

95-865 Unstructured Data Analytics

Lecture 12: Wrap up neural net basics;
image analysis with convolutional neural
nets (also called CNNs or convnets)

Slides by George H. Chen

Architecting Neural Nets

- Basic building block that is often repeated:
linear layer followed by *nonlinear* activation
 - Without nonlinear activation, two consecutive linear layers is mathematically equivalent to having a single linear layer!
- How to select # of nodes in a layer, or # of layers?
 - These are hyperparameters! *Infinite* possibilities!
 - Choose between different hyperparameter settings by using the strategy from last lecture (choose based on validation accuracy)
 - Very expensive in practice!
(Active area of research: neural architecture search)
- Much more common in practice: modify existing architectures that are known to work well
(e.g., ResNet for image classification/object recognition)

PyTorch Has Lots of Examples

→ ↺ 🔒 pytorch.org/examples/ 🔗 ☆ ℒX 📄 🏠

PyTorch

Get Started Ecosystem PyTorch Edge ▾ Blog Tutorials Docs ▾ Resou

🔍 Search Docs

PyTorch Examples

Docs > PyTorch Examples

PYTORCH EXAMPLES

This pages lists various PyTorch examples that you can use to learn and experiment with PyTorch.

Image Classification using Vision Transformer

This example shows how to train a **Vision Transformer** from scratch on the **CIFAR10** database.
[GO TO EXAMPLE](#) ↗

Image Classification Using ConvNets

This example demonstrates how to run image classification with **Convolutional Neural Networks ConvNets** on the **MNIST** database.
[GO TO EXAMPLE](#) ↗

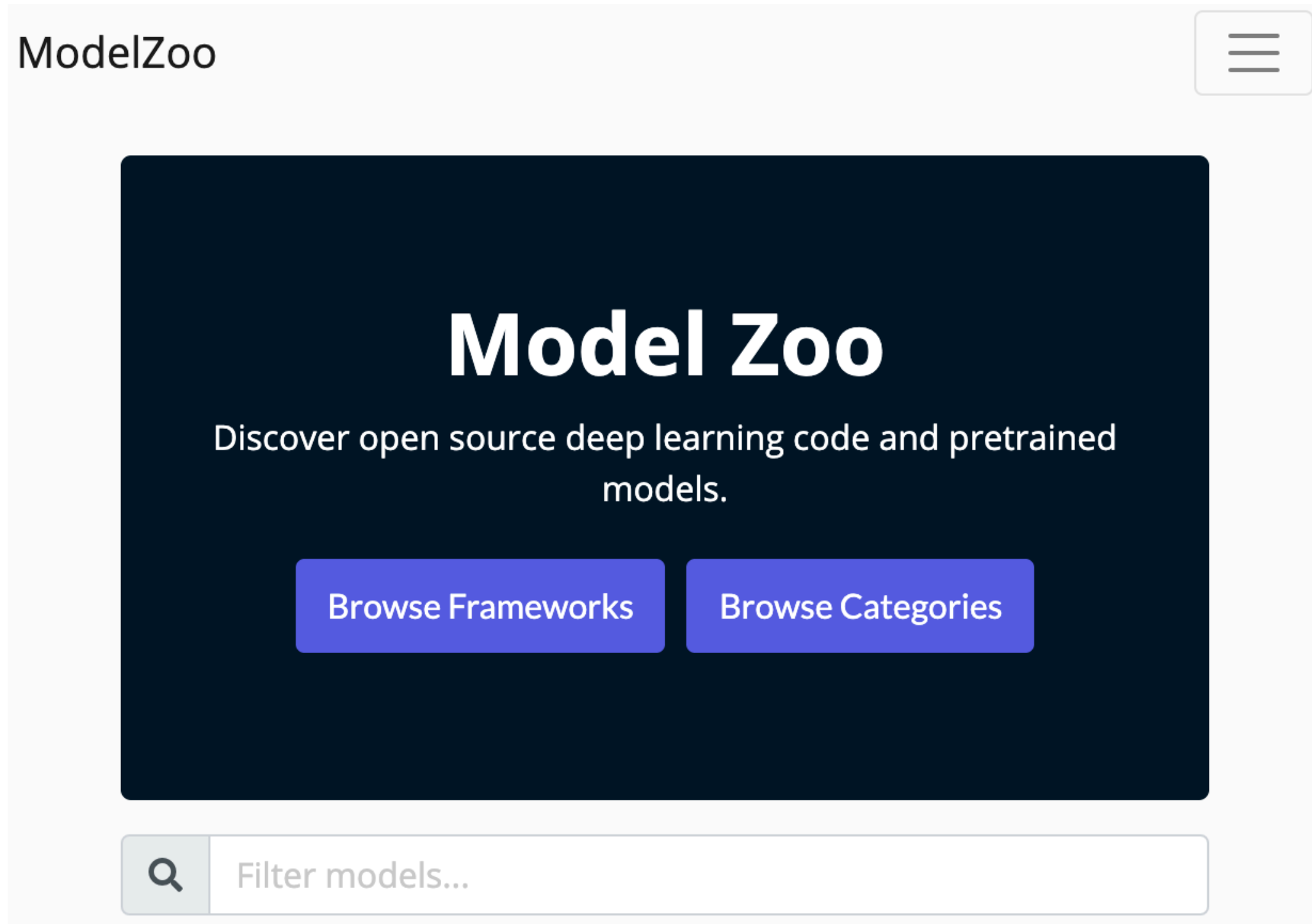
Measuring Similarity using Siamese Network

This example demonstrates how to measure similarity between two images using **Siamese**

Word-level Language Modeling using RNN and Transformer

This example demonstrates how to train a multi-

Find a Massive Collection of Models at the Model Zoo



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

Multinomial logistic regression:

```
def f(input):
```

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

Multinomial logistic regression:

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

Multilayer perceptron:

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

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

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

Complexity of a Neural Net?

Increasing number of layers (depth) makes neural net more “complex”

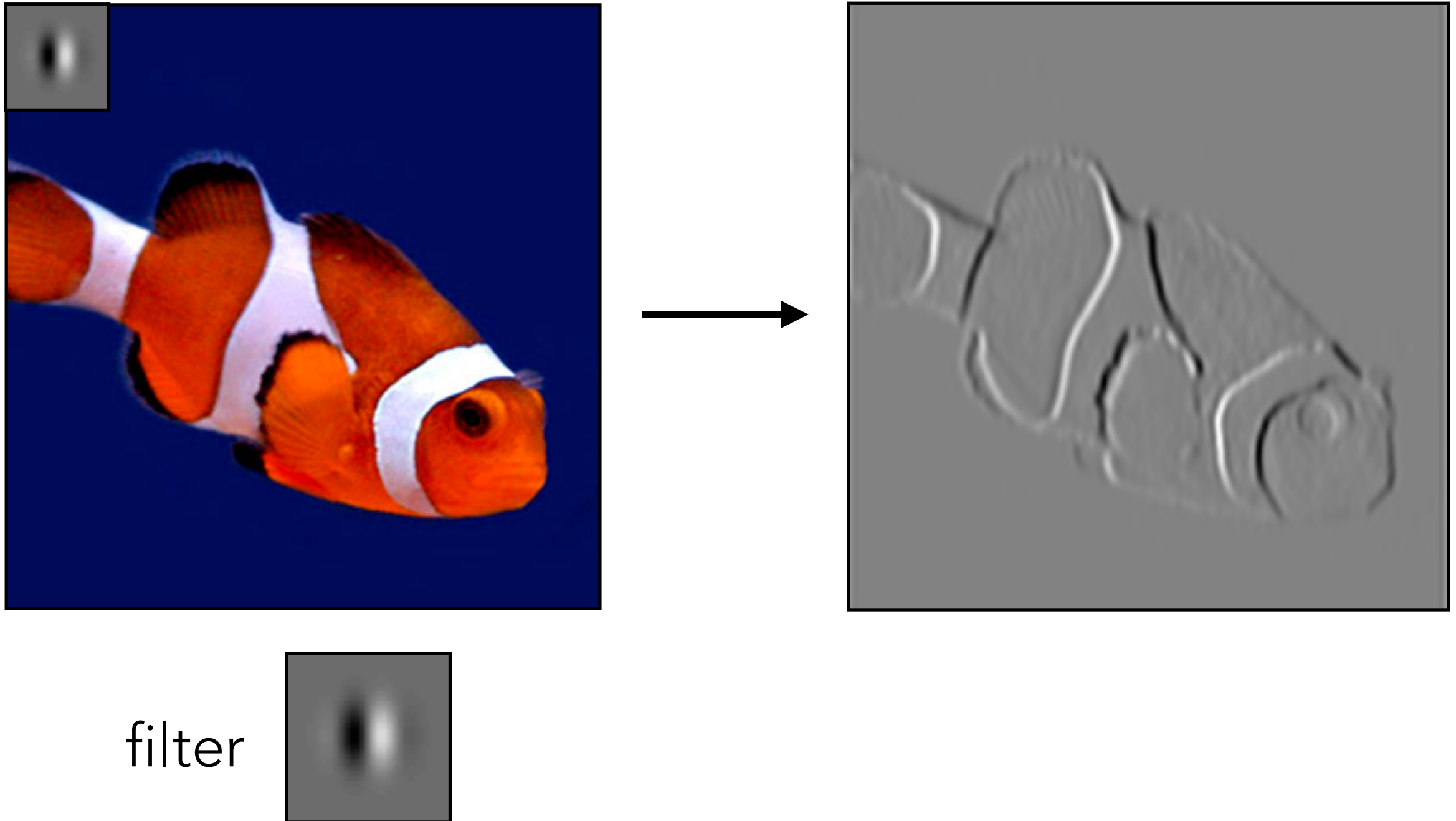
⇒ Learn computer program that has more lines of code

Earlier: MLP had more parameters than logistic regression

Upcoming: we'll see an example where a deeper network has *fewer* parameters than a shallower one

Accounting for image structure:
convolutional neural nets
(CNNs or convnets)

Convolution



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

Take dot product!

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

Output image

Convolution

Take dot product!

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

Output image

Convolution

Take dot product!

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

Output image

Convolution

Take dot product!

0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	1	1	0	1	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!

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

Output image

Convolution

Take dot product!

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

Output image

Convolution

Take dot product!

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

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

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

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

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

$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

$$= \frac{1}{9}$$

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

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

Very commonly used for:

- Blurring an image



$$\begin{array}{|c|c|c|} \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array} =$$



- Finding edges



$$\begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline 2 & 2 & 2 \\ \hline -1 & -1 & -1 \\ \hline \end{array} =$$



(this example finds horizontal edges)

Convolution Layer

Activation layer
(such as ReLU)

Conv2d
layer



convolve with
each filter

$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$



add bias

apply
activation

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



add bias

apply
activation

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

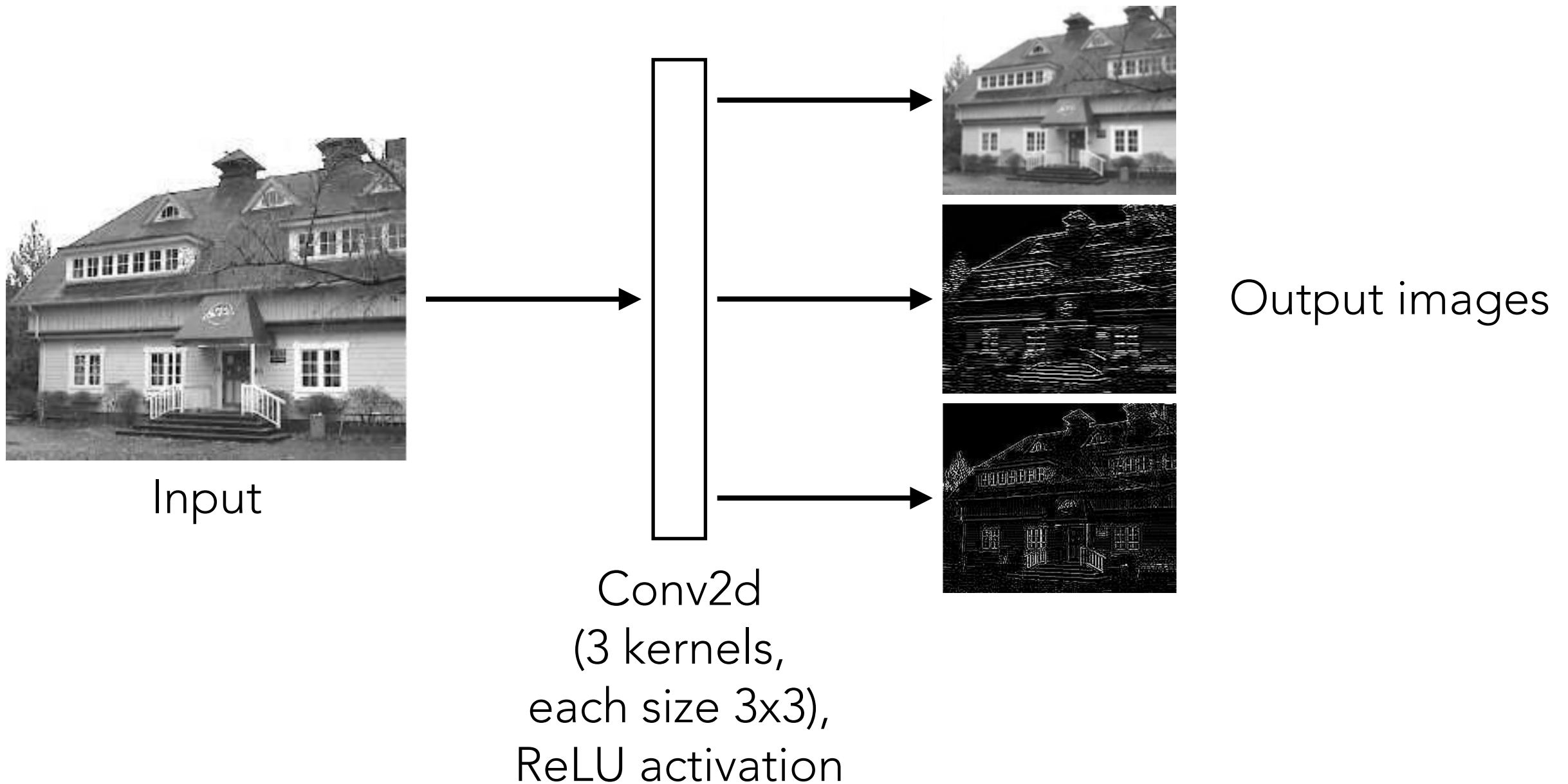


add bias

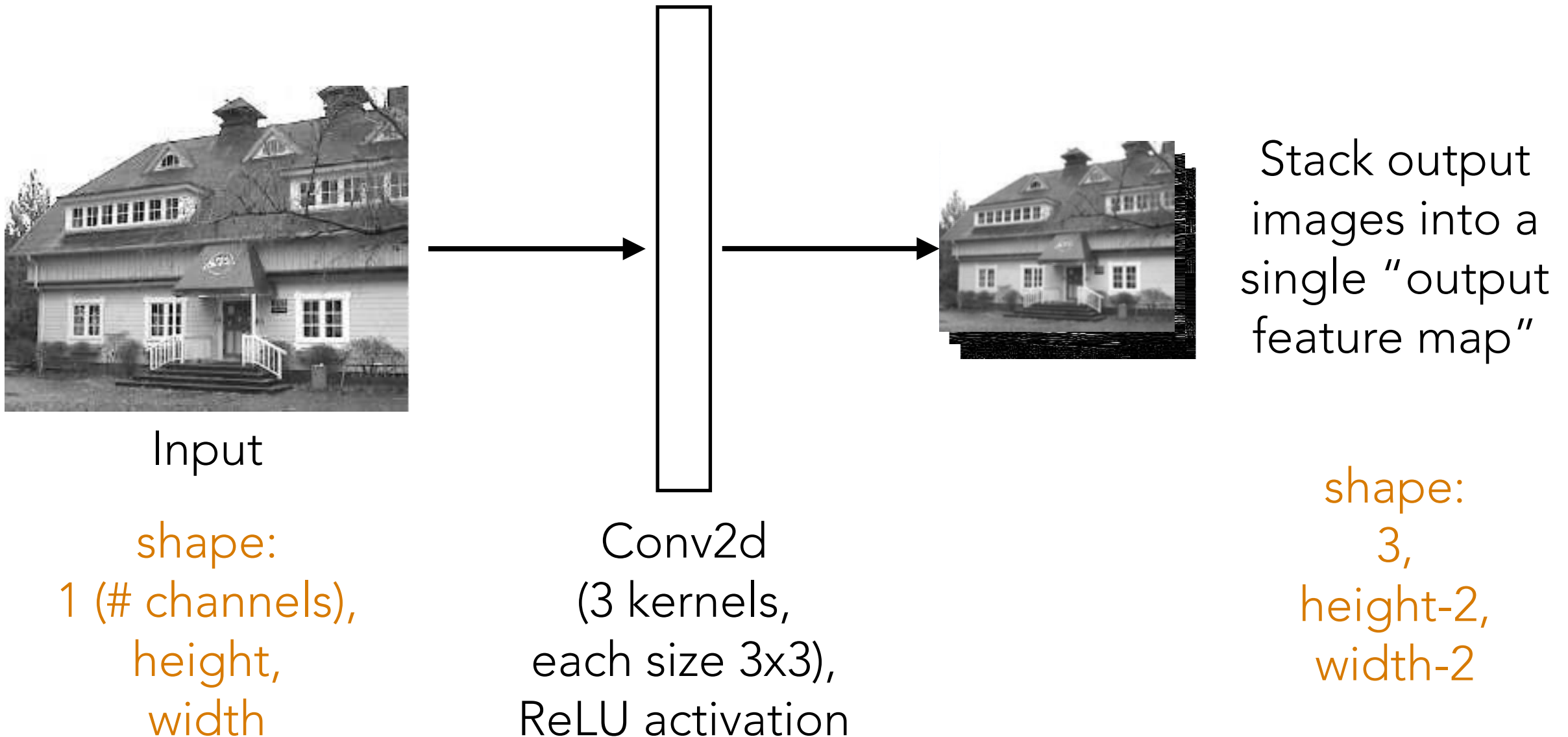
apply
activation

filters & biases (1 bias number per filter)
are unknown and are learned!

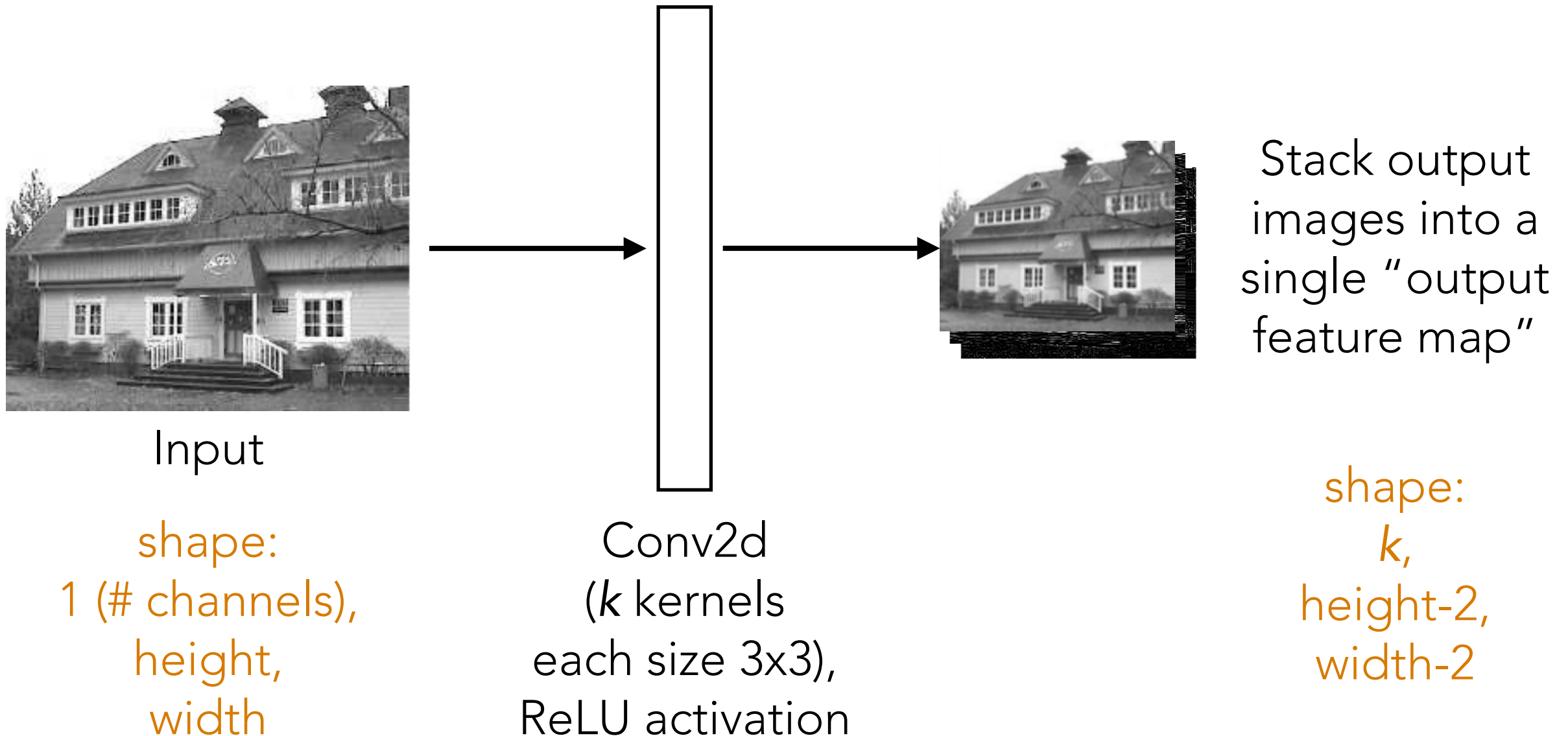
Convolution Layer



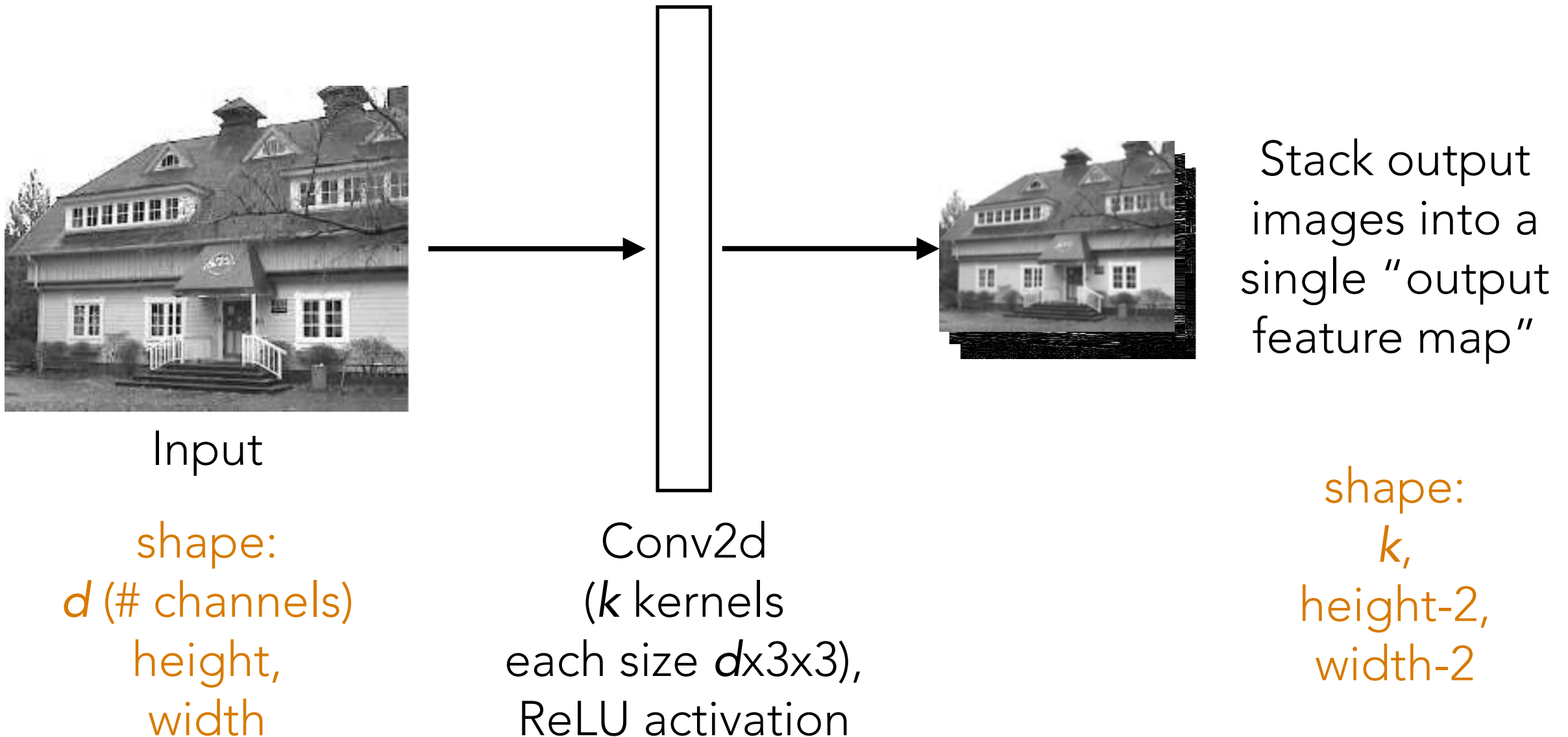
Convolution Layer



Convolution Layer



Convolution Layer

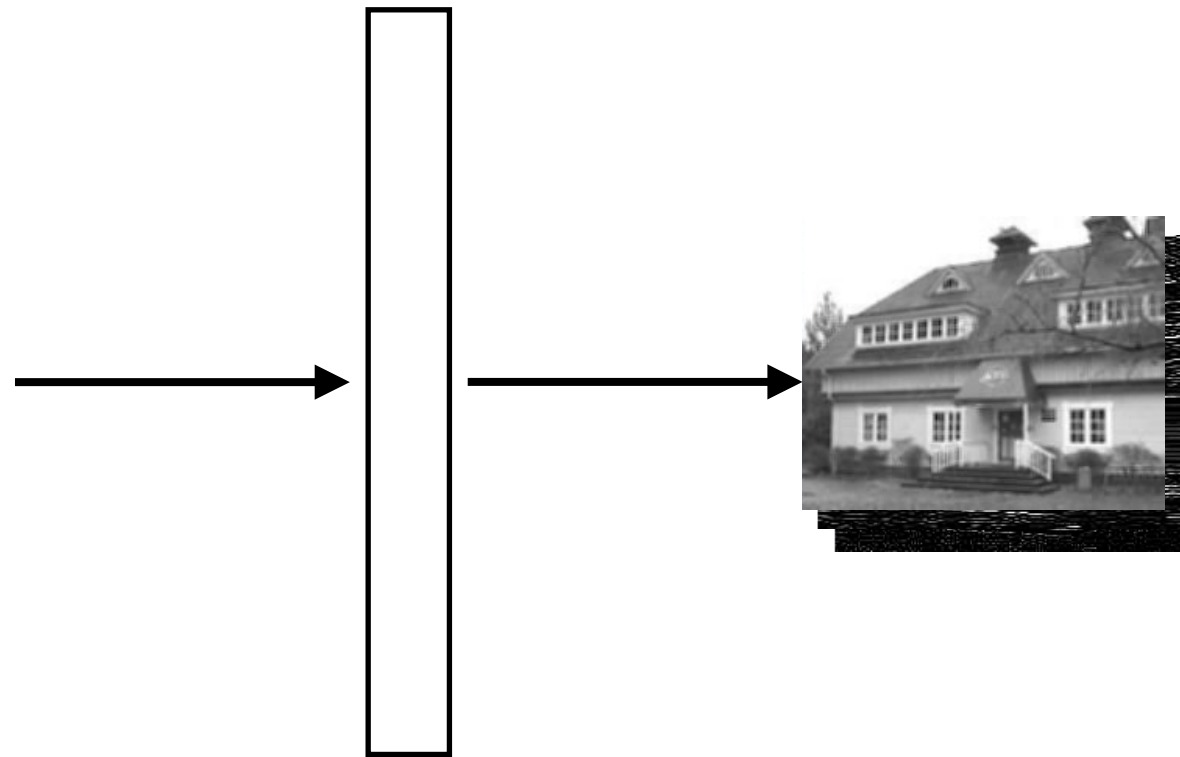


Convolution Layer



Input

shape:
 d (# channels)
height,
width

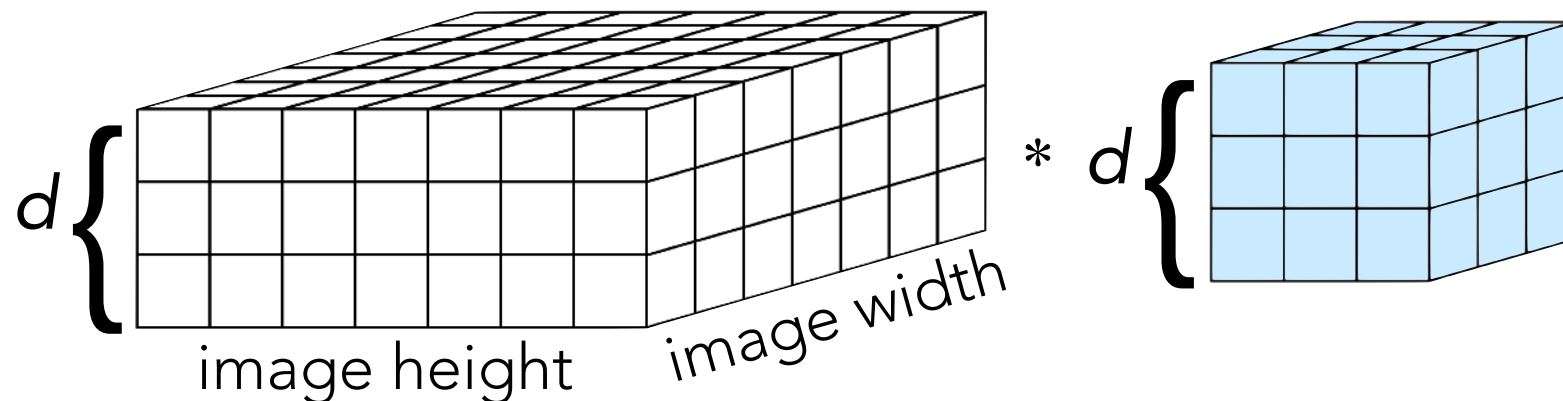


Conv2d
(k kernels
each size $d \times 3 \times 3$),
ReLU activation

Stack output
images into a
single "output
feature map"

shape:
 k ,
height-2,
width-2

Each filter:

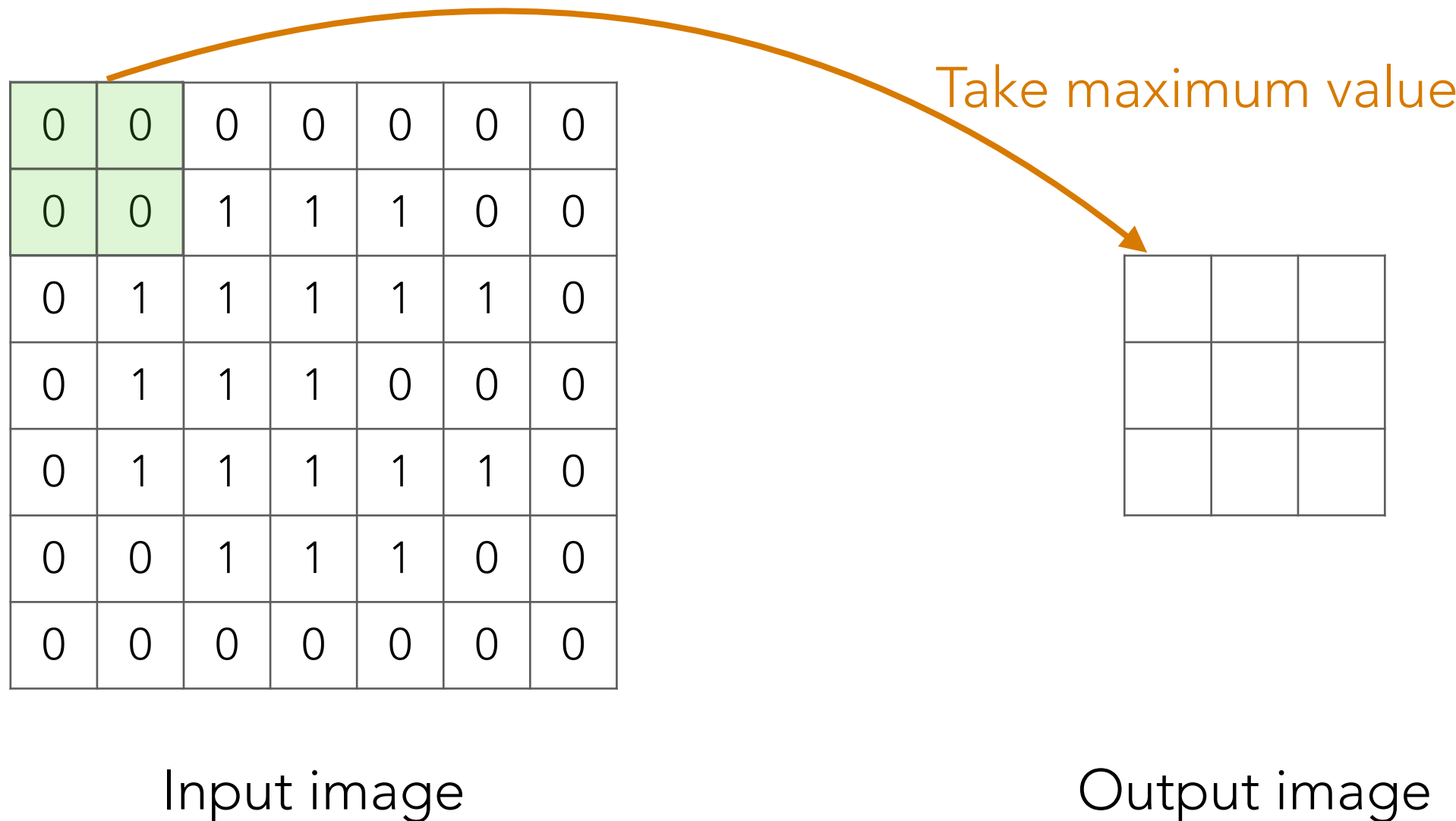


Pooling

- Produces smaller image summarizing original larger image
- To produce this smaller image, need to aggregate or “pool” together information
- If “object” in input image shifts by a little bit, want output to stay about the same

Max Pooling

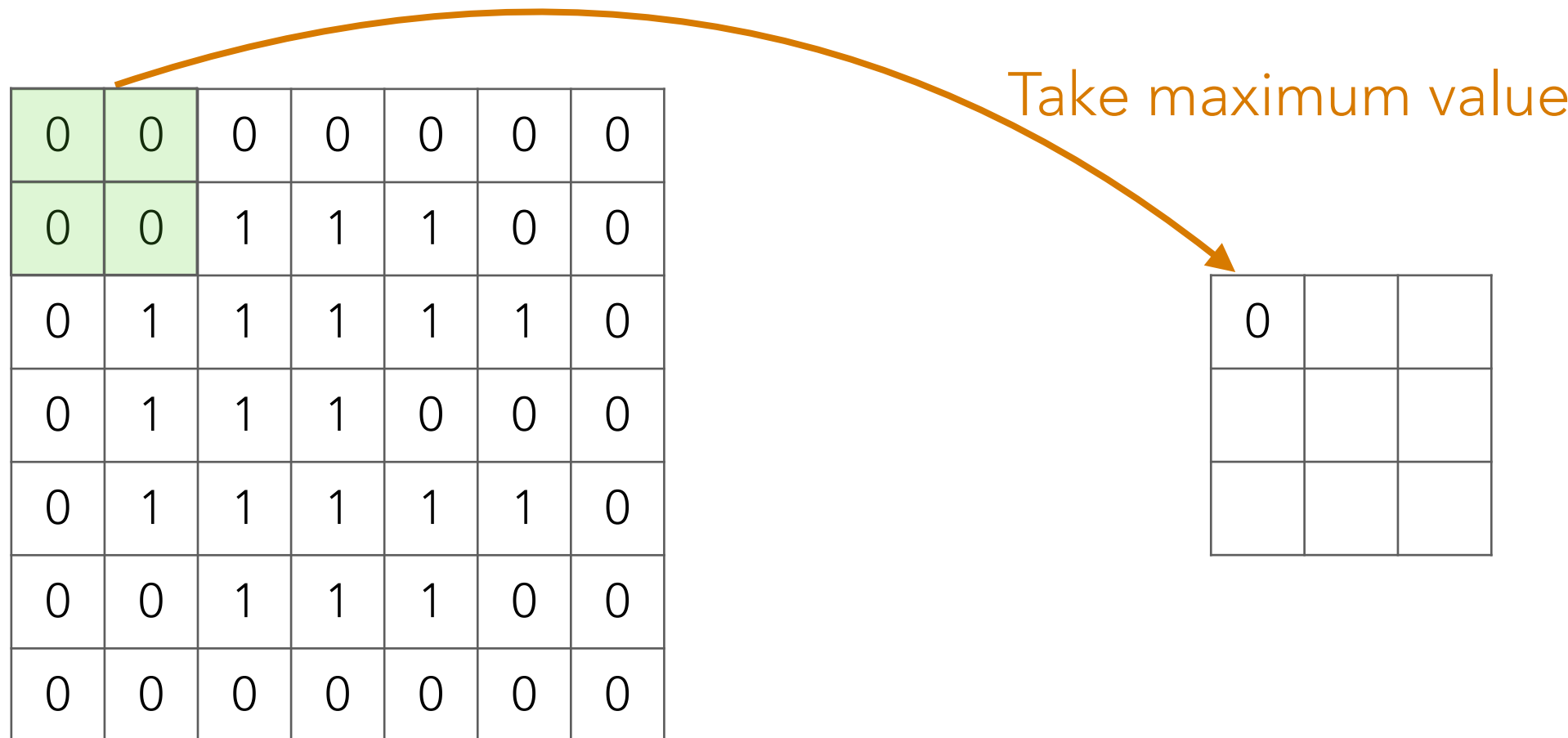
Called "2-by-2" max pooling since this green box is 2 rows by 2 columns



3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc

Max Pooling

Called "2-by-2" max pooling since this green box is 2 rows by 2 columns



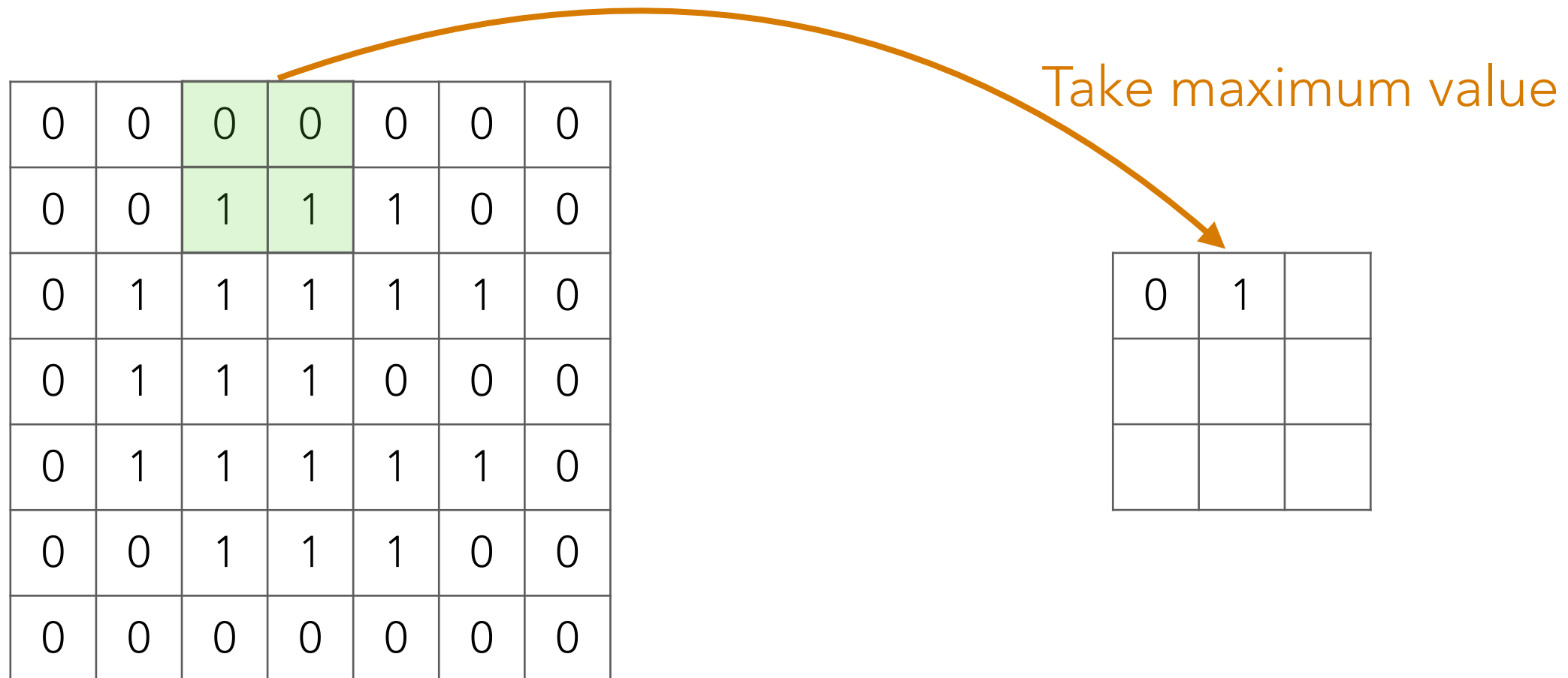
Input image

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc

Max Pooling

Called "2-by-2" max pooling since this green box is 2 rows by 2 columns



Input image

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc

Max Pooling

Called "2-by-2" max pooling since this green box is 2 rows by 2 columns

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

Take maximum value

0	1	1

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc

Max Pooling

Called "2-by-2" max pooling since this green box is 2 rows by 2 columns

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

Take maximum value

0	1	1
1		

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc

Max Pooling

Called "2-by-2" max pooling since this green box is 2 rows by 2 columns

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	1	1
1	1	1
1	1	1

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc

Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 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

$$\begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline 2 & 2 & 2 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

=

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

Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 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

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} =$$

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

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
2-by-2 max pooling

Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 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

-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
2-by-2 max pooling

Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 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

-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
2-by-2 max pooling

Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 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

-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
2-by-2 max pooling

Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 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

-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
2-by-2 max pooling

Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 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

-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?

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

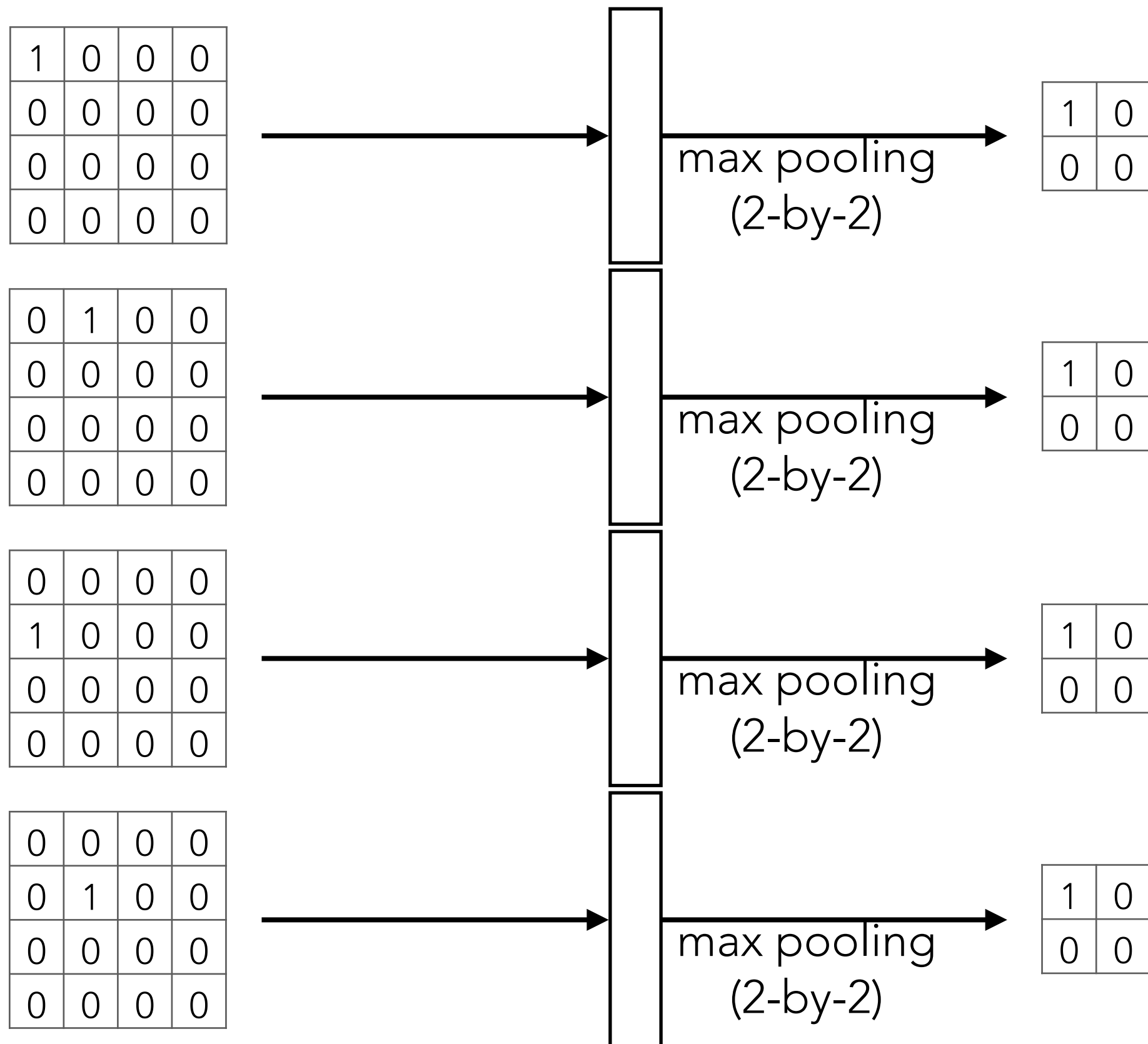
Example: applying max pooling again results in a single pixel that captures info from entire input image!



1	3
1	3

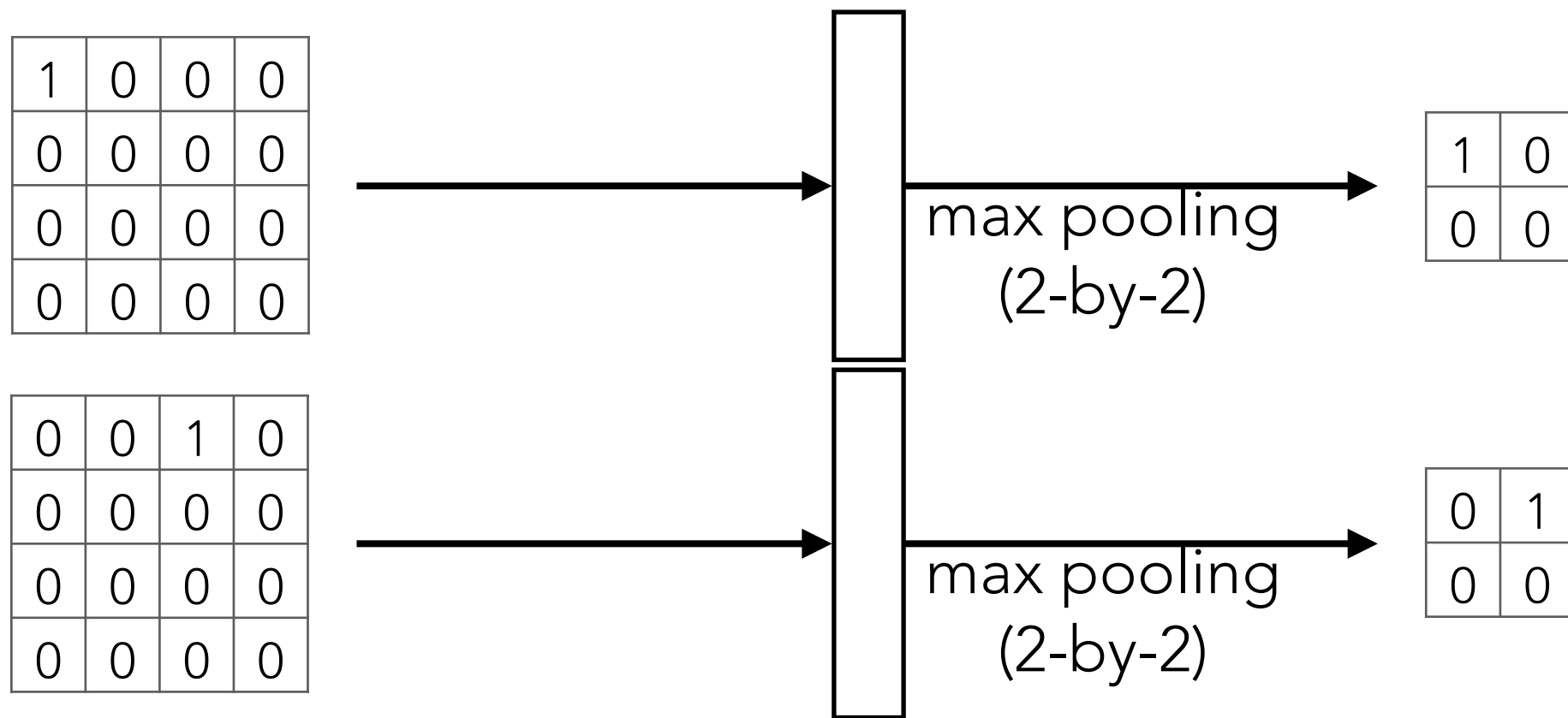
Output after
2-by-2 max pooling

Small Shifts & Max Pooling



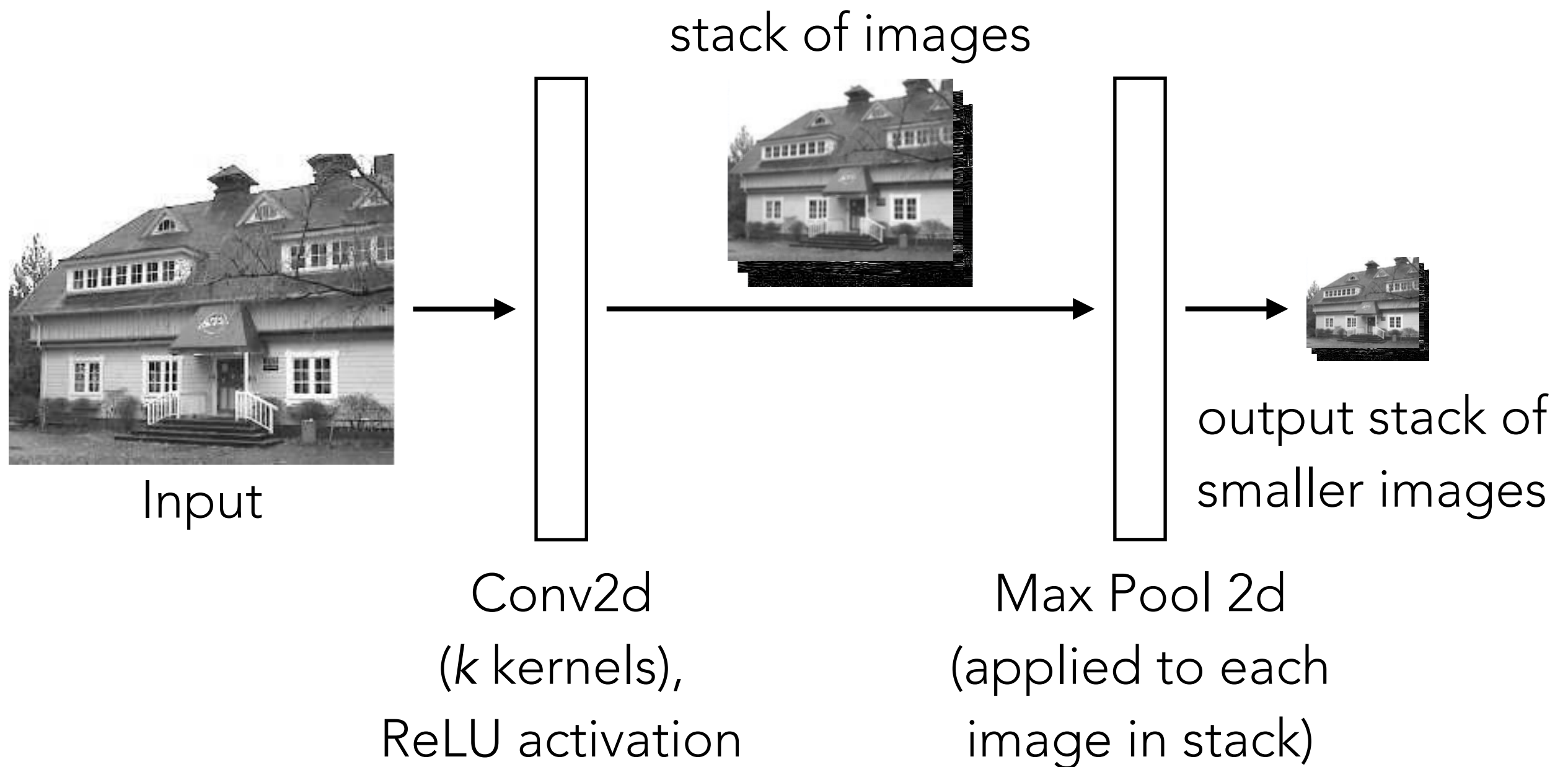
Small shift in
input object of
interest results
in same output

Small Shifts & Max Pooling



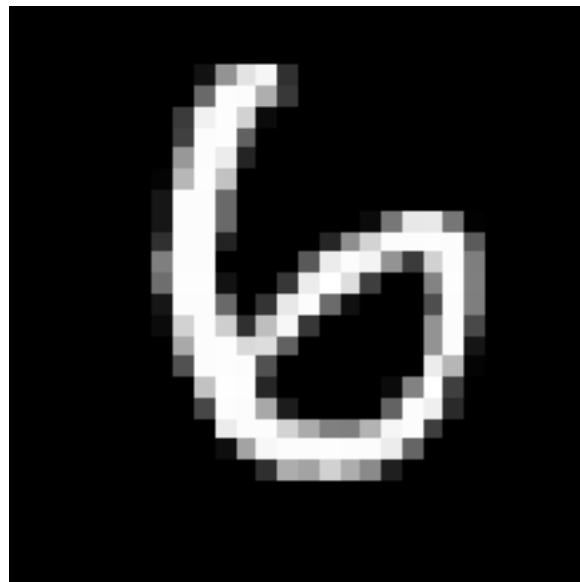
A bigger shift in the input results in a different output

Common Building Block of CNNs

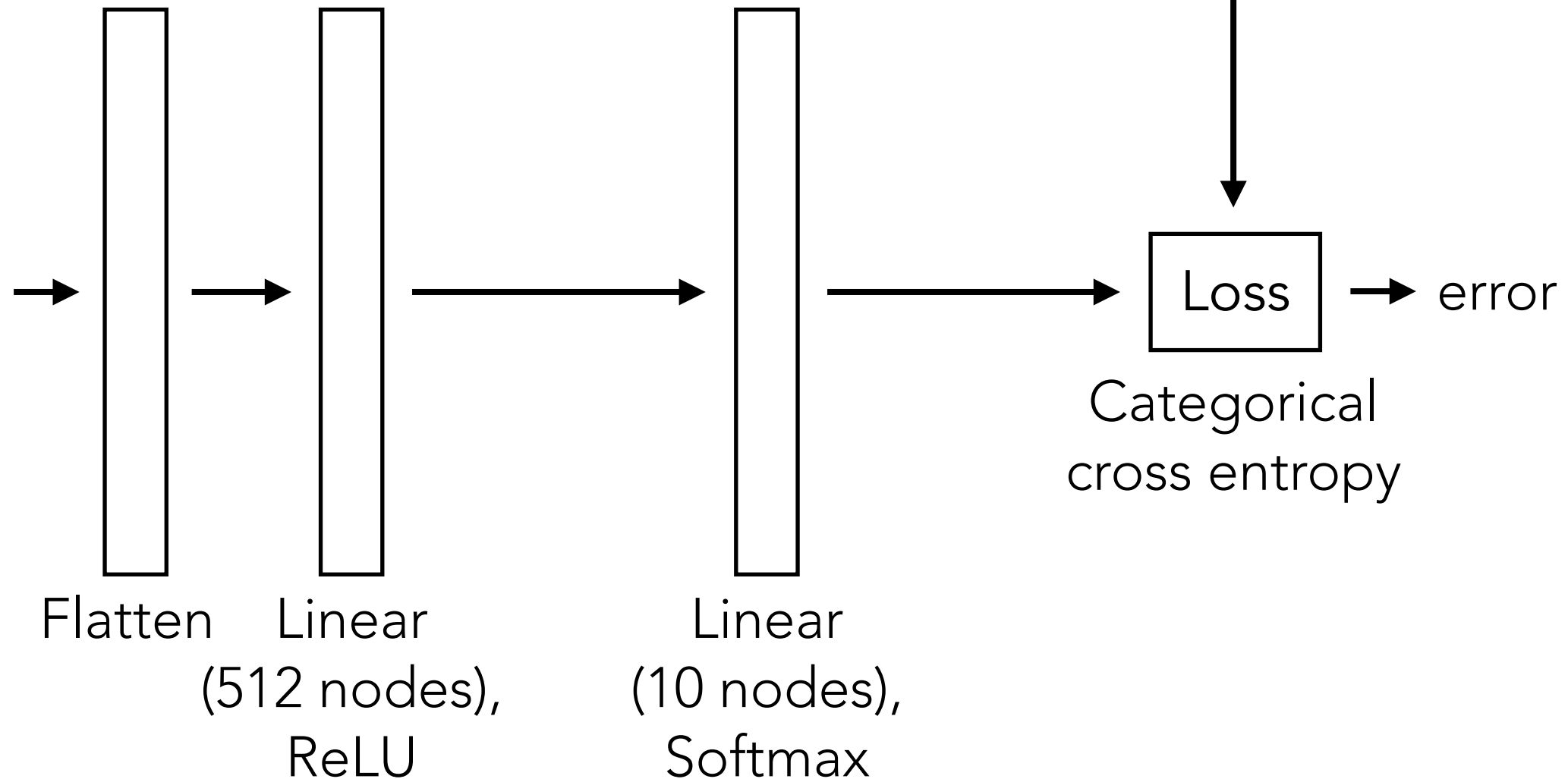


Handwritten Digit Recognition

Training label: 6

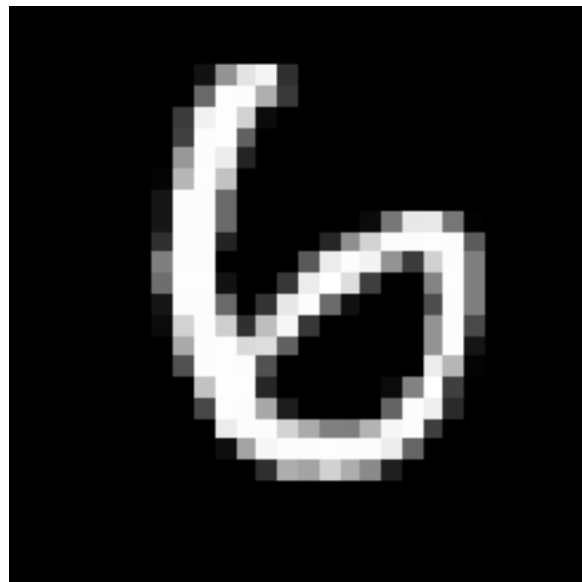


Input

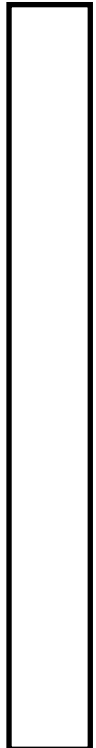


Handwritten Digit Recognition

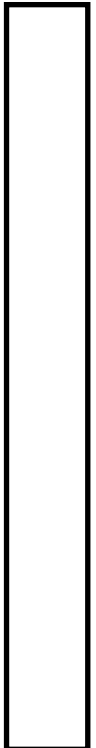
Training label: 6



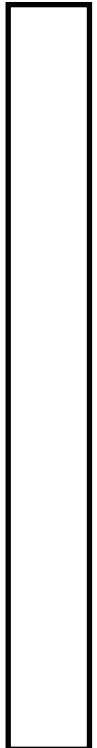
Input

→ 
Conv2d,
ReLU

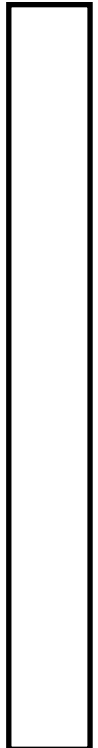



Max
Pool
2d





Flatten

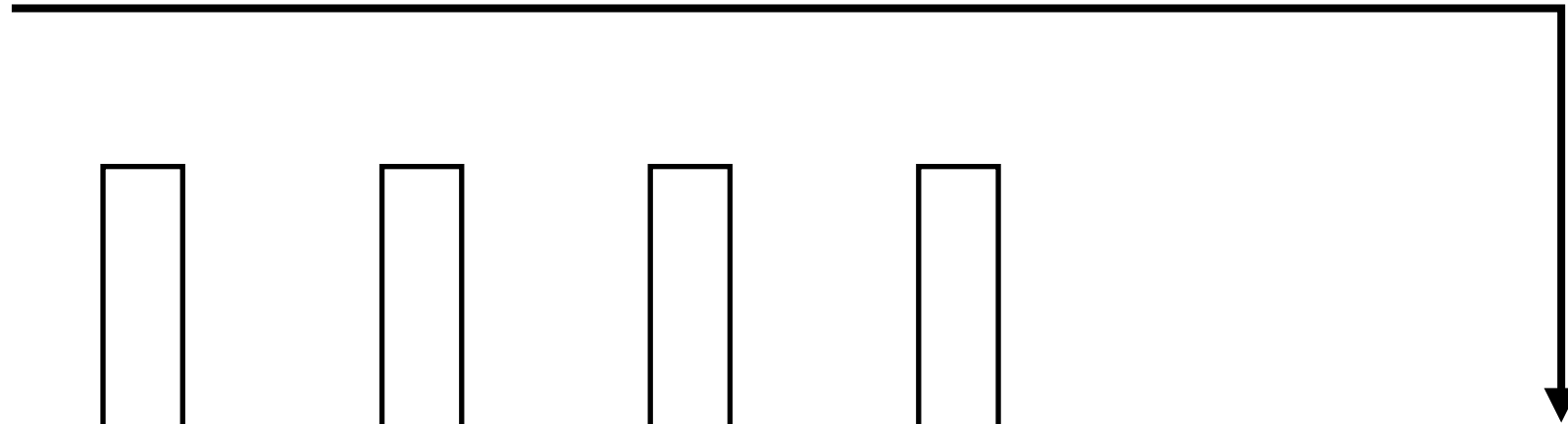



Linear
(10 nodes),
Softmax

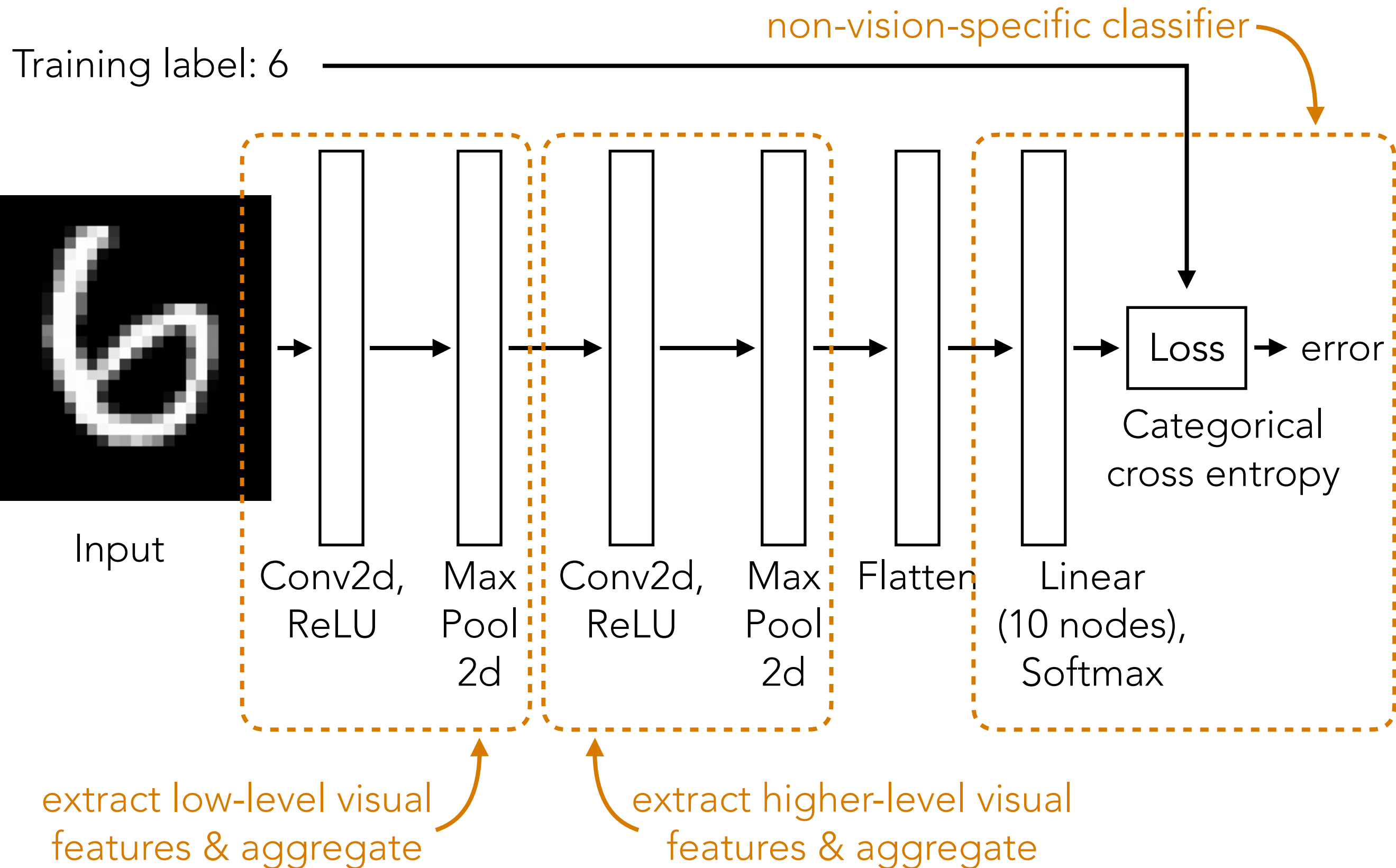



Categorical
cross entropy

→ error



Handwritten Digit Recognition



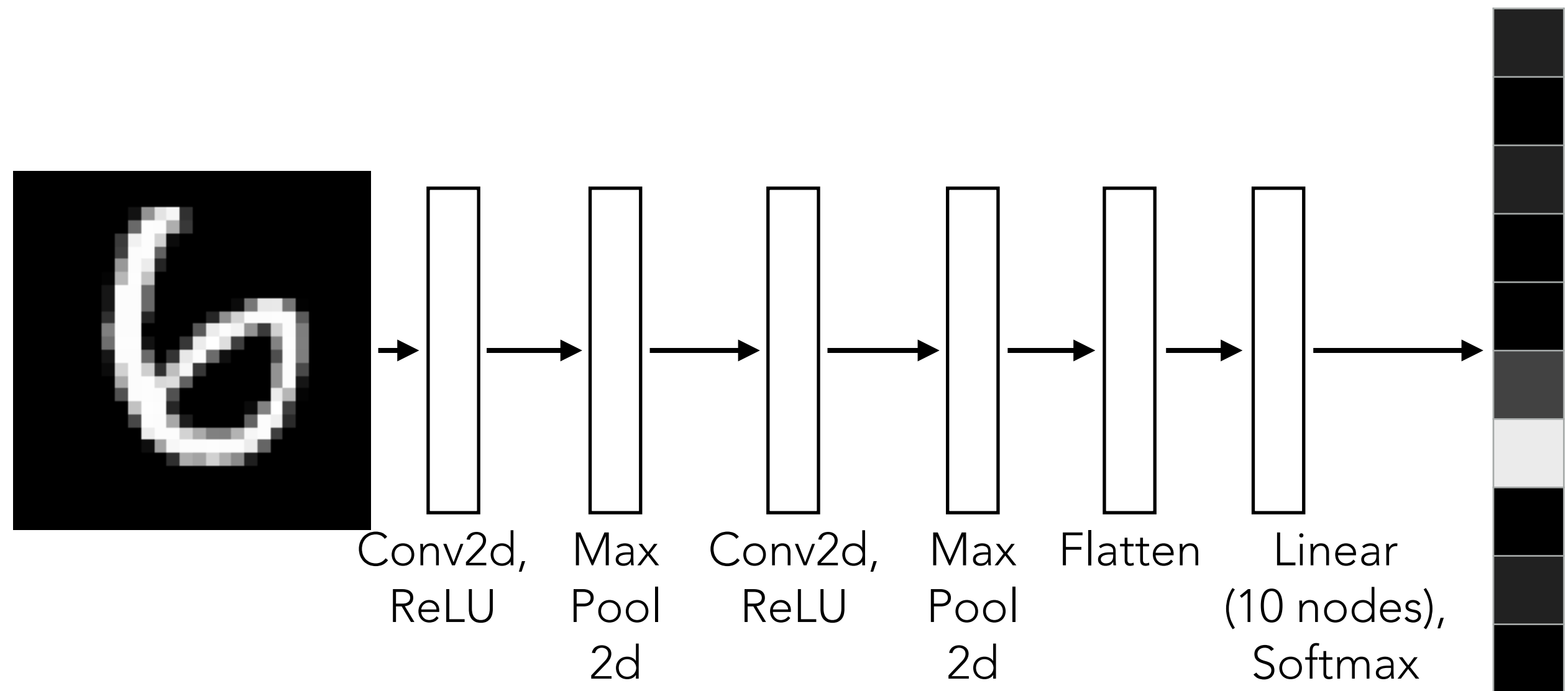
CNNs

Demo

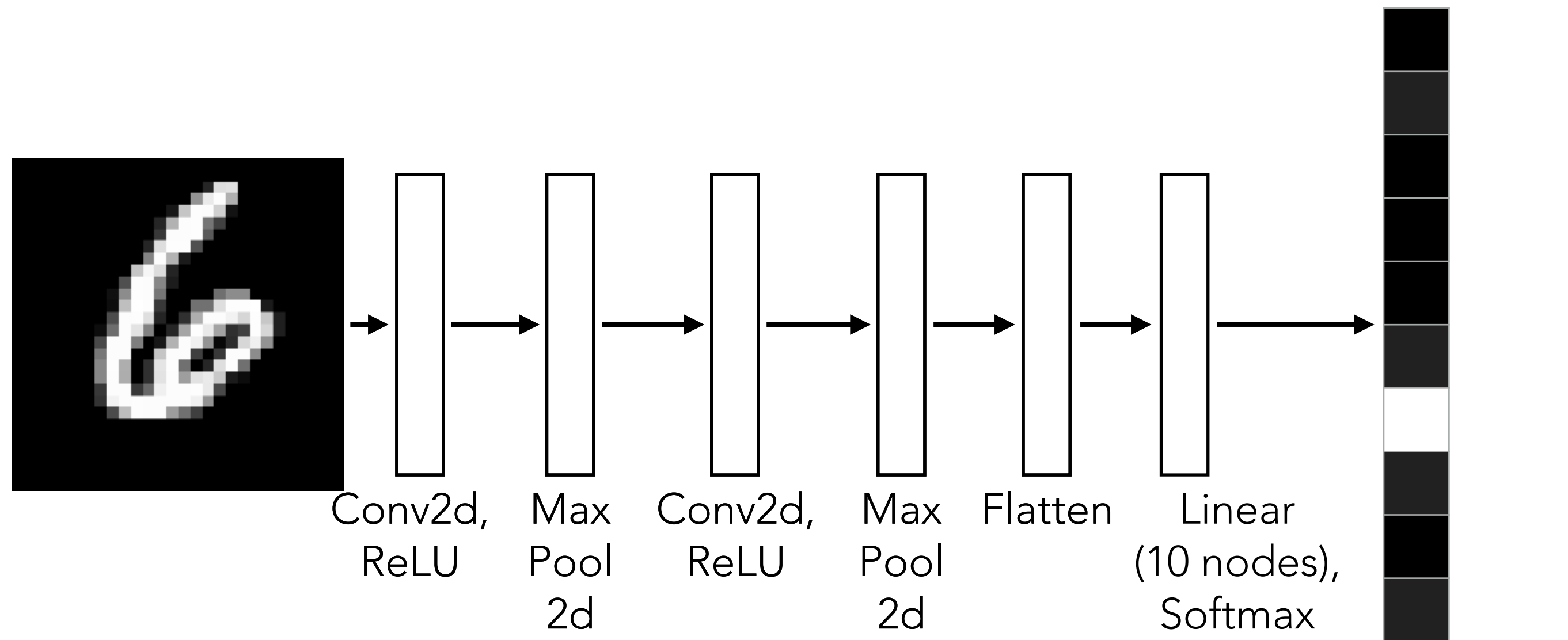
Recap

- A convolution filter processes an input image to produce an output image by taking weighted sums
(examples: blurring an image, finding edges in an image)
- Max pooling produces a *smaller* summary output and is somewhat invariant to small shifts in input “objects”
 - For examples where max pooling fails to achieve this and for a better way to do pooling, see Richard Zhang’s fix for max pooling linked on the course webpage
- Repeat convolution → nonlinear activation → pooling to learn increasingly higher-level features

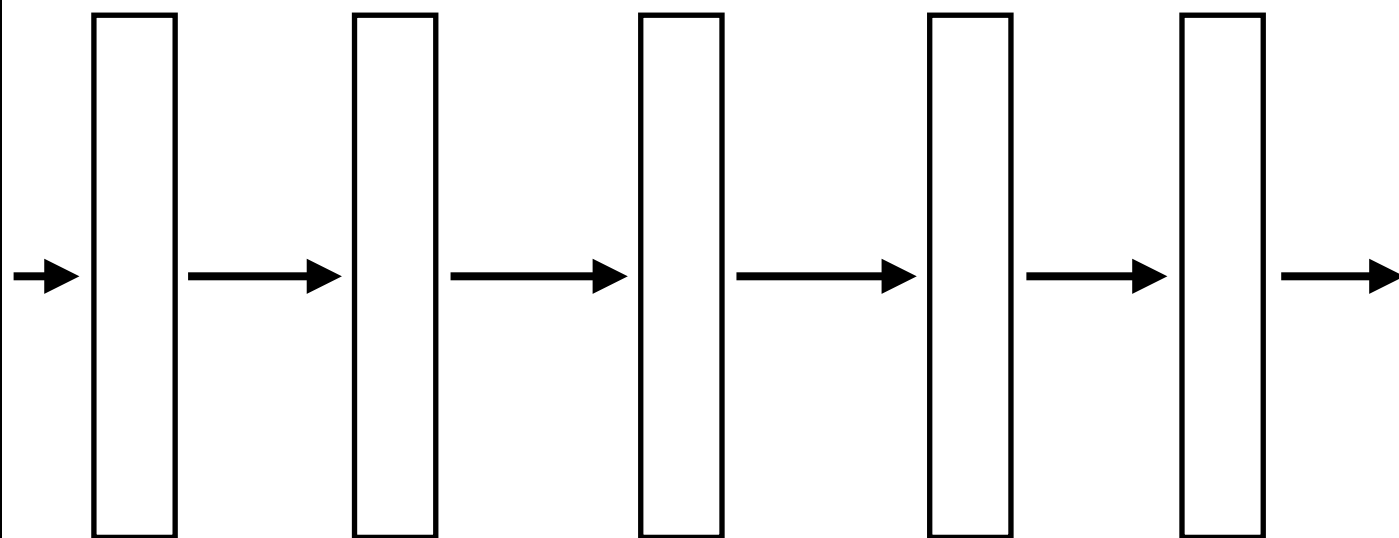
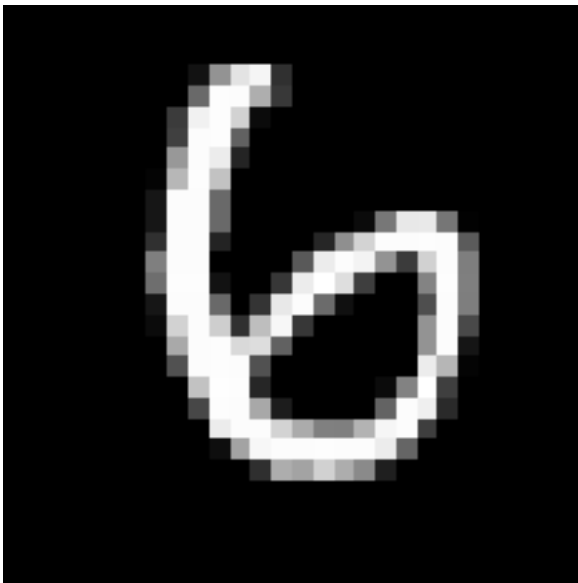
CNNs Encode Semantic Structure for Images



CNNs Encode Semantic Structure for Images

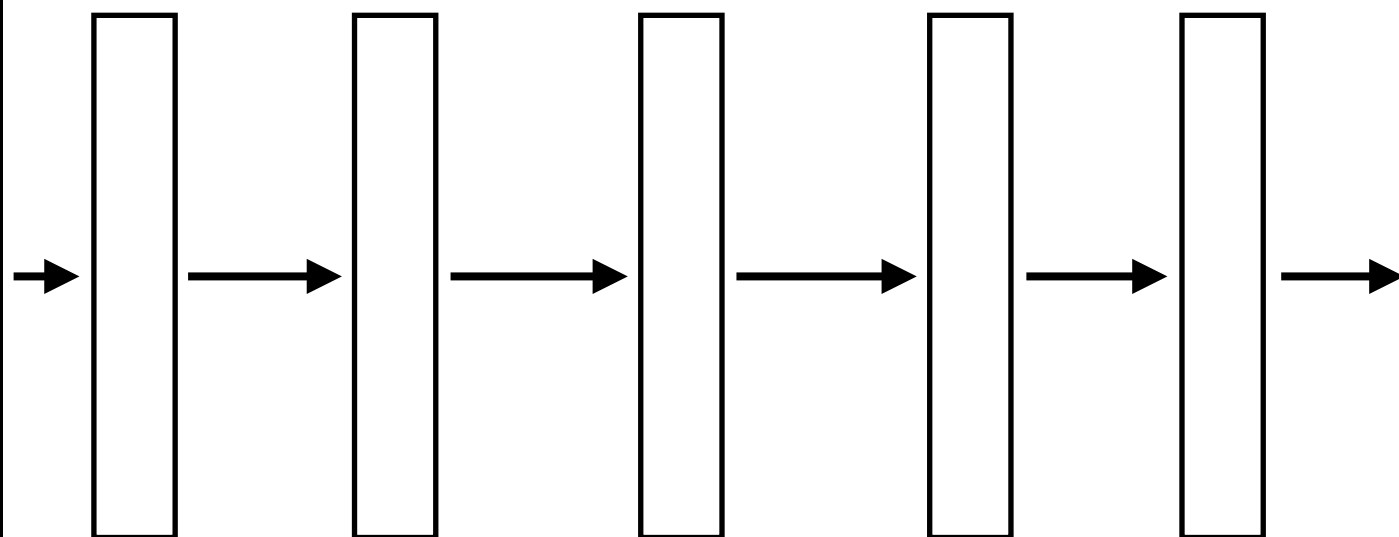
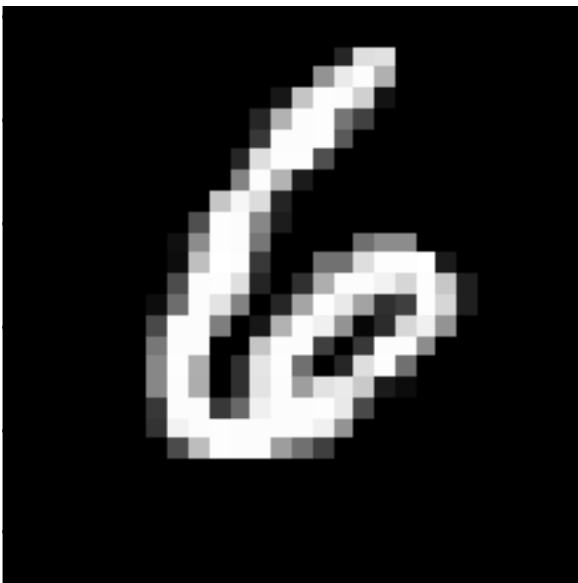


final output for different input
6's is similar



Conv2d, ReLU Max Pool 2d Conv2d, ReLU Max Pool 2d Flatten

actually, intermediate representations close to the last layer are also similar!



Conv2d, ReLU Max Pool 2d Conv2d, ReLU Max Pool 2d Flatten

(intuition: recall the crumpled paper analogy!)

One more PyTorch thing...

Constructing PyTorch Models with `nn.Module`

```
deeper_model = nn.Sequential(nn.Flatten(),
                             nn.Linear(in_features=784, out_features=512),
                             nn.ReLU(),
                             nn.Linear(in_features=512, out_features=10))
```

Another way to write this (we'll need this level of detail for next lecture):

```
class DeeperModel(nn.Module):
    def __init__(self, num_in_features, num_intermediate_features, num_out_features):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear1 = nn.Linear(num_in_features, num_intermediate_features)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(num_intermediate_features, num_out_features)

    def forward(self, inputs):
        flatten_output = self.flatten(inputs)
        linear1_output = self.linear1(flatten_output)
        relu_output = self.relu(linear1_output)
        linear2_output = self.linear2(relu_output)
        return linear2_output

deeper_model = DeeperModel(784, 512, 10)
```