The Mathematics of Machine Learning 2026

Lecture 12

Tim van Erven

University of Amsterdam

Outline

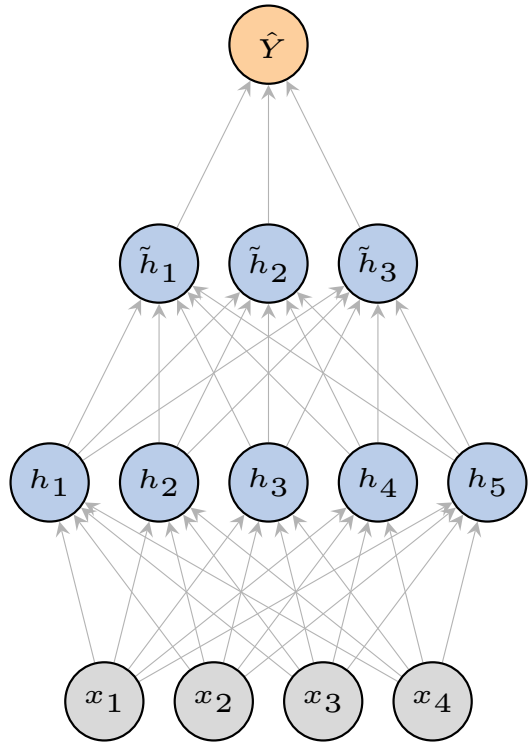
Dropout Regularization for Fully Connected Layers

1. Dropout Regularization

Neural Networks: Convolutional Layers

2. High-level Network Structure
3. Convolutional Layers for Grayscale Images
4. Dimensions and Padding
5. Multiple Input Channels
6. Multiple Output Channels

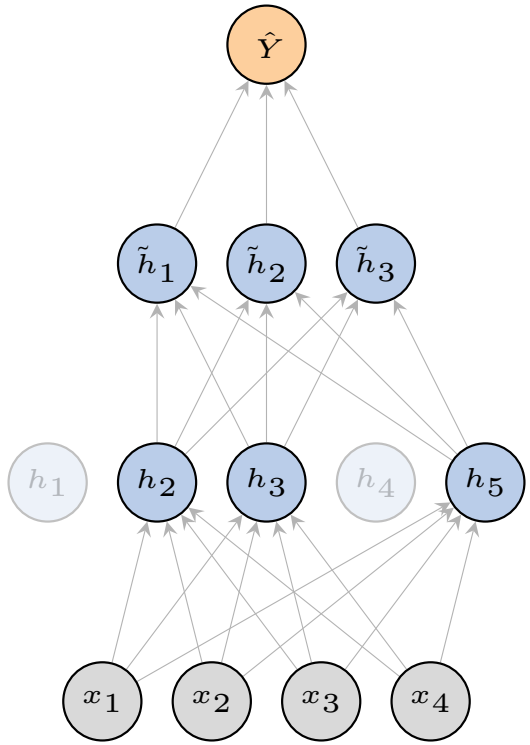
Dropout Regularization



Dropout for a hidden layer:

- Every time we compute a gradient during training, **randomly remove some neurons** from the layer and update only the remaining part of the network
- Scale activations of units that are not removed to avoid introducing bias
- Do not remove any neurons at test time

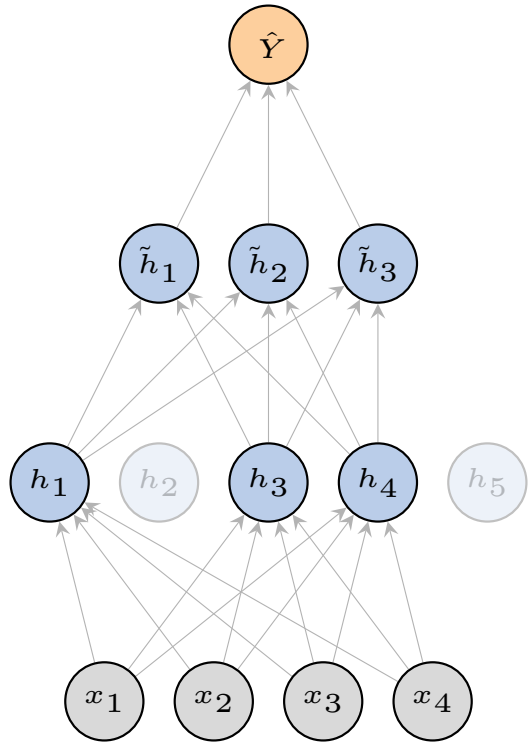
Dropout Regularization



Dropout for a hidden layer:

- Every time we compute a gradient during training, **randomly remove some neurons** from the layer and update only the remaining part of the network
- Scale activations of units that are not removed to avoid introducing bias
- Do not remove any neurons at test time

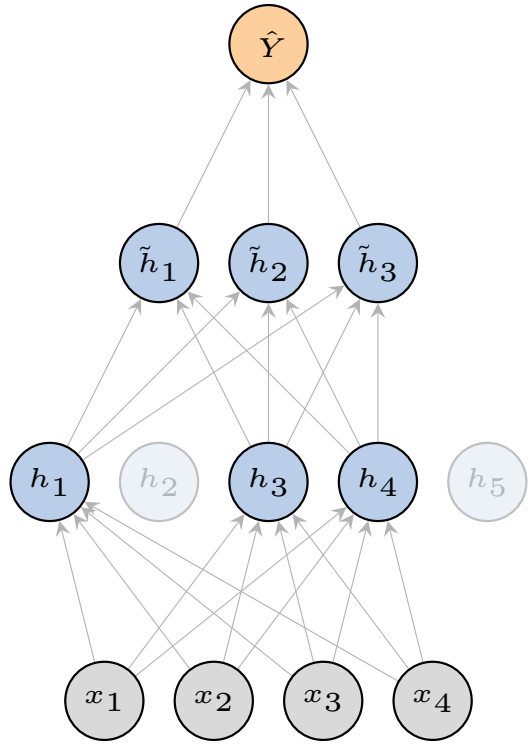
Dropout Regularization



Dropout for a hidden layer:

- Every time we compute a gradient during training, **randomly remove some neurons** from the layer and update only the remaining part of the network
- Scale activations of units that are not removed to avoid introducing bias
- Do not remove any neurons at test time

Dropout Regularization



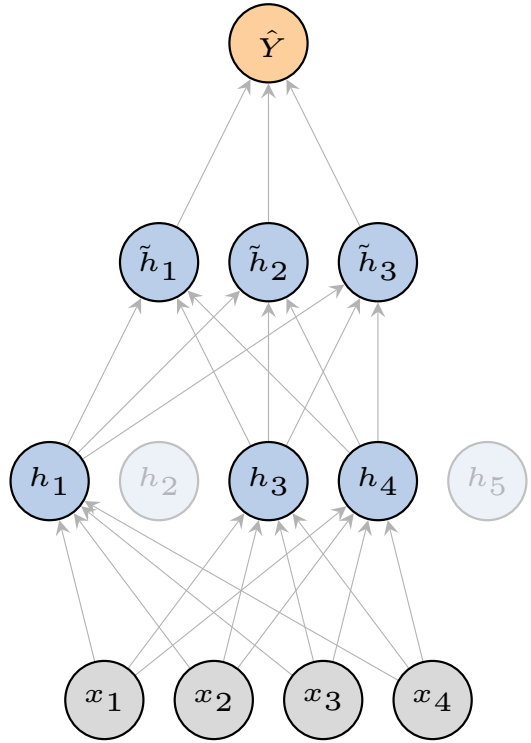
Dropout for a hidden layer:

- Specifically, for each neuron h in the hidden layer, randomly replace its activation a by

$$a' = \begin{cases} 0 & \text{with probability } p \\ \frac{a}{1-p} & \text{with probability } 1-p. \end{cases}$$

- Unbiased: $\mathbb{E}[a'] = a$

Dropout Regularization



Motivation:

- A type of **regularization** added when training neural networks
- Appears to promote [1]:
  - Small weights (measured in a data-dependent way)
  - More confident predictions, i.e. output probabilities closer to 0 or 1
- Also adds additional randomness, analogously to smaller minibatch sizes [2]

Outline

Dropout Regularization for Fully Connected Layers

1. Dropout Regularization

Neural Networks: Convolutional Layers

2. High-level Network Structure
3. Convolutional Layers for Grayscale Images
4. Dimensions and Padding
5. Multiple Input Channels
6. Multiple Output Channels

High-level Network Structure

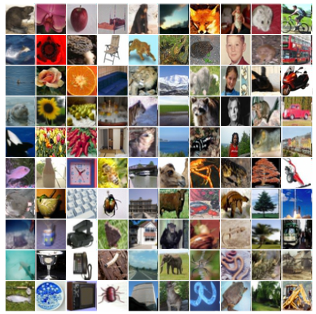


Figure 18.8 Examples of natural images. The `CIFAR-100` database consists of 100 color image classes, with 600 examples in each class (500 train, 100 test). Each image is $32 \times 32 \times 3$ (red, green, blue). Here we display a randomly chosen image from each class. The classes are organized by hierarchical structure, with 20 coarse levels and five subclasses within each. So, for example, the first five images in the first column are `aquatic mammals`, namely `beaver`, `dolphin`, `otter`, `seal` and `whale`.

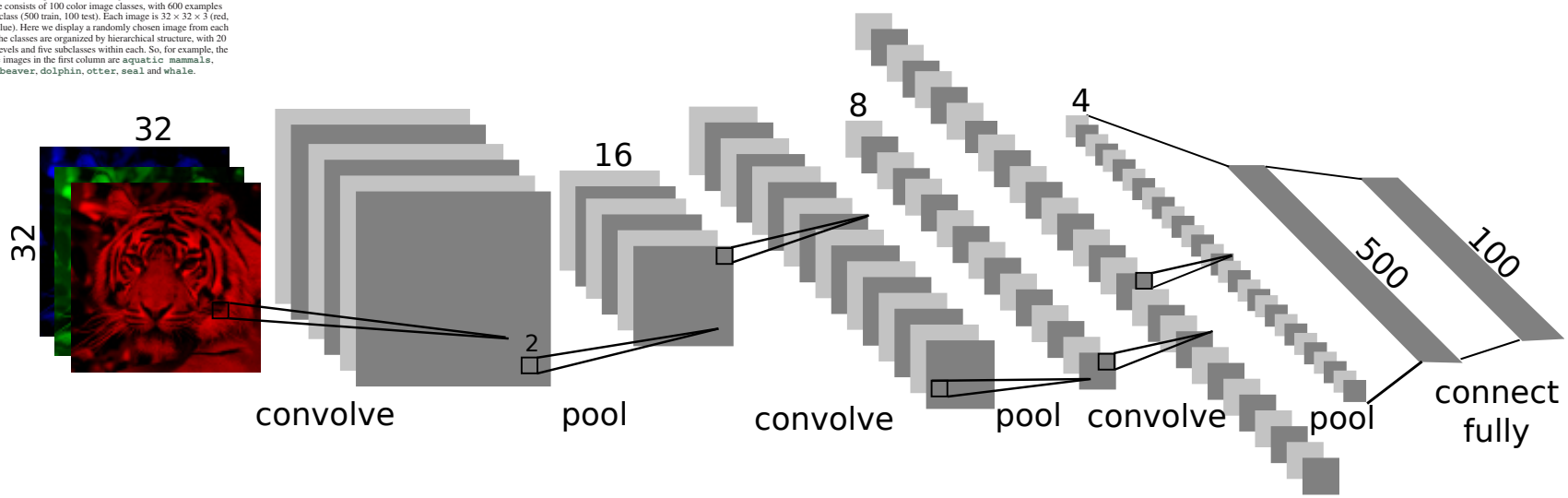


Figure 18.9 Architecture of a deep-learning network for the `CIFAR-100` image classification task. The input layer and

- Activation function also after every convolutional layer (before pooling)

Figures from Efron, Hastie, Computer Age Statistical Inference, 2016.

Convolutional Layers for Grayscale Images

Input image

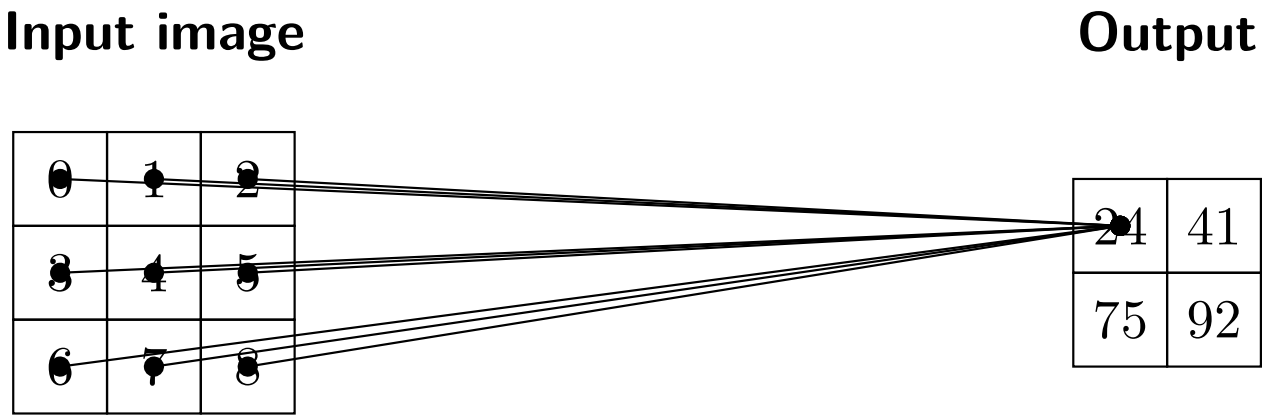
0	1	2
3	4	5
6	7	8

Output

24	41
75	92

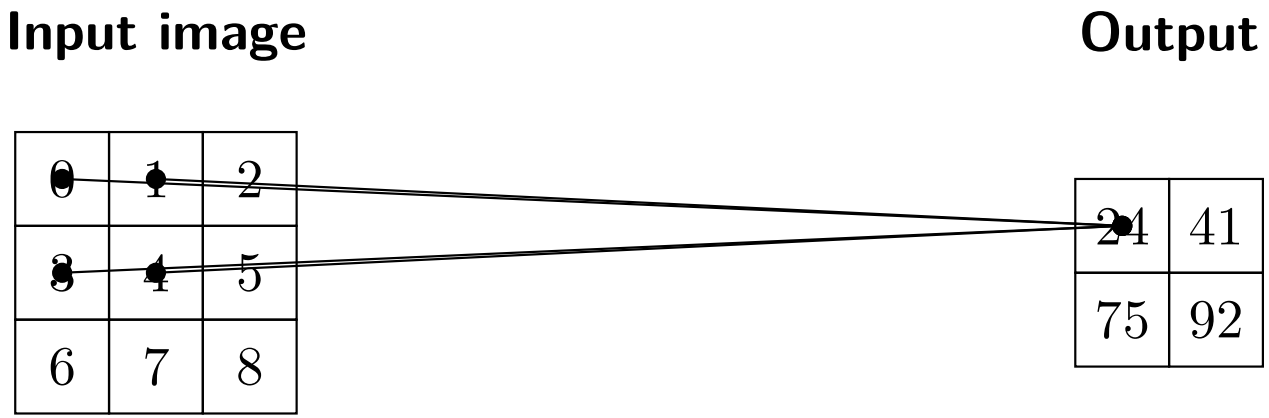
Fully connected layer: every output unit extracts information from the whole input image.

Convolutional Layers for Grayscale Images



Fully connected layer: every output unit extracts information from the whole input image.

Convolutional Layers for Grayscale Images



Convolutional layer: every output unit extracts information from a **local square region** in the input image.

Convolutional Layers for Grayscale Images

Input image

0	1	2
3	4	5
6	7	8

$*$

Filter

4	9
1	3

$=$

Output

24	41
75	92

Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Convolutional Layers for Grayscale Images

Input image

0	1	2
3	4	5
6	7	8

$*$

Filter

4	9
1	3

$=$

Output

24	41
75	92

$$0 \times 4 + 1 \times 9 + 3 \times 1 + 4 \times 3 = 24$$

Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Convolutional Layers for Grayscale Images

Input image

0	1	2
3	4	5
6	7	8

$*$

Filter

4	9
1	3

$=$

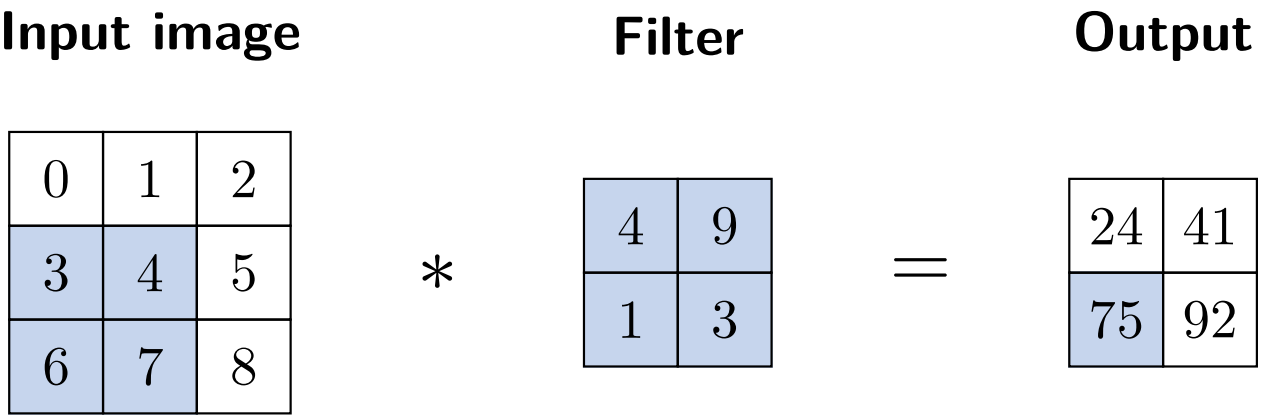
Output

24	41
75	92

$$1 \times 4 + 2 \times 9 + 4 \times 1 + 5 \times 3 = 41$$

Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Convolutional Layers for Grayscale Images



$$3 \times 4 + 4 \times 9 + 6 \times 1 + 7 \times 3 = 75$$

Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Convolutional Layers for Grayscale Images

Input image

0	1	2
3	4	5
6	7	8

$*$

Filter

4	9
1	3

$=$

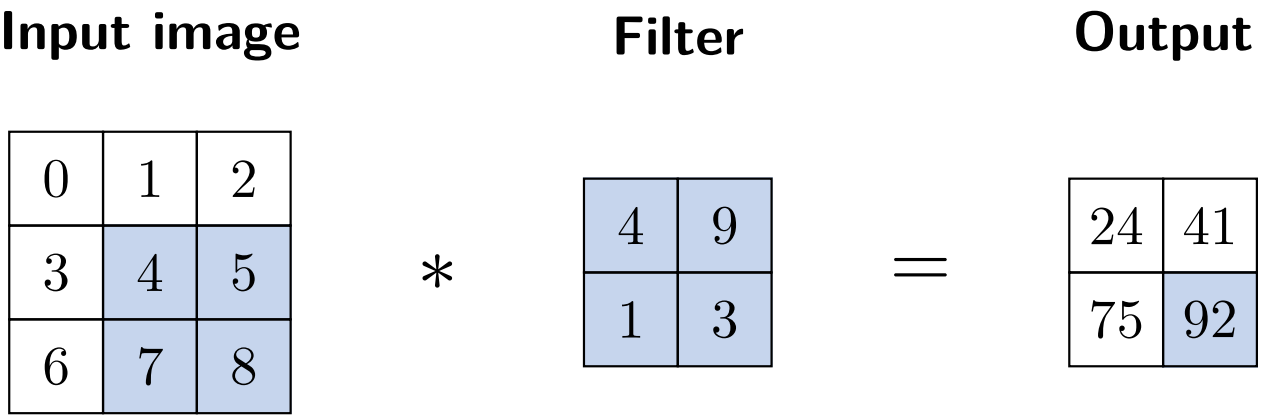
Output

24	41
75	92

$$4 \times 4 + 5 \times 9 + 7 \times 1 + 8 \times 3 = 92$$

Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Convolutional Layers for Grayscale Images

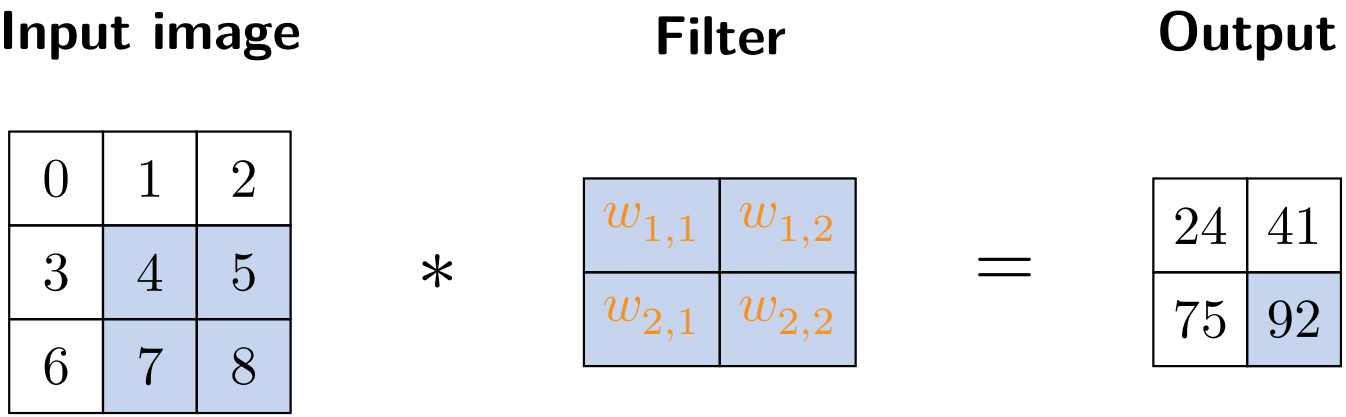


Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Can view as constraints on fully connected layer:

- Weights coming from outside square region are 0
- Weights in filter are shared for all output units. Much fewer parameters!

Convolutional Layers for Grayscale Images

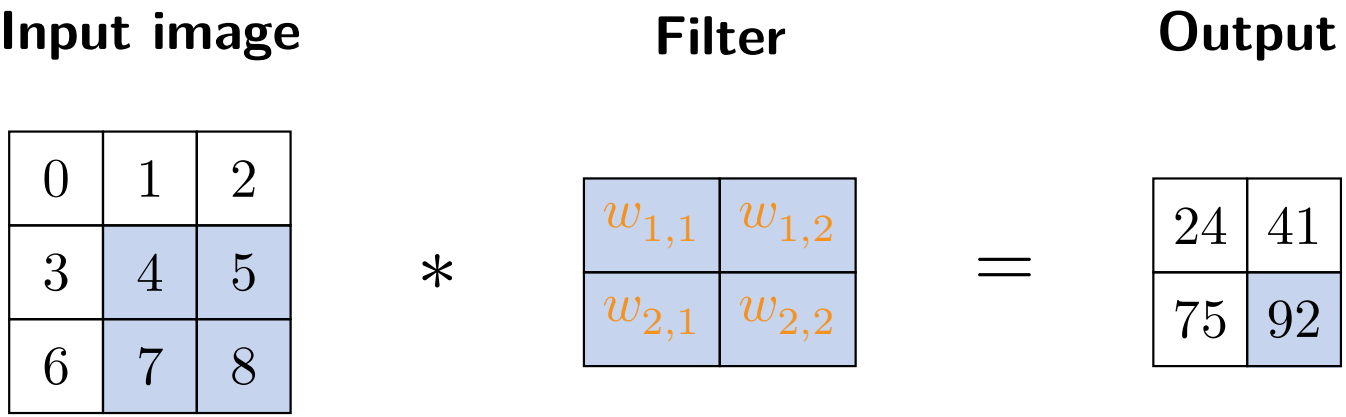


Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Parameters:

- **Filter weights** are **learned from data**

Convolutional Layers for Grayscale Images

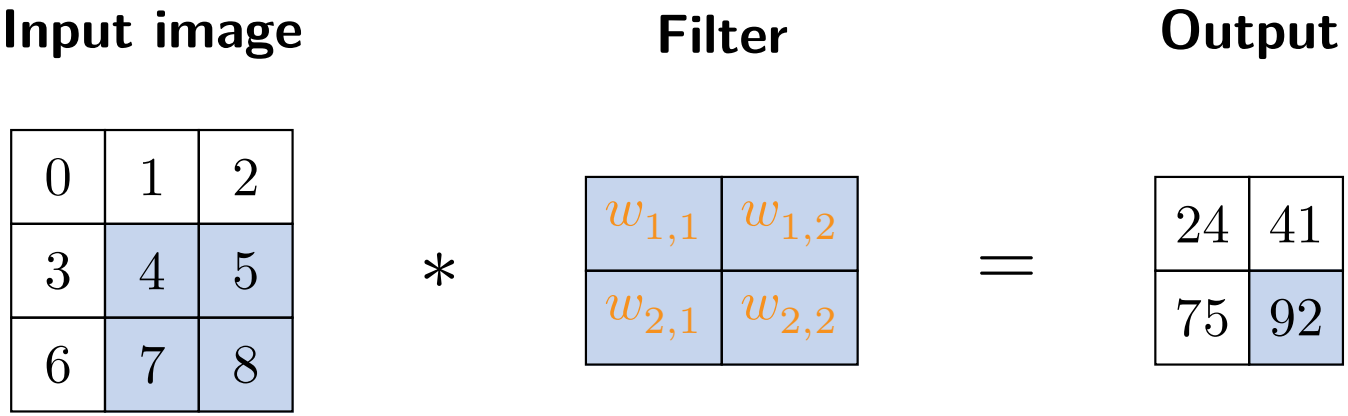


Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Parameters:

- **Filter weights** are **learned from data**
- Backpropagation can be modified to compute gradients in networks that include convolutional layers

Convolutional Layers for Grayscale Images



Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

Exploits prior knowledge:

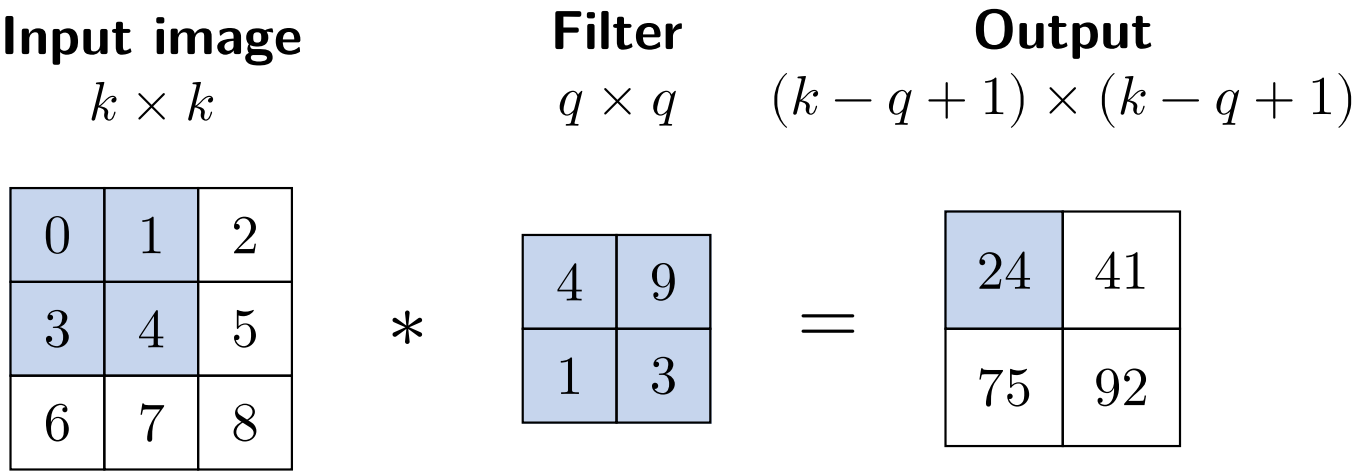
- Prior knowledge: same object can be detected the same way in different parts of the image
- NB Filter should be large enough to cover size of typical objects in image (this depends on image resolution)

Dimensions and Padding

Input image $k \times k$				**Filter** $q \times q$			**Output** $(k-q+1) \times (k-q+1)$	
0	1	2	$*$	4	9	$=$	24	41
3	4	5		1	3		75	92
6	7	8						

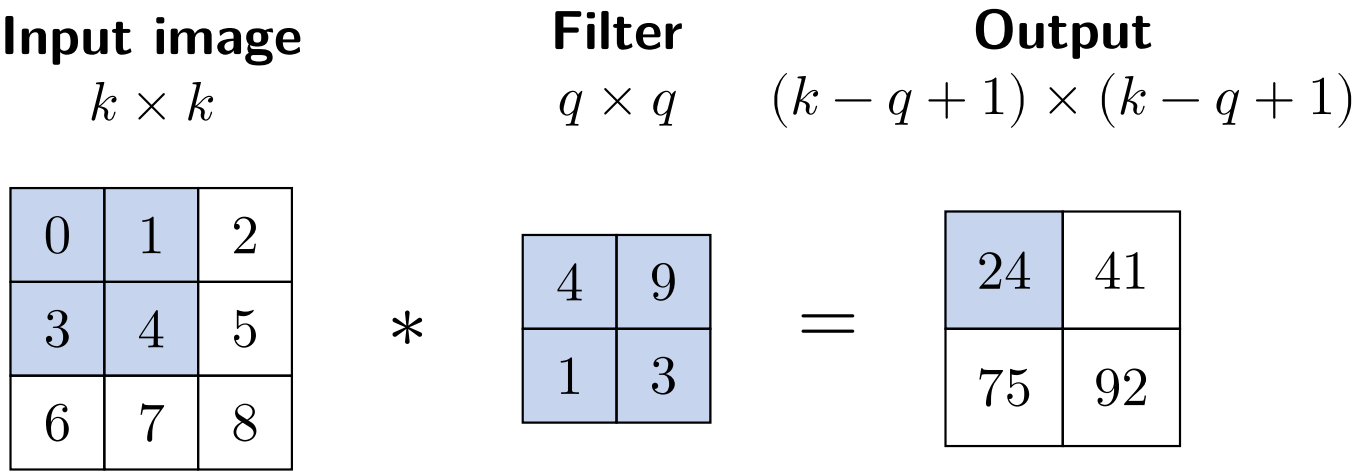
- **Very small** q, much smaller than k. For instance, $q = 2$ or $q = 3$. So output nearly same dimensions as input.

Dimensions and Padding



- **Very small** q, much smaller than k. For instance, $q = 2$ or $q = 3$. So output nearly same dimensions as input.
- Smaller output than input $\Rightarrow$ pixels on boundaries and especially in corners underused compared to other pixels.

Dimensions and Padding



- **Very small** q, much smaller than k. For instance, $q = 2$ or $q = 3$. So output nearly same dimensions as input.
- Smaller output than input $\Rightarrow$ pixels on boundaries and especially in corners underused compared to other pixels.
- Difference is small for small q, but effect accumulates if we concatenate multiple convolutional layers.

Dimensions and Padding

Input image
$k \times k$

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

$*$

Filter
$q \times q$

4	9
1	3

$=$

Output
$(k-q+1) \times (k-q+1)$

0	3	7	2
9	24	41	33
45	75	92	28
54	87	100	32

- **Pad input image with zeros** to make output dimensions (nearly) equal to input dimensions

Dimensions and Padding

Input image
$k \times k$

0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0

$*$

Filter
$q \times q$

4	9
1	3

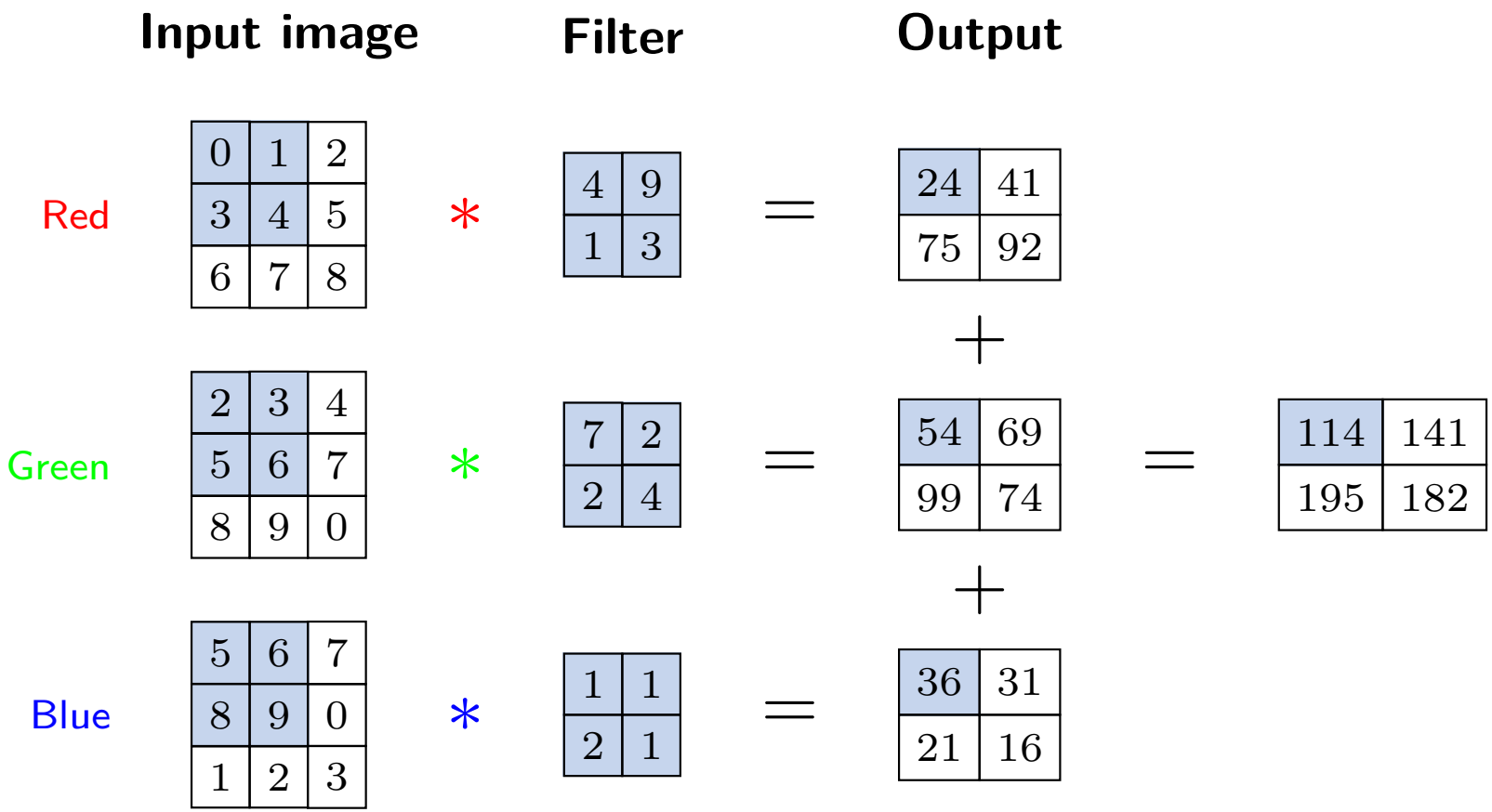
$=$

Output
$(k-q+1) \times (k-q+1)$

0	3	7	2
9	24	41	33
45	75	92	28
54	87	100	32

- **Pad input image with zeros** to make output dimensions (nearly) equal to input dimensions
- If q is odd, adding $(q-1)/2$ zeros on all sides makes **output dimensions equal to input dimensions**

Multiple Input Channels



- Input consists of multiple matrices = **channels** with the same dimensions (for instance: red, green and blue in a color image)
- Perform separate convolution per input channel and add up the results

Multiple Output Channels I

- Often want to extract different aspects from inputs. For instance, detect different features in image.
- Solution: duplicate the operation from the previous slide m times with **separate filters** to get m **output channels**
- SGD then optimizes filters to be jointly useful.

Multiple Output Channels II

Input image

Red

0	1	2
3	4	5
6	7	8

Green

2	3	4
5	6	7
8	9	0

Blue

5	6	7
8	9	0

Output channel 1

$$
\begin{array}{ccccc}
 & \begin{array}{|cc|}\hline 4 & 9 \\ 1 & 3 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 24 & 41 \\ 75 & 92 \\ \hline\end{array} & \\
 & & & + & \\
* & \begin{array}{|cc|}\hline 7 & 2 \\ 2 & 4 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 54 & 69 \\ 99 & 74 \\ \hline\end{array} & = \quad \begin{array}{|cc|}\hline 114 & 141 \\ 195 & 182 \\ \hline\end{array} \\
 & & & + & \\
 & \begin{array}{|cc|}\hline 1 & 1 \\ 2 & 1 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 36 & 31 \\ 21 & 16 \\ \hline\end{array} &
\end{array}
$$

Multiple Output Channels II

Input image

Red

0	1	2
3	4	5
6	7	8

Green

2	3	4
5	6	7
8	9	0

Blue

5	6	7
8	9	0
1	2	3

Output channel 1

$$
\begin{array}{ccccc}
\begin{array}{|cc|}\hline 4 & 9 \\ 1 & 3 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 24 & 41 \\ 75 & 92 \\ \hline\end{array} & & \\
 & & + & & \\
\begin{array}{|cc|}\hline 7 & 2 \\ 2 & 4 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 54 & 69 \\ 99 & 74 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 114 & 141 \\ 195 & 182 \\ \hline\end{array} \\
 & & + & & \\
\begin{array}{|cc|}\hline 1 & 1 \\ 2 & 1 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 36 & 31 \\ 21 & 16 \\ \hline\end{array} & &
\end{array}
$$

Output channel 2

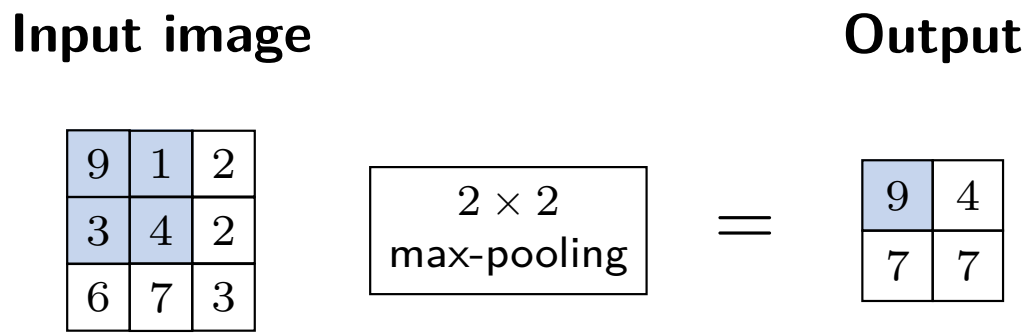
$$
\begin{array}{ccccc}
\begin{array}{|cc|}\hline 2 & 1 \\ 3 & 1 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 14 & 21 \\ 35 & 42 \\ \hline\end{array} & & \\
 & & + & & \\
\begin{array}{|cc|}\hline 1 & 3 \\ 2 & 1 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 27 & 34 \\ 48 & 45 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 92 & 83 \\ 115 & 116 \\ \hline\end{array} \\
 & & + & & \\
\begin{array}{|cc|}\hline 2 & 1 \\ 1 & 3 \\ \hline\end{array} & = & \begin{array}{|cc|}\hline 51 & 28 \\ 32 & 29 \\ \hline\end{array} & &
\end{array}
$$

Pooling

- The outputs of a convolutional layer depend on a very local region of the input by construction
- Often want to **downsample** these outputs to get a **representation that depends on a slightly larger region of the input**

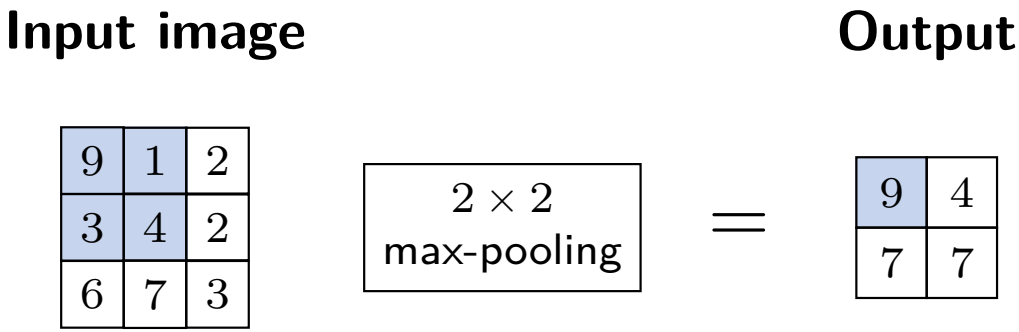
Pooling

- The outputs of a convolutional layer depend on a very local region of the input by construction
- Often want to **downsample** these outputs to get a **representation that depends on a slightly larger region of the input**



Pooling

- The outputs of a convolutional layer depend on a very local region of the input by construction
- Often want to **downsample** these outputs to get a **representation that depends on a slightly larger region of the input**



- Max-pooling: take the maximum over a $q \times q$ region
- **No parameters!**
- It is also possible, but less common nowadays, to take the average instead of the max: average pooling

High-level Network Structure

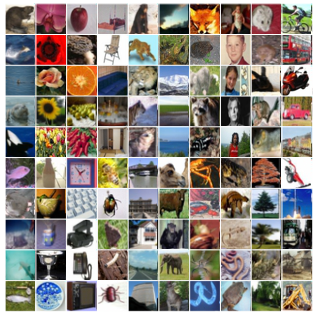


Figure 18.8 Examples of natural images. The `CIFAR-100` database consists of 100 color image classes, with 600 examples in each class (500 train, 100 test). Each image is $32 \times 32 \times 3$ (red, green, blue). Here we display a randomly chosen image from each class. The classes are organized by hierarchical structure, with 20 coarse levels and five subclasses within each. So, for example, the first five images in the first column are `aquatic mammals`, namely `beaver`, `dolphin`, `otter`, `seal` and `whale`.

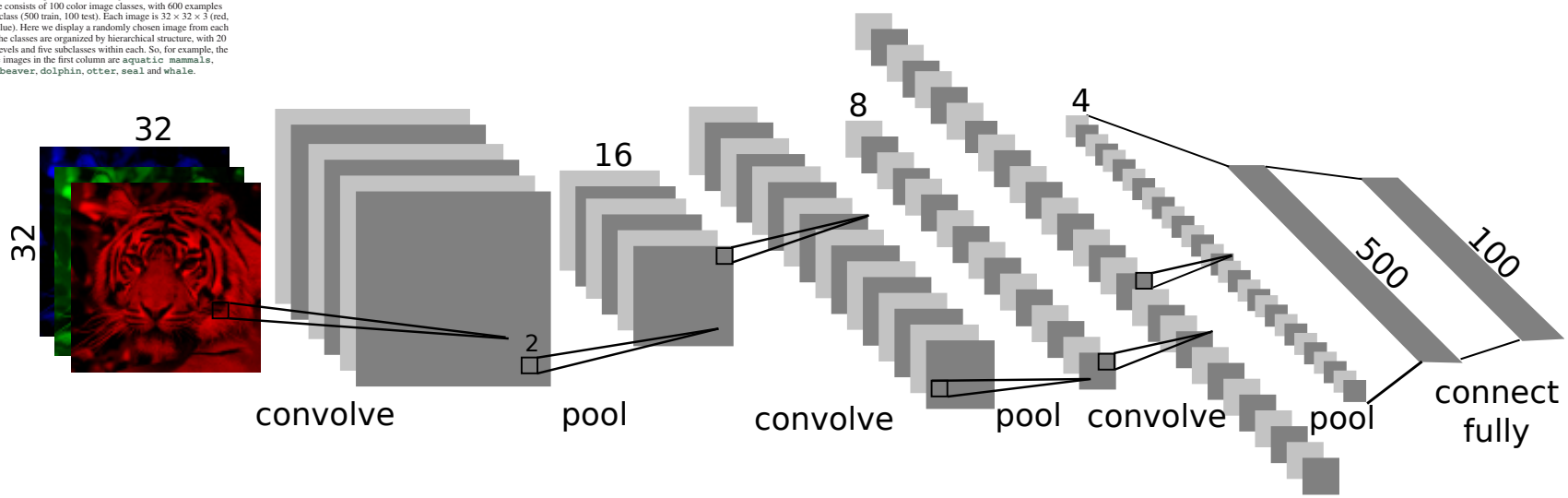


Figure 18.9 Architecture of a deep-learning network for the `CIFAR-100` image classification task. The input layer and

- Activation function also after every convolutional layer (before pooling)

Figures from Efron, Hastie, Computer Age Statistical Inference, 2016.

References

[1] S. Wager, S. Wang, and P. S. Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, volume 26, 2013.

[2] C. Wei, S. Kakade, and T. Ma. The implicit and explicit regularization effects of dropout. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10181–10192. PMLR, 13–18 Jul 2020.