Deep Learning for Analyzing Images and Time Series

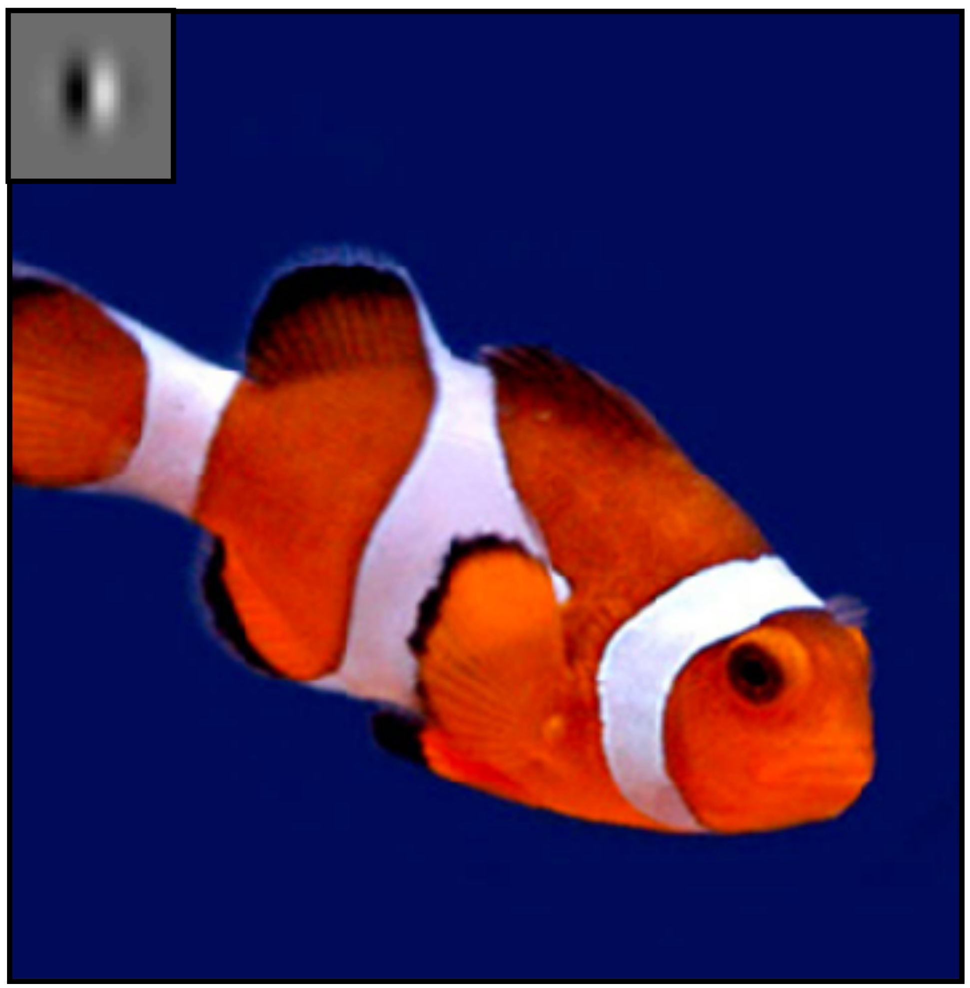
most slides are by George Chen (CMU)

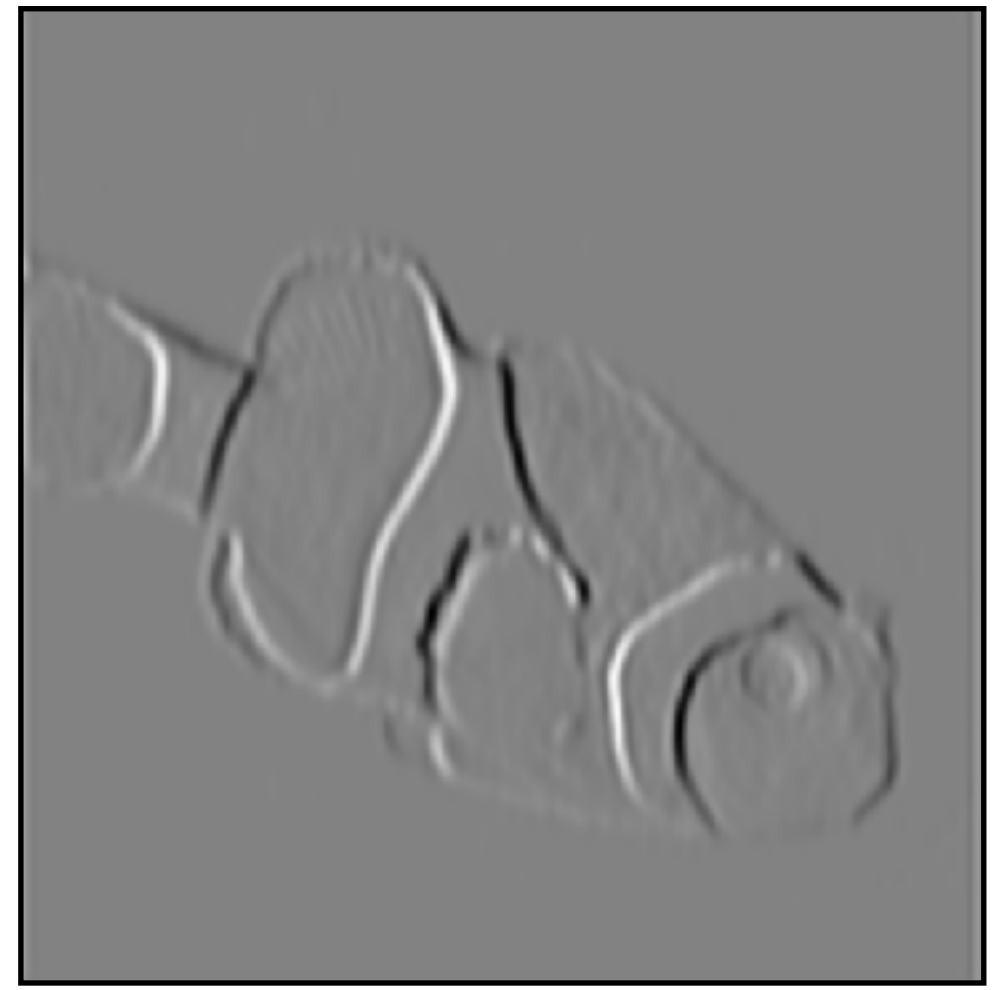
some slides are by Phillip Isola (OpenAI, UC Berkeley)

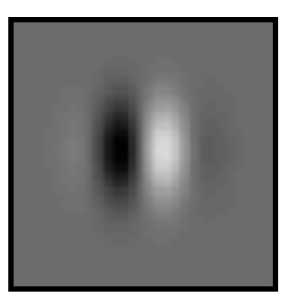
CMU 95-865 Fall 2017

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

Convolution





filter 

Slide by Phillip Isola

Convolution

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

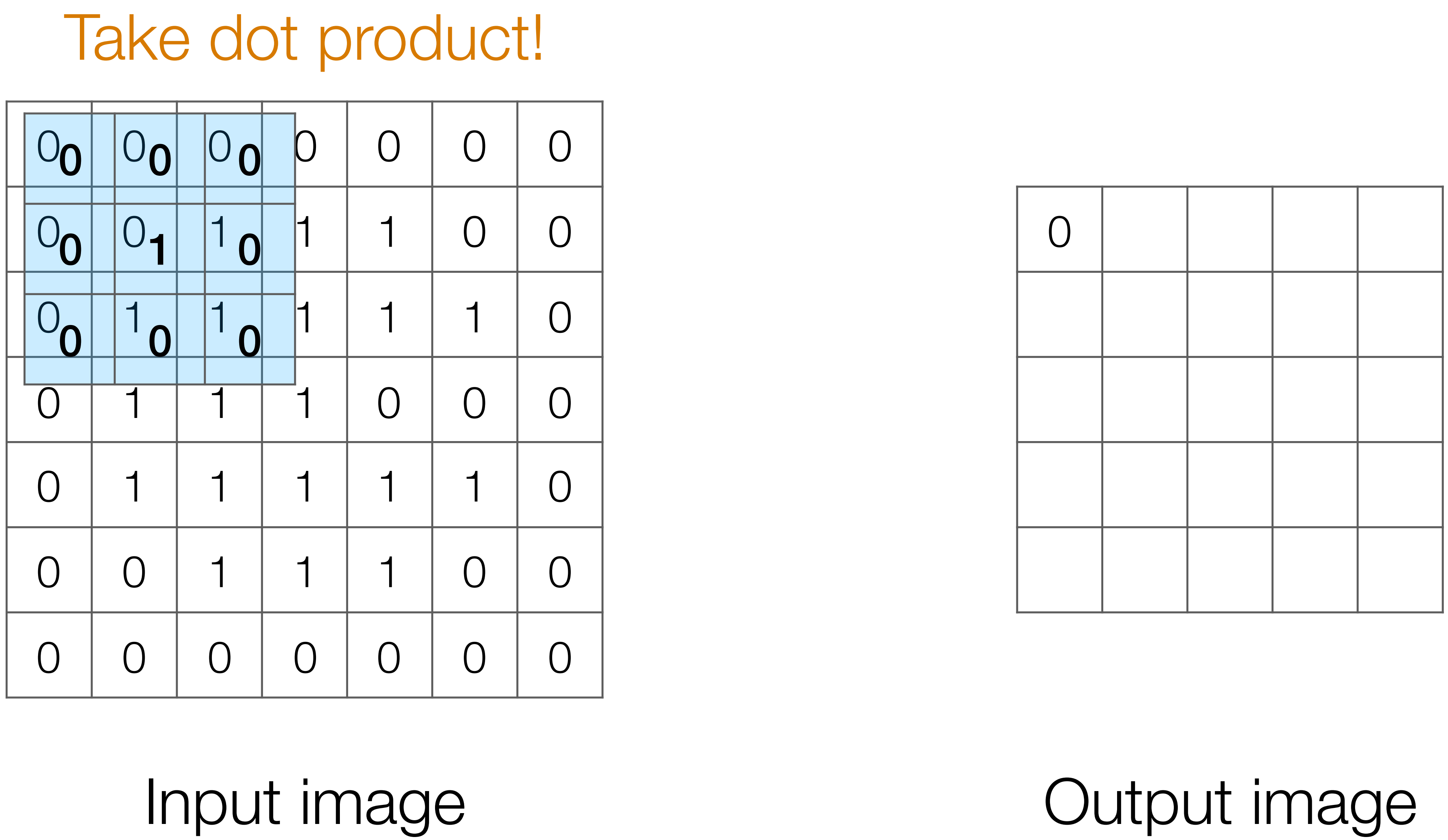
0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0
0	1	1	1	1	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input image

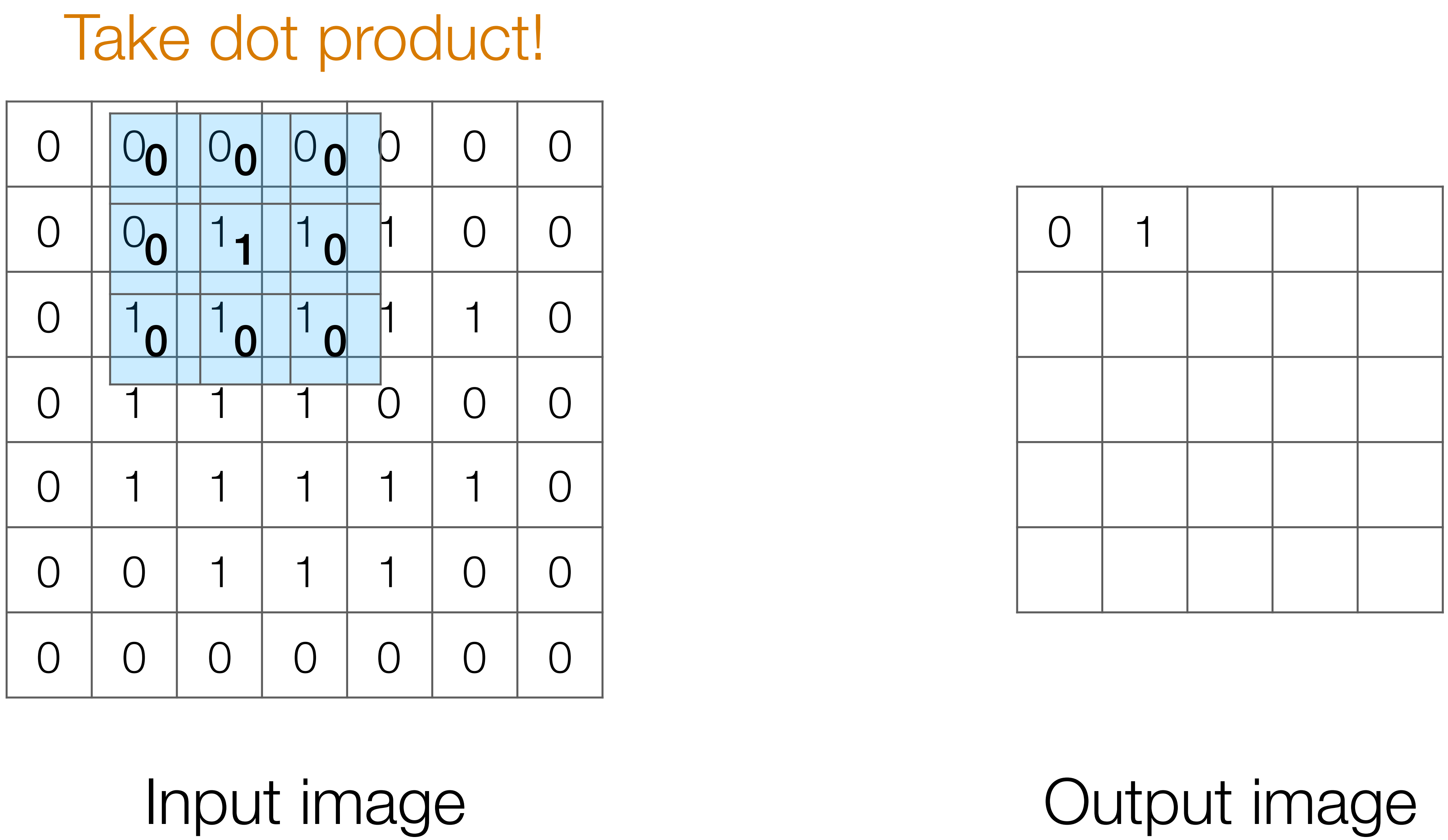
0	0	0
0	1	0
0	0	0

Filter
(also called “kernel”)

Convolution

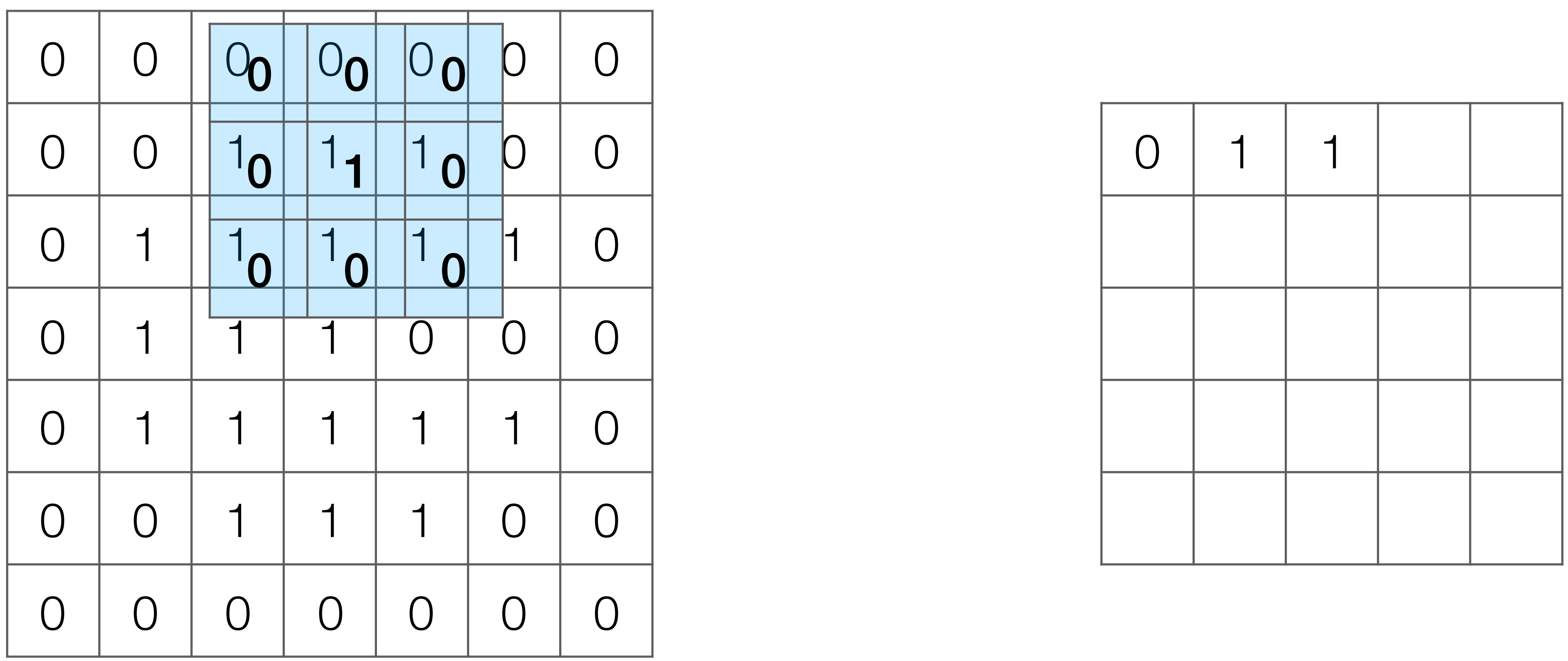


Convolution



Convolution

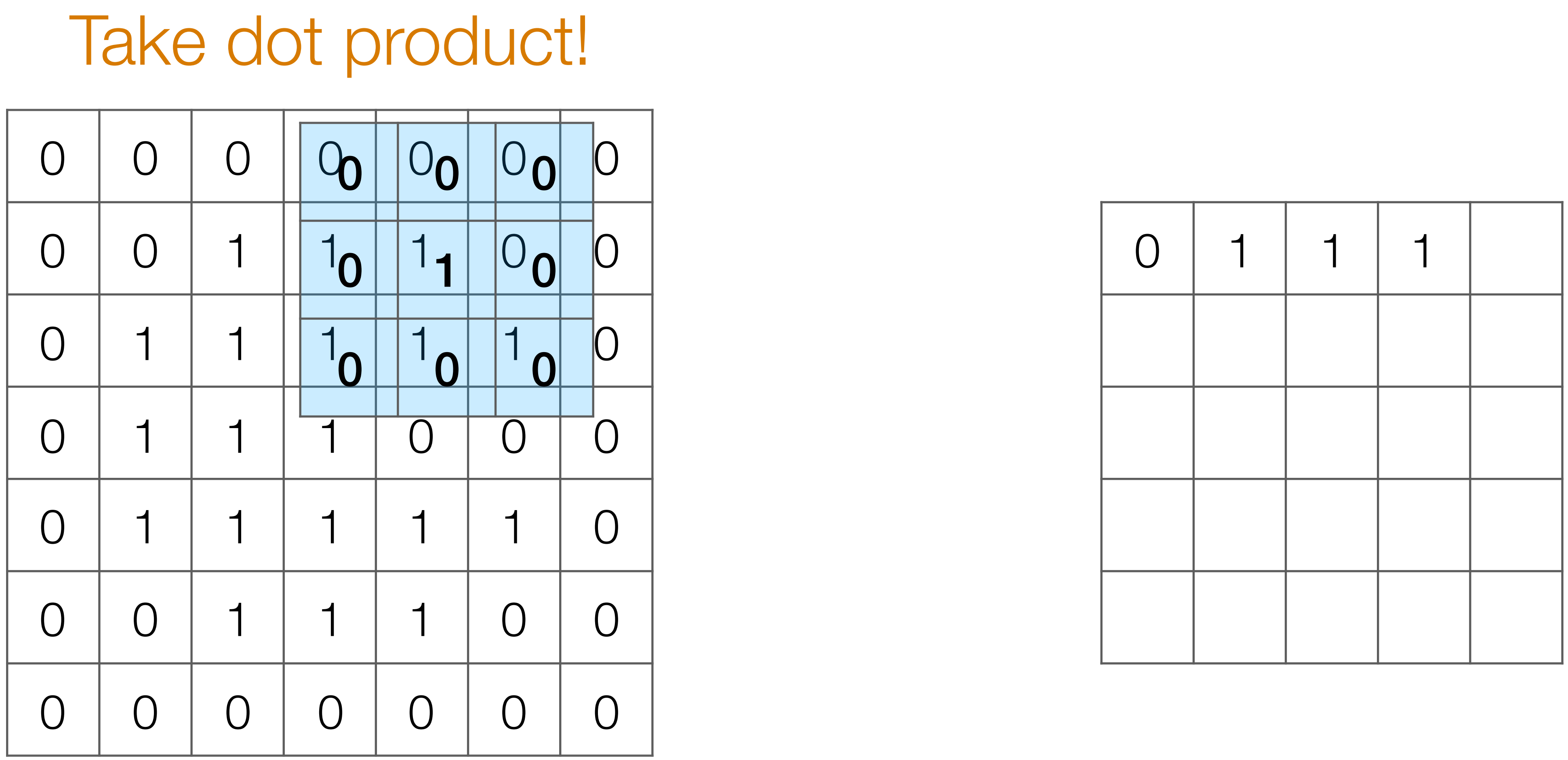
Take dot product!



Input image

Output image

Convolution

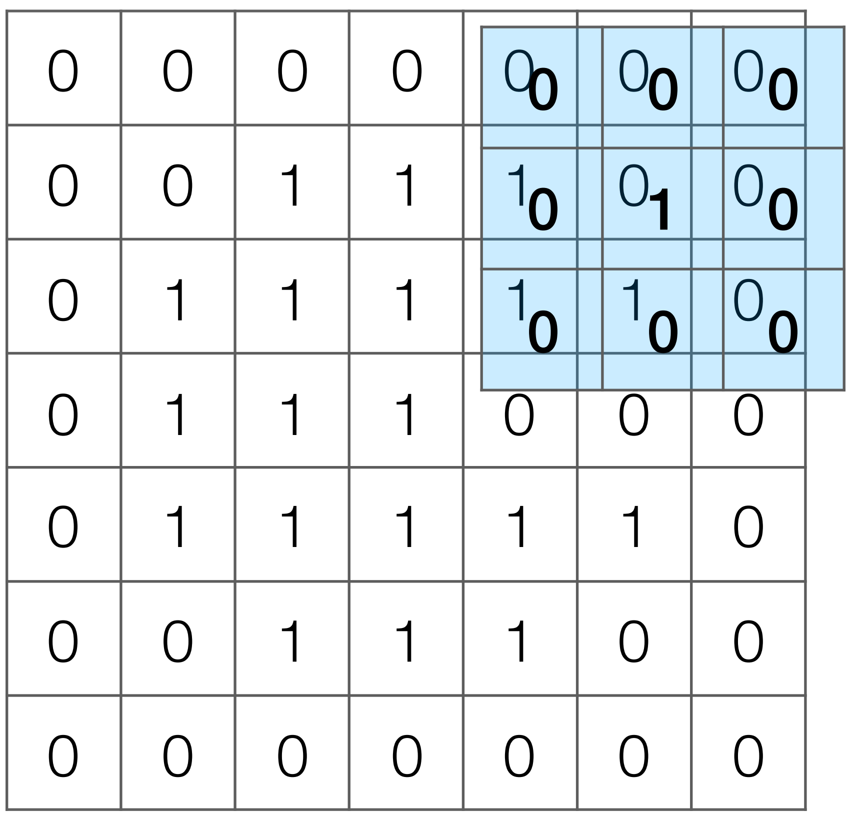


Input image

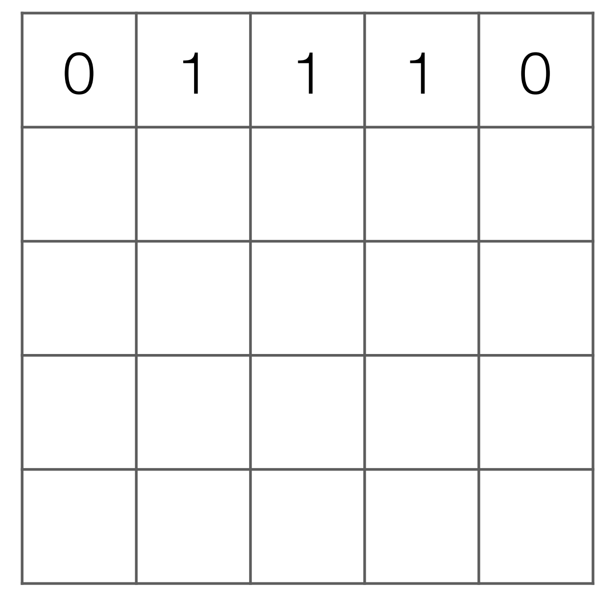
Output image

Convolution

Take dot product!



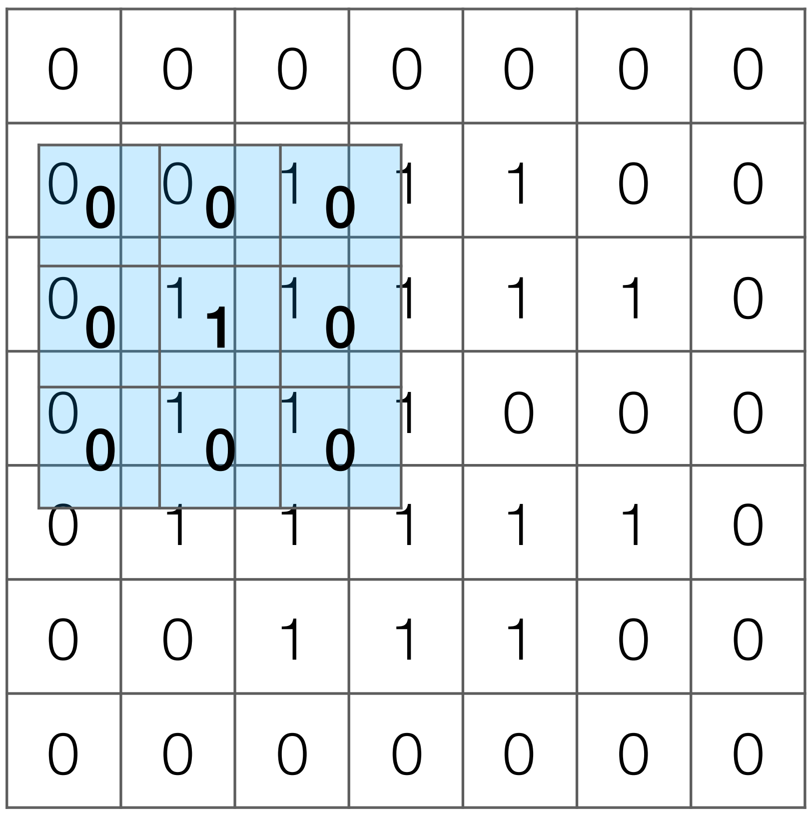
Input image



Output image

Convolution

Take dot product!

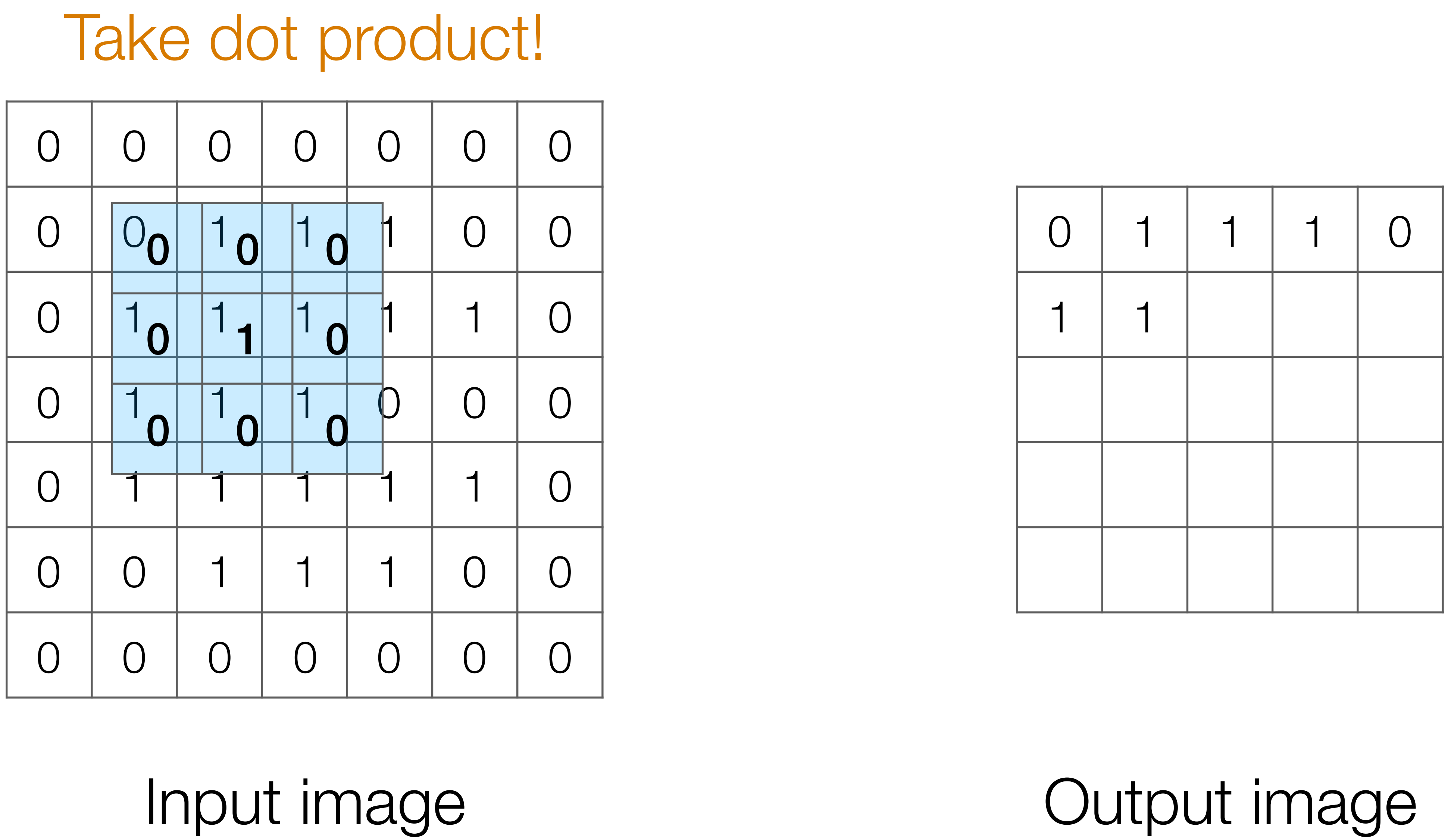


Input image

0	1	1	1	0
1				

Output image

Convolution



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

$$
\begin{array}{|c|c|c|c|c|c|c|c|c|}
\hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline
\mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} \\ \hline
\mathbf{0} & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \mathbf{0} \\ \hline
\mathbf{0} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & \mathbf{0} \\ \hline
\mathbf{0} & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \mathbf{0} \\ \hline
\mathbf{0} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & \mathbf{0} \\ \hline
\mathbf{0} & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \mathbf{0} \\ \hline
\mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} \\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline
\end{array}
\;*\;
\begin{array}{|c|c|c|}
\hline
\mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline
\mathbf{0} & \mathbf{1} & \mathbf{0} \\ \hline
\mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline
\end{array}
\;=\;
\begin{array}{|c|c|c|c|c|c|c|}
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline
0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline
0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline
0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline
0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline
\end{array}
$$

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

*

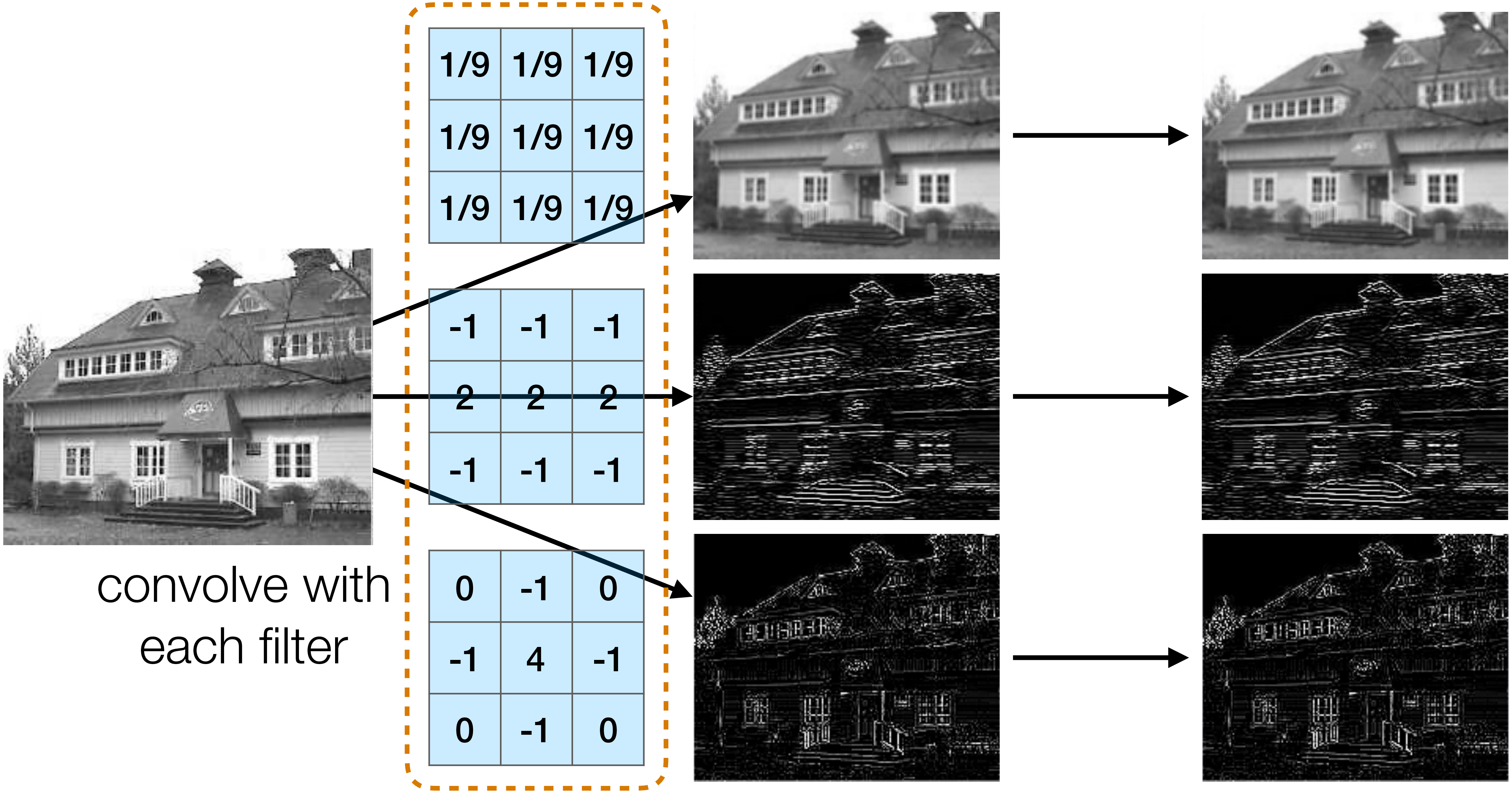
-1	-1	-1
2	2	2
-1	-1	-1

=

(this example finds horizontal edges)

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

Convolution Layer

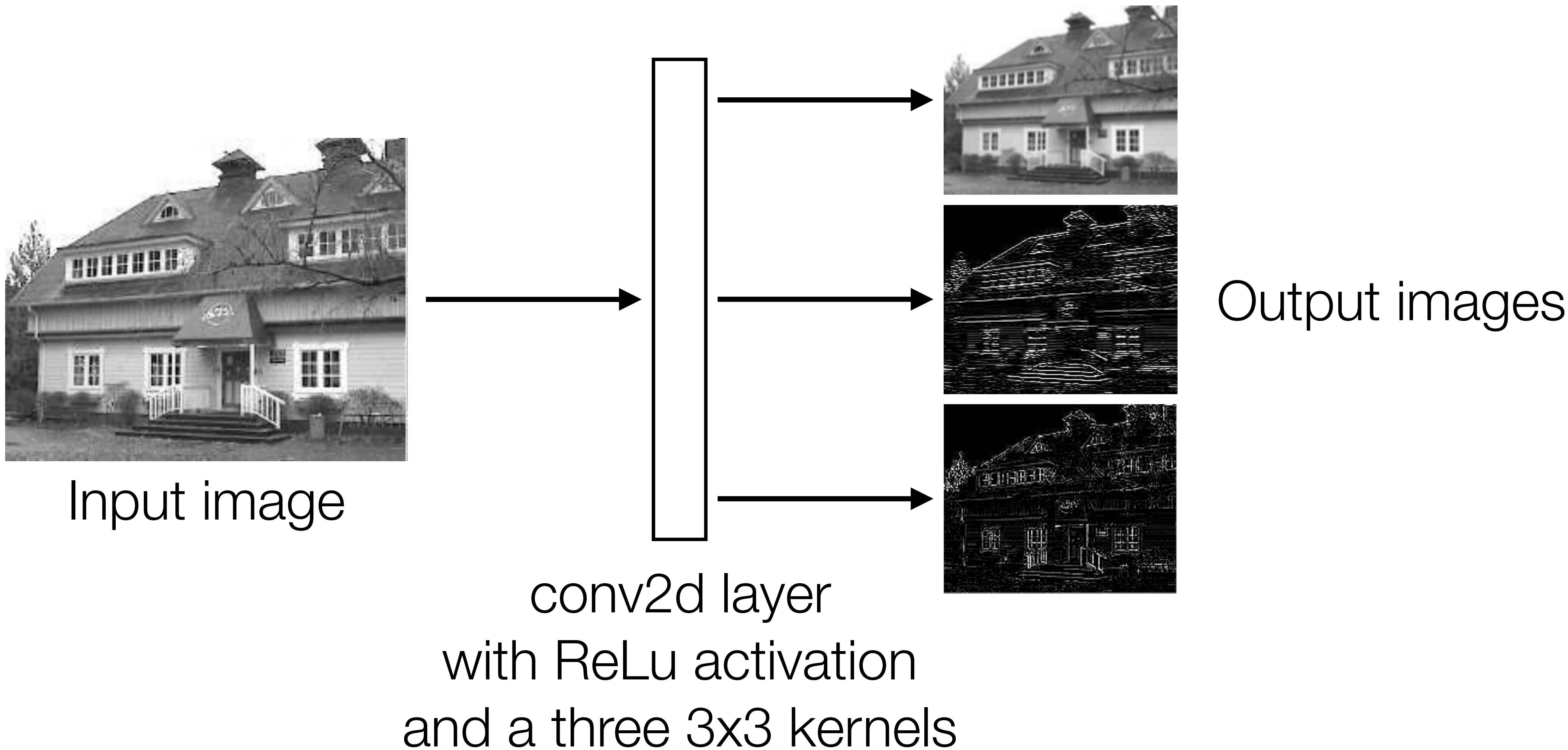


filters are actually unknown and are learned!

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

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

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)

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

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

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

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?

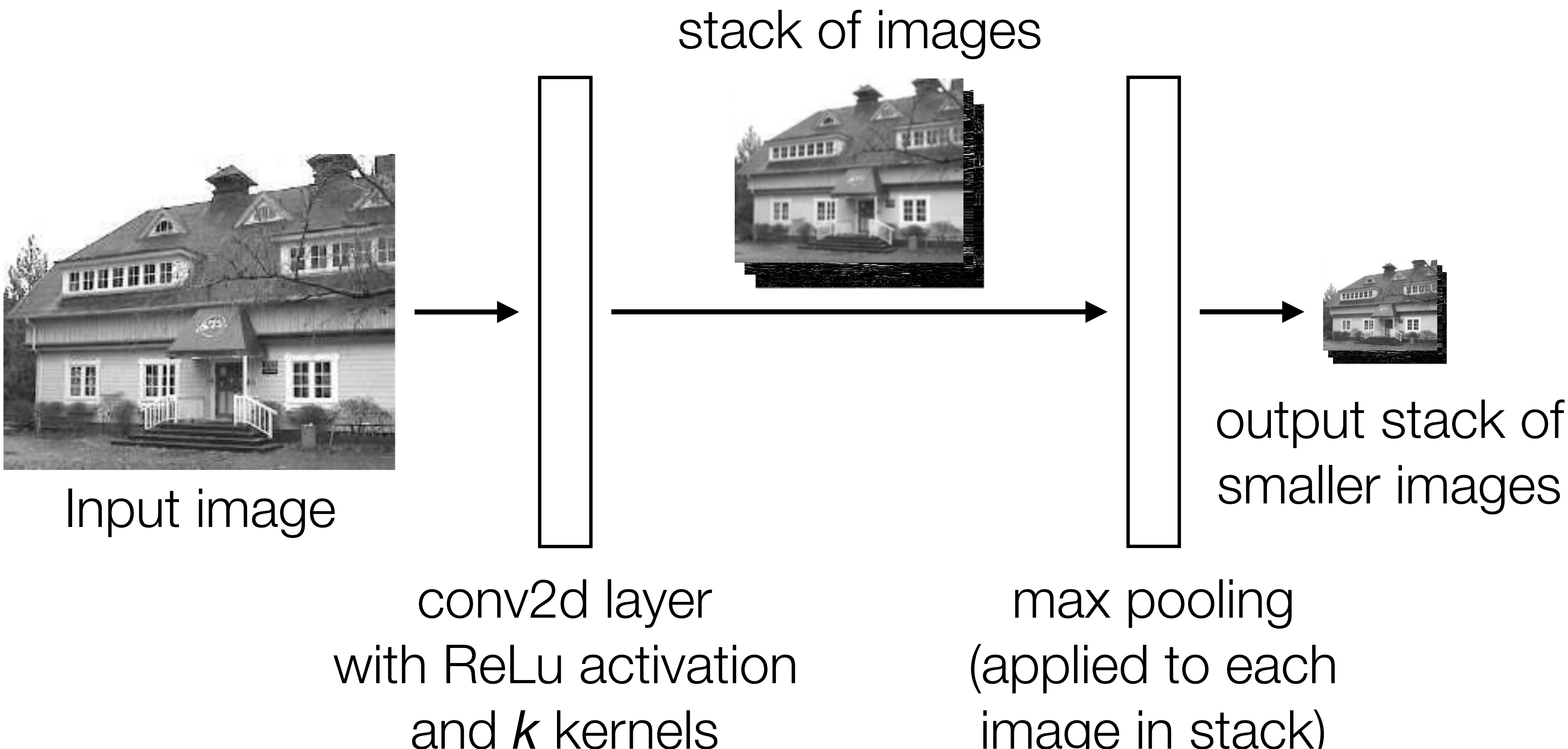
In this example: 1 pixel in max pooling output captures information from 16 input pixels!

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

1	3
1	3

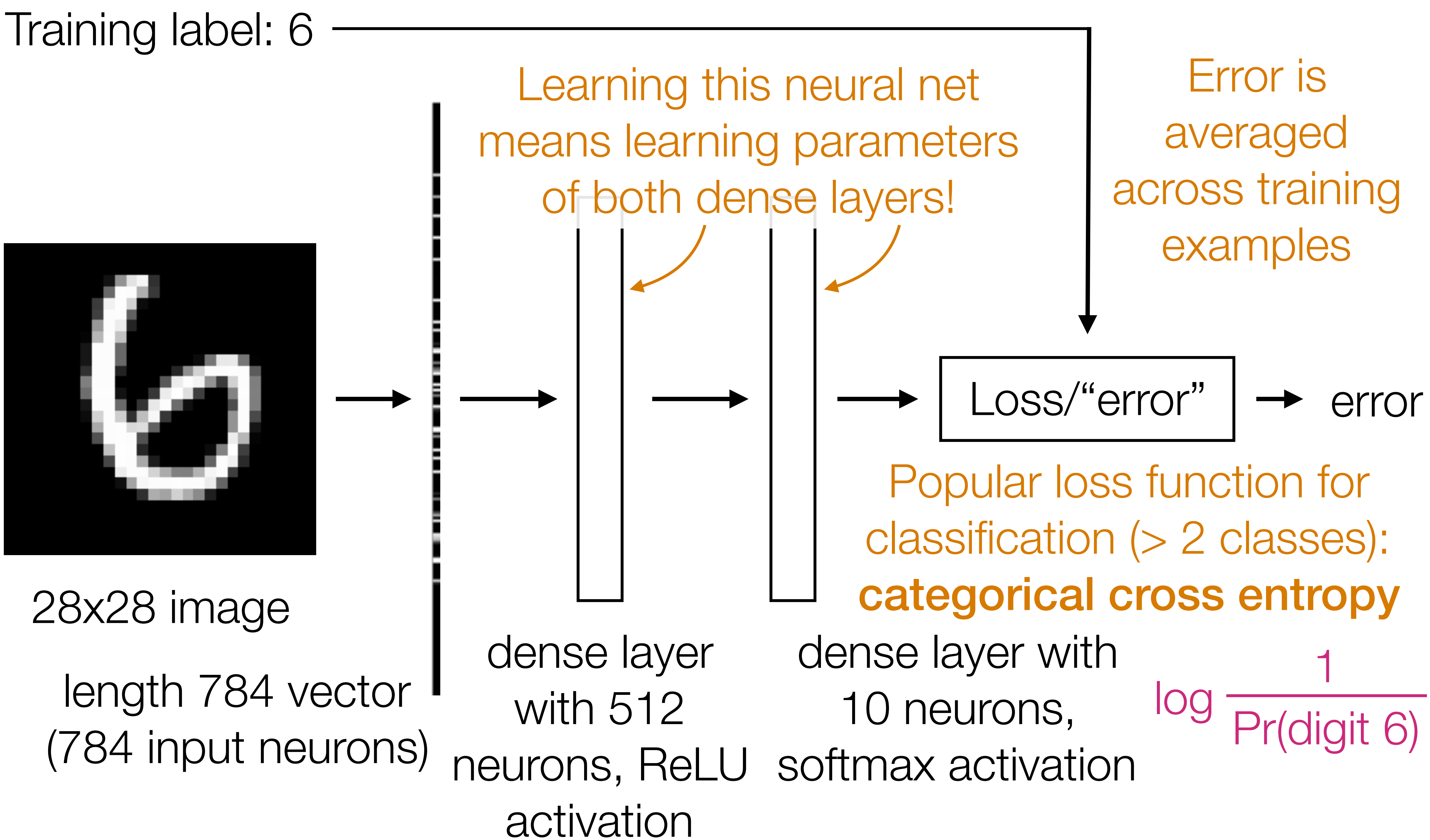
Output after max pooling

Basic Building Block of CNN’s

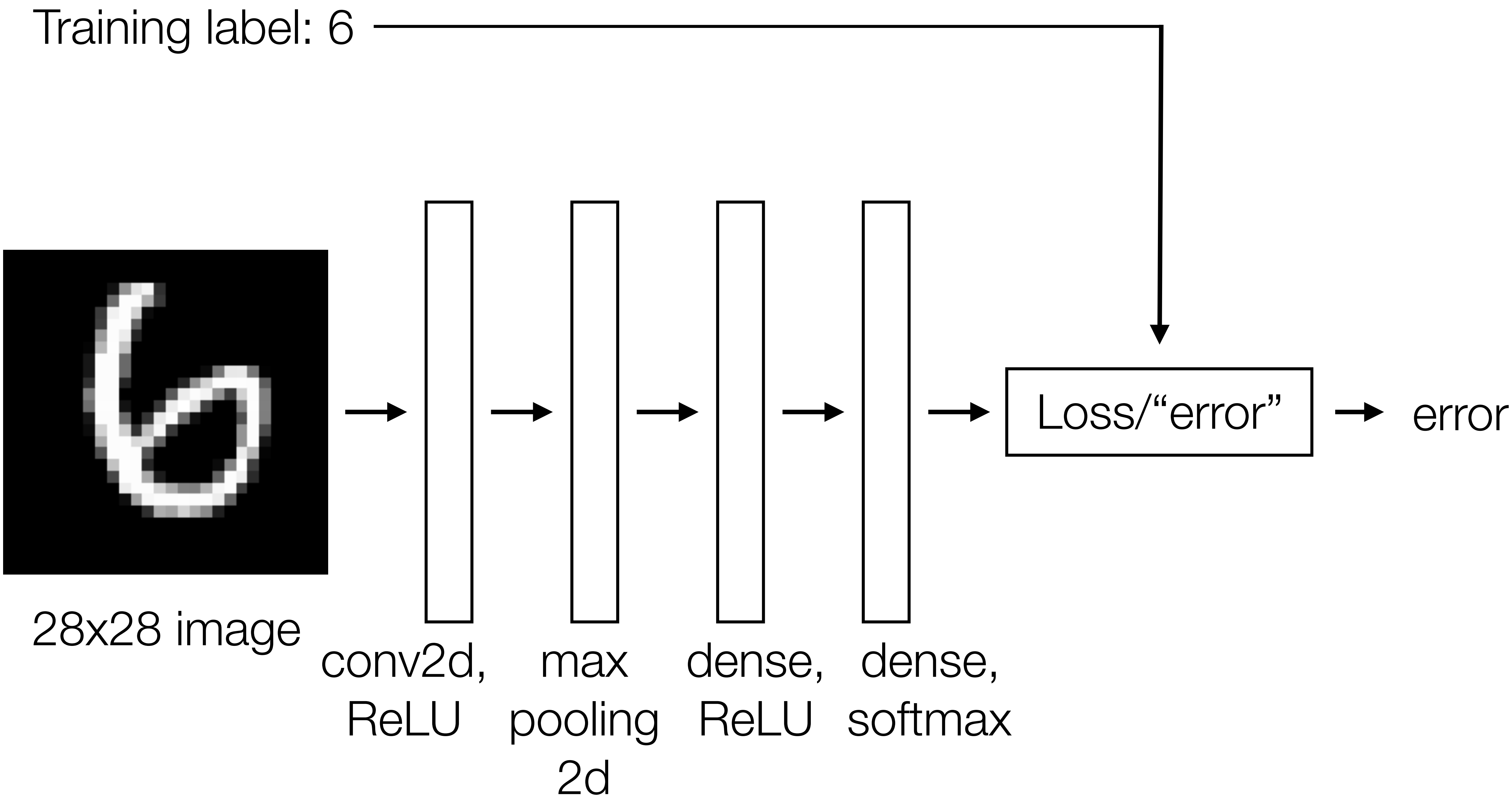


Images from: http://aishack.in/tutorials/image-convolution-examples/

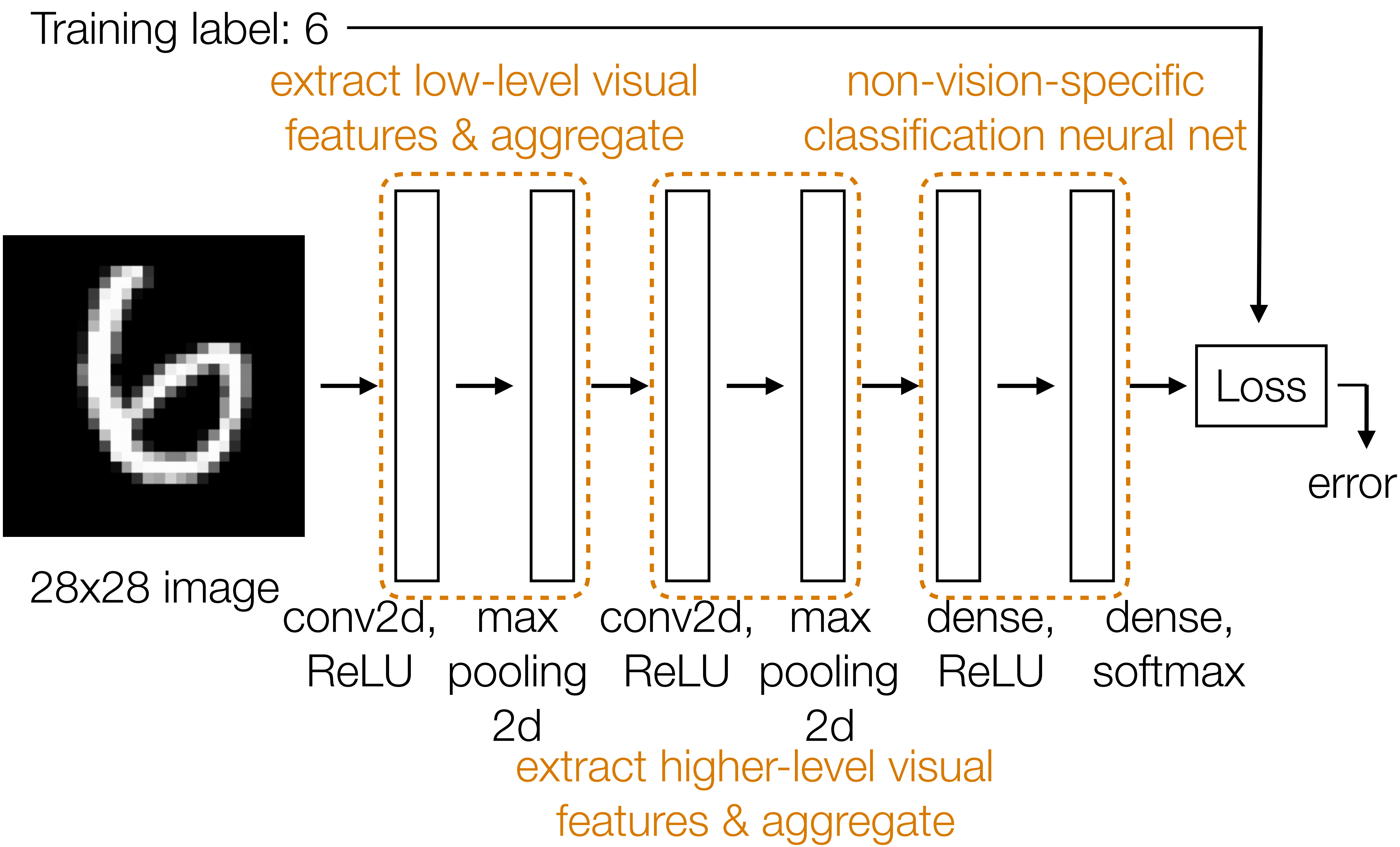
Handwritten Digit Recognition



Handwritten Digit Recognition



Handwritten Digit Recognition



CNN Demo

CNN's

- Learn convolution filters for extracting simple features

- Max pooling aggregates local information

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

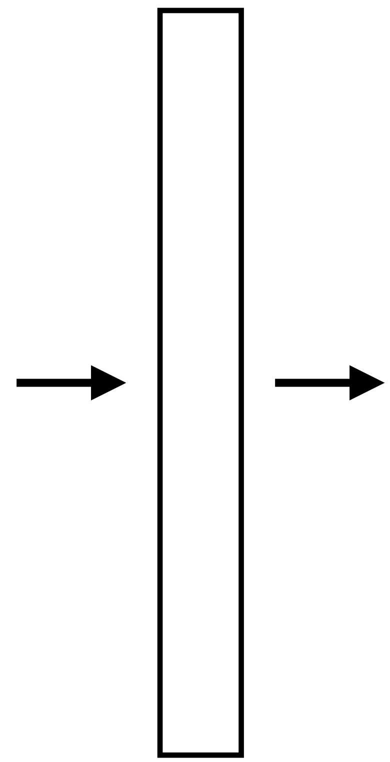
- Convolution filters are shift-invariant

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

Recurrent Neural Networks (RNNs)

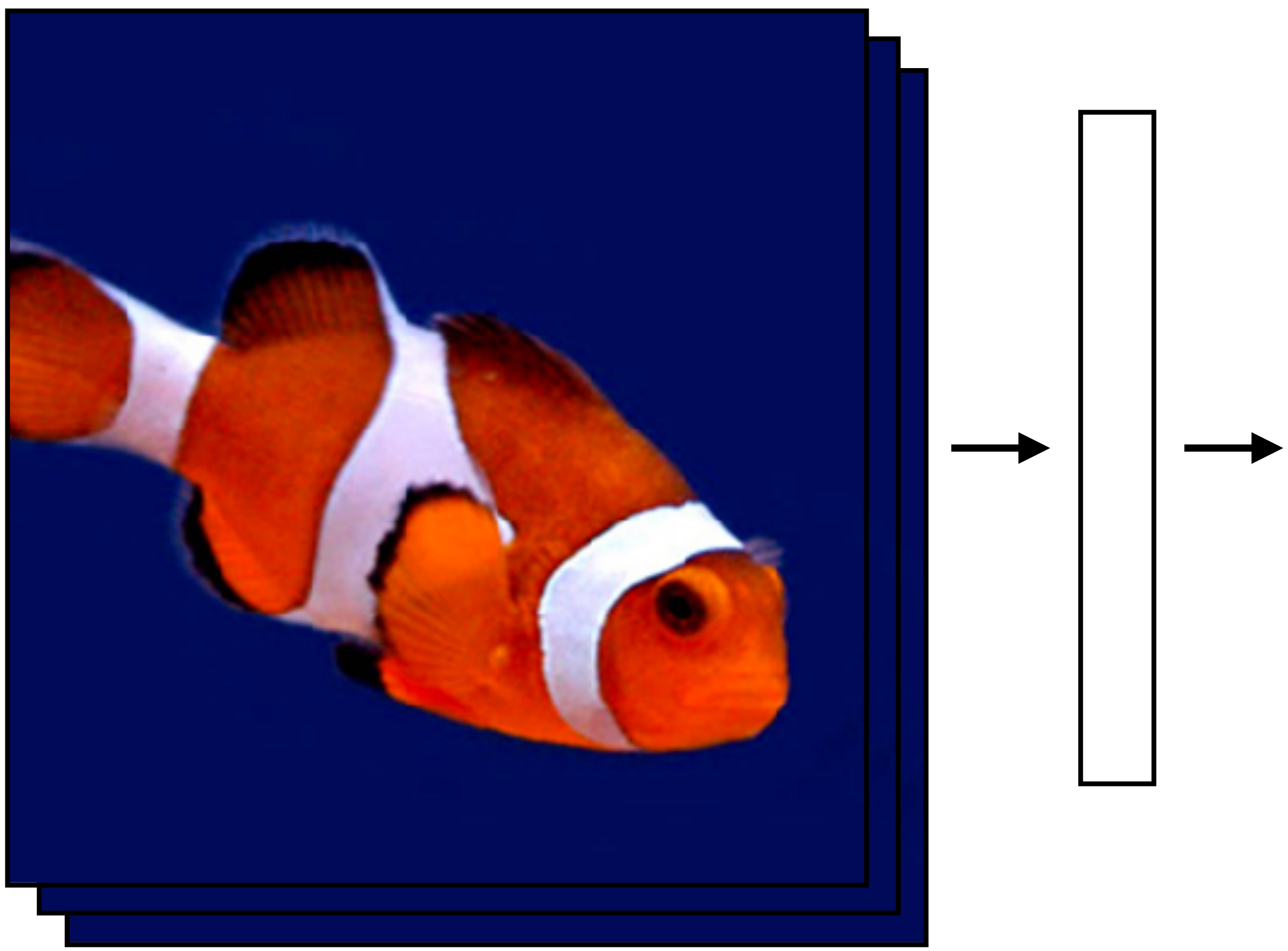
RNNs

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



RNNs

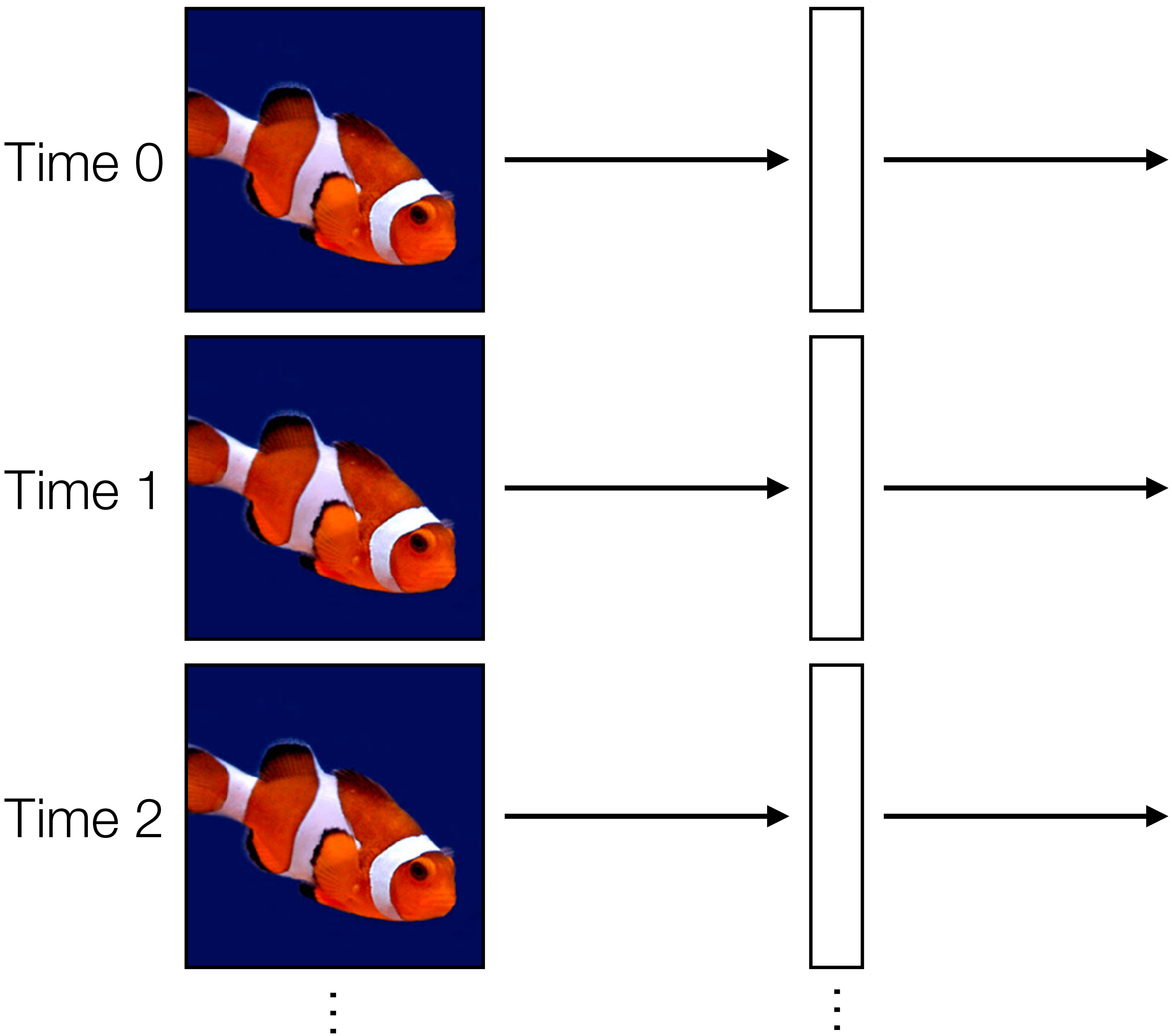
What we've seen so far are "feedforward" NNs



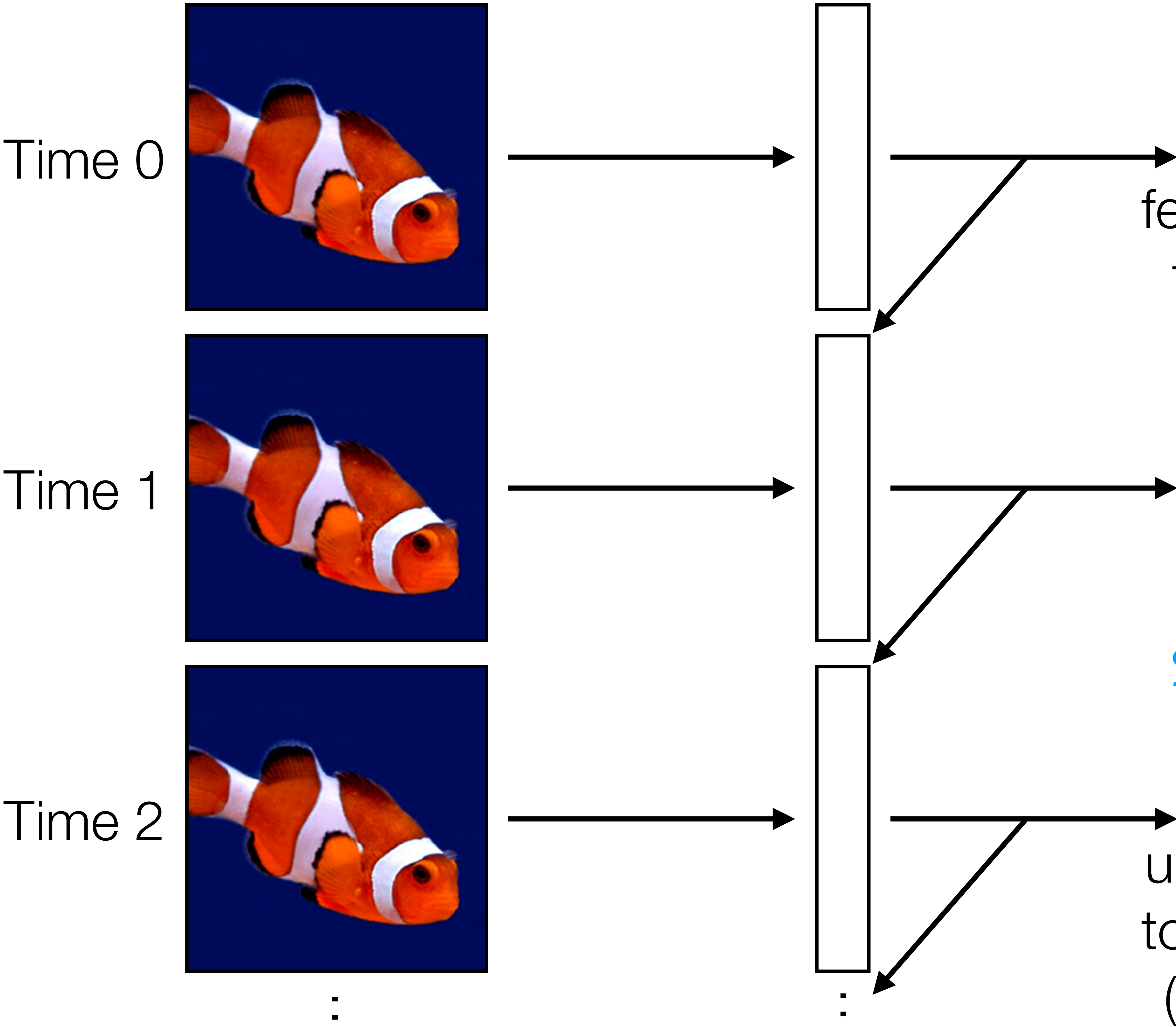
What if we had a video?

RNNs

Feedforward NN's:
treat each video frame
separately



RNNs



Feedforward NN's:
treat each video frame separately

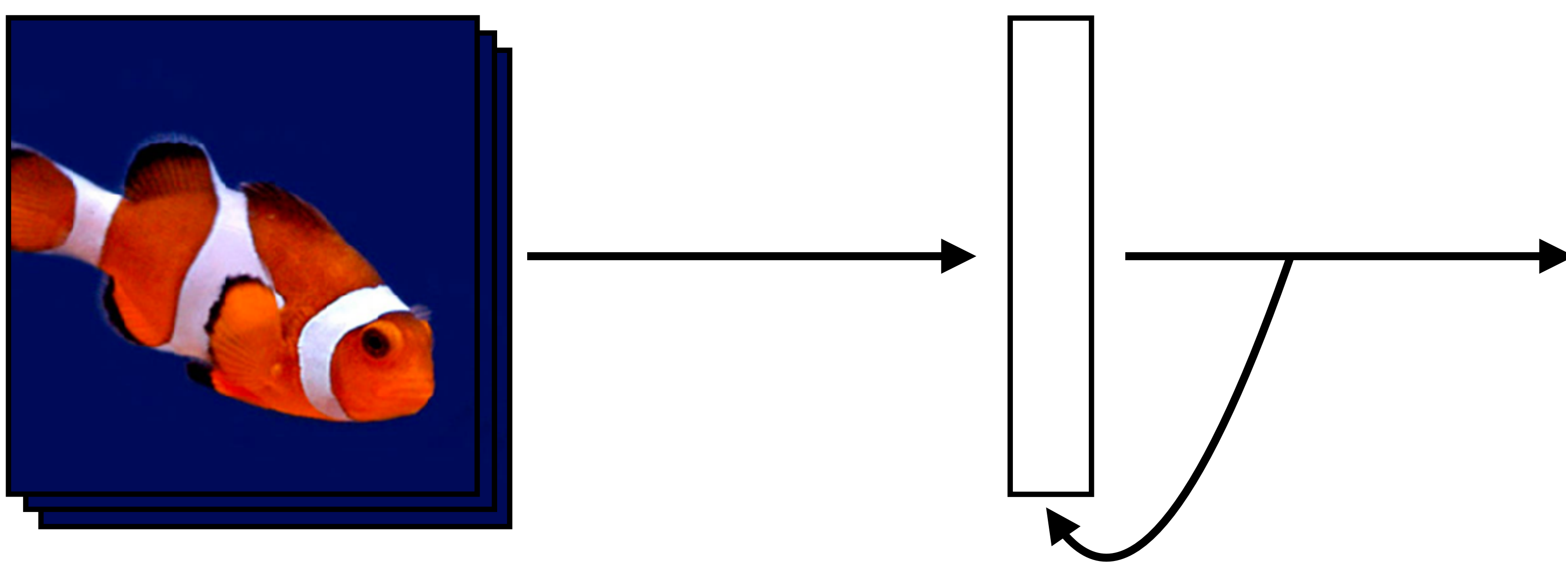
RNN's:
feed output at previous time step as input to RNN layer at current time step

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

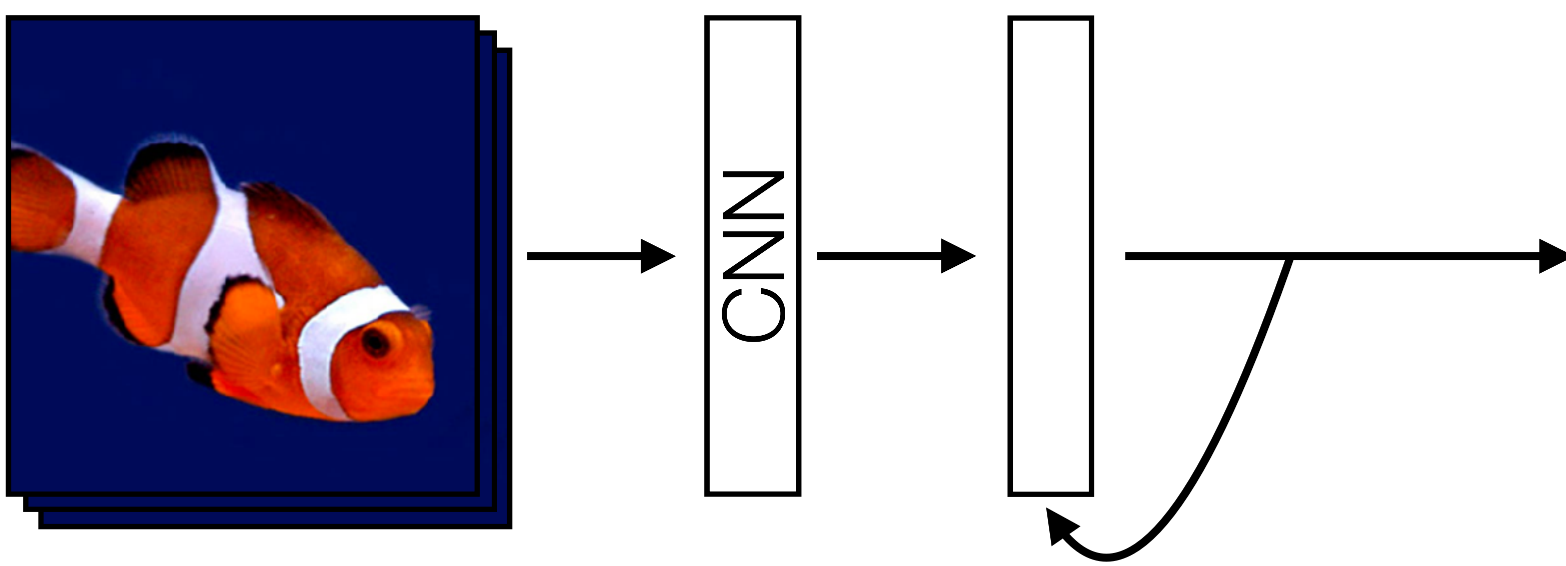
RNN's: feed output at previous time step as input to RNN layer at current time step

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

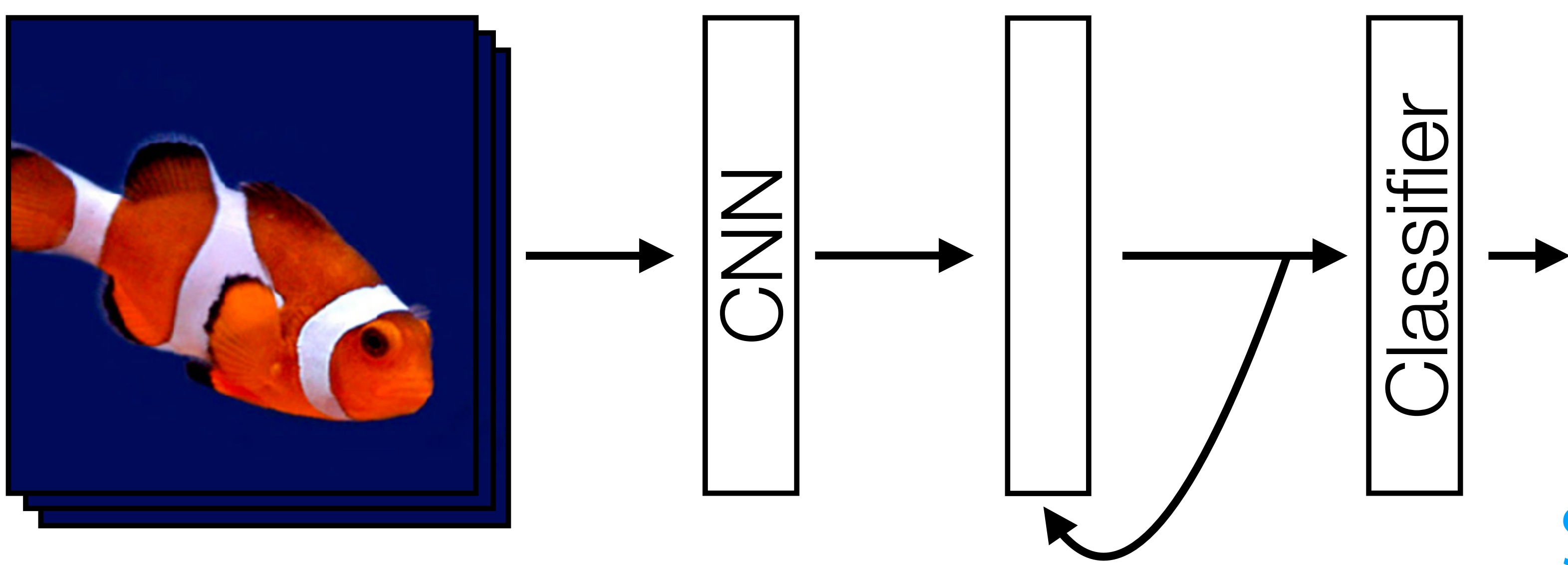
RNN's:
feed output at previous time step as input to RNN layer at current time step

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



like a dense layer that has memory

Feedforward NN's: treat each video frame separately

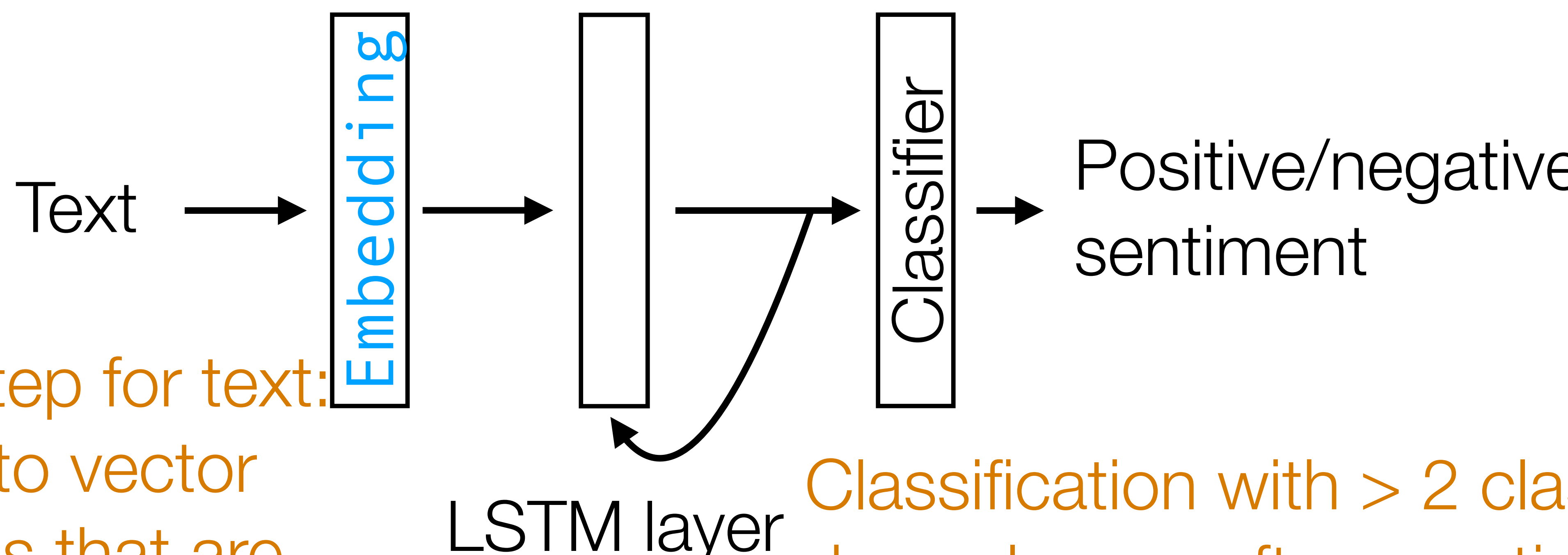
RNN's: feed output at previous time step as input to RNN layer at current time step

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)



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

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

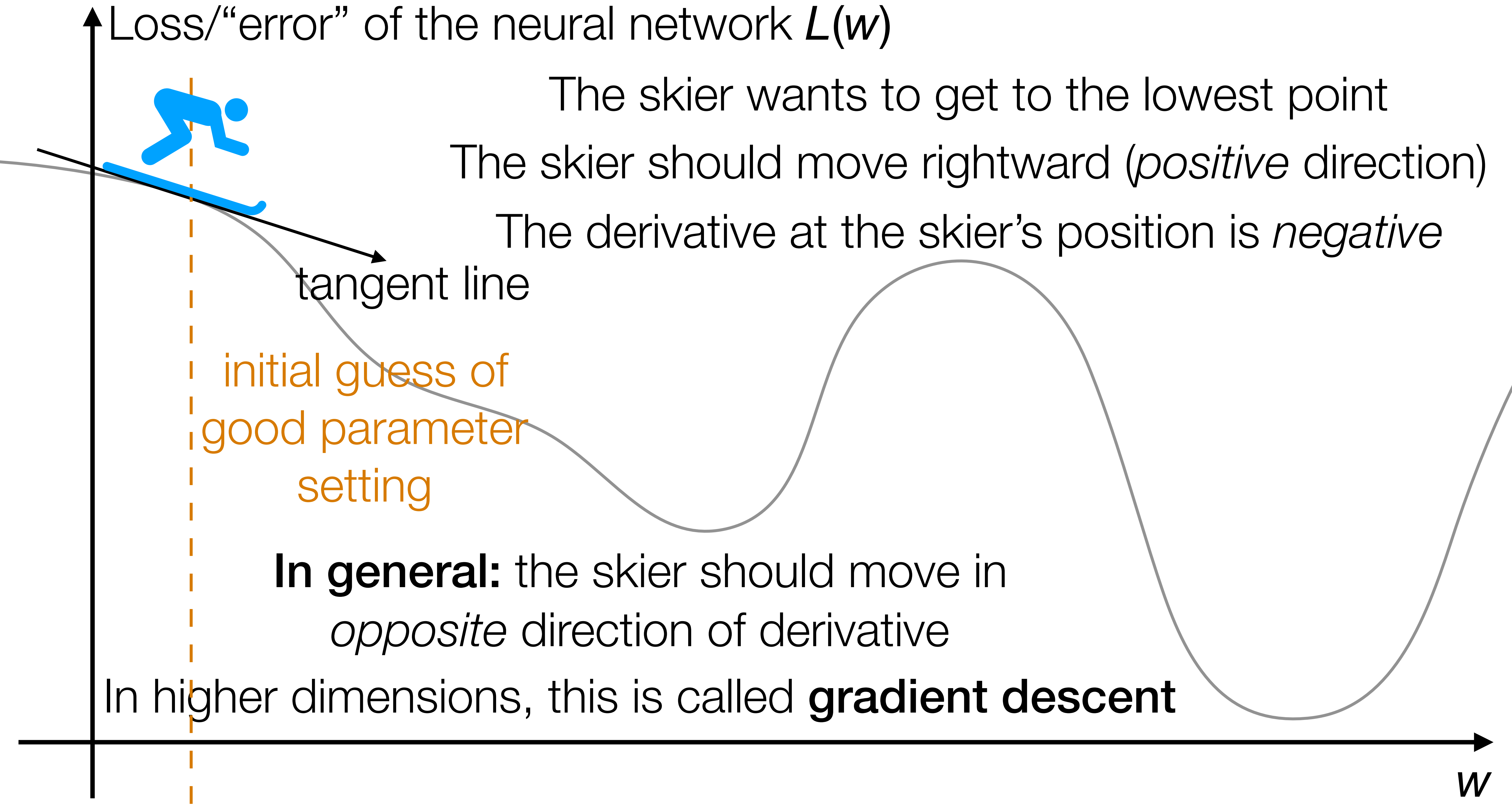
    - If time series doesn't actually have global structure, performance gain from using RNNs could be little compared to using 1D CNNs

- 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)

Learning a Deep Net

Learning a Deep Net

Suppose the neural network has a single real number parameter *w*

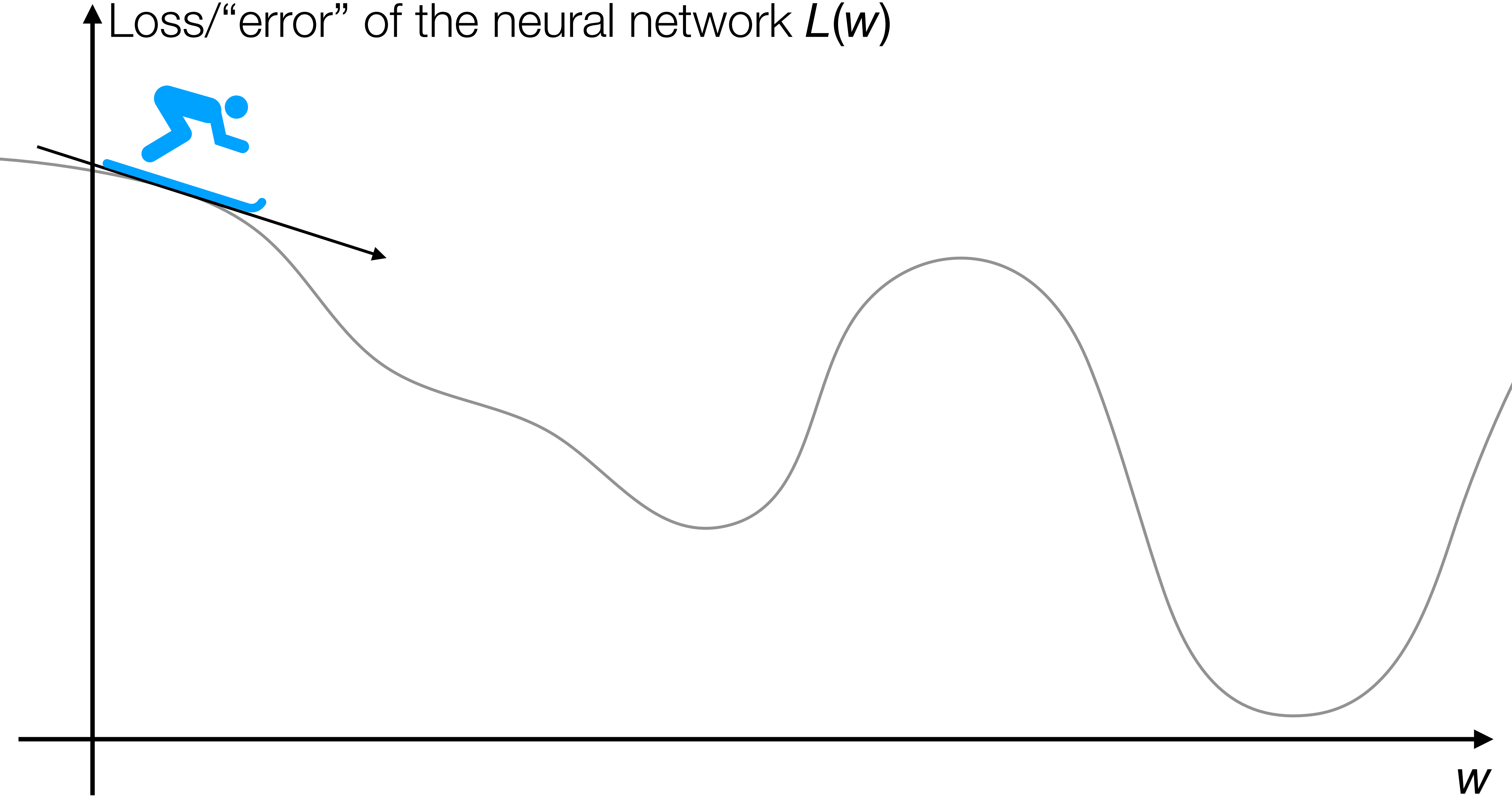


In general: the skier should move in *opposite* direction of derivative

In higher dimensions, this is called **gradient descent**

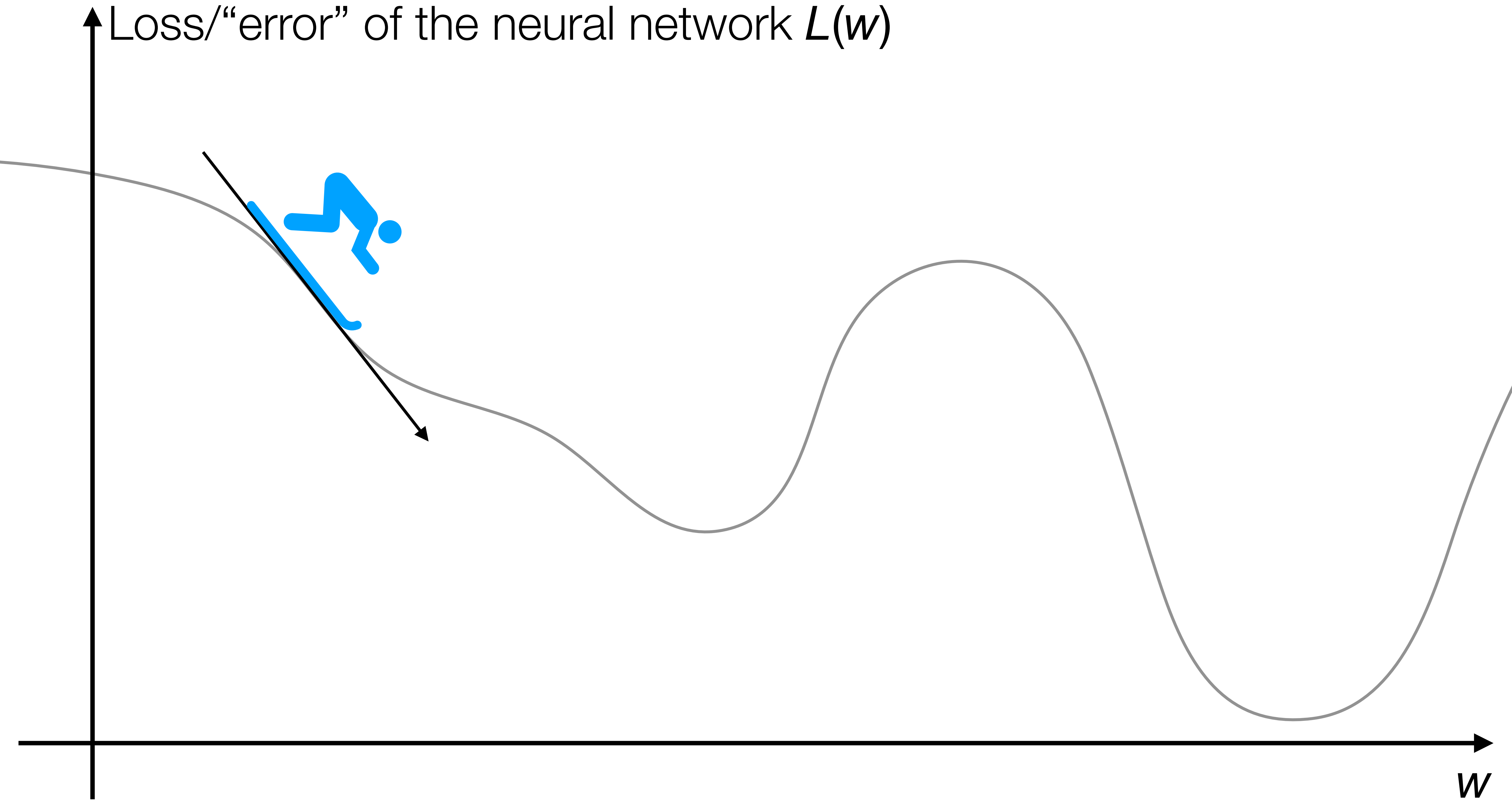
Learning a Deep Net

Suppose the neural network has a single real number parameter ***w***



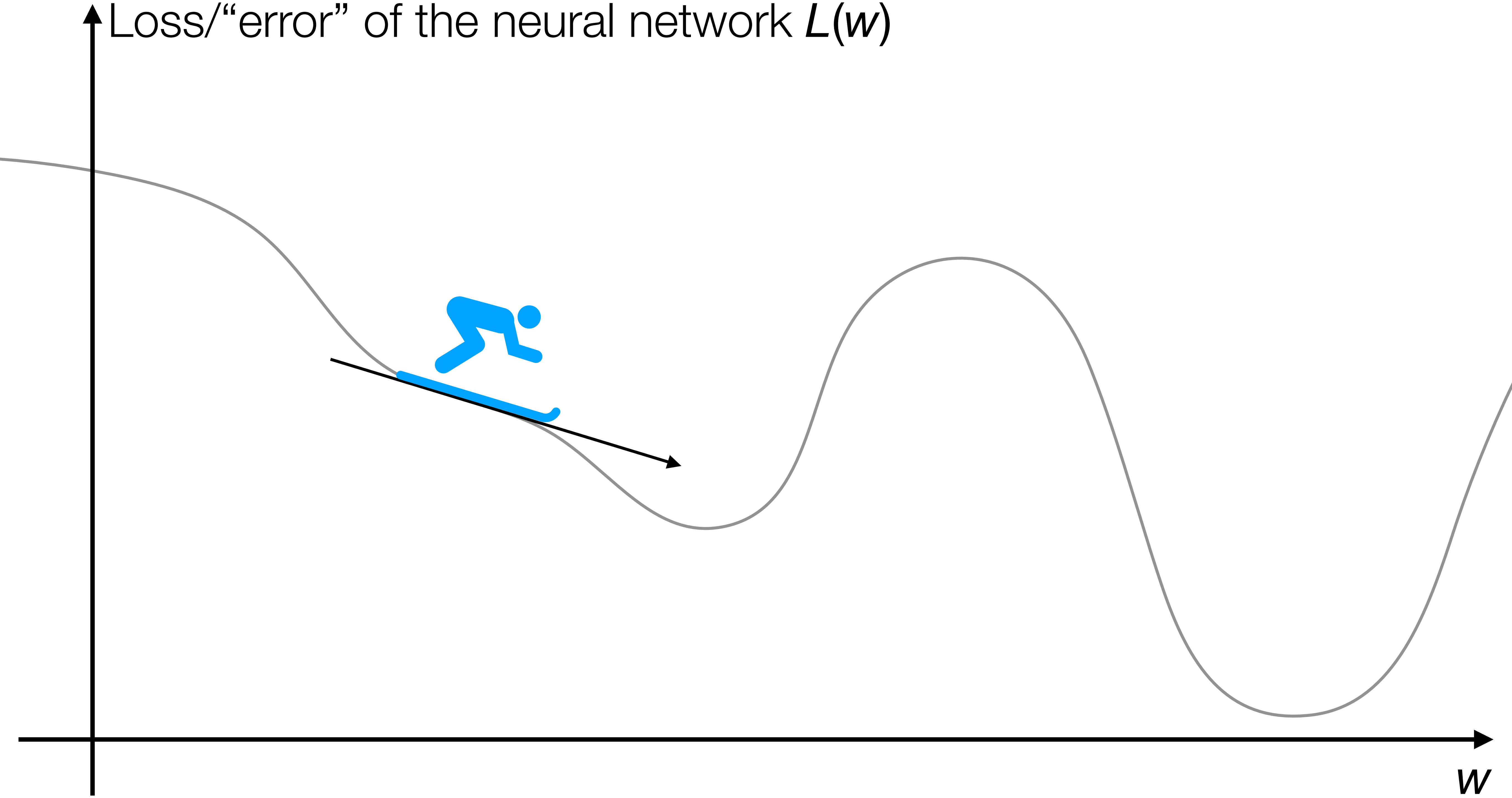
Learning a Deep Net

Suppose the neural network has a single real number parameter ***w***



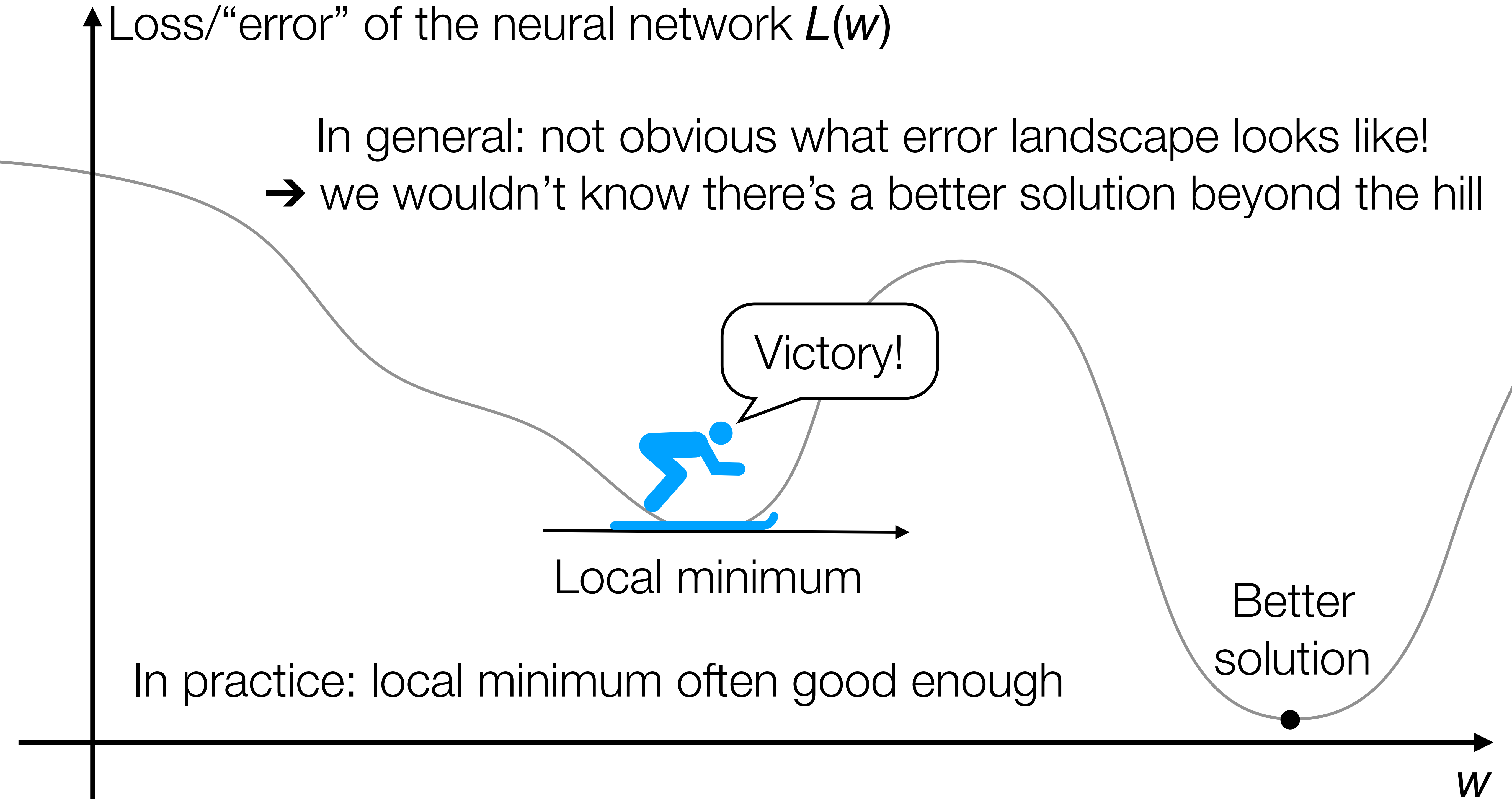
Learning a Deep Net

Suppose the neural network has a single real number parameter *w*



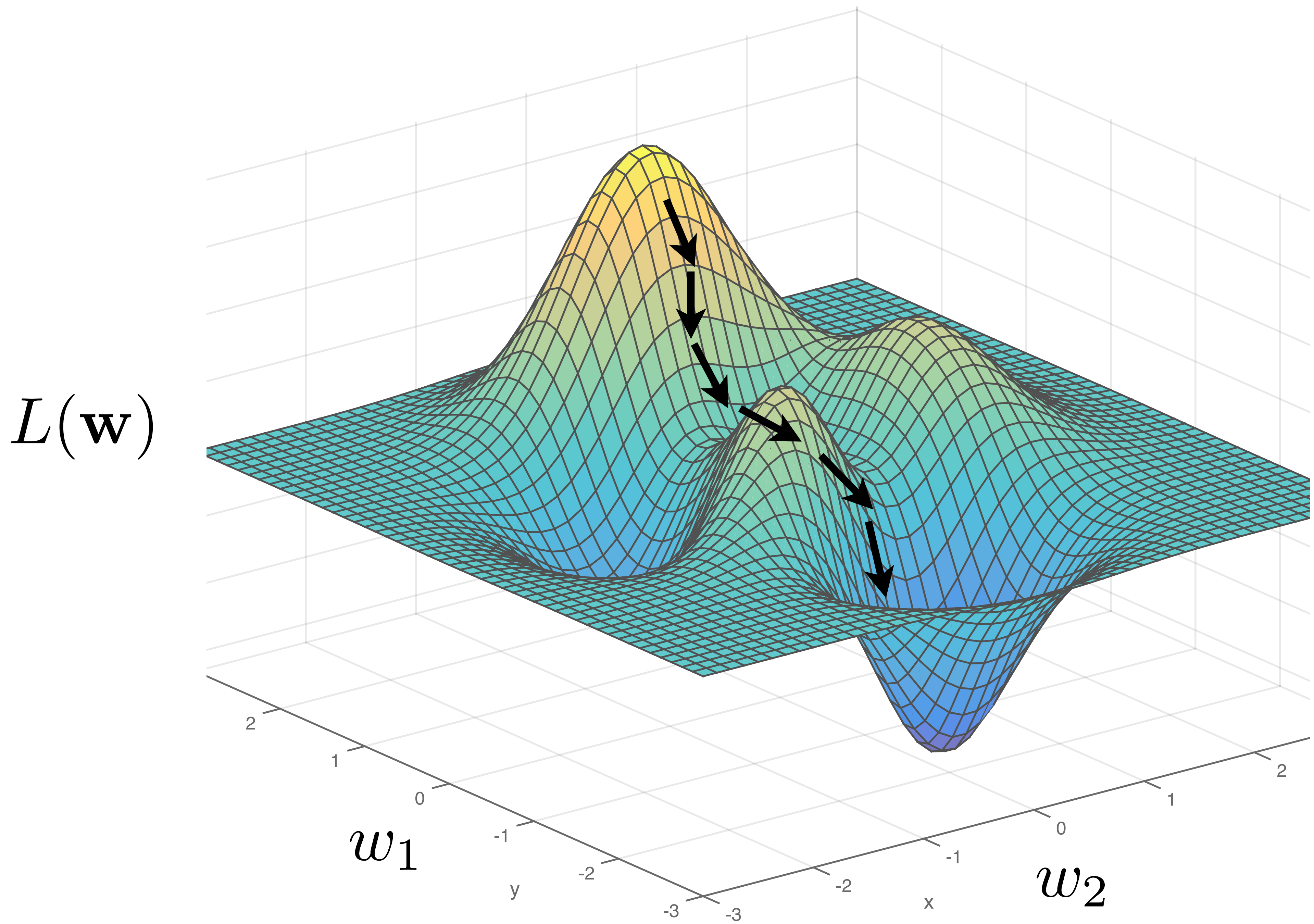
Learning a Deep Net

Suppose the neural network has a single real number parameter *w*



Learning a Deep Net

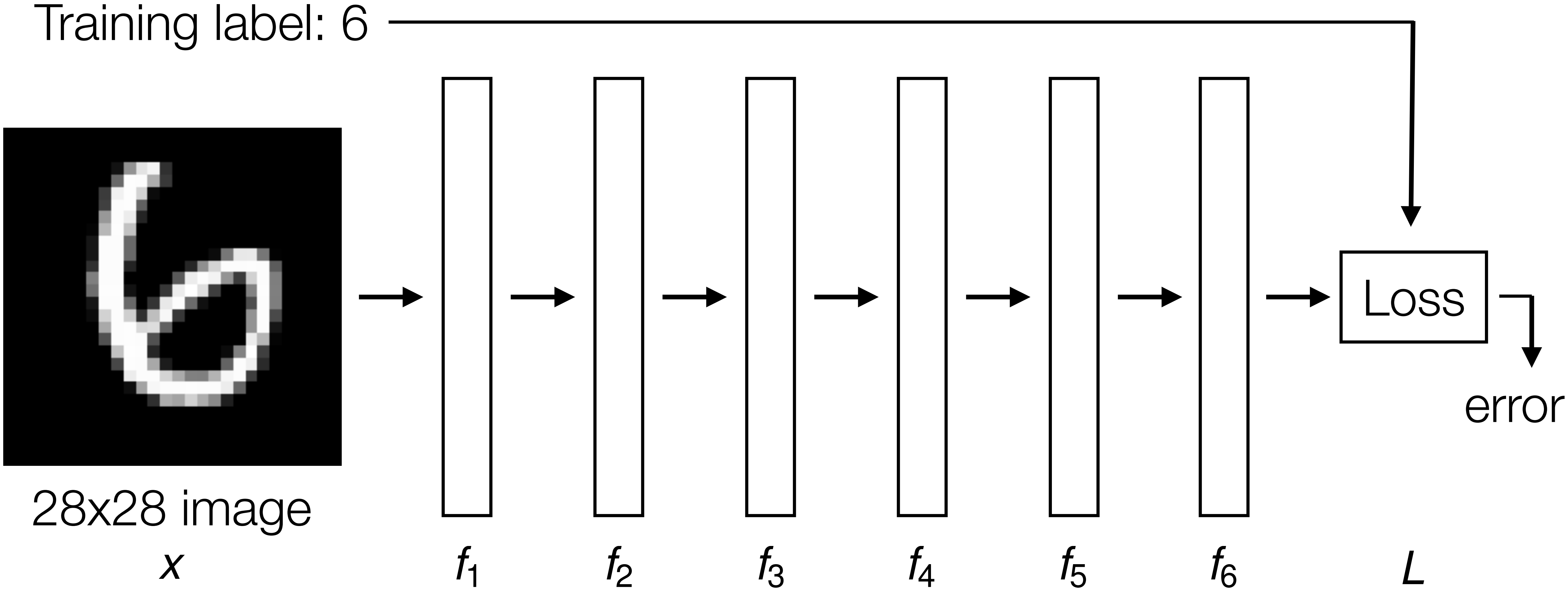
2D example of gradient descent



Slide by Phillip Isola

Remark: In practice, deep nets often have > *millions* of parameters, so *very* high-dimensional gradient descent

Handwritten Digit Recognition



All parameters: θ

Error: $L(f_6(f_5(f_4(f_3(f_2(f_1(x)))))))$

Gradient: $\frac{\partial L(f_6(f_5(f_4(f_3(f_2(f_1(x)))))))}{\partial \theta}$

Automatic differentiation is a crucial component to learning deep nets!

Careful derivative chain rule calculation: back-propagation algorithm

Dealing with Small Datasets

- Data augmentation
  - Generate perturbed versions of your training data (e.g., for images, add mirrored versions of images, rotated versions, etc) to get larger training dataset
- Fine tune
  - Is there an existing pre-trained neural net on a similar task? If so, reuse pre-trained model and modify the neural net slightly and train (using existing weights as initialization)

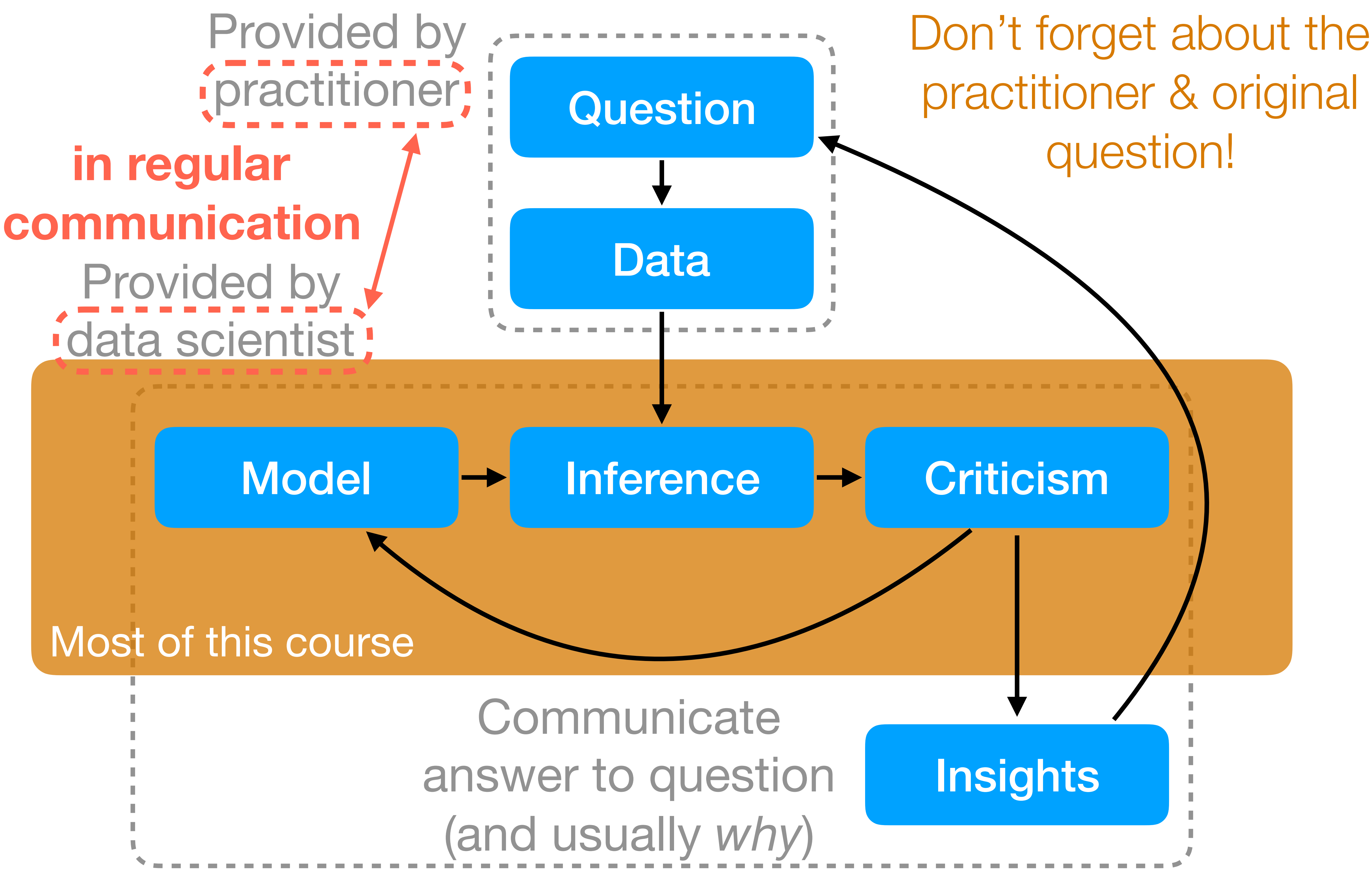
Lots More to Deep Learning

- Extremely important bit we haven't covered: visualizing what the deep net learned
- Some other cool ideas:
    - Self-supervised learning: remove parts of the data and predict the missing parts from the other parts (this is the key idea for word2vec!) — no training labels required!
    - Generative adversarial networks: 2 deep nets, one that learns a generative process for data, and another that tries to classify whether a data point is generated (synthetic) or real
    - Deep reinforcement learning: train AI to play Go and other games, also important in robotics

The Future of Deep Learning

- Deep learning currently is still limited in what it can do — the layers do simple operations and have to be differentiable
  - How do we make deep nets that generalize better?
- Still lots of engineering and expert knowledge used to design some of the best systems (e.g., AlphaGo)
  - How do we get away with using less expert knowledge?
- How to properly do lifelong learning?

95-865


Provided by
practitioner
in regular
communication
Provided by
data scientist
Question
Data
Don’t forget about the
practitioner & original
question!
Model
Inference
Criticism
Most of this course
Communicate
answer to question
(and usually *why*)
Insights

95-865 Some Parting Thoughts

- Remember to visualize different steps of your data analysis pipeline — very helpful when you're still debugging
- Often times in practice there may be little or no training labels
  - Is it possible to predict certain parts of the data from other parts? (Some times, we can set up a self-supervised task)
  - If we have to manual label, what's the best way to do it?
- Usually there are *tons* of models that you could try
  - It's good practice to come up with <u>quantitative metrics</u> that make sense for the problem you're trying to solve, and for which you can <u>evaluate models using a prediction task on held-out data</u>

Thanks for being a beta tester!