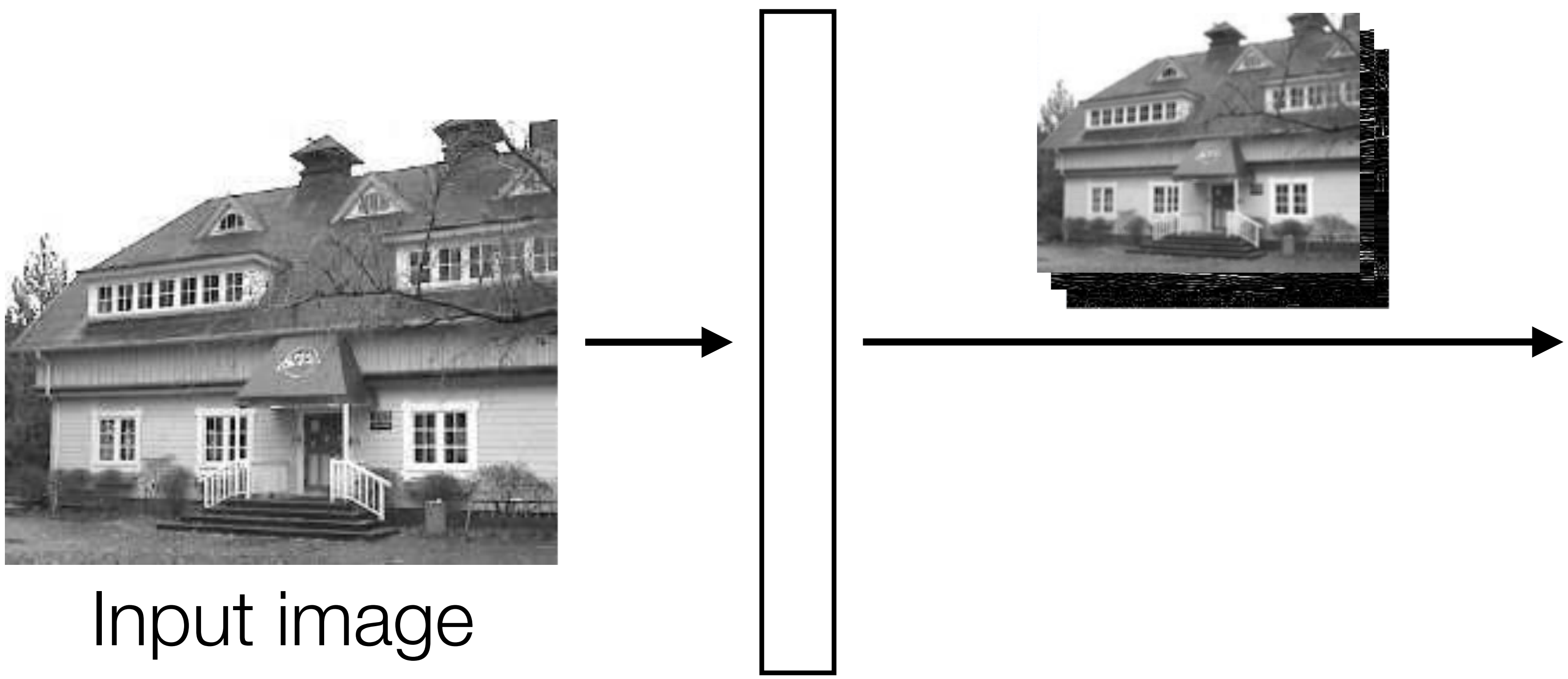


conv2d layer
with ReLu activation
and *k* kernels

Images from: http://aishack.in/tutorials/image-convolution-examples/

Basic Building Block of CNN's

stack of images

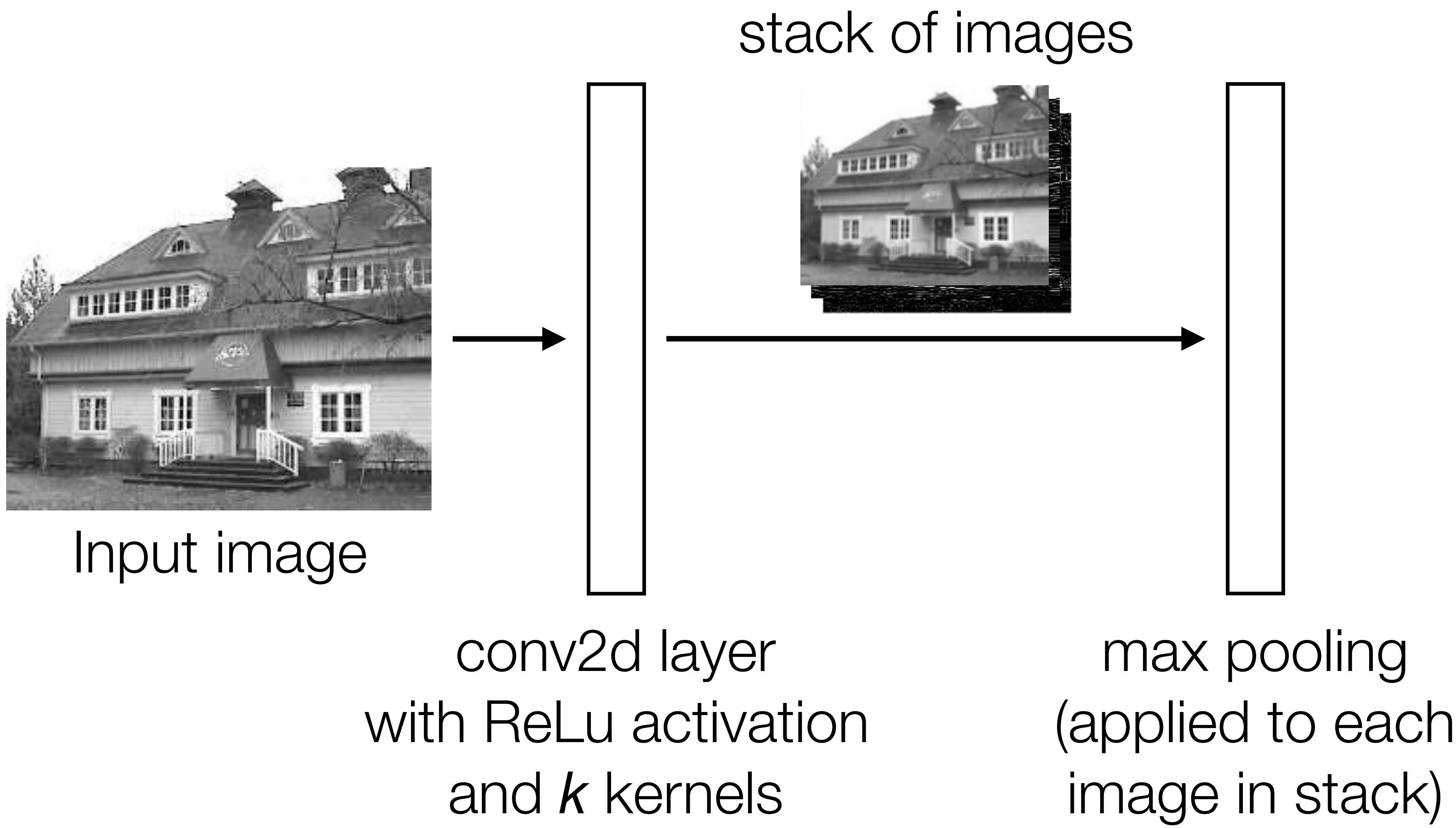


Input image

conv2d layer
with ReLu activation
and *k* kernels

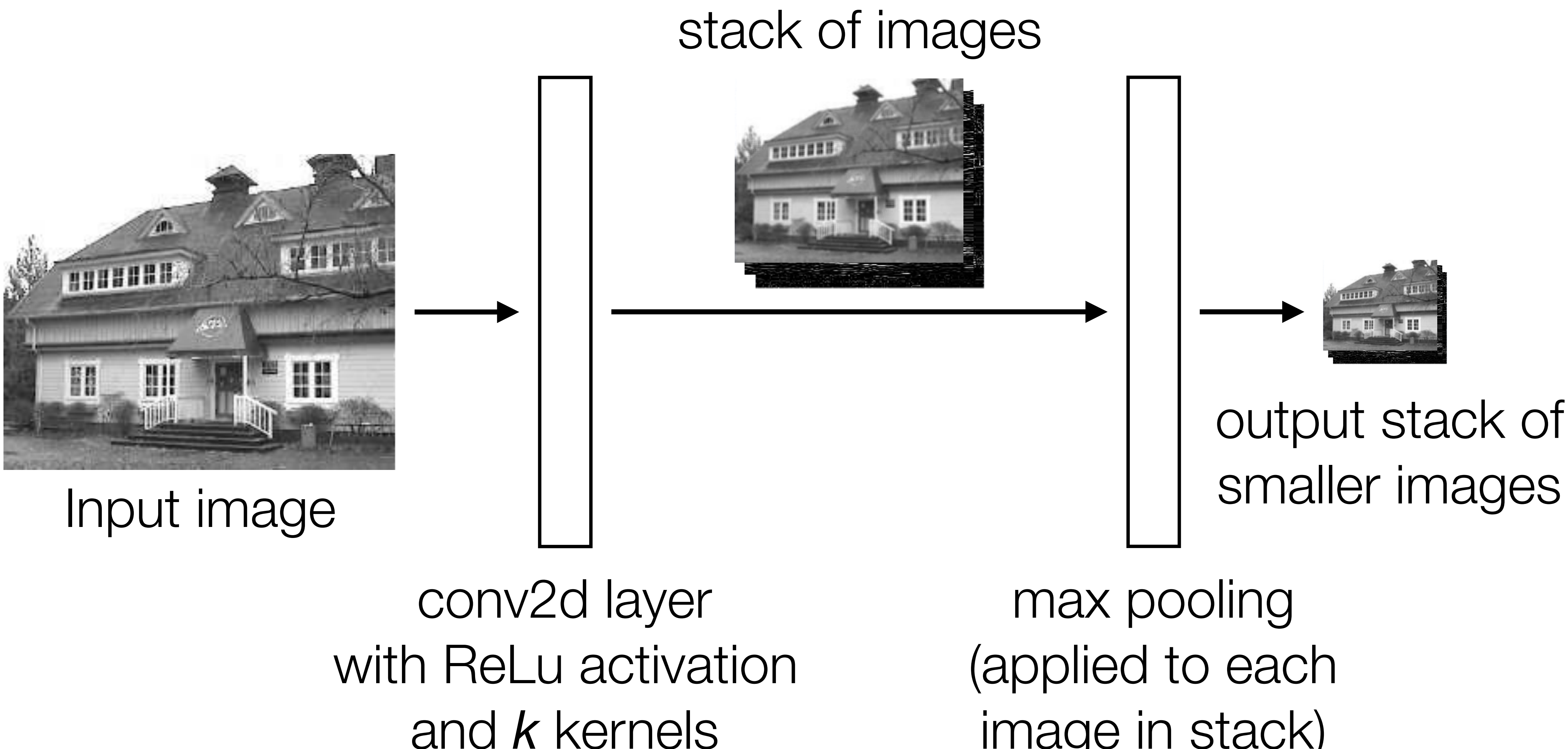
Images from: http://aishack.in/tutorials/image-convolution-examples/

Basic Building Block of CNN's



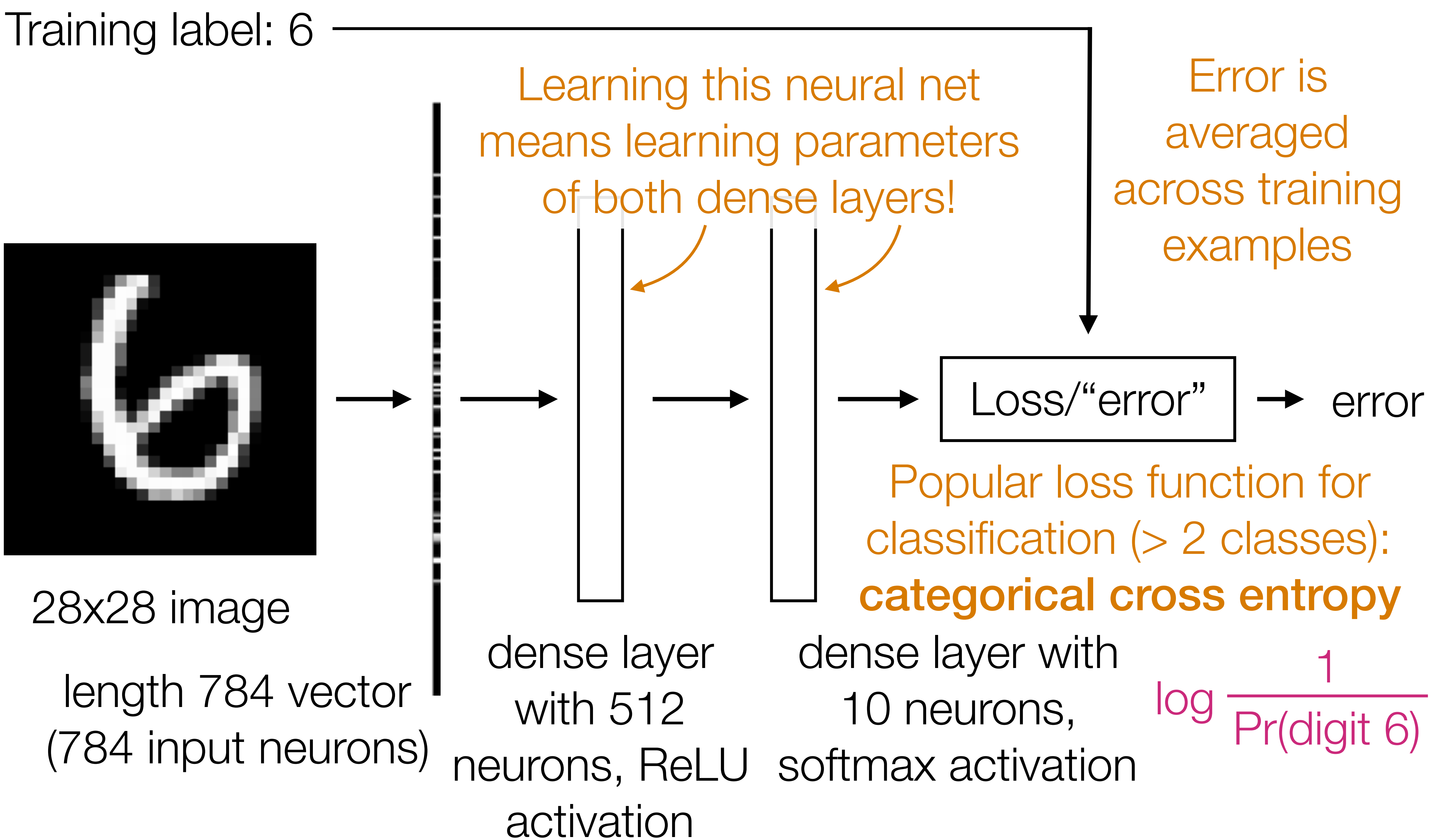
Images from: http://aishack.in/tutorials/image-convolution-examples/

Basic Building Block of CNN's

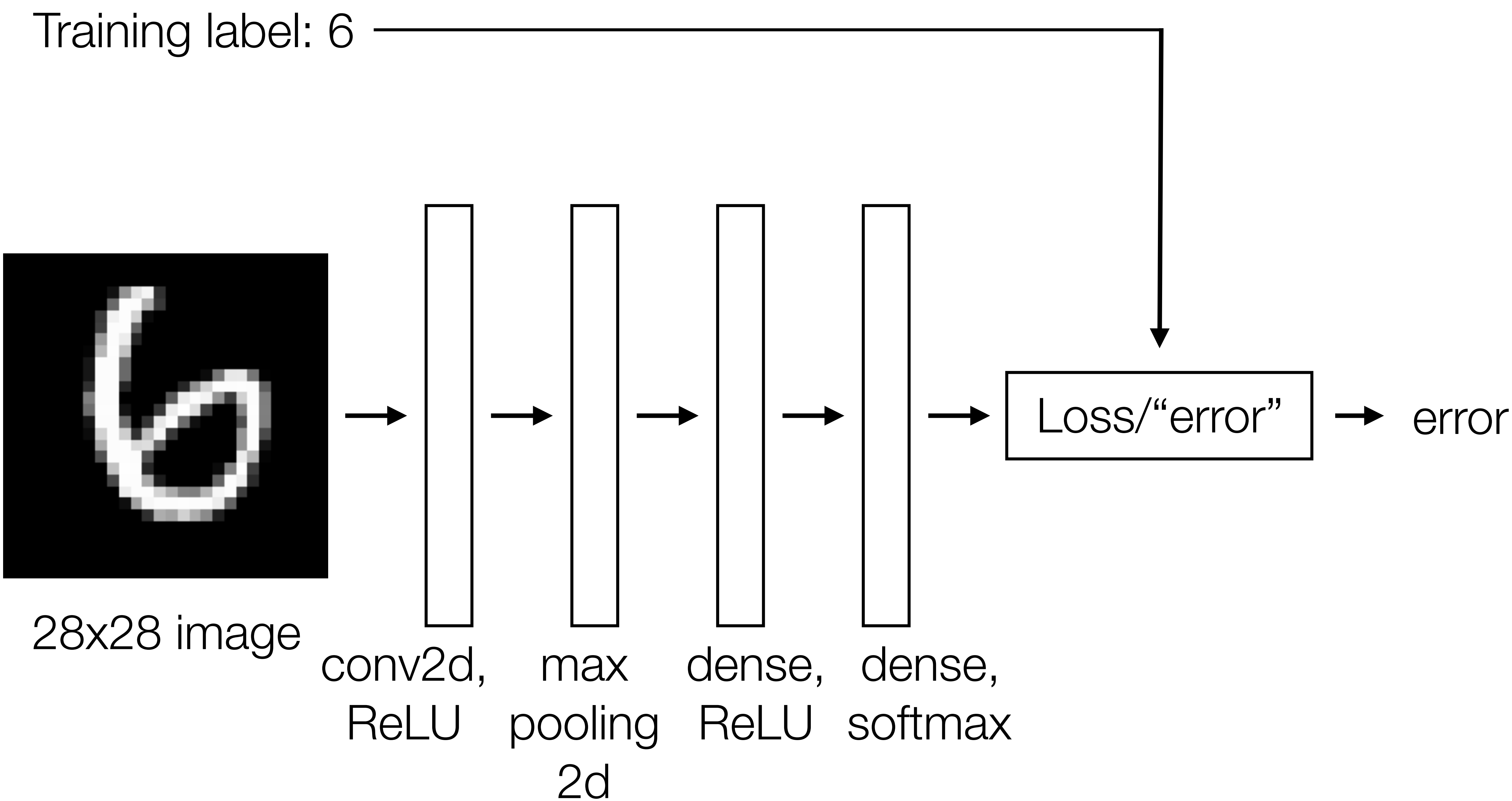


Images from: http://aishack.in/tutorials/image-convolution-examples/

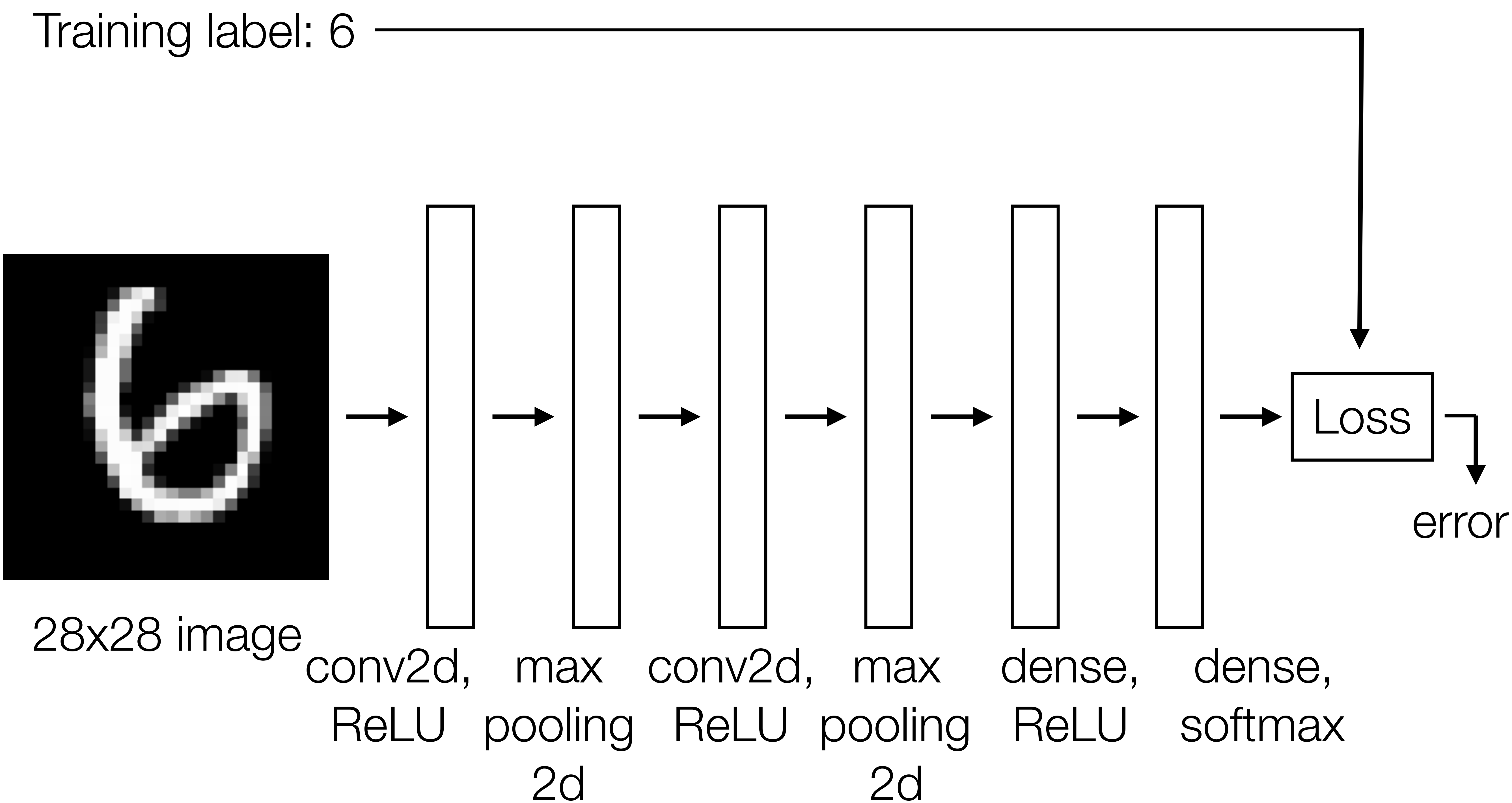
Handwritten Digit Recognition



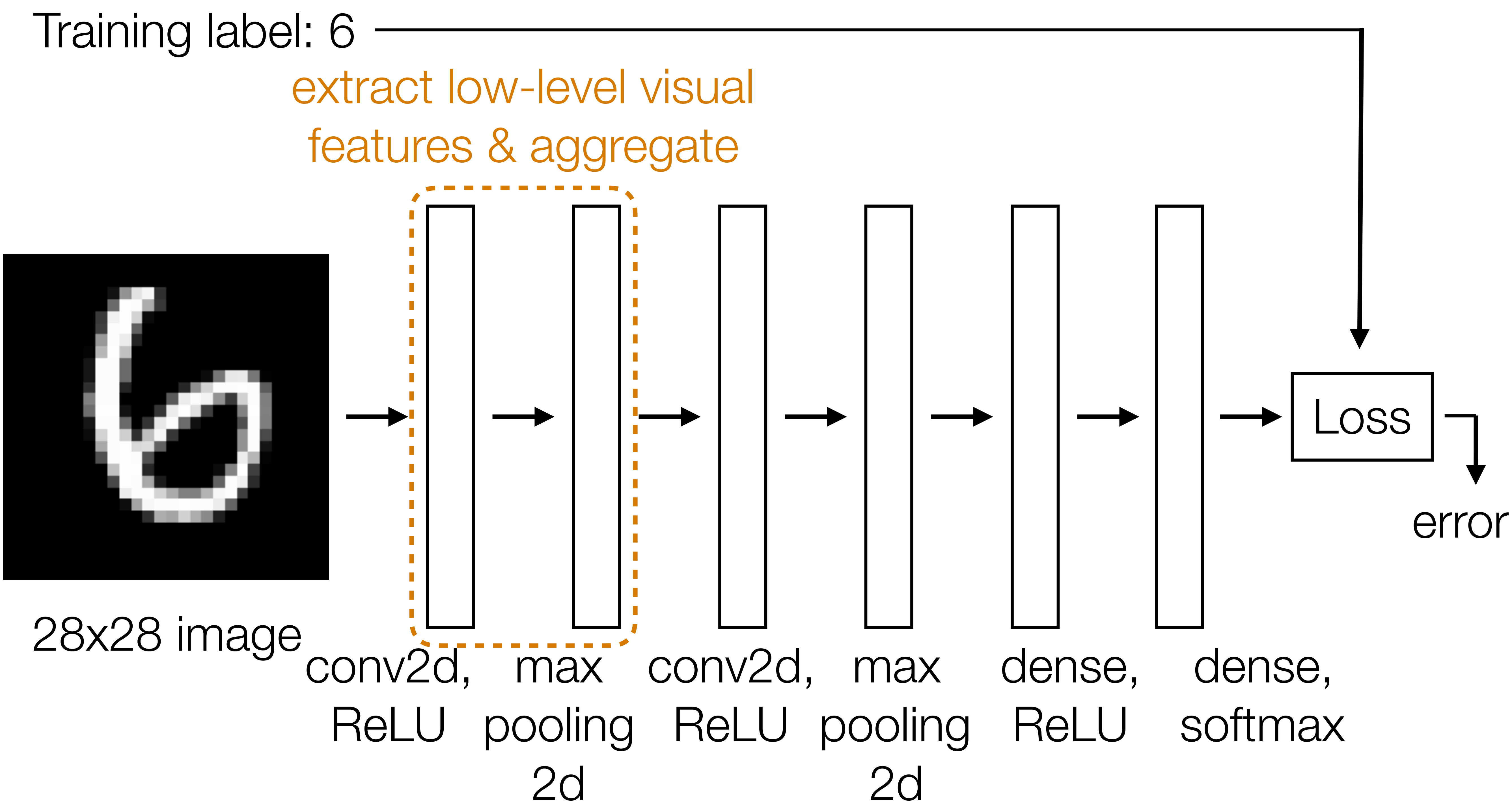
Handwritten Digit Recognition



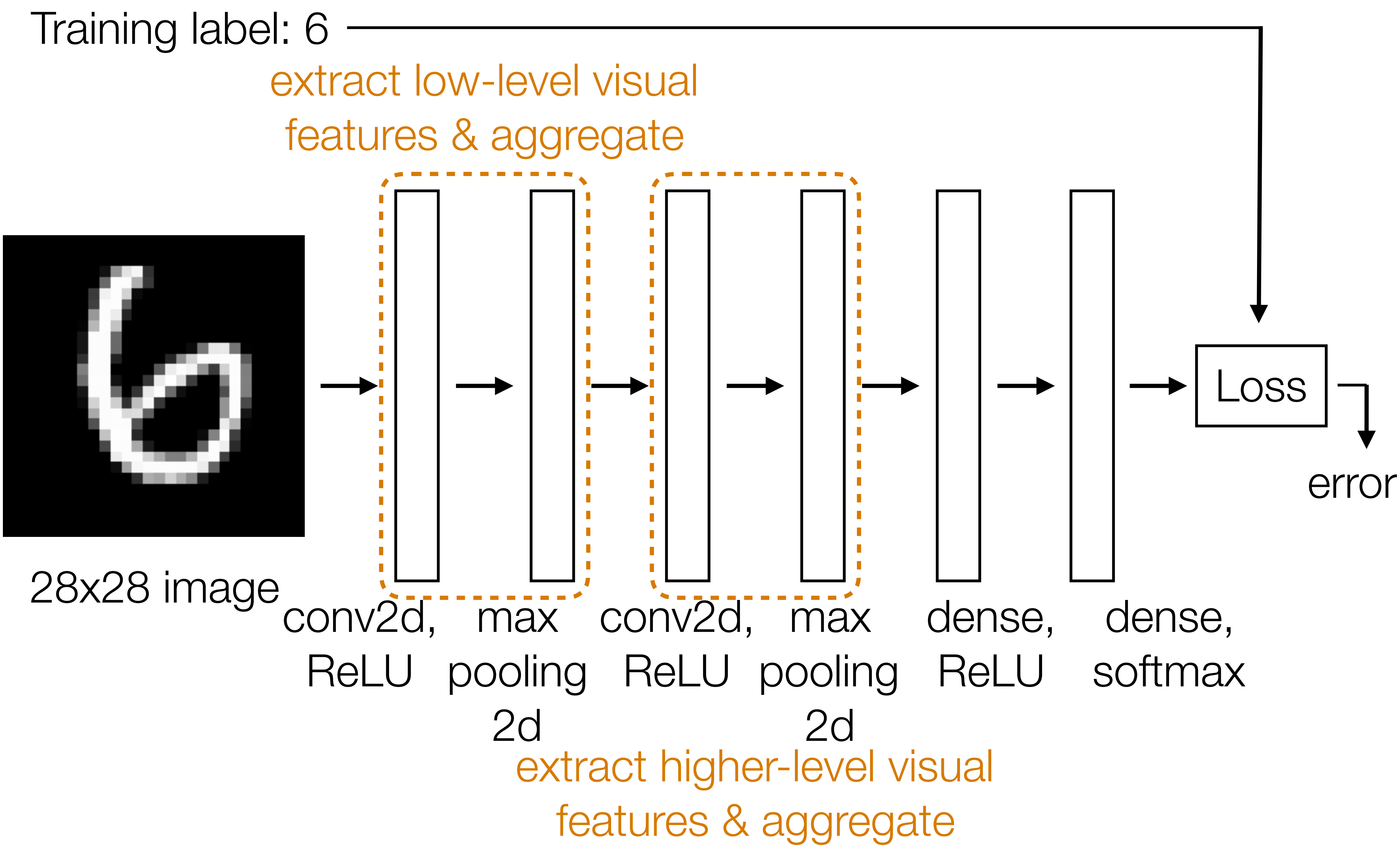
Handwritten Digit Recognition



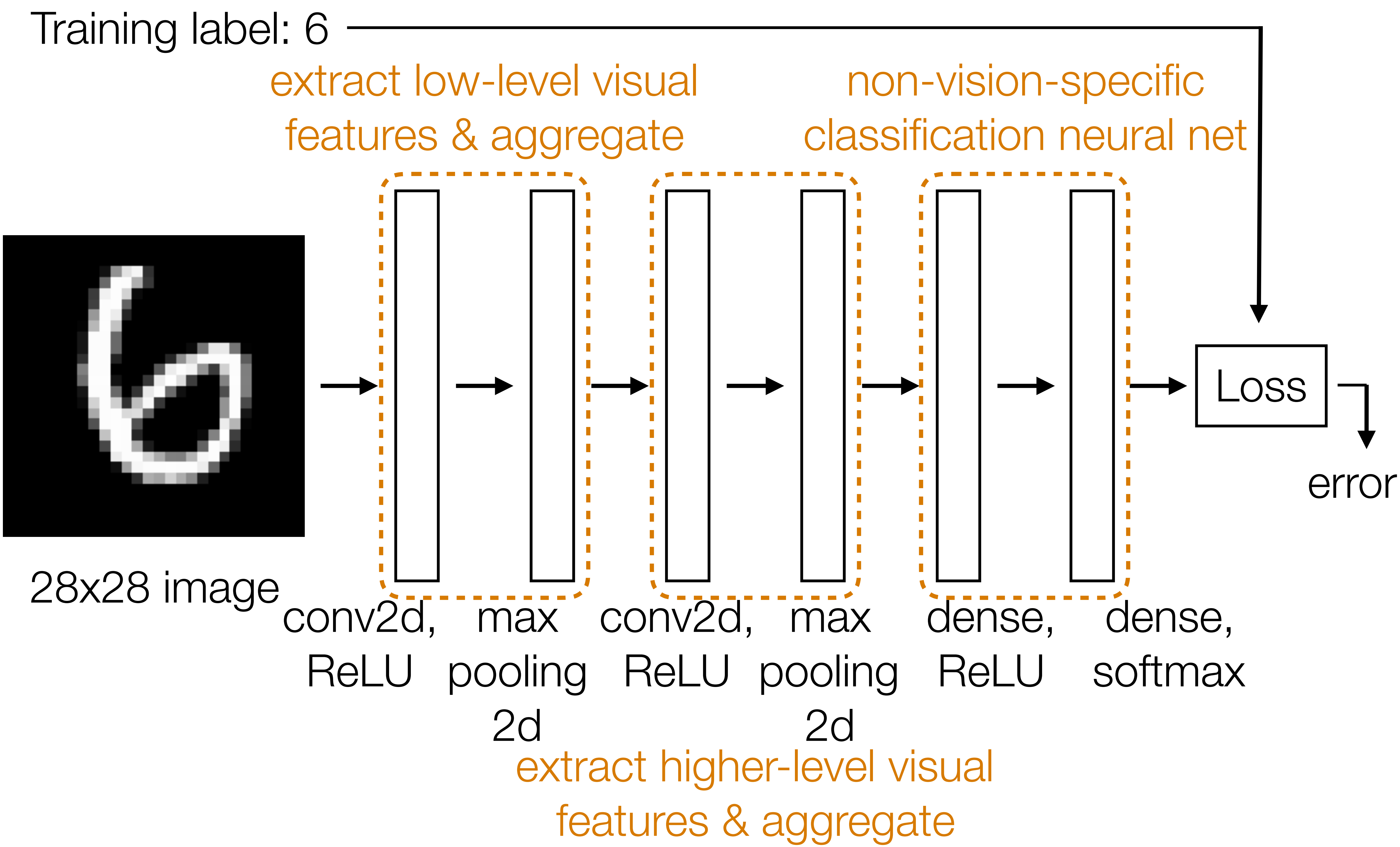
Handwritten Digit Recognition



Handwritten Digit Recognition



Handwritten Digit Recognition



CNN Demo

CNN’s

CNN's

- Learn convolution filters for extracting simple features

CNN’s

- Learn convolution filters for extracting simple features

- Max pooling aggregates local information

CNN's

- Learn convolution filters for extracting simple features
- Max pooling aggregates local information
- Can then repeat the above two layers to learn features from increasingly higher-level representations

CNN’s

- Learn convolution filters for extracting simple features

- Max pooling aggregates local information

- Can then repeat the above two layers to learn features from increasingly higher-level representations

- Convolution filters are shift-invariant

CNN’s

- Learn convolution filters for extracting simple features

- Max pooling aggregates local information

- Can then repeat the above two layers to learn features from increasingly higher-level representations

- Convolution filters are shift-invariant

- In terms of invariance to an object shifting within the input image, this is roughly achieved by pooling

Recurrent Neural Networks (RNNs)

RNNs

RNNs

What we’ve seen so far are “feedforward” NNs

RNNs

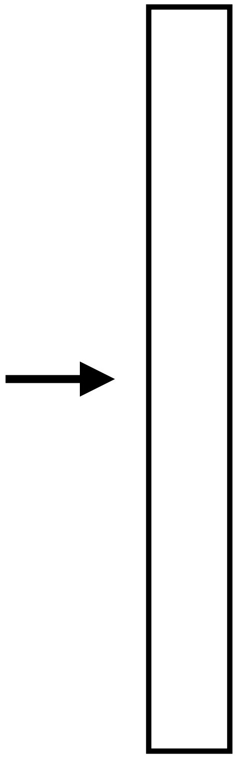
What we've seen so far are "feedforward" NNs

RNNs

What we've seen so far are "feedforward" NNs

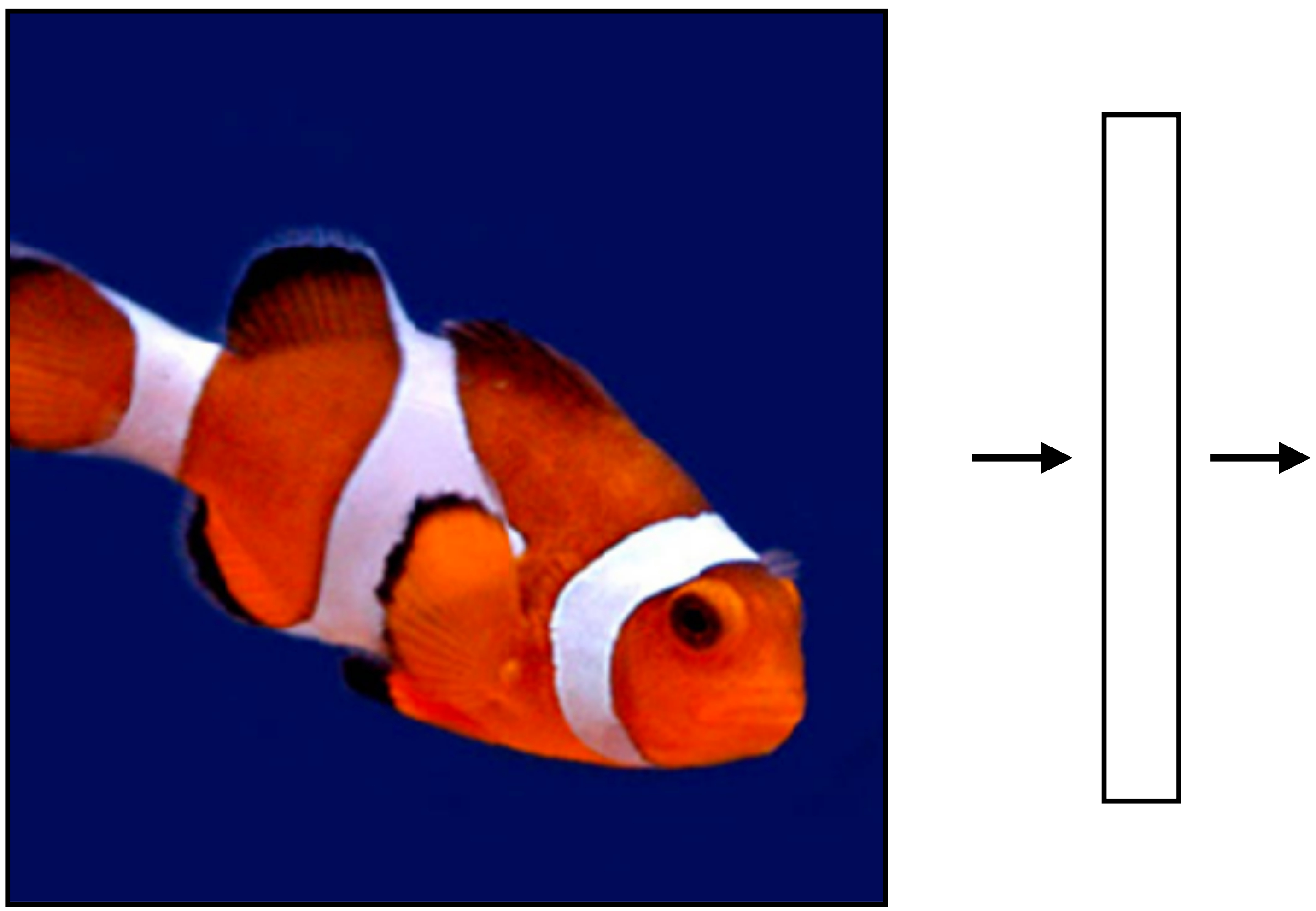
RNNs

What we've seen so far are "feedforward" NNs



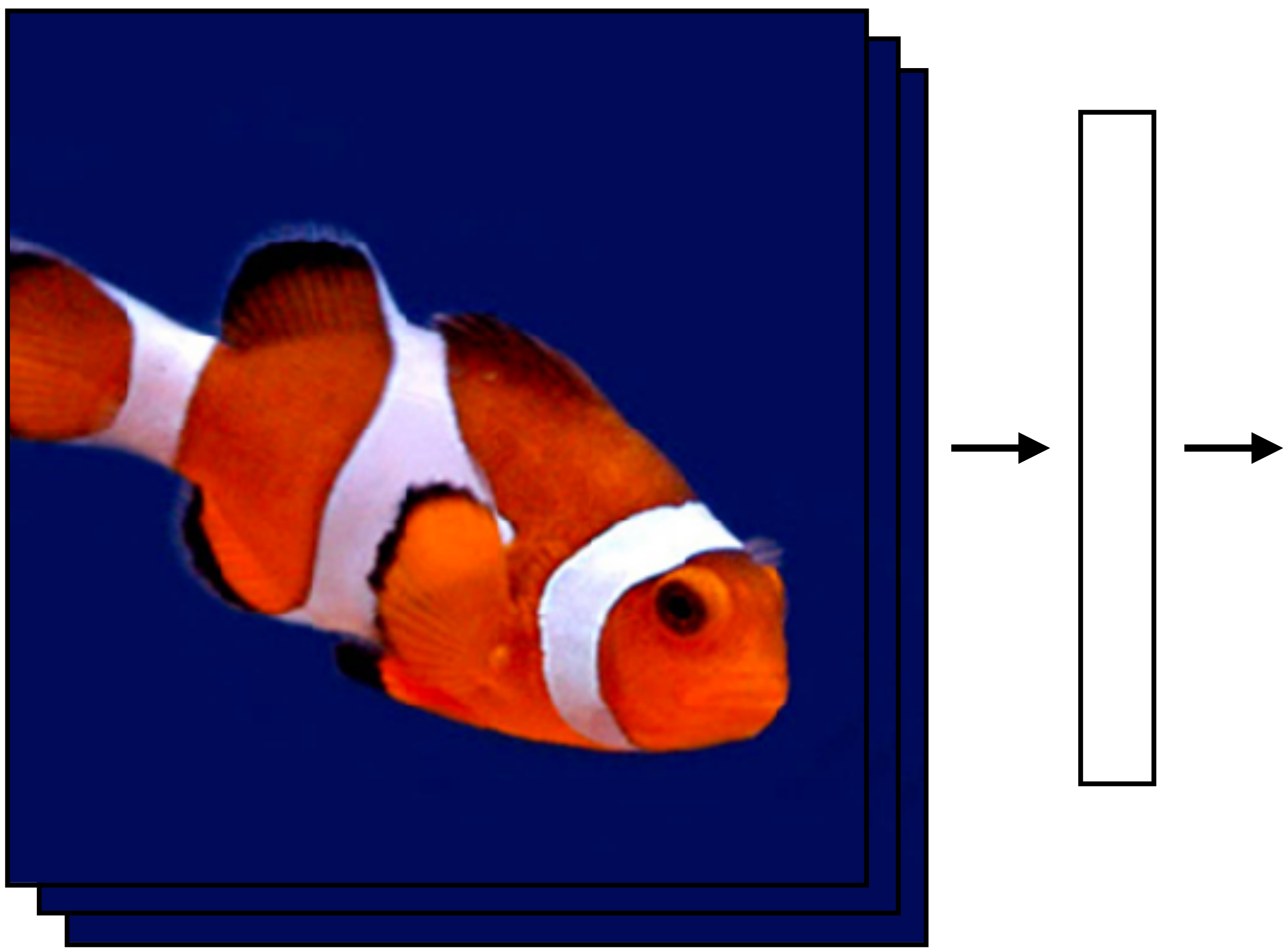
RNNs

What we've seen so far are "feedforward" NNs



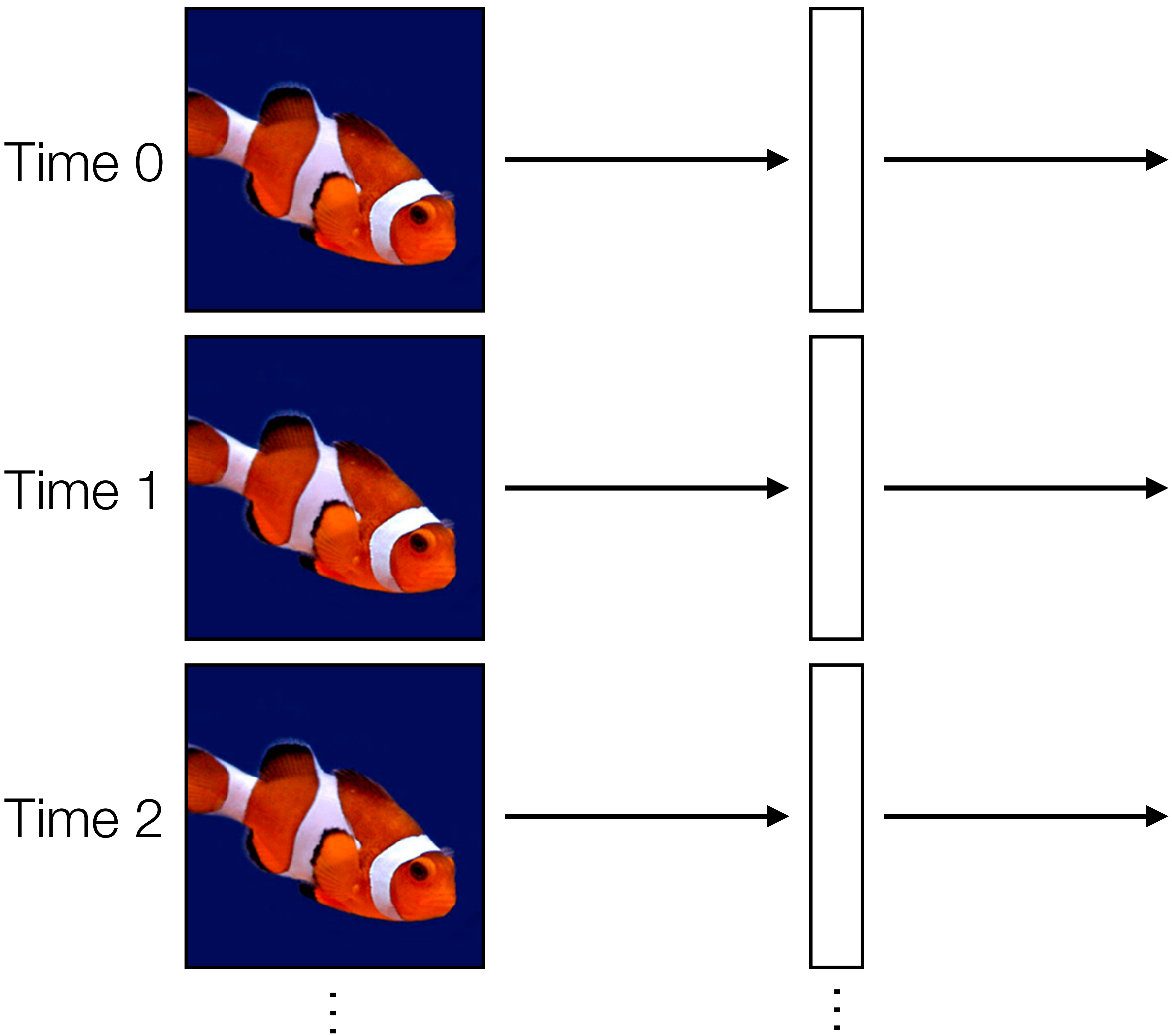
RNNs

What we've seen so far are "feedforward" NNs



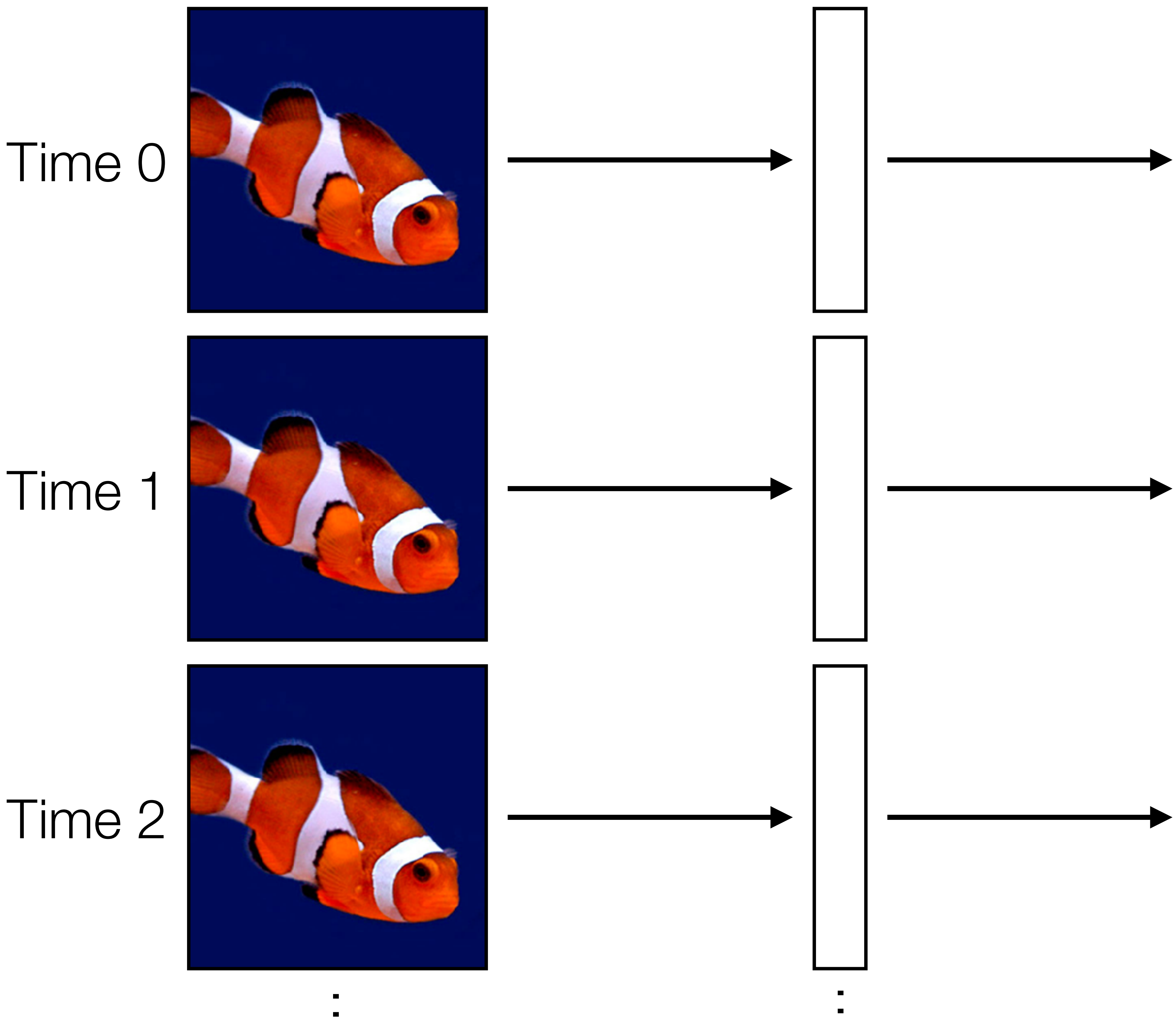
What if we had a video?

RNNs

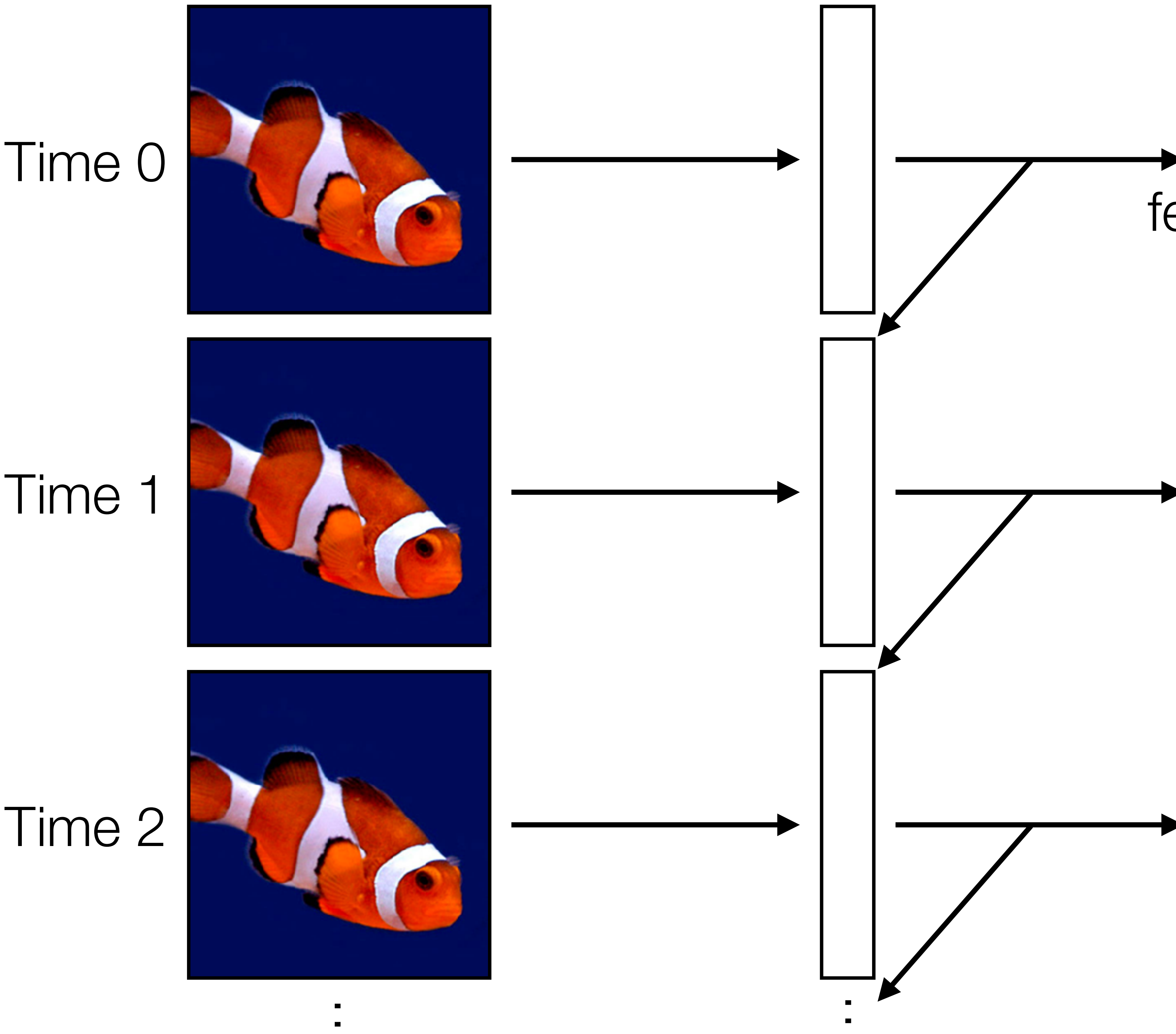


RNNs

Feedforward NN's:
treat each video frame
separately



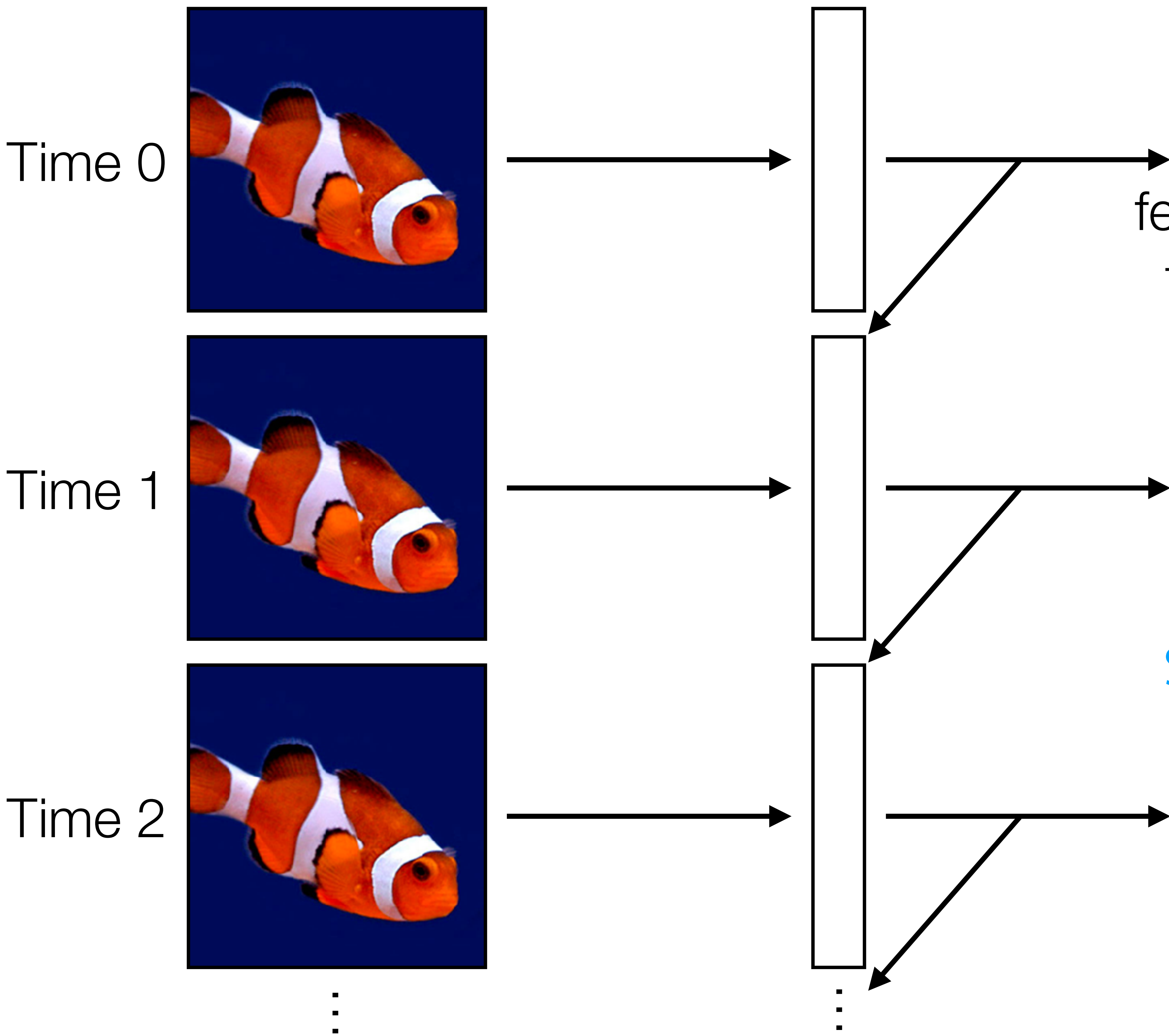
RNNs



Feedforward NN's:
treat each video frame separately

RNN's:
feed output at previous time step as input to RNN layer at current time step

RNNs

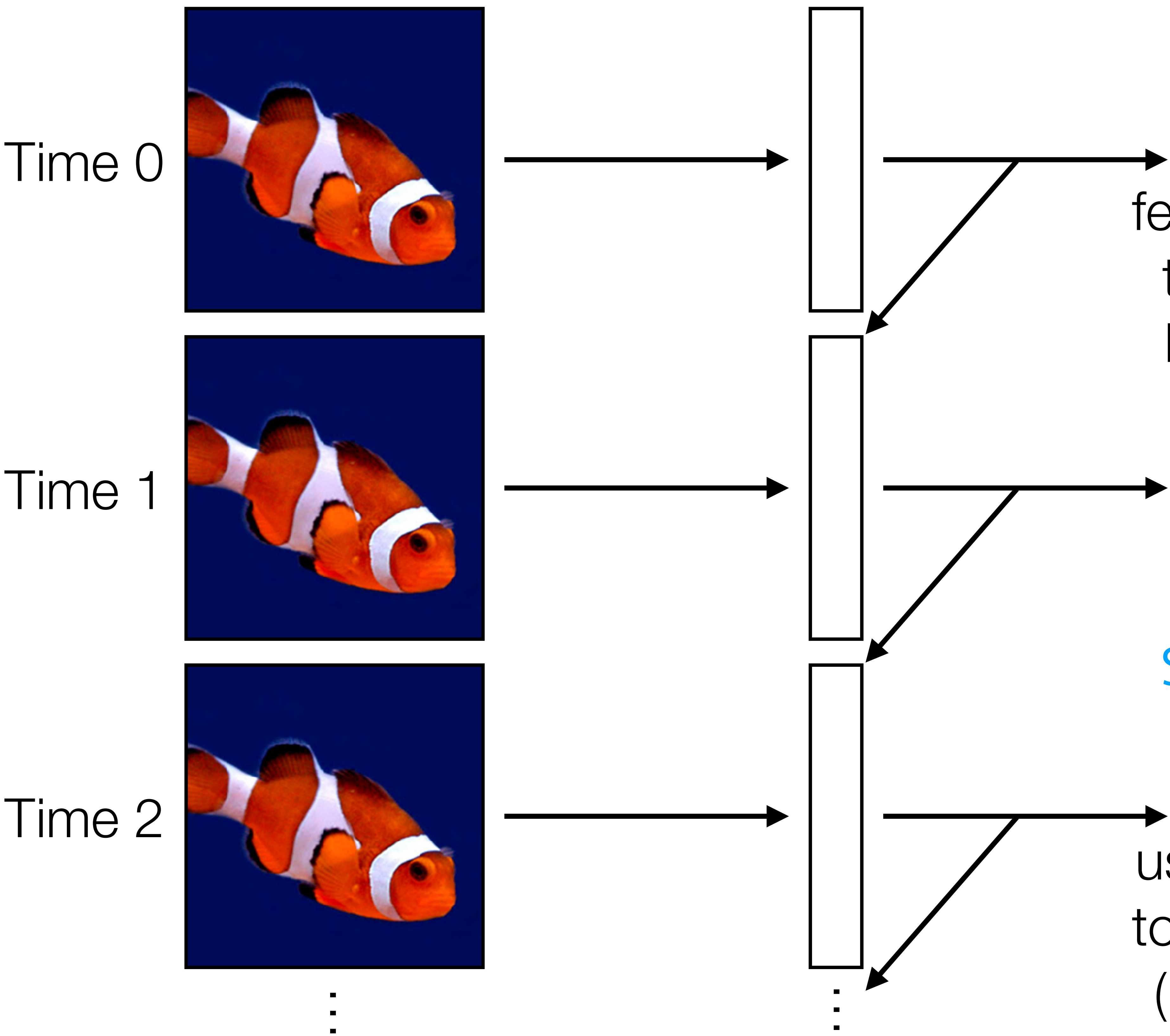


Feedforward NN's:
treat each video frame separately

RNN's:
feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options:
`SimpleRNN`, `LSTM`

RNNs



Feedforward NN's:
treat each video frame separately

RNN's:
feed output at previous time step as input to RNN layer at current time step

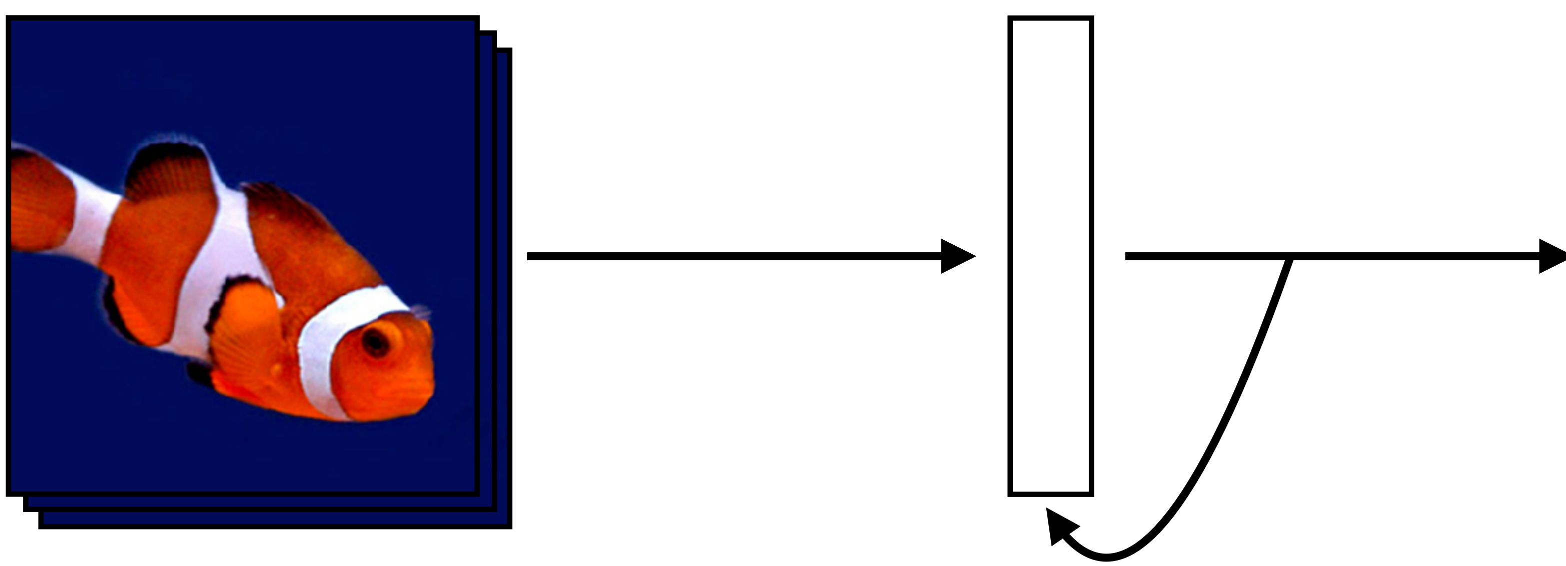
In `keras`, different RNN options:
`SimpleRNN, LSTM`

Recommendation:
use LSTMs if you want to have longer memory (long range structure)

RNNs

Feedforward NN's:
treat each video frame separately

RNN's:
feed output at previous time step as input to RNN layer at current time step

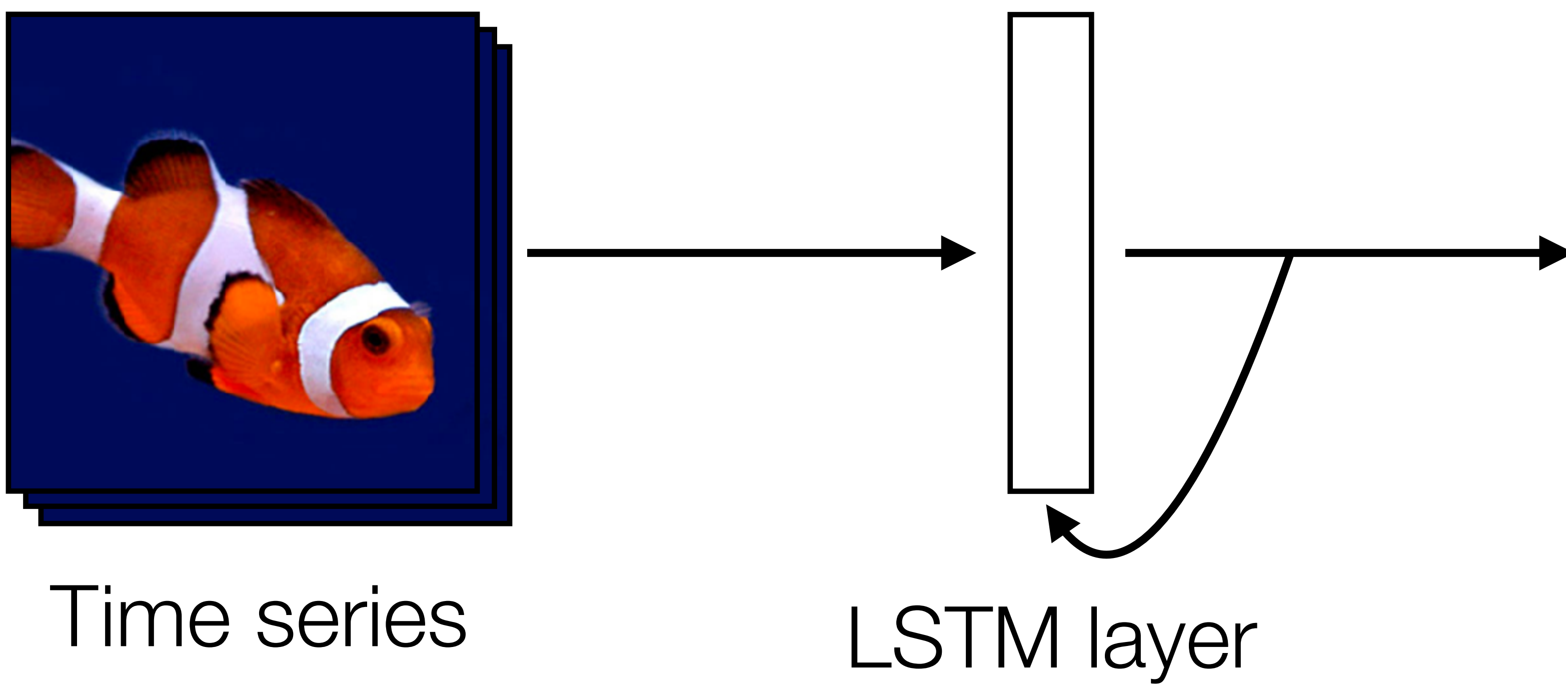


Time series

In `keras`, different RNN options:
`SimpleRNN, LSTM`

Recommendation:
use LSTMs if you want to have longer memory (long range structure)

RNNs



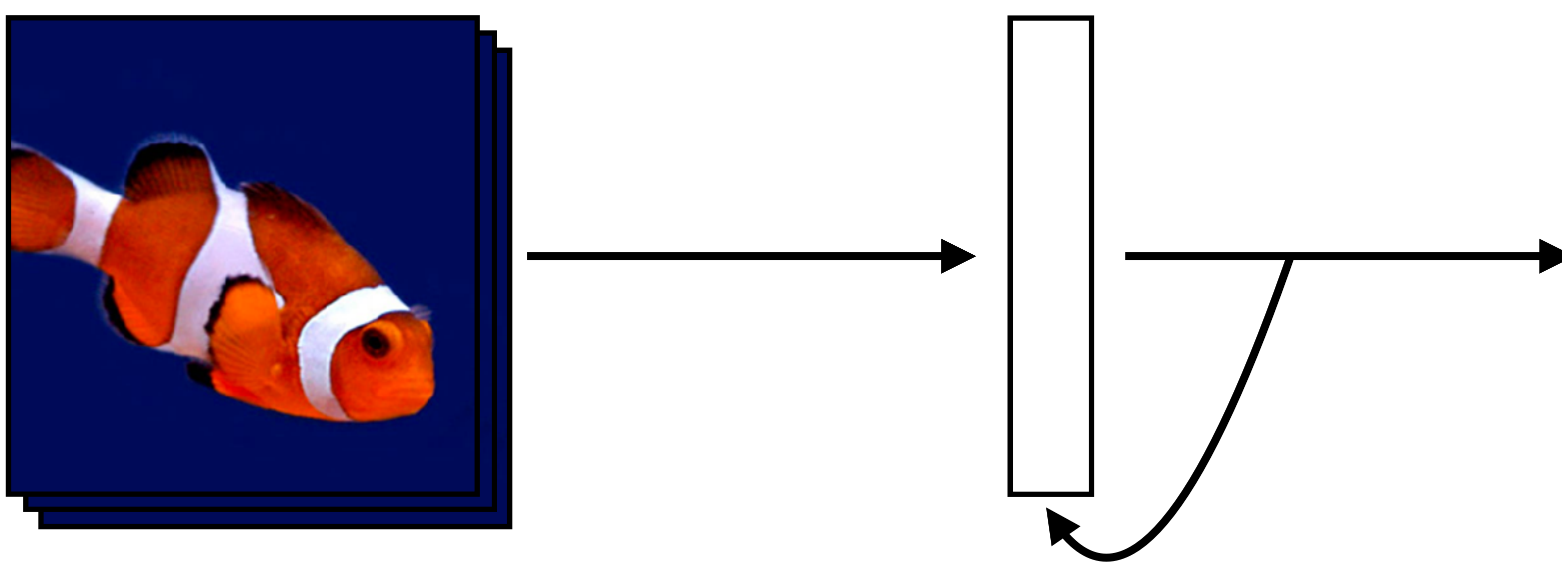
Feedforward NN's:
treat each video frame separately

RNN's:
feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options:
`SimpleRNN, LSTM`

Recommendation:
use LSTMs if you want to have longer memory (long range structure)

RNNs



Time series

LSTM layer

like a dense layer
that has memory

Feedforward NN's:
treat each video frame separately

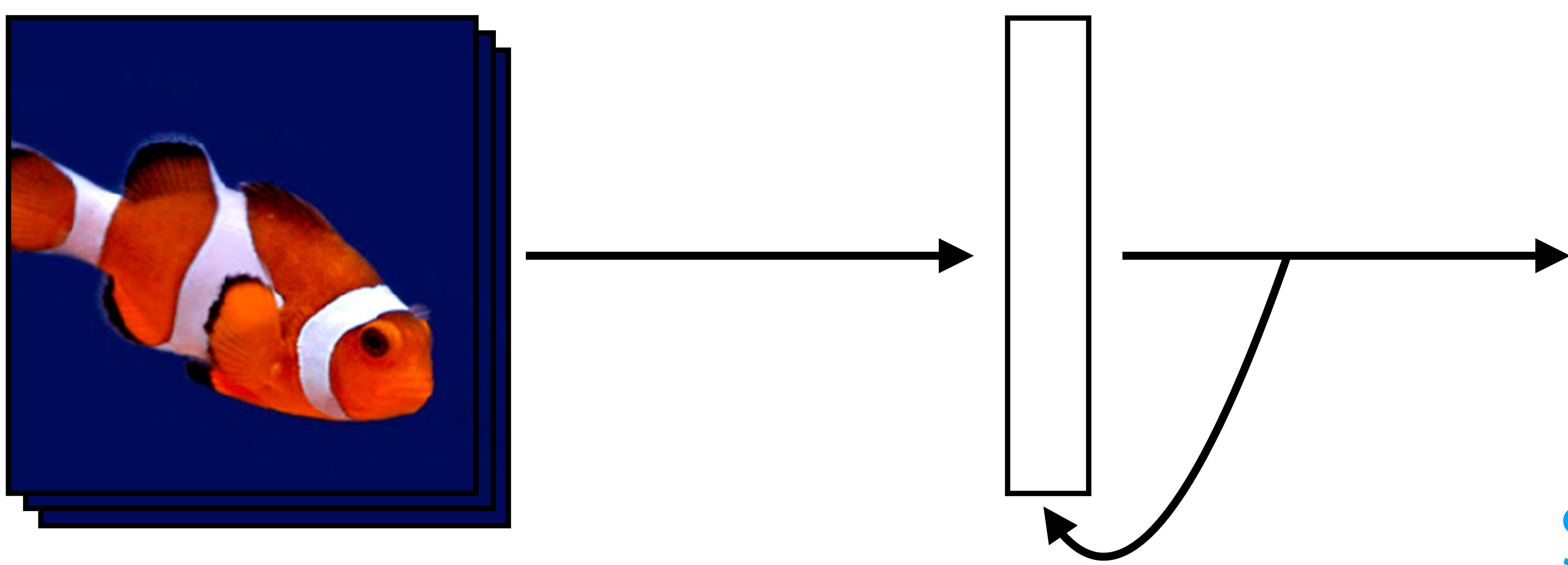
RNN's:
feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options:
`SimpleRNN`, `LSTM`

Recommendation:
use LSTMs if you want to have longer memory (long range structure)

RNNs

readily chains together with other neural net layers



Time series

LSTM layer

like a dense layer that has memory

Feedforward NN's: treat each video frame separately

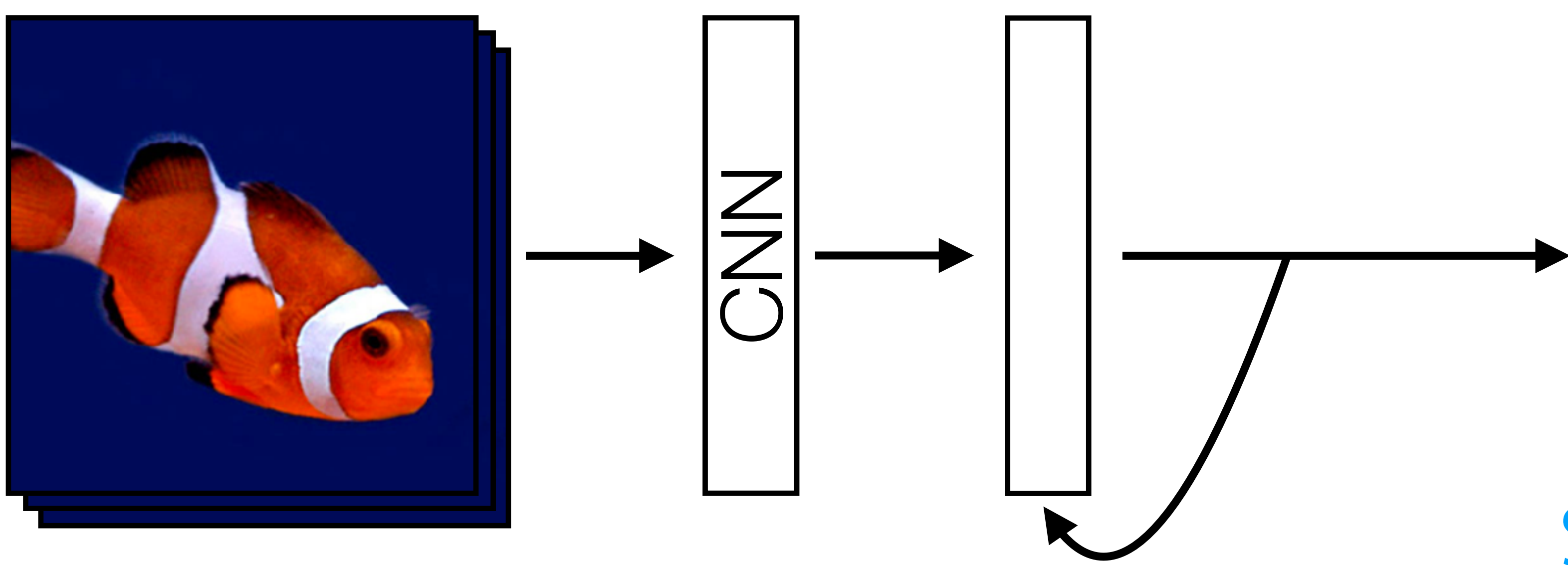
RNN's: feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options: `SimpleRNN, LSTM`

Recommendation: use LSTMs if you want to have longer memory (long range structure)

RNNs

readily chains together with
other neural net layers



Time series

LSTM layer

like a dense layer
that has memory

Feedforward NN's:
treat each video frame
separately

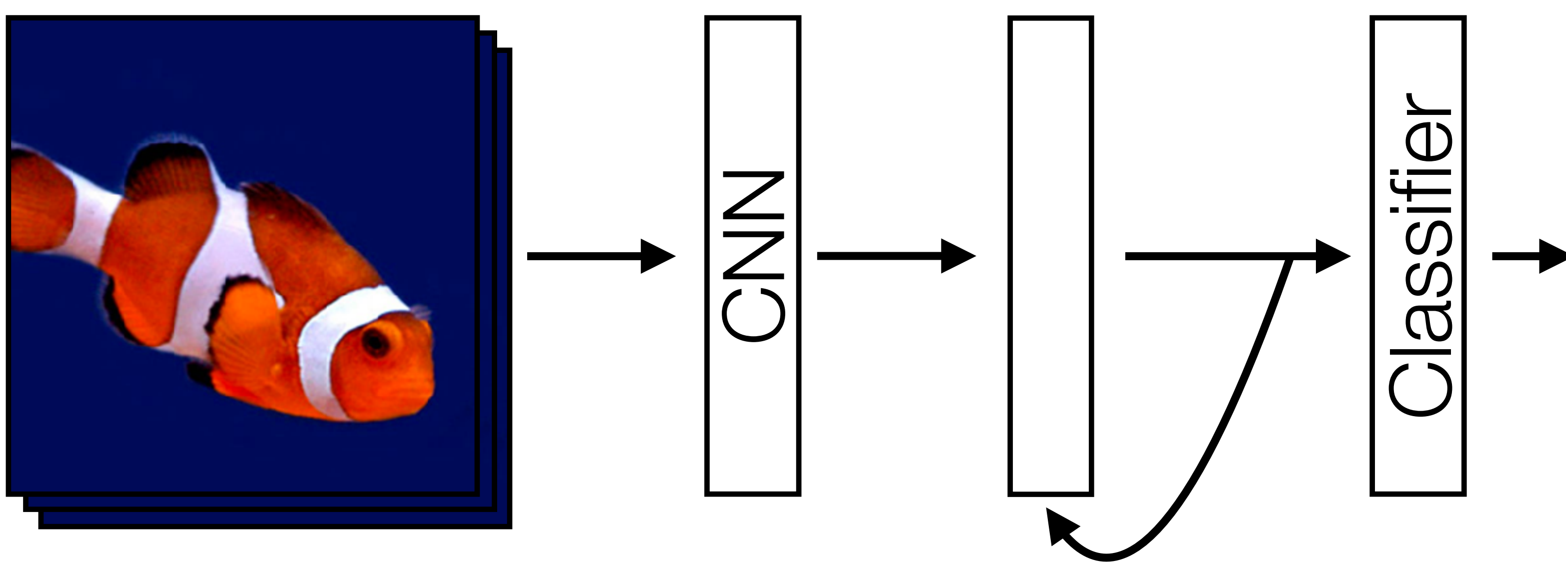
RNN's:
feed output at previous
time step as input to
RNN layer at current
time step

In `keras`, different
RNN options:
`SimpleRNN`, `LSTM`

Recommendation:
use LSTMs if you want
to have longer memory
(long range structure)

RNNs

readily chains together with other neural net layers



Time series

LSTM layer

like a dense layer that has memory

Feedforward NN's: treat each video frame separately

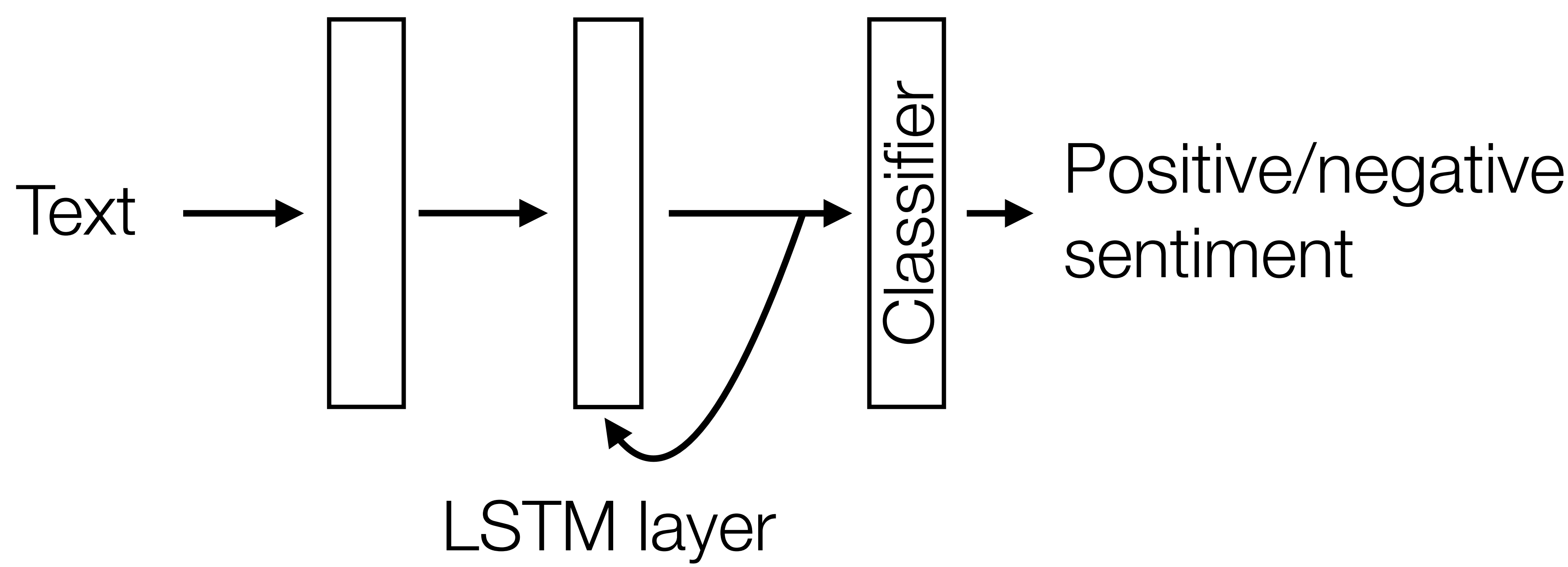
RNN's: feed output at previous time step as input to RNN layer at current time step

In `keras`, different RNN options: `SimpleRNN`, `LSTM`

Recommendation: use LSTMs if you want to have longer memory (long range structure)

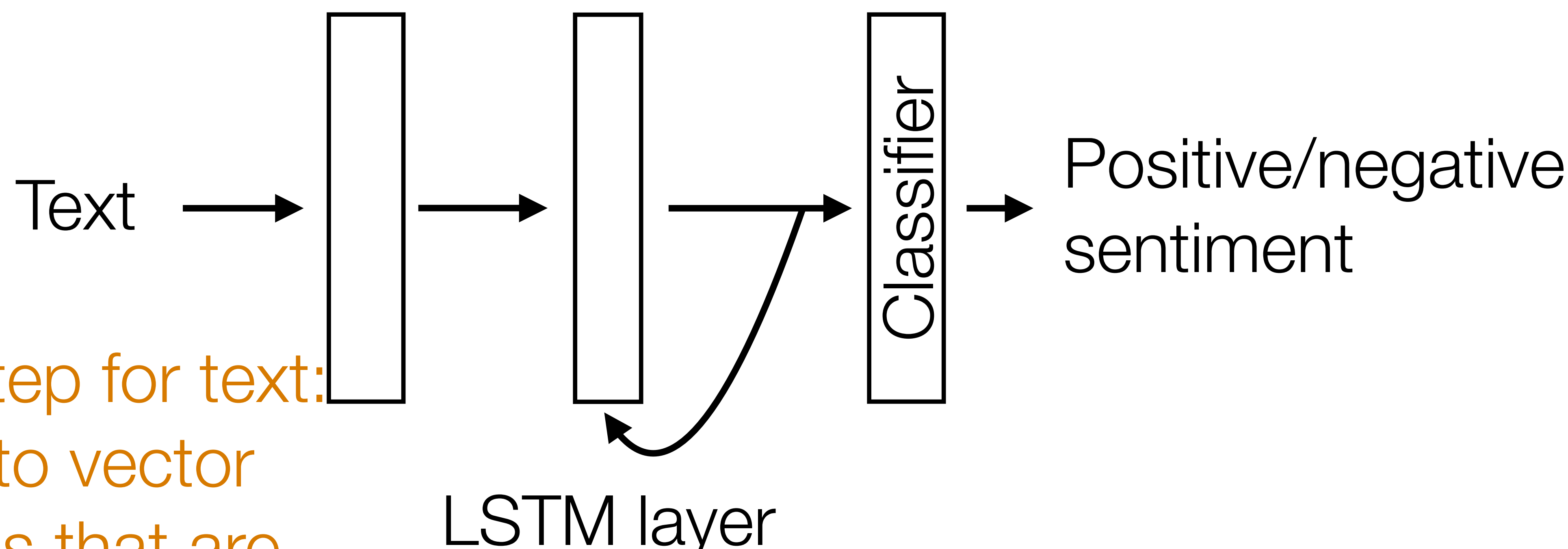
RNNs

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)



RNNs

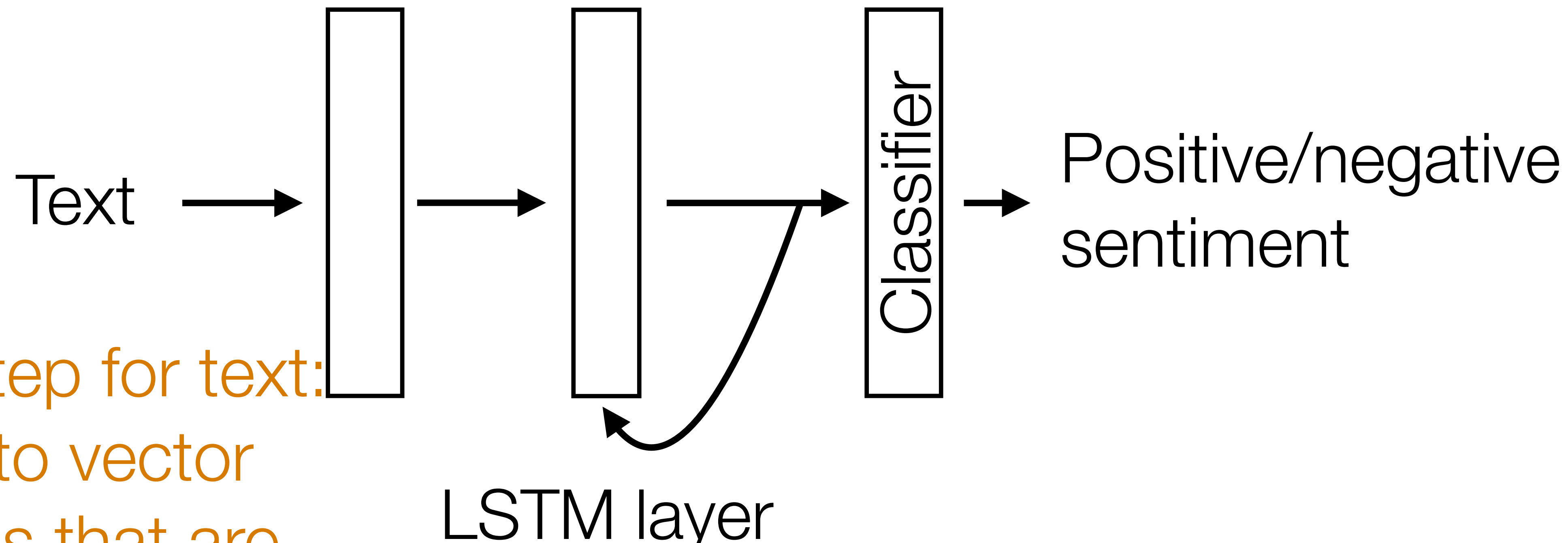
Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)



Common first step for text: turn words into vector representations that are semantically meaningful

RNNs

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)

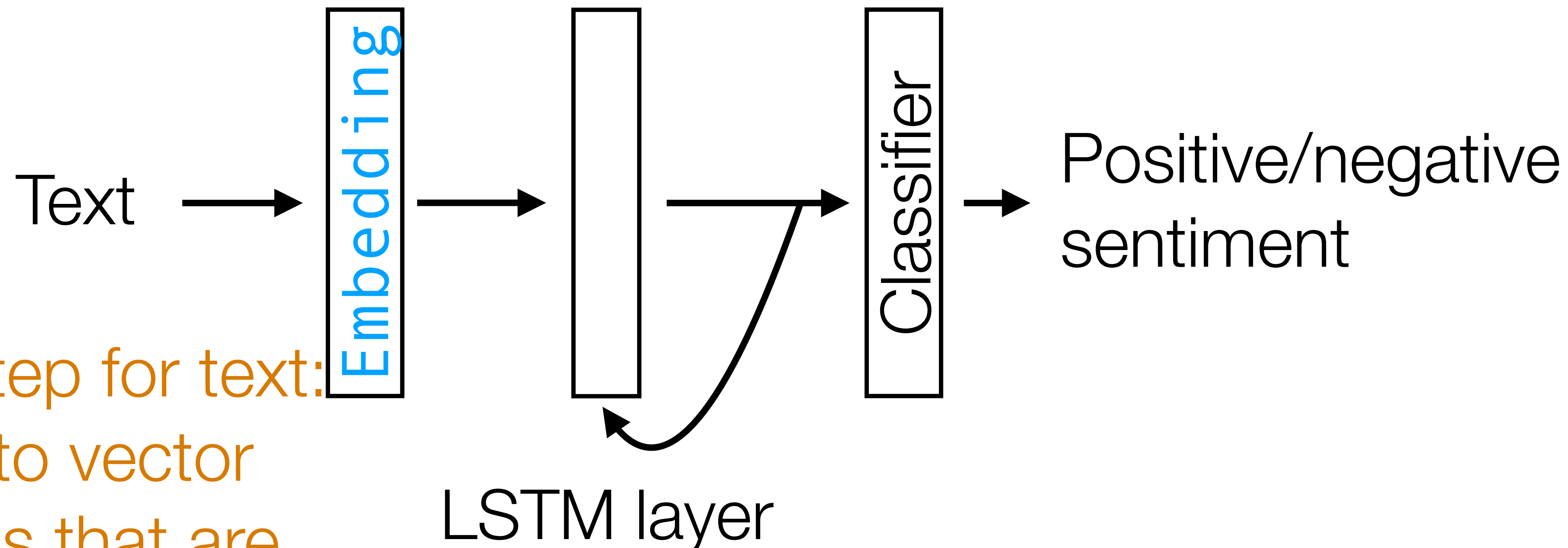


Common first step for text: turn words into vector representations that are semantically meaningful

In `keras`, use `Embedding` layer

RNNs

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)

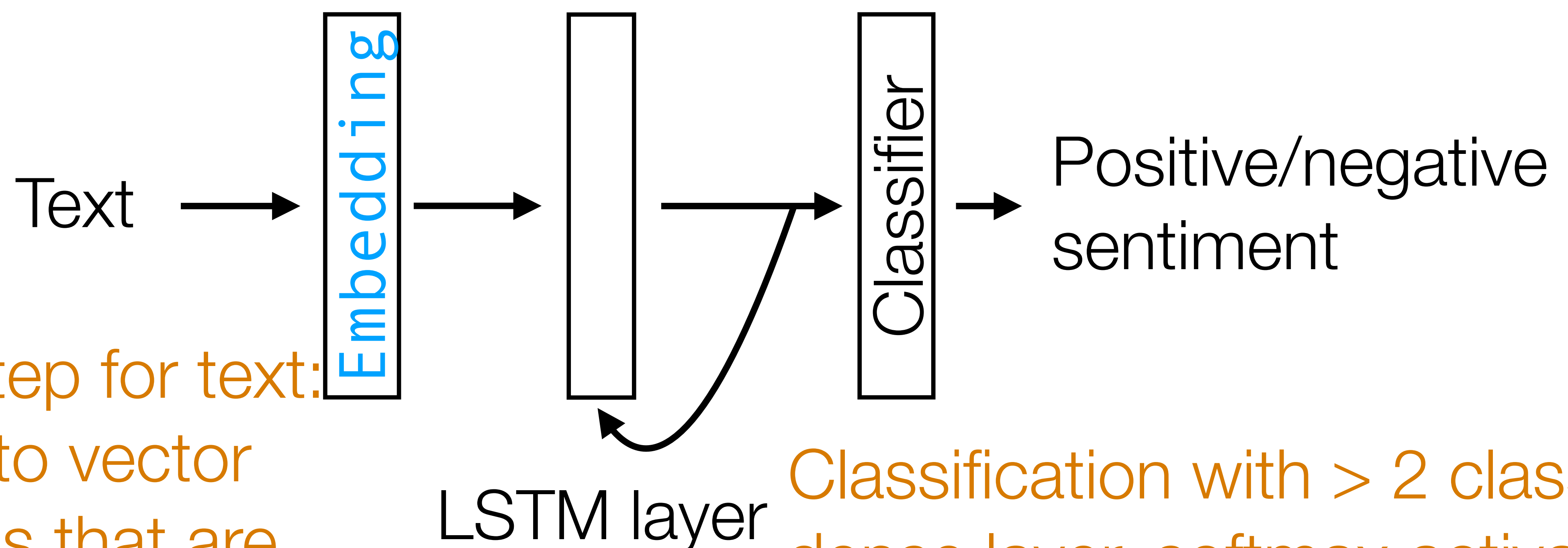


Common first step for text: turn words into vector representations that are semantically meaningful

In `keras`, use `Embedding` layer

RNNs

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)



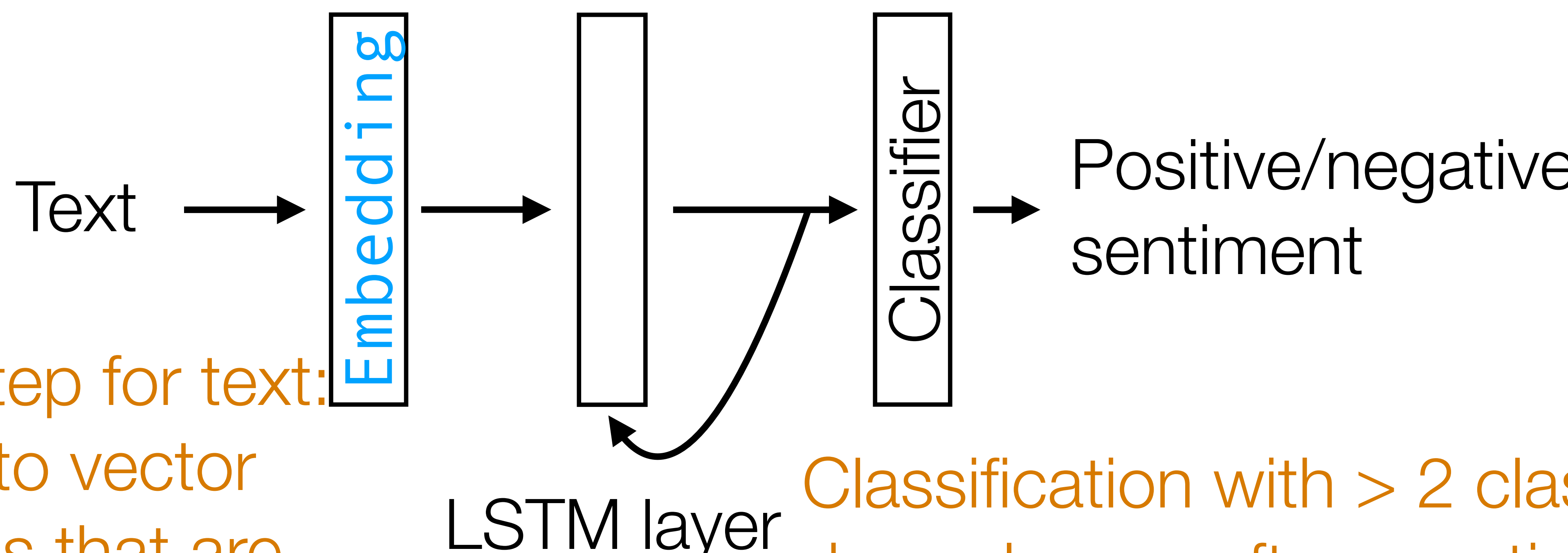
Common first step for text: turn words into vector representations that are semantically meaningful

Classification with > 2 classes: dense layer, softmax activation

In `keras`, use `Embedding` layer

RNNs

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)



Common first step for text: turn words into vector representations that are semantically meaningful

In `keras`, use `Embedding` layer

Classification with > 2 classes: dense layer, softmax activation

Classification with 2 classes: dense layer with 1 neuron, sigmoid activation

RNNs

Demo

RNNs

RNNs

- Neatly handles time series in which there is some sort of global structure, so memory helps

RNNs

- Neatly handles time series in which there is some sort of global structure, so memory helps
  - If time series doesn't actually have global structure, performance gain from using RNNs could be little compared to using 1D CNNs

RNNs

- Neatly handles time series in which there is some sort of global structure, so memory helps
    - If time series doesn't actually have global structure, performance gain from using RNNs could be little compared to using 1D CNNs
- An RNN layer should be chained together with other layers that learn a semantically meaningful interpretation from data (e.g., CNNs for images, word embeddings like word2vec/GloVe for text)