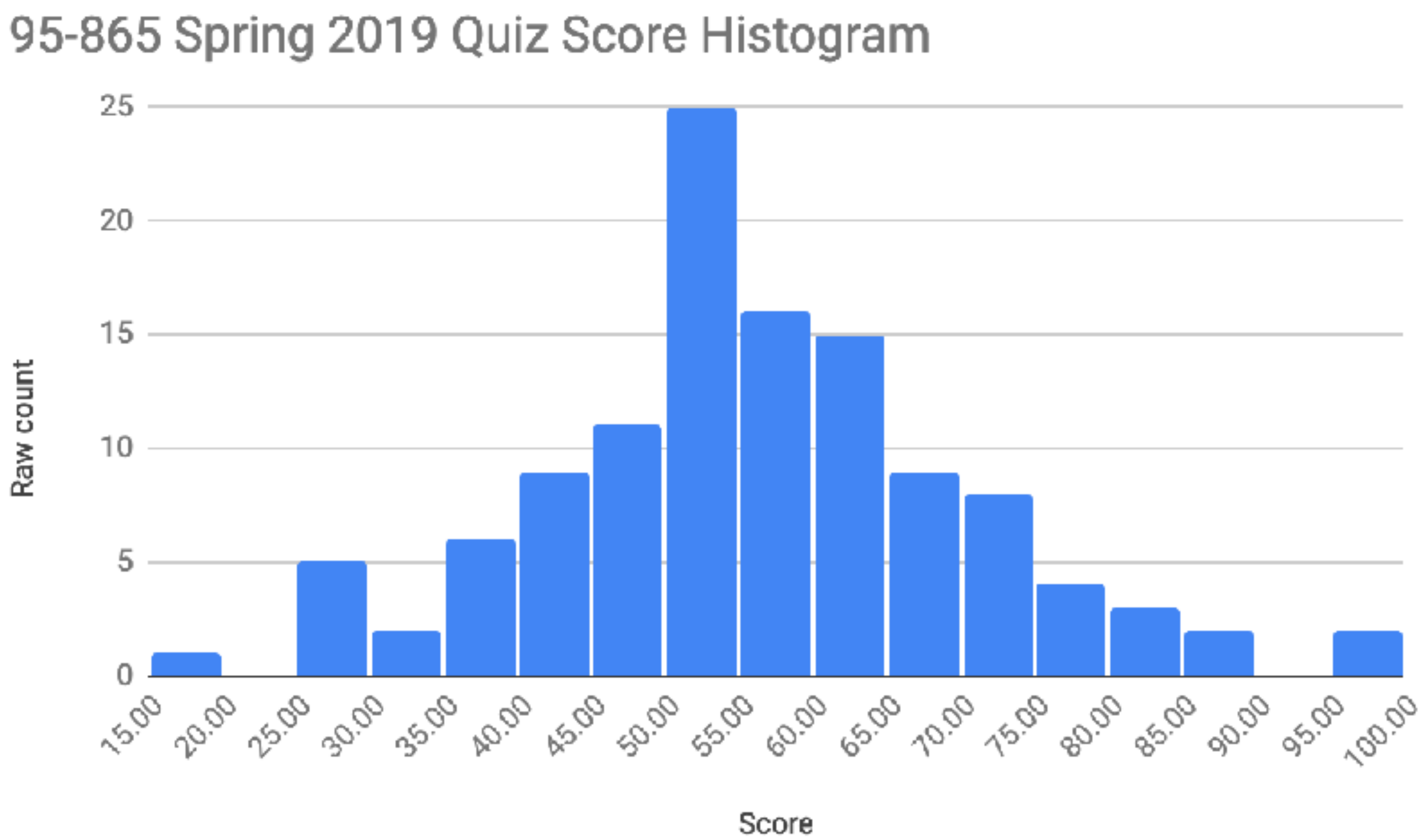
# 94-775/95-865 Lecture 11: Image Analysis With Convolutional Neural Nets

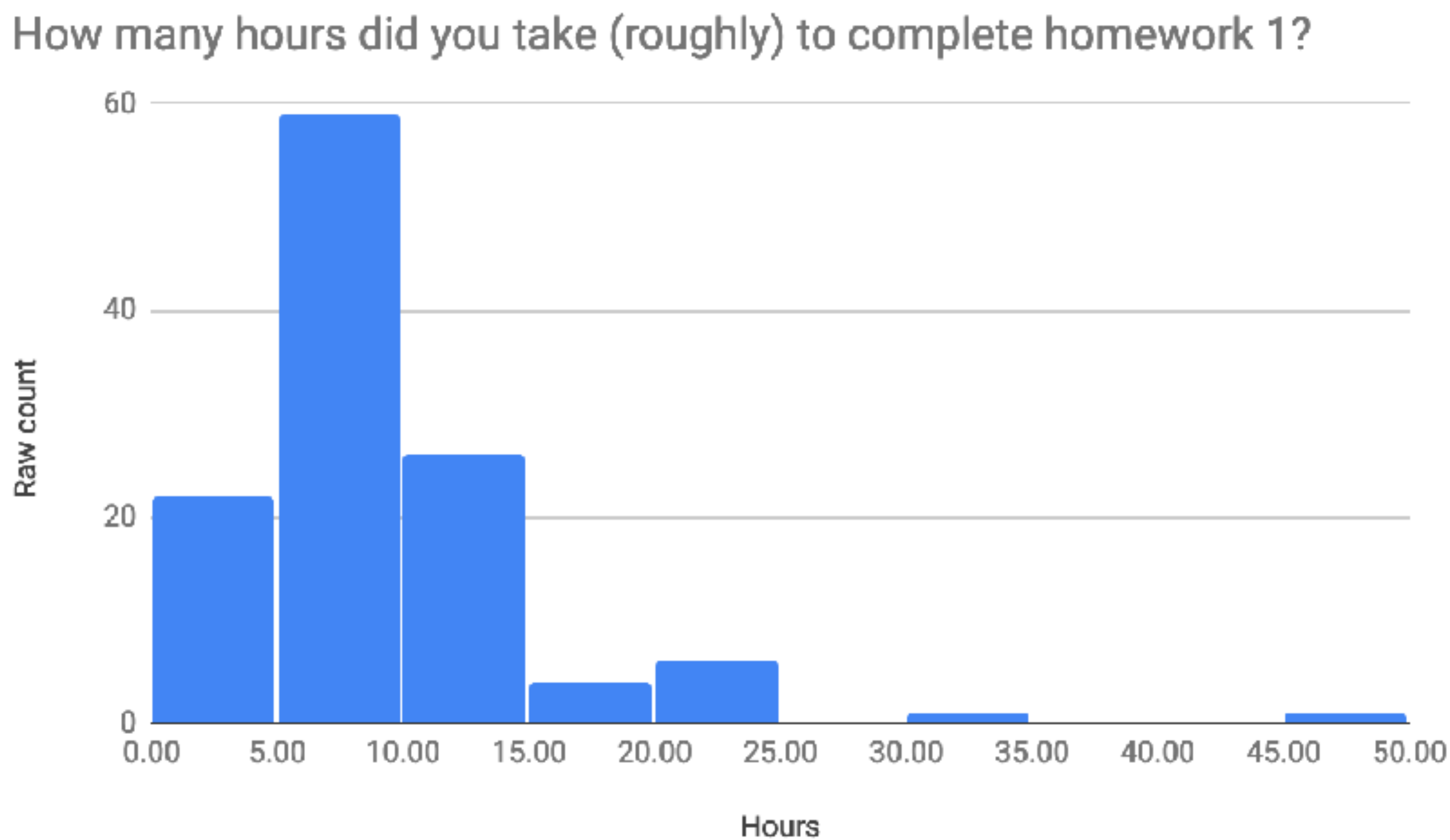
George Chen

# Quiz Results

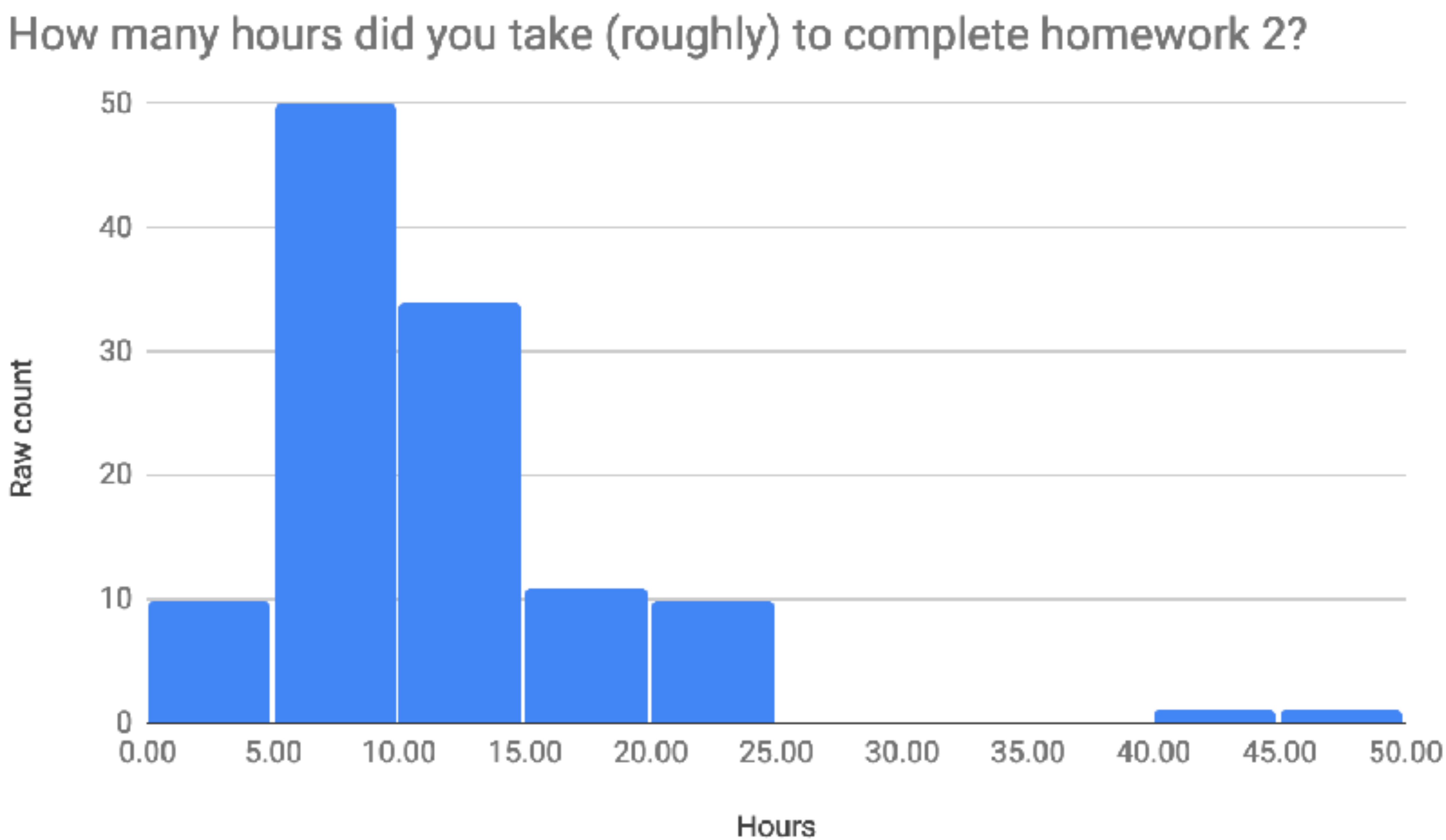



Mean 56.1, std dev 14.2, max 95.5

# Questionnaire Results



# Questionnaire Results



# Questionnaire Results

- Some people want to see more demos

- Some people want to see more math

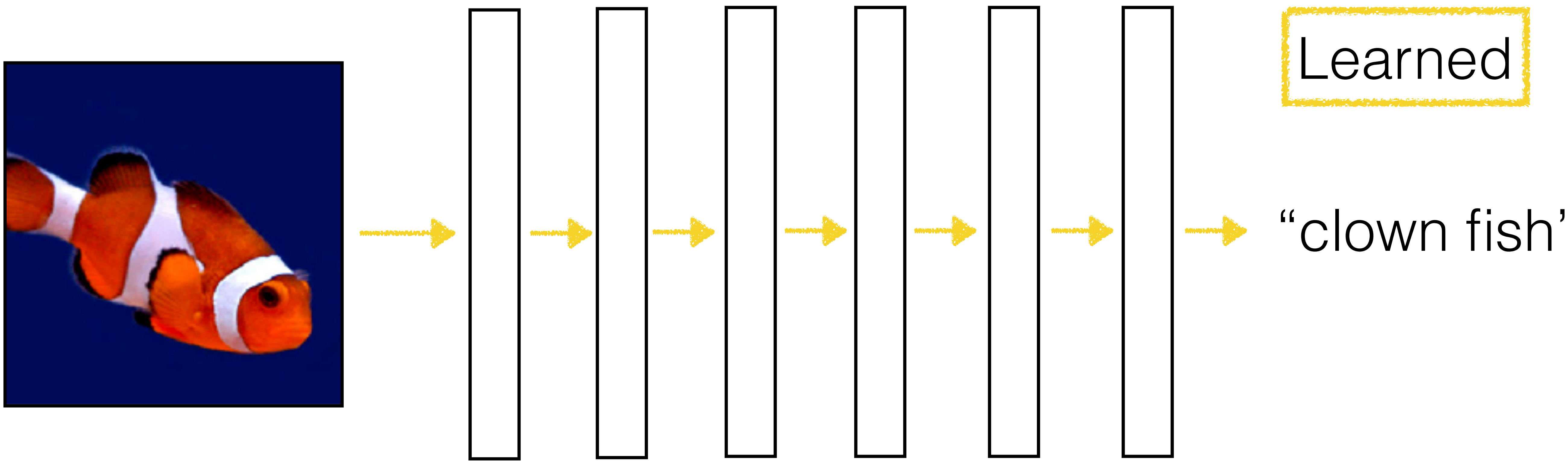
- Some people want to see more algorithms

A mini is quite short—can’t have more of everything…

# Announcements

- Start HW3 (takes something like **50%** longer than HW2)
- Yes, AWS takes a while to get used to
- Quiz regrades: due Monday 11:59pm
- Some past final exams have been posted (an additional one in recitation this Friday)

# Deep Learning




- Inspired by biological neural nets *but otherwise not the same at all* (biological neural nets do *not* work like deep nets)
- Learns a layered representation
    - Tries to get rid of manual feature engineering
    - Need to design constraints for what features are learned to account for structure in data (e.g., images, text, …)

# Learning a neural net amounts to curve fitting

We’re just estimating a function

# Neural Net as Function Approximation

Given `input`, learn a computer program that computes `output`

this is a **function**

Single-layer neural net example:

```
def f(input):
    output = softmax(np.dot(input, W) + b)
    return output
```

the only things that we are learning
(we fix their dimensions in advance)

We are fixing what the function `f` looks like in code
and are only adjusting `W` and `b`!!!

# Neural Net as Function Approximation

Given `input`, learn a computer program that computes `output`

Single-layer neural net example:

```
output = softmax(np.dot(input, W) + b)
```

Two-layer neural net example:

```
layer1_output = relu(np.dot(input, W1) + b1)

output = softmax(np.dot(layer1_output, W2) + b2)
```

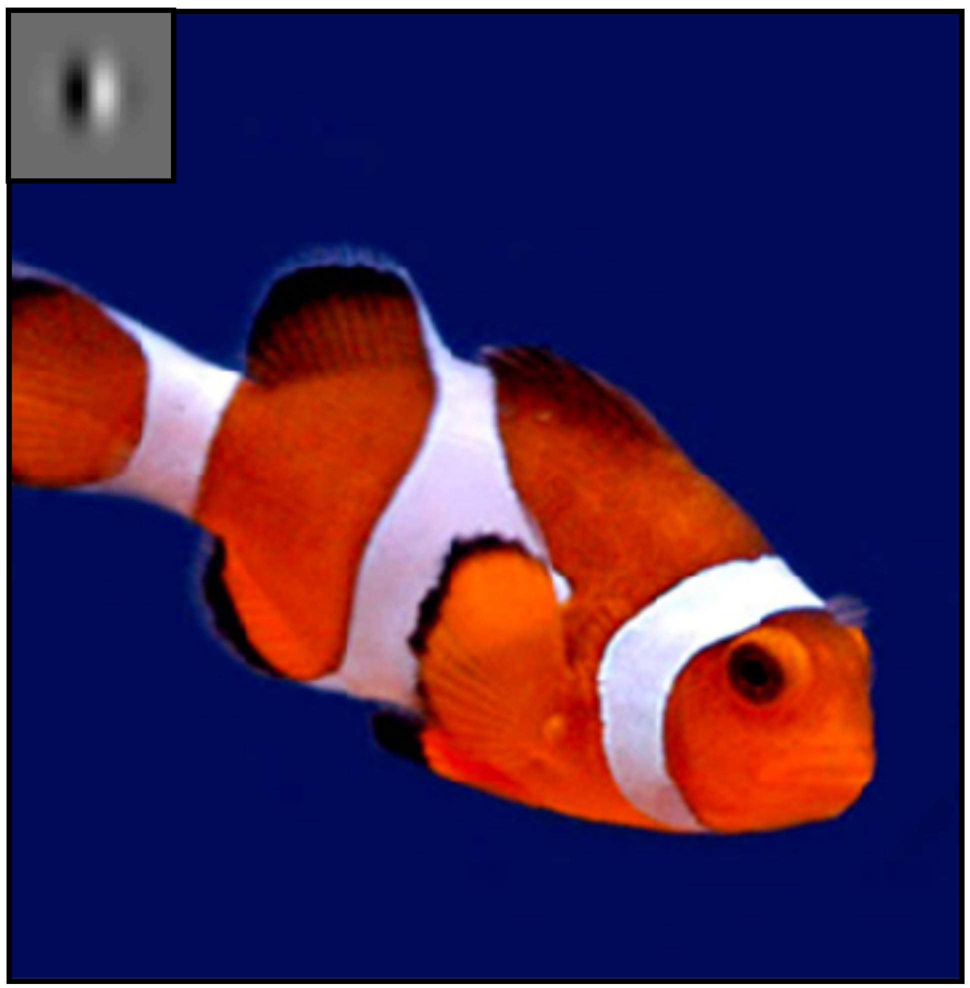
Learning a neural net: learning a simple computer program that maps inputs (raw feature vectors) to outputs (predictions)

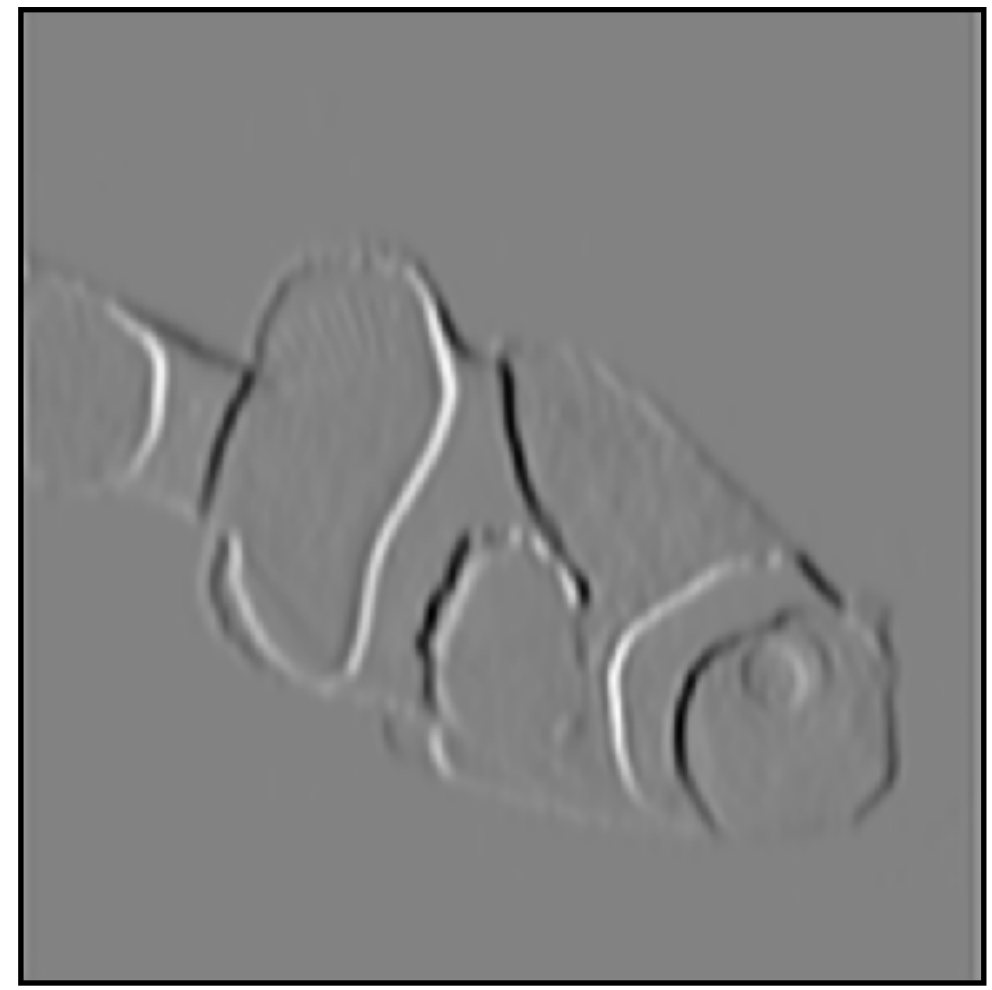
# Architecting Neural Nets

- Increasing number of layers (depth) makes neural net more complex
    - Can approximate more functions
    - More parameters needed
        - More training data may be needed
- Designing neural net architectures is a bit of an art
    - How to select the number of neurons for intermediate layers?
    - Very common in practice: modify existing architectures that are known to work well (e.g., VGG-16 for computer vision/image processing)

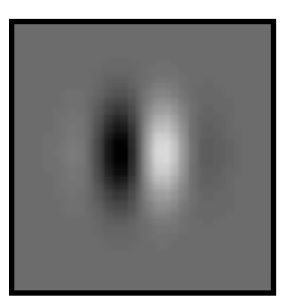
# Image analysis with Convolutional Neural Nets (CNNs, also called convnets)

# 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

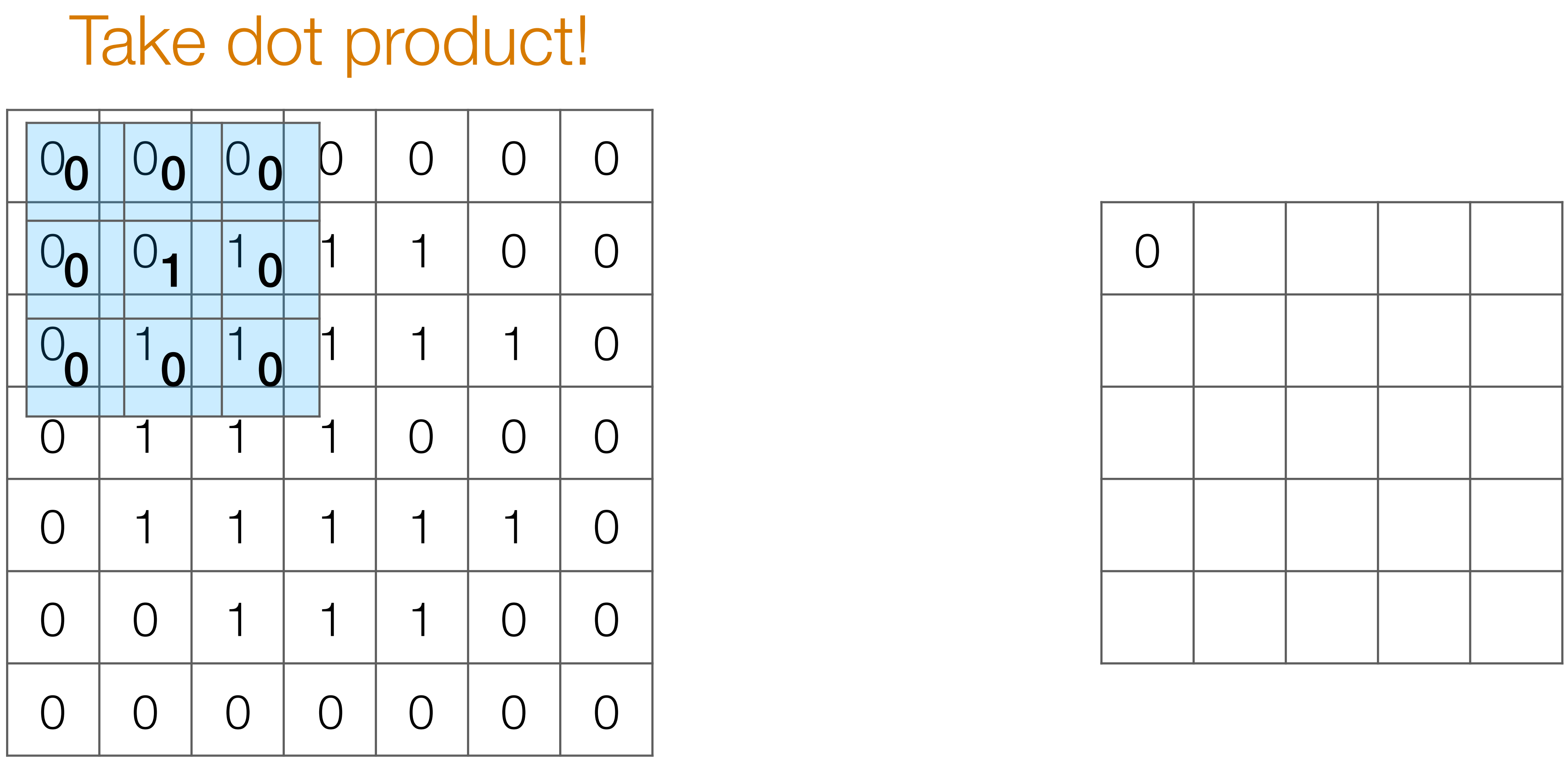
| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Input image

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

Filter
(also called “kernel”)

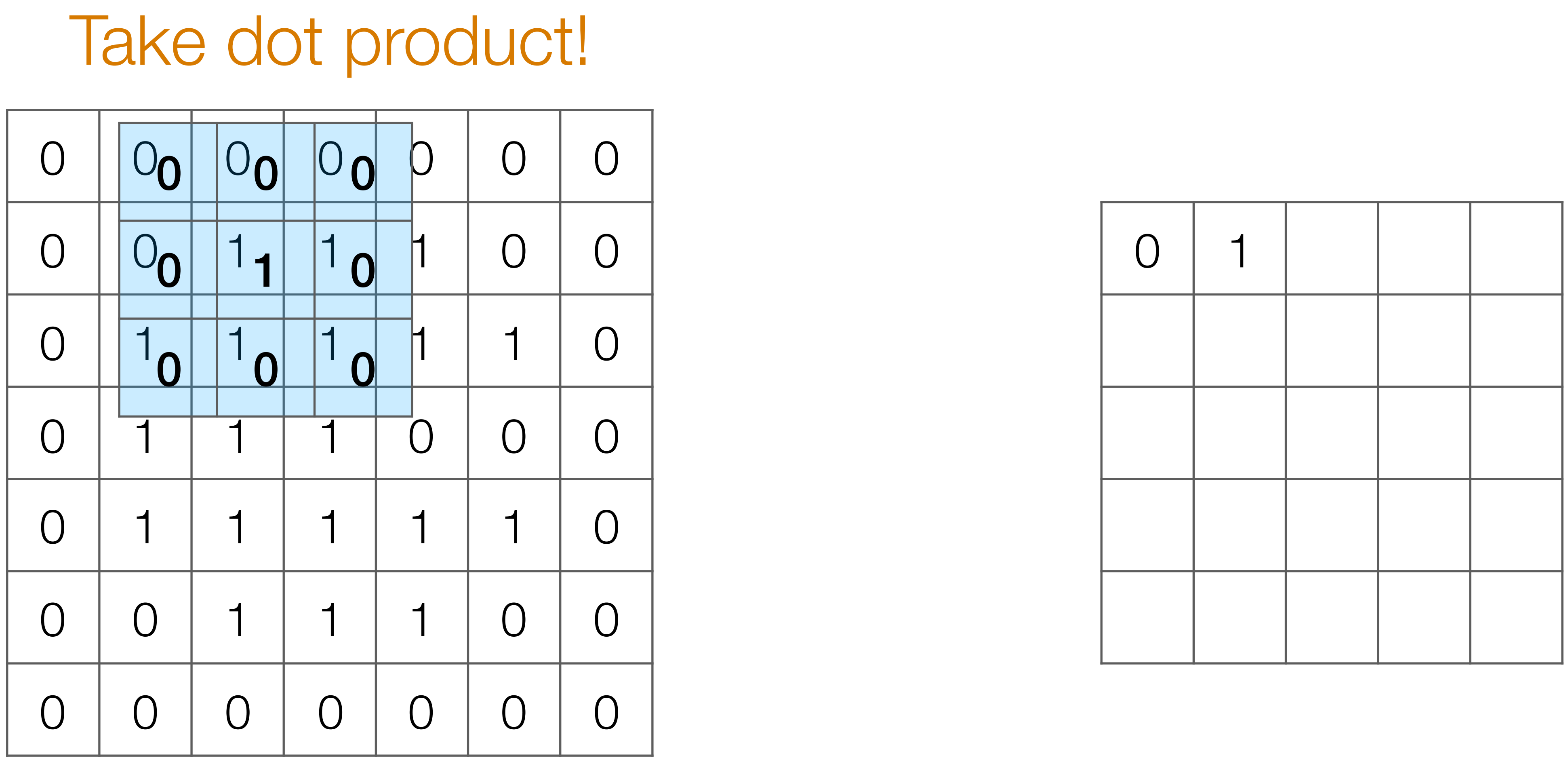
# Convolution




Input image

Output image

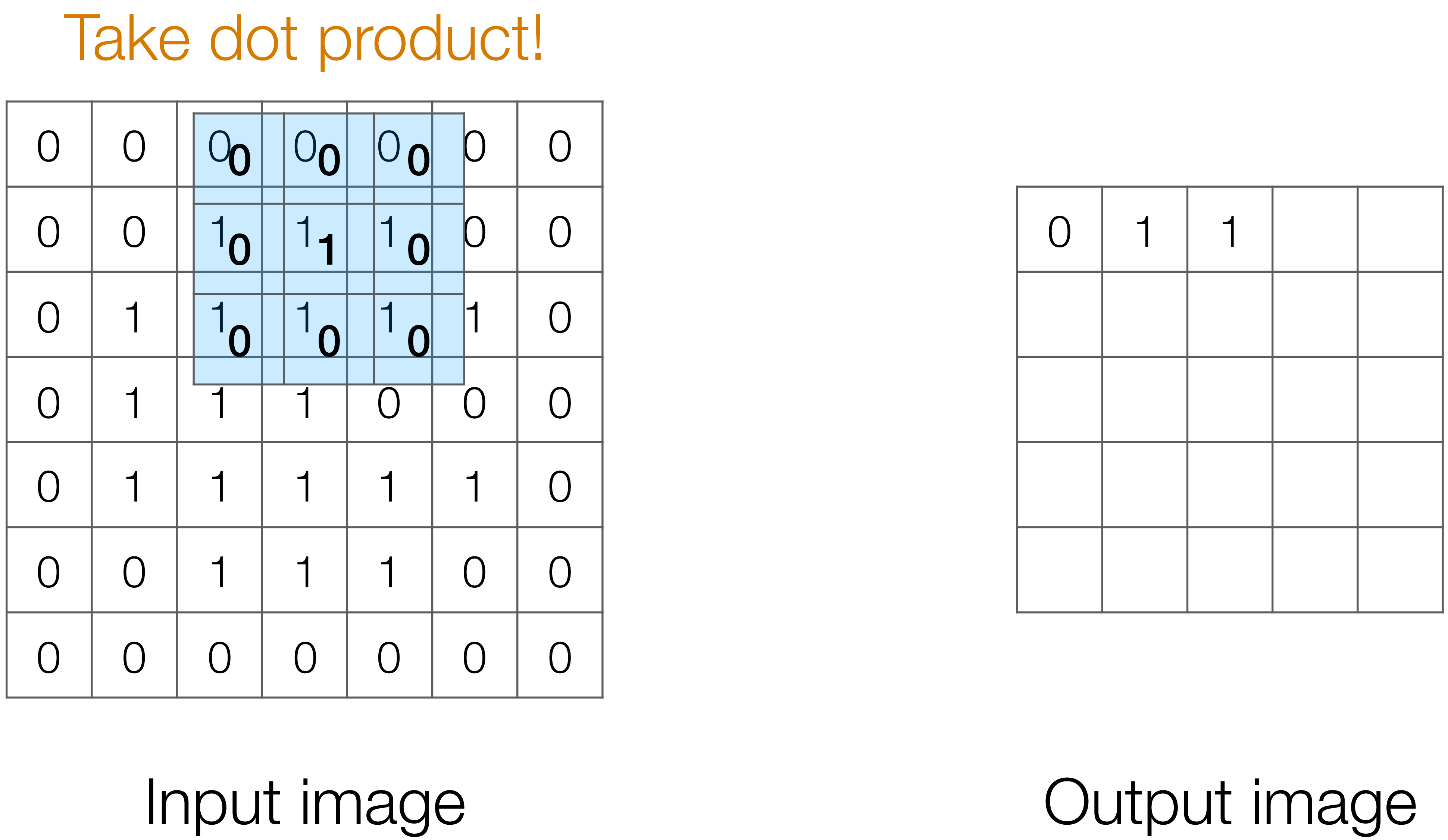
# Convolution



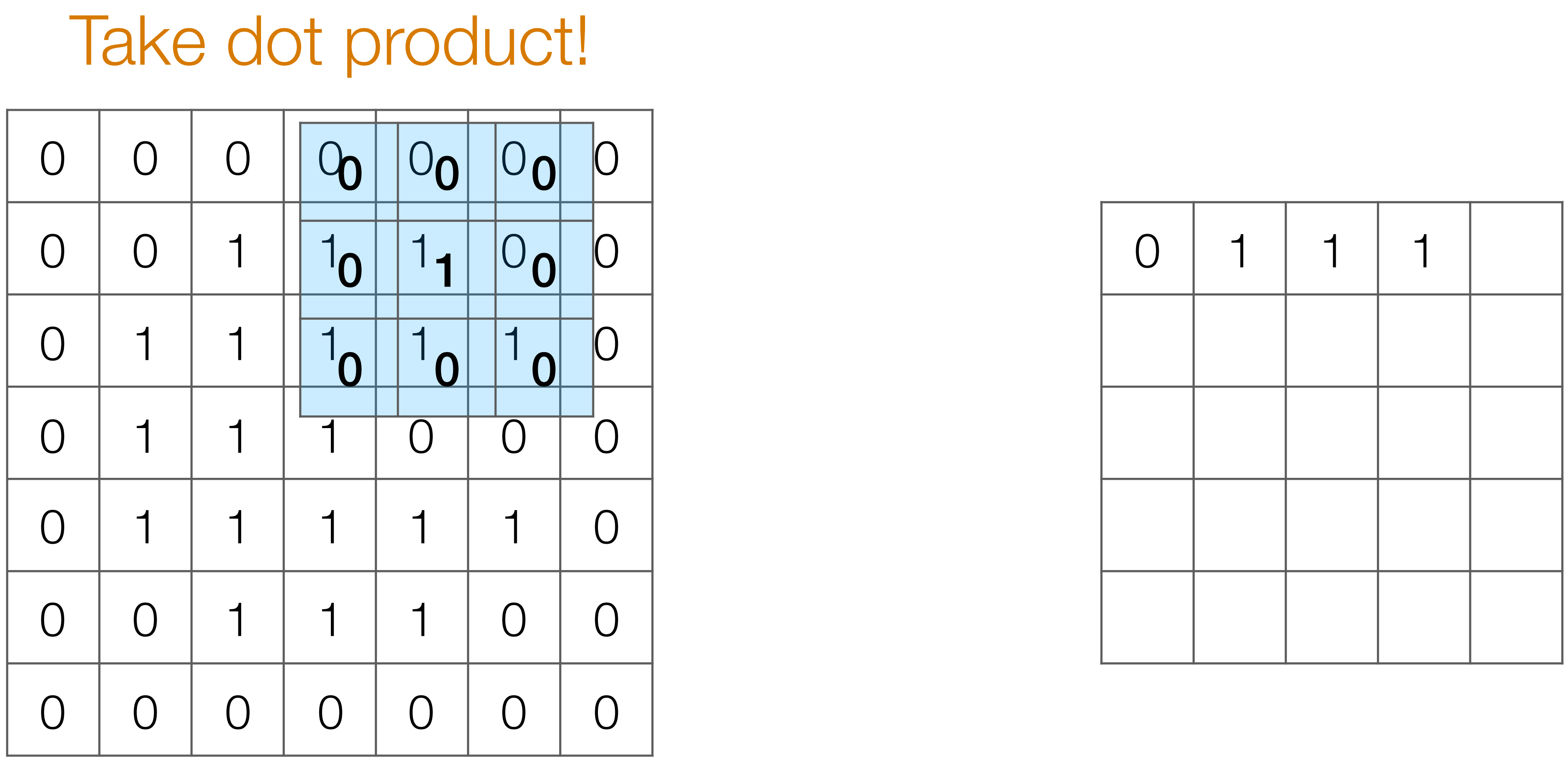

Input image

Output image

# Convolution



# Convolution

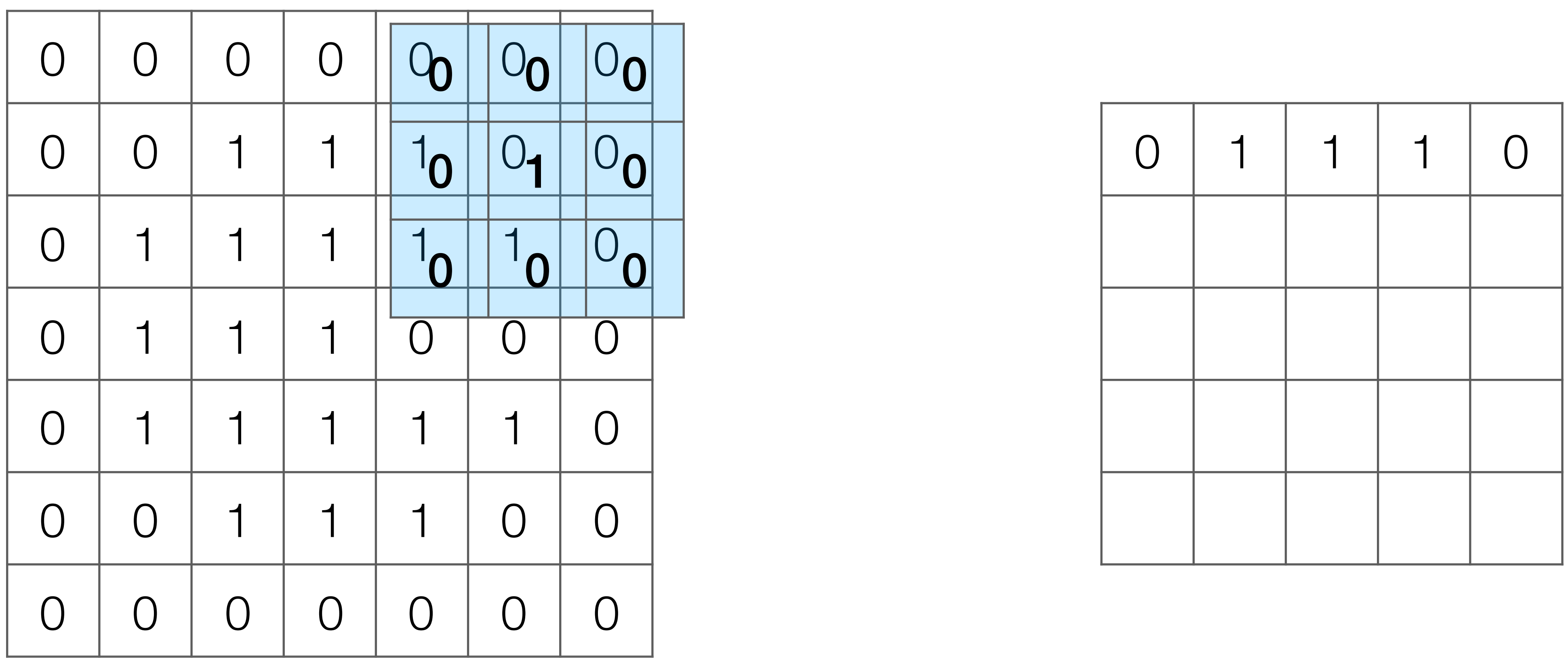



Input image

Output image

# Convolution

Take dot product!

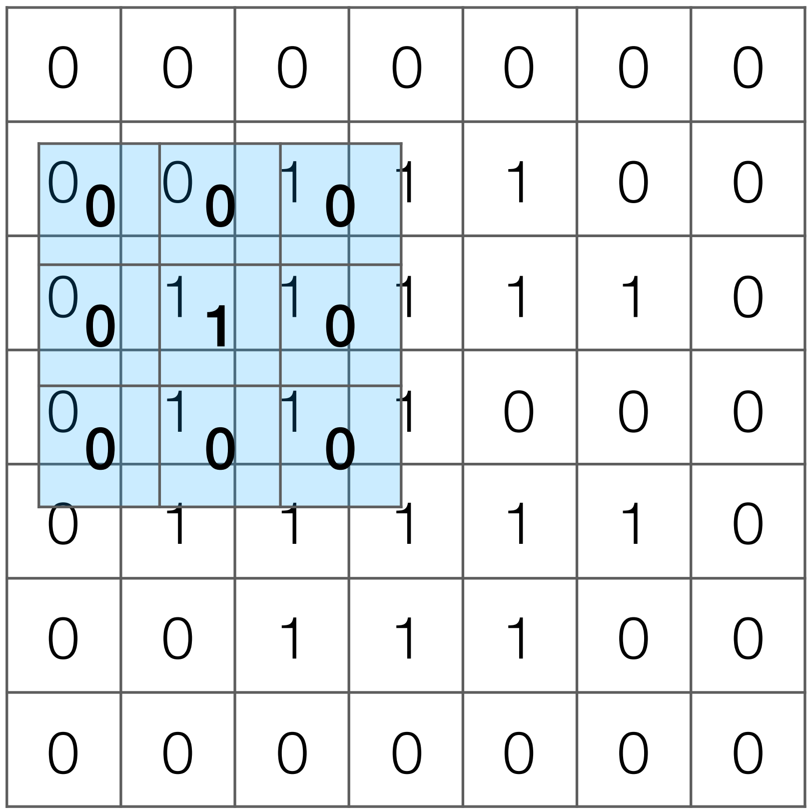



Input image

Output image

# Convolution

Take dot product!

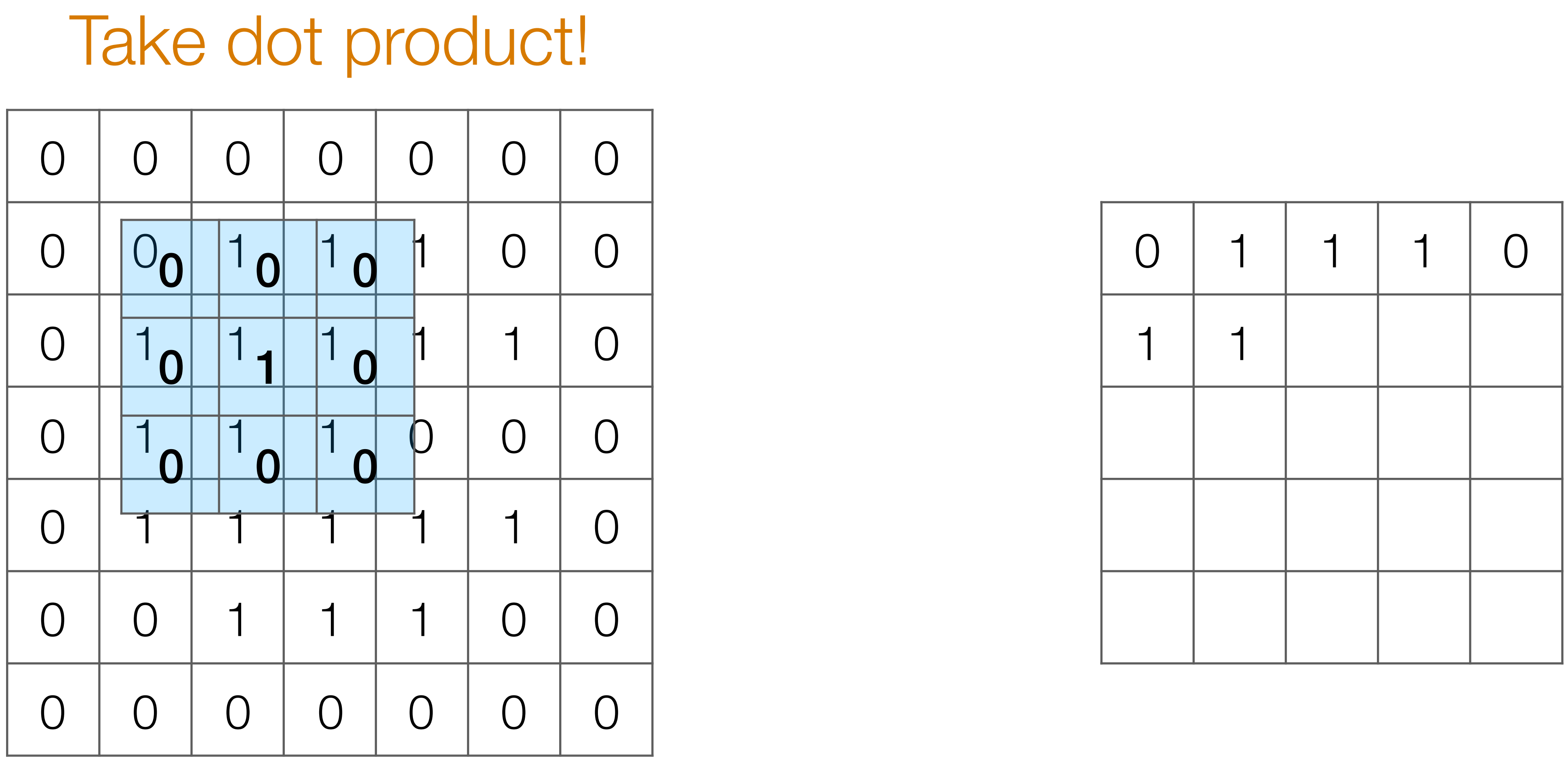


Input image

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 |
| 1 | | | | |
| | | | | |
| | | | | |
| | | | | |

Output image

# Convolution




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 |

*

| | | |
|---|---|---|
| **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

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** |

*

| | | |
|---|---|---|
| **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 |

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

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

Input image

Output image

# Convolution

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

Input image

Output image

# Convolution

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

Input image Output image

# 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

*

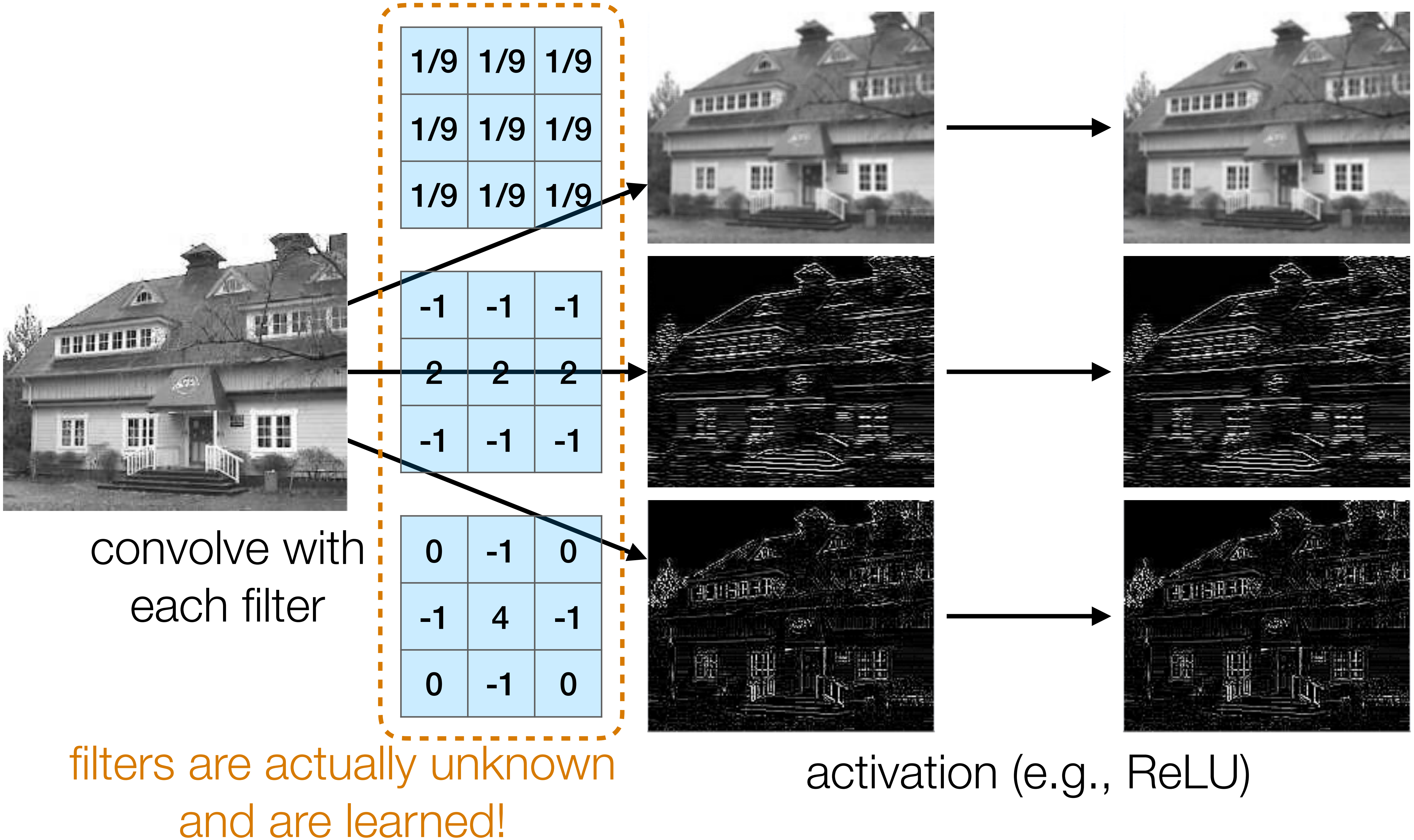
| | | |
|---|---|---|
| -1 | -1 | -1 |
| 2 | 2 | 2 |
| -1 | -1 | -1 |

=

(this example finds horizontal edges)

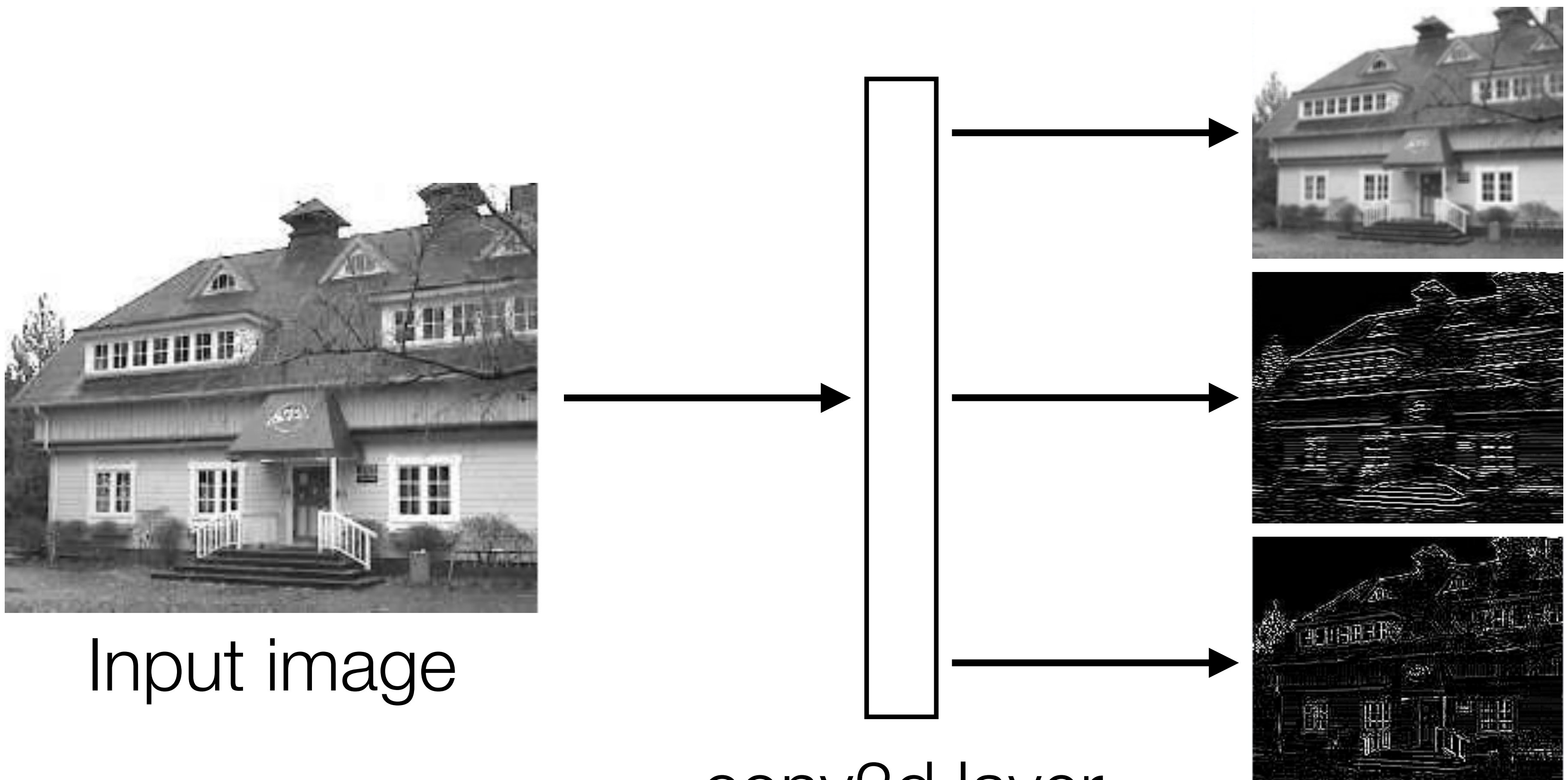
Images from: http://aishack.in/tutorials/image-convolution-examples/

# Convolution Layer




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

# Convolution Layer




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

# Convolution Layer

Input image

dimensions:
height,
width

conv2d layer
with ReLu activation
and three 3x3 kernels

Stack output images into a single “output feature map”

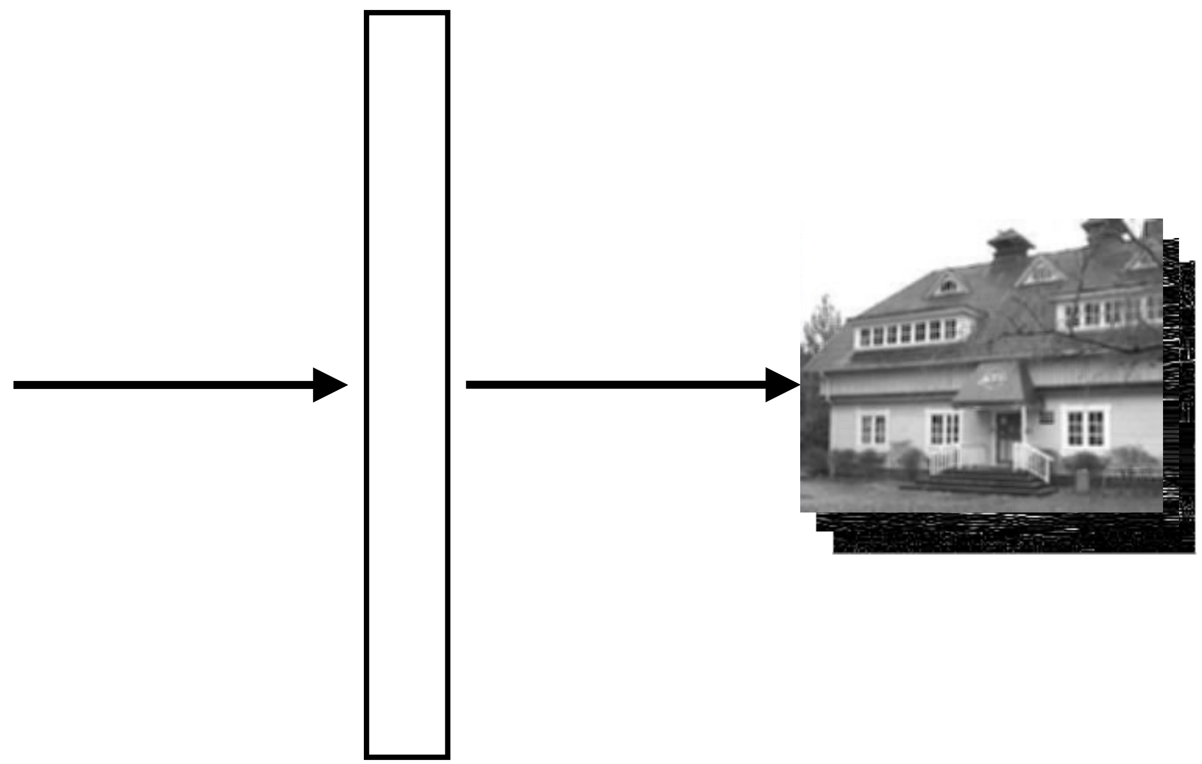
dimensions:
height-2,
width-2,
number of kernels
(3 in this case)

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

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

- Aggregate local information
- Produces a smaller image
  (each resulting pixel captures some “global” information)
- If object in input image shifts a little, output is the same

# Max Pooling

$$
\begin{array}{|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline
0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline
0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline
0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline
0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}
*
\begin{array}{|c|c|c|}
\hline
\mathbf{-1} & \mathbf{-1} & \mathbf{-1} \\ \hline
\mathbf{2} & \mathbf{2} & \mathbf{2} \\ \hline
\mathbf{-1} & \mathbf{-1} & \mathbf{-1} \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|c|}
\hline
0 & 1 & 3 & 1 & 0 \\ \hline
1 & 1 & 1 & 3 & 3 \\ \hline
0 & 0 & -2 & -4 & -4 \\ \hline
1 & 1 & 1 & 3 & 3 \\ \hline
0 & 1 & 3 & 1 & 0 \\ \hline
\end{array}
$$

Input image

# 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

| | |
|---|---|
| | |
| | |

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 | |
| | |

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

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

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

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

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?

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

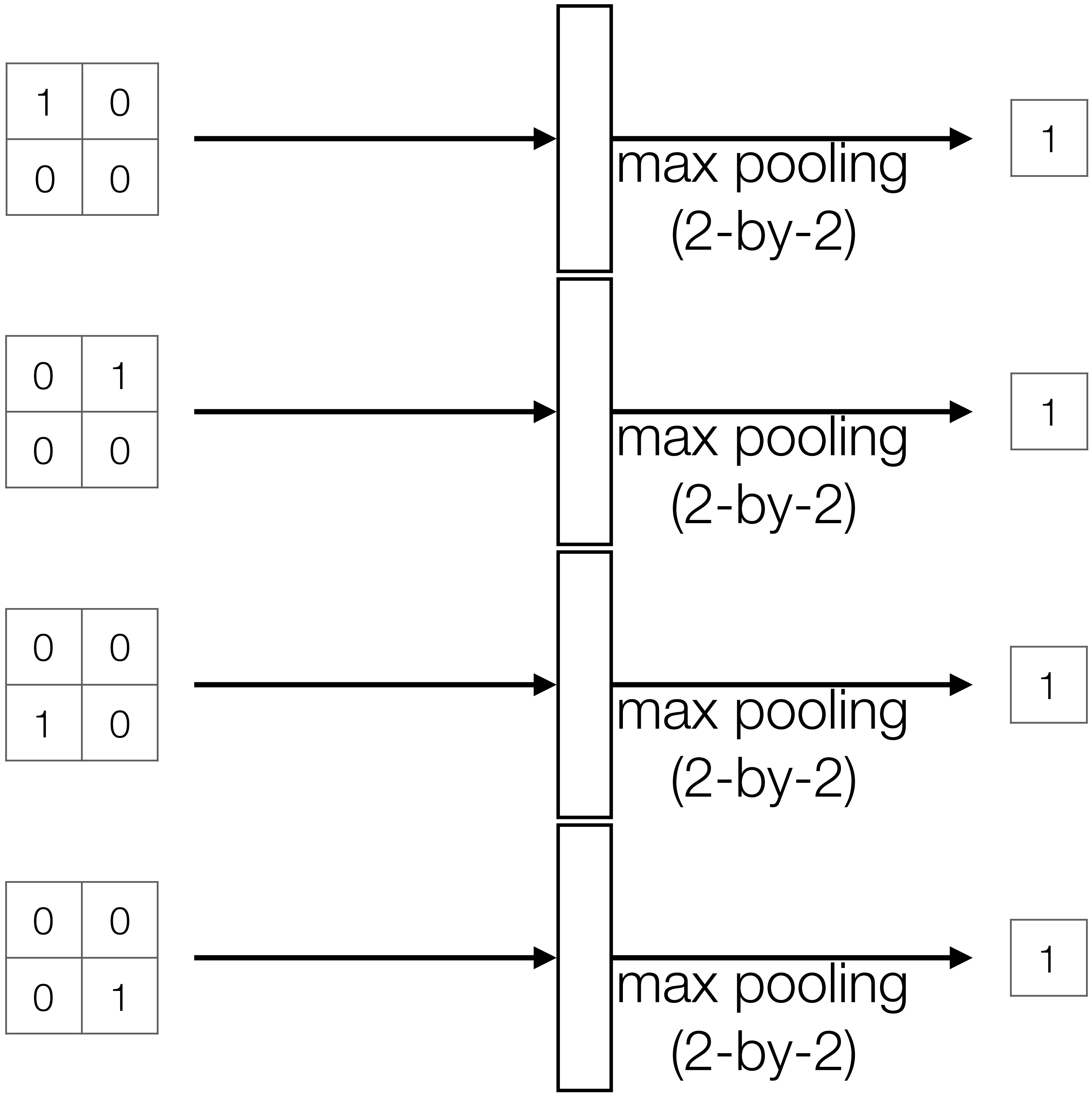
Output after max pooling

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

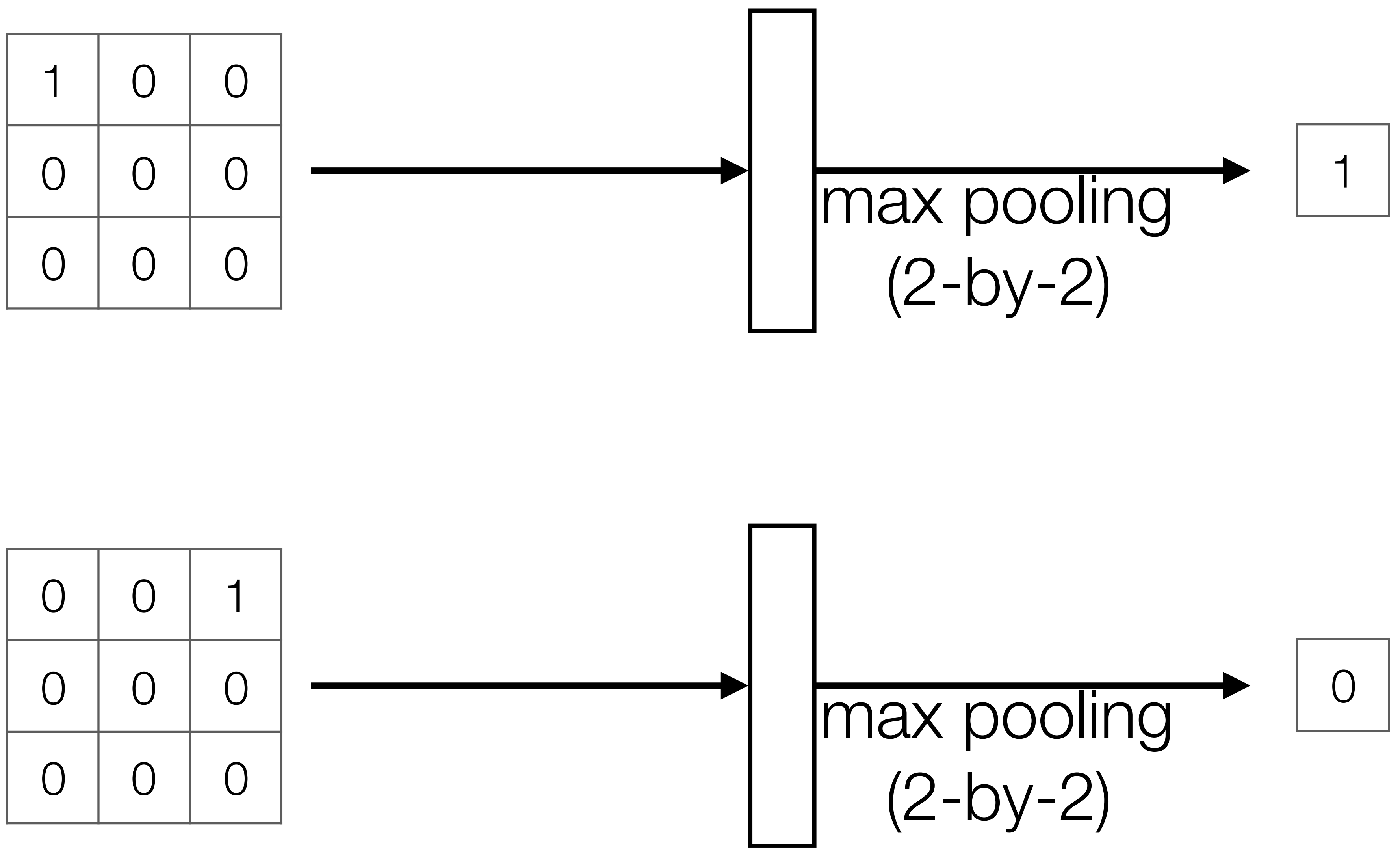
Example: applying max pooling again results in a single pixel that captures info from entire input image!

# Max Pooling and (Slight) Shift Invariance

Small shift of object in input image results in same output

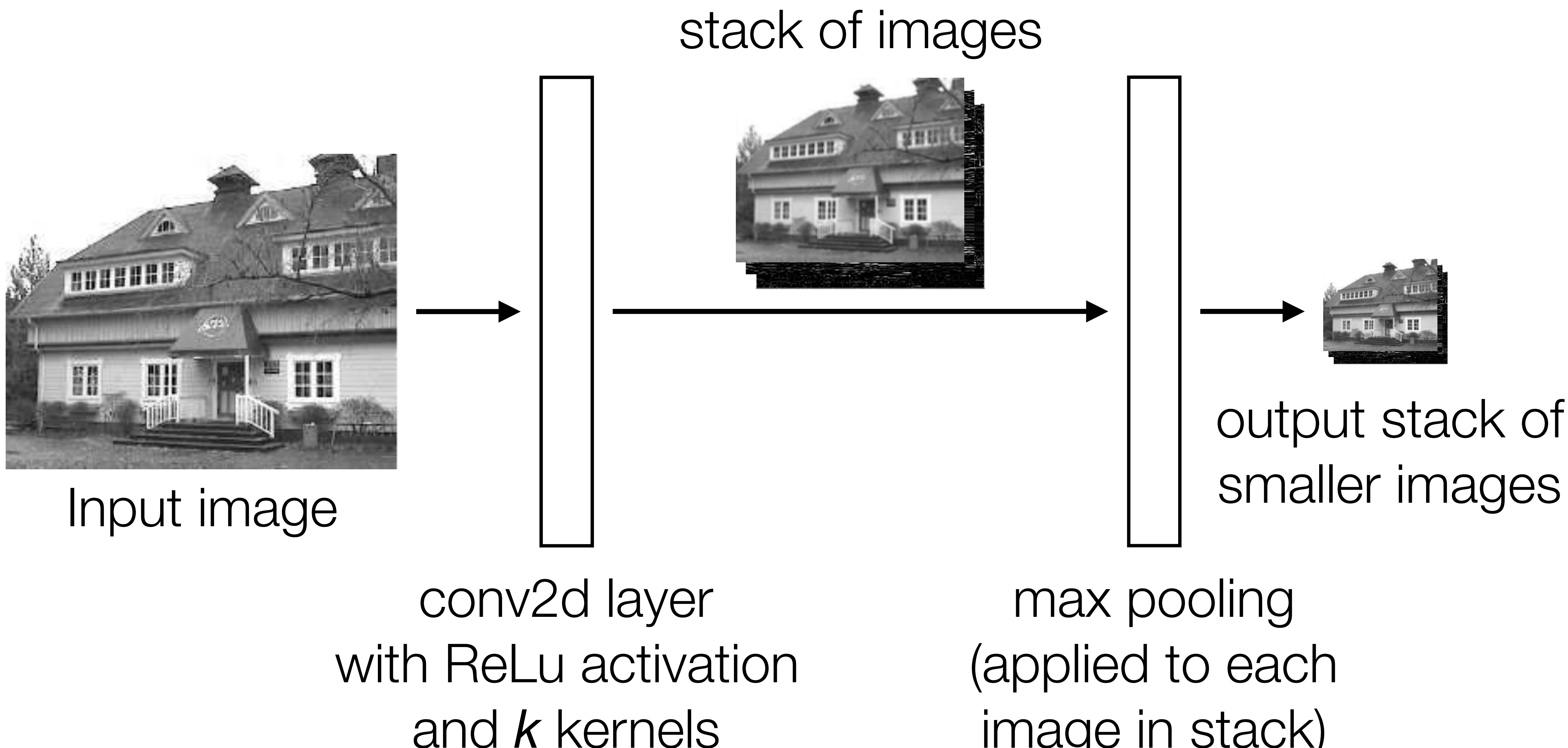


# Max Pooling and (Slight) Shift Invariance



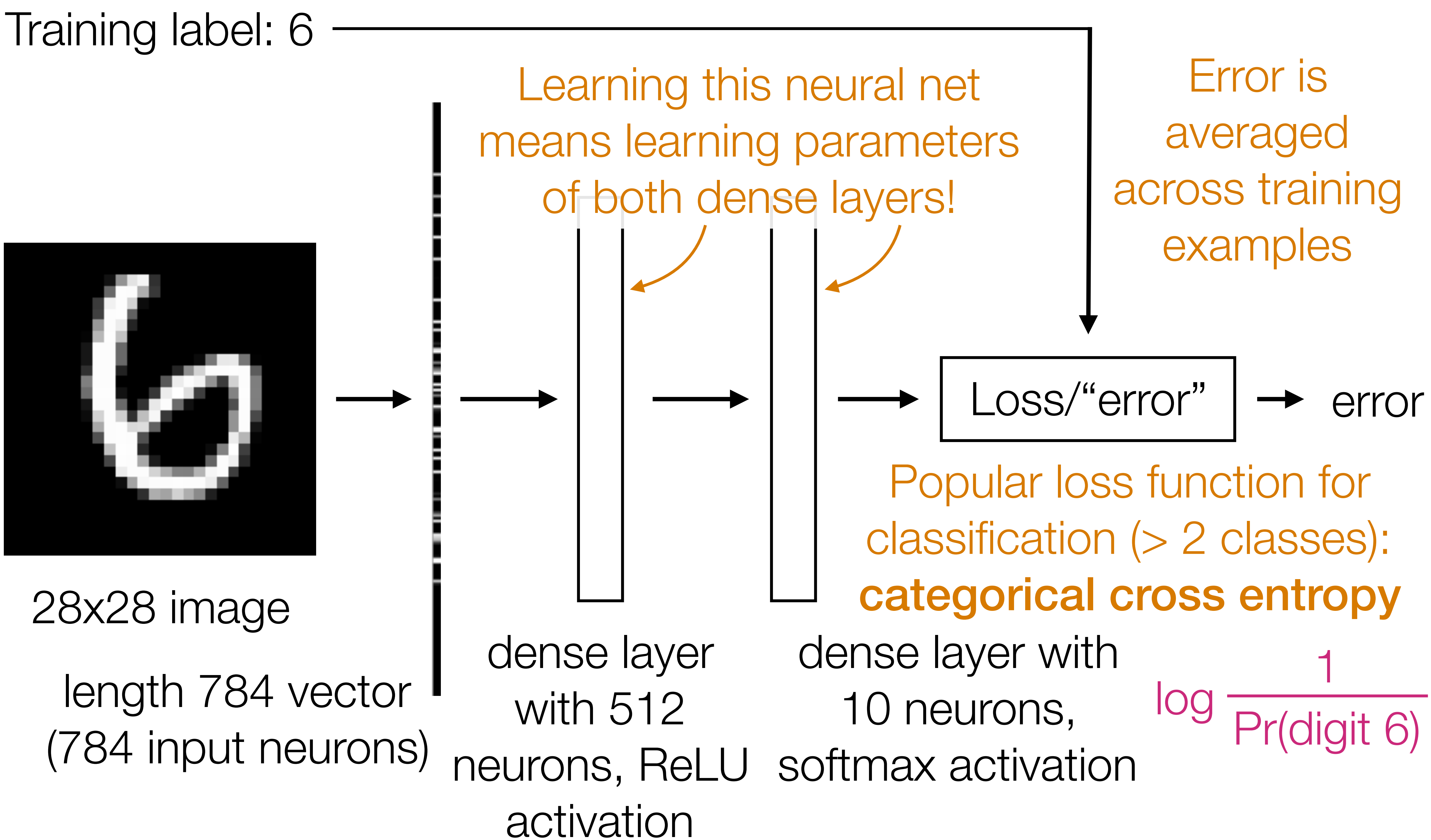

Big shift in input can still change output

# Basic Building Block of CNN’s

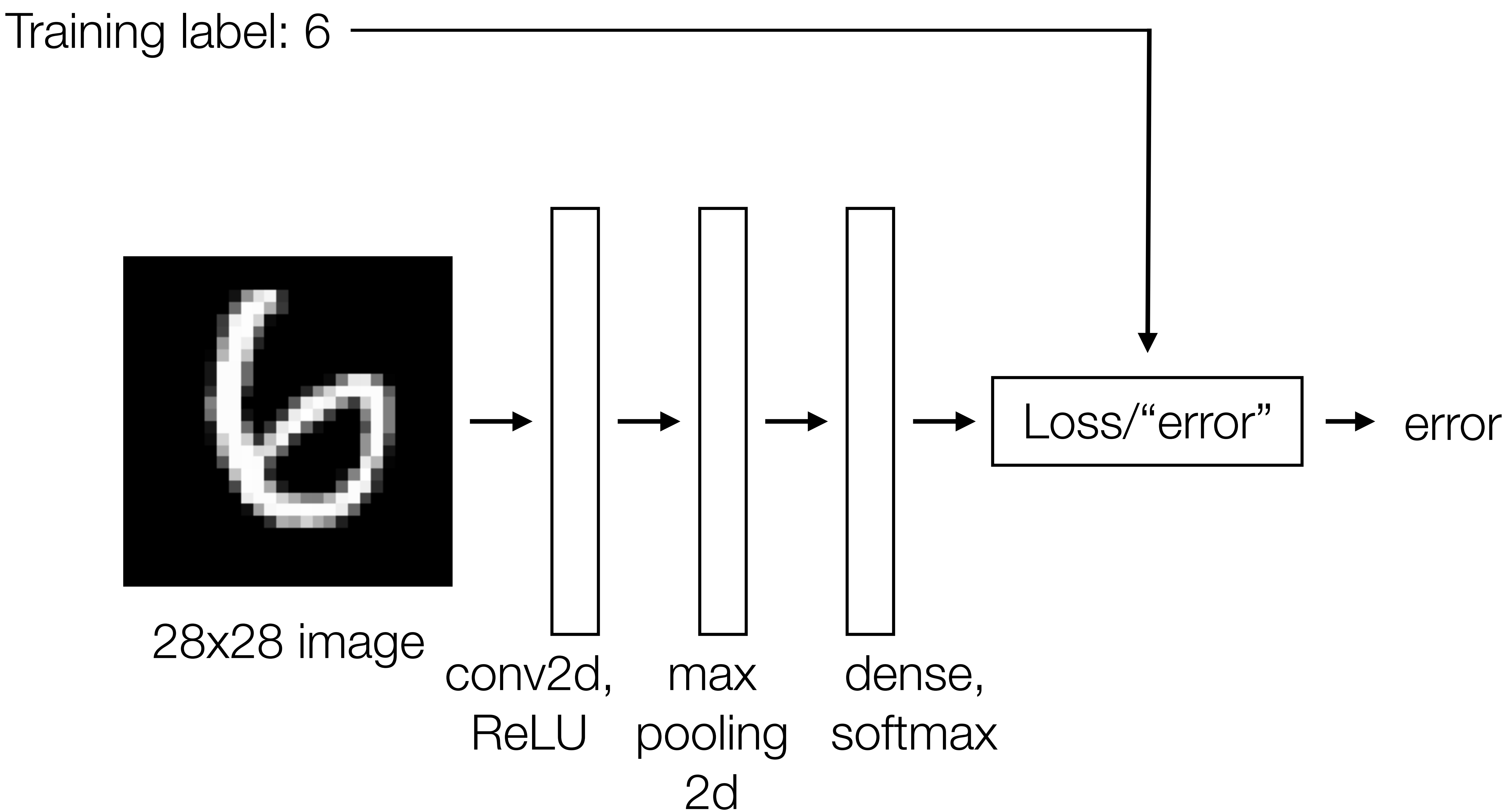



Images from: http://aishack.in/tutorials/image-convolution-examples/

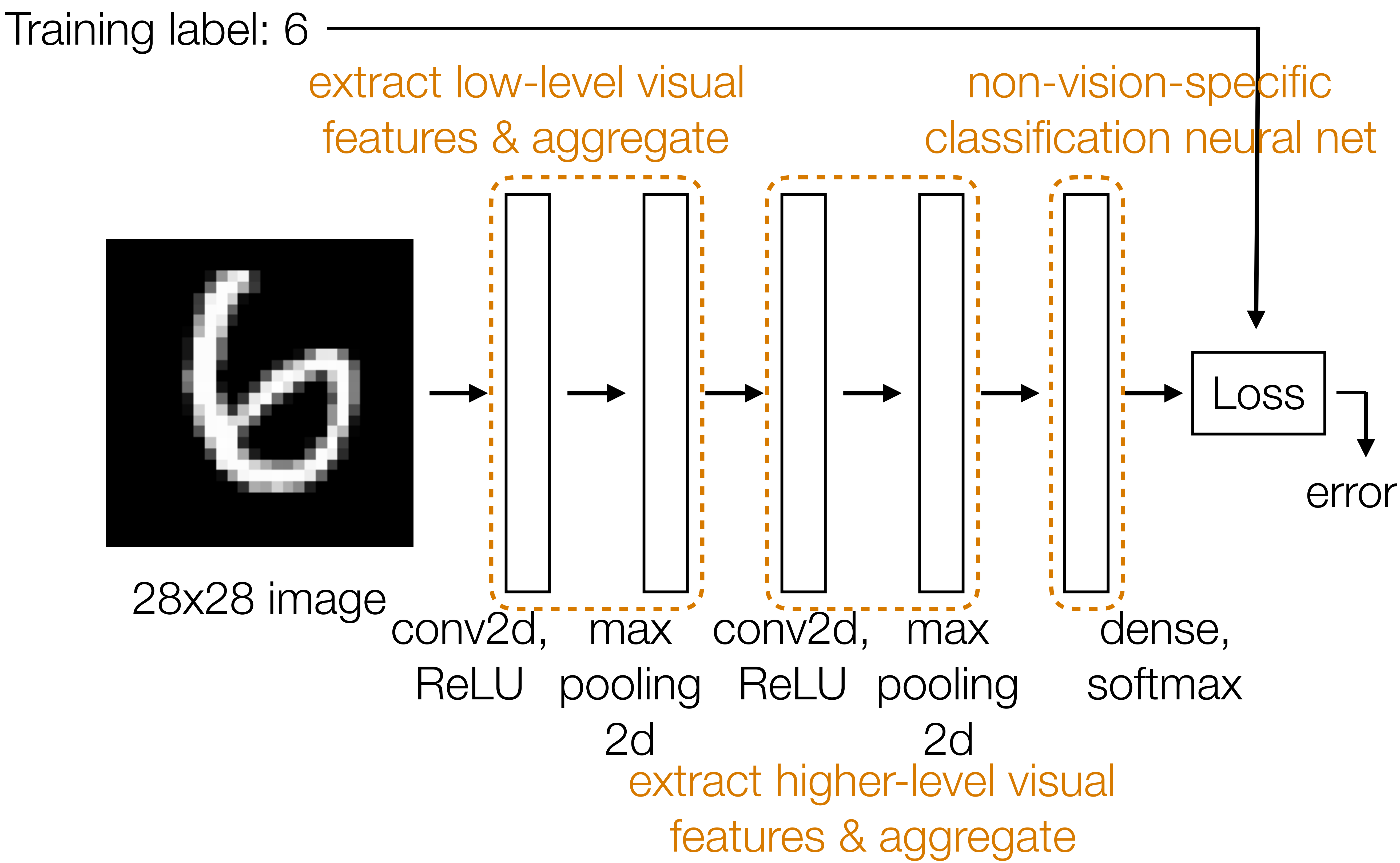
# Handwritten Digit Recognition



# Handwritten Digit Recognition



# Handwritten Digit Recognition



# CNN Demo

# CNN's

- Learn convolution filters for extracting simple features

- Max pooling summarizes information and produces a *smaller* output and is invariant to small shifts in input objects

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