The Mathematics of Machine Learning 2025

Lecture 12

Tim van Erven

University of Amsterdam

Outline

Optimization of Neural Networks

1. Backpropagation

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

Final Remarks

7. Remarks about ChatGPT

Q&A

Backpropagation for Fully Connected Layers

How to compute gradients $\nabla_{\boldsymbol{\beta}} L(Y_i, f_{\boldsymbol{\beta}}(X_i))$?

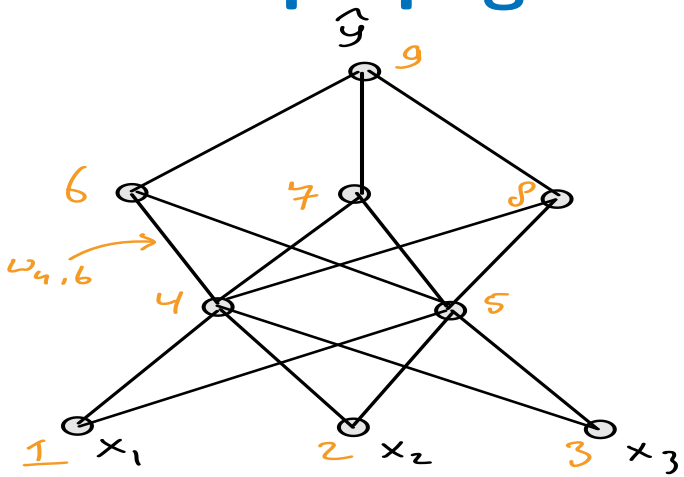
- 20 million parameters: only have time for a fixed number of computations per parameter!

Backpropagation for Fully Connected Layers

How to compute gradients $\nabla_{\boldsymbol{\beta}} L(Y_i, f_{\boldsymbol{\beta}}(X_i))$?

- 20 million parameters: only have time for a fixed number of computations per parameter!
- Solution: **Backpropagation**
  $\leftarrow$ use chain rule for derivatives many times: $\frac{\mathrm{d}a}{\mathrm{d}b} = \frac{\mathrm{d}a}{\mathrm{d}c}\frac{\mathrm{d}c}{\mathrm{d}b}$

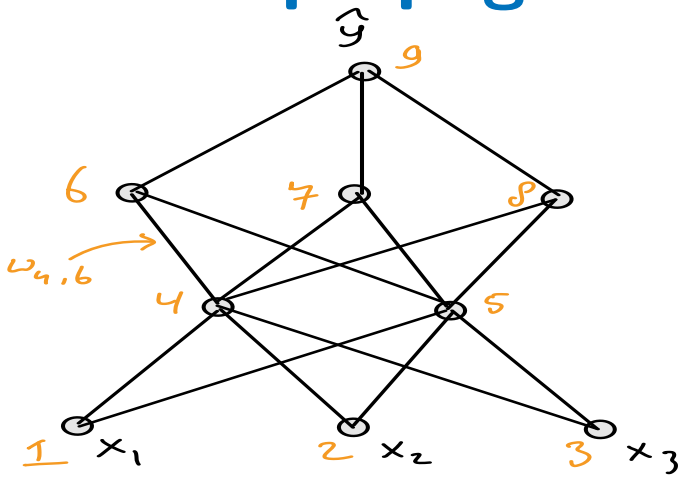
Backpropagation for Fully Connected Layers



$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$

$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \ldots, 8$$

Backpropagation for Fully Connected Layers

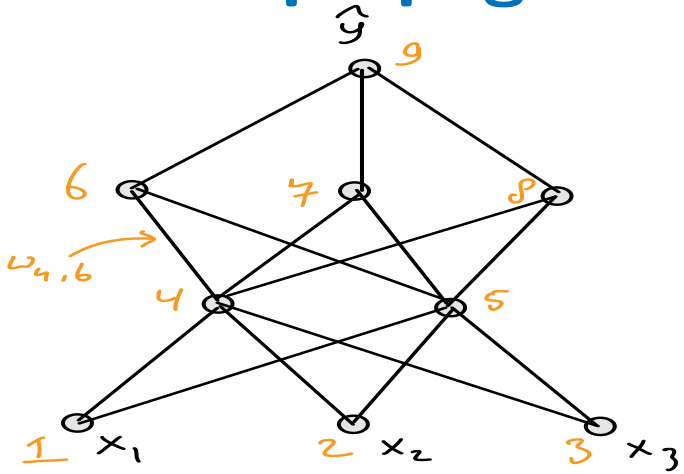


$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$

$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \ldots, 8$$

1. $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_9} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}\hat{Y}}$ is derivative of the loss

Backpropagation for Fully Connected Layers



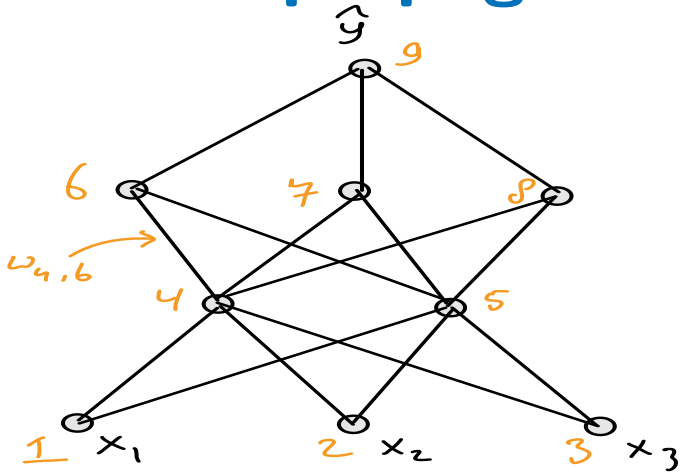
$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$

$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \ldots, 8$$

1. $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_9} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}\hat{Y}}$ is derivative of the loss
2. Suppose we have $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_6}$. Then we can efficiently compute

$$\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_6} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_6}\frac{\mathrm{d}o_6}{\mathrm{d}a_6} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_6}\sigma'(a_6)$$

Backpropagation for Fully Connected Layers



$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$
$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \ldots, 8$$

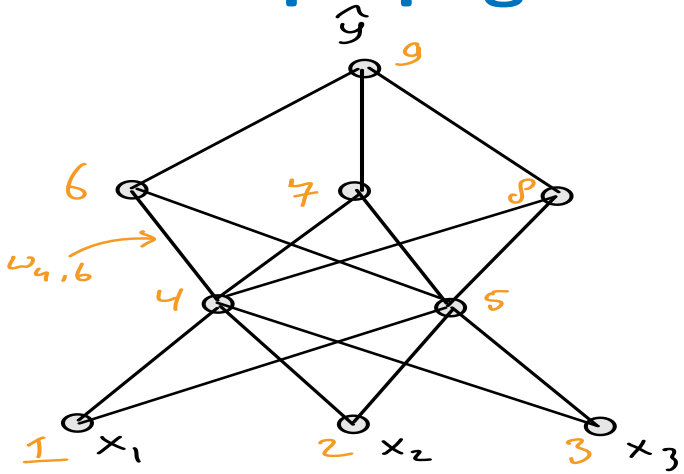
1. $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_9} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}\hat{Y}}$ is derivative of the loss
2. Suppose we have $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_6}$. Then we can efficiently compute

$$\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_6} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_6}\frac{\mathrm{d}o_6}{\mathrm{d}a_6} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_6}\sigma'(a_6)$$

3. Suppose we have e.g. $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_6}$, then we can efficiently compute derivatives w.r.t. all incoming weights to unit 6:

$$\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}w_{4,6}} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_6}\frac{\mathrm{d}\sum_{i=4}^{5} w_{i,6}o_i + b_6}{\mathrm{d}w_{4,6}} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_6}o_4$$

Backpropagation for Fully Connected Layers



$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$
$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \dots, 8$$

1. $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_9} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}\hat{Y}}$ is derivative of the loss
2. Suppose we have $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_m}$. Then we can efficiently compute

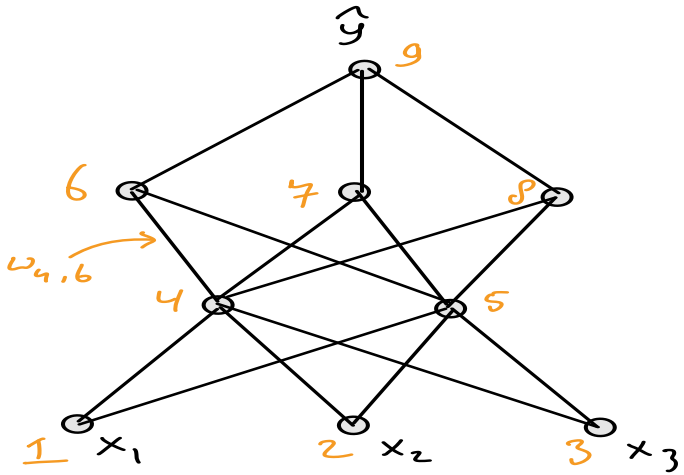
> Summary so far: if we could efficiently compute $\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_m}$ for all $m = 4, \dots, 8$, then we could efficiently compute derivatives w.r.t. all weights.

3. S

derivatives w.r.t. all incoming weights to unit 6:

$$\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}w_{4,6}} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_6} \frac{\mathrm{d}\sum_{i=4}^{5} w_{i,6} o_i + b_6}{\mathrm{d}w_{4,6}} = \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}a_6} o_4$$

Backpropagation for Fully Connected Layers

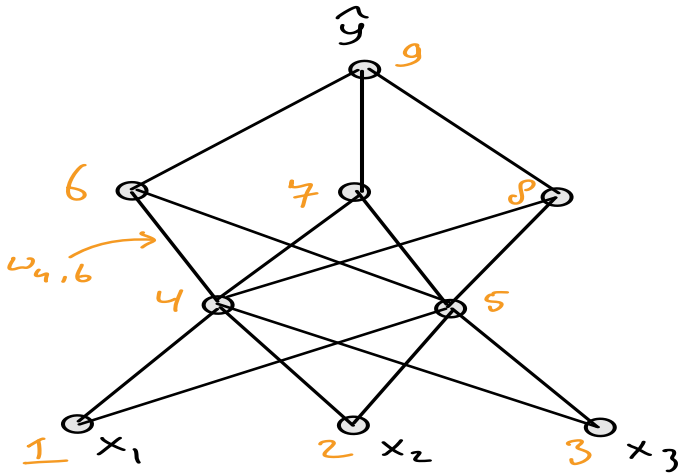


$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$

$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \ldots, 8$$

4. Work backwards from output to inputs to compute derivatives w.r.t. each o_m if we already have these derivatives higher up in the network.

Backpropagation for Fully Connected Layers



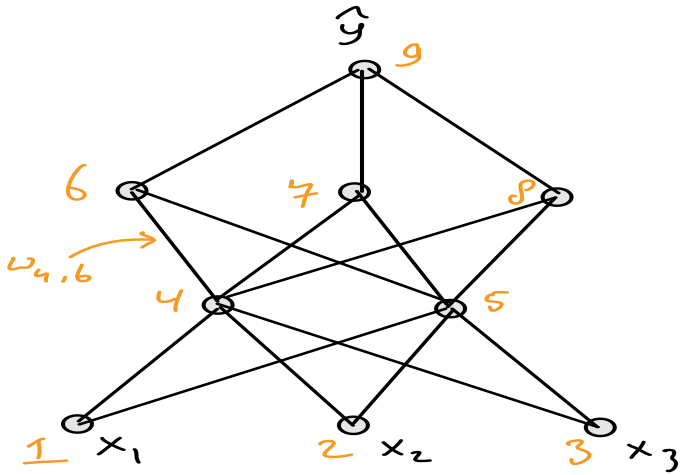
$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$

$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \ldots, 8$$

4. Work backwards from output to inputs to compute derivatives w.r.t. each o_m if we already have these derivatives higher up in the network.

Vector chain rule: $$\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_5} = \sum_{i=6}^{8} \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_i} \frac{\mathrm{d}o_i}{\mathrm{d}o_5}$$

Backpropagation for Fully Connected Layers



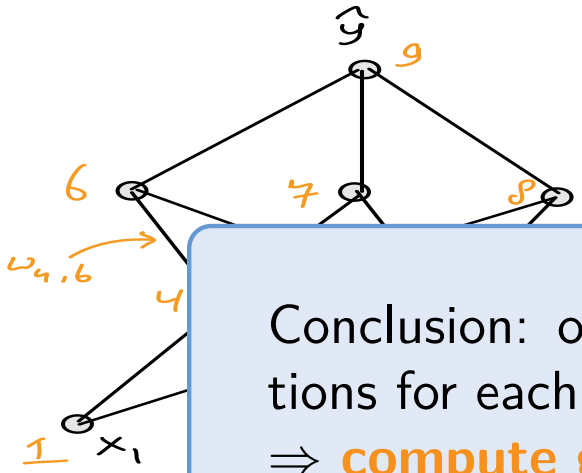
$$\hat{Y} = f_{\boldsymbol{\beta}}(X) = a_9$$
$$o_m = \sigma(a_m) \qquad \text{for } m = 4, \ldots, 8$$

4. Work backwards from output to inputs to compute derivatives w.r.t. each o_m if we already have these derivatives higher up in the network.

Vector chain rule: $$\frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_5} = \sum_{i=6}^{8} \frac{\mathrm{d}L(Y,\hat{Y})}{\mathrm{d}o_i} \frac{\mathrm{d}o_i}{\mathrm{d}o_5}$$

$$\frac{\mathrm{d}o_6}{\mathrm{d}o_5} = \frac{\mathrm{d}o_6}{\mathrm{d}a_6}\frac{\mathrm{d}a_6}{\mathrm{d}o_5} = \sigma'(a_6)\frac{\mathrm{d}\sum_{i=4}^{5} w_{i,6}o_i + b_6}{\mathrm{d}o_5} = \sigma'(a_6)w_{5,6}$$

Backpropagation for Fully Connected Layers



$4, \ldots, 8$

4. V

e

n

w.r.t.

V

Conclusion: only need to do fixed number of operations for each element of the network
$\Rightarrow$ **compute gradient vector quickly**, in number of computation steps that scales **linearly in the number of parameters**

- Programming all these derivatives is very challenging
- Modern software libraries do all this differentiation automatically under the hood!

$$\frac{\mathrm{d}o_6}{\mathrm{d}o_5} = \frac{\mathrm{d}o_6}{\mathrm{d}a_6}\frac{\mathrm{d}a_6}{\mathrm{d}o_5} = \sigma'(a_6)\frac{\mathrm{d}\sum_{i=4} w_{i,6}o_i + b_6}{\mathrm{d}o_5} = \sigma'(a_6)w_{5,6}$$

Outline

Optimization of Neural Networks

1. Backpropagation

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

Final Remarks

7. Remarks about ChatGPT

Q&A

High-level Network Structure

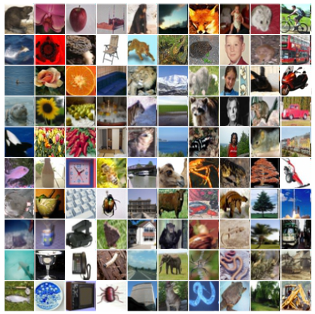


Figure 18.8 Examples of natural images. The `CIFAR-100` database consists of 100 color image classes, with 600 examples in each class (500 train, 100 test). Each image is $32 \times 32 \times 3$ (red, green, blue). Here we display a randomly chosen image from each class. The classes are organized by 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`.

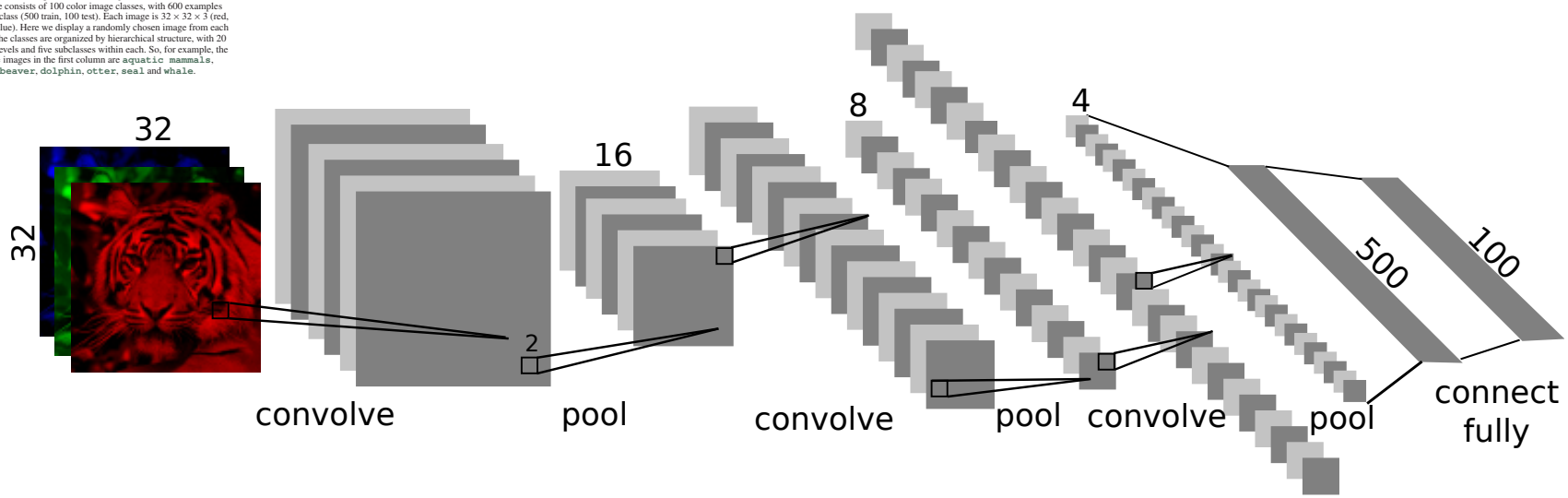


Figure 18.9 Architecture of a deep-learning network for the `CIFAR-100` image classification task. The input layer and

- 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

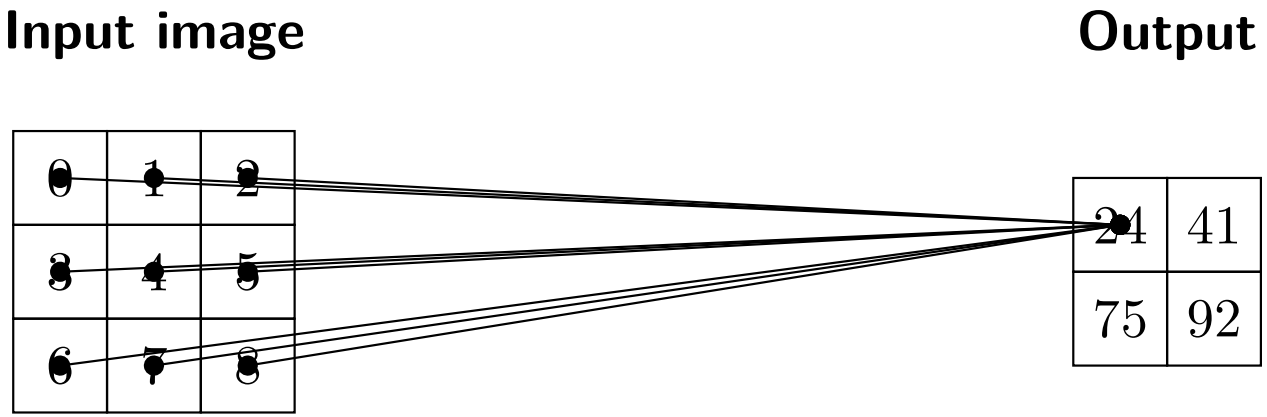
0	1	2
3	4	5
6	7	8

Output

24	41
75	92

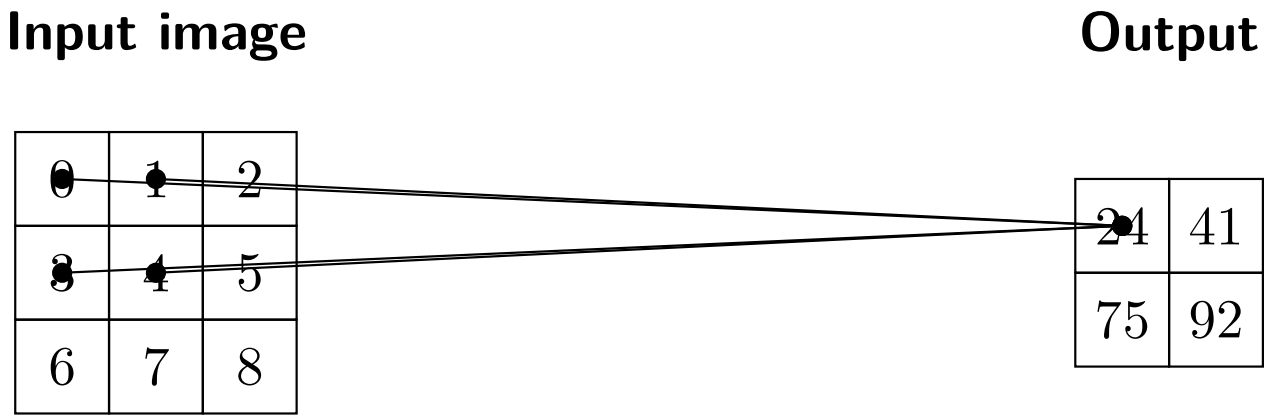
Fully connected layer: every output unit extracts information from the whole input image.

Convolutional Layers for Grayscale Images



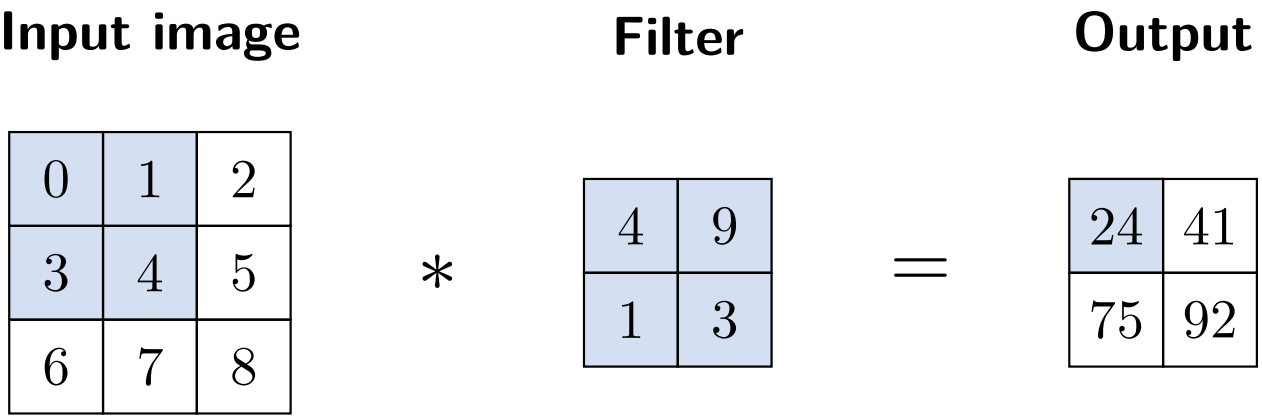
Fully connected layer: every output unit extracts information from the whole input image.

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



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

Input image

0	1	2
3	4	5
6	7	8

$*$

Filter

4	9
1	3

$=$

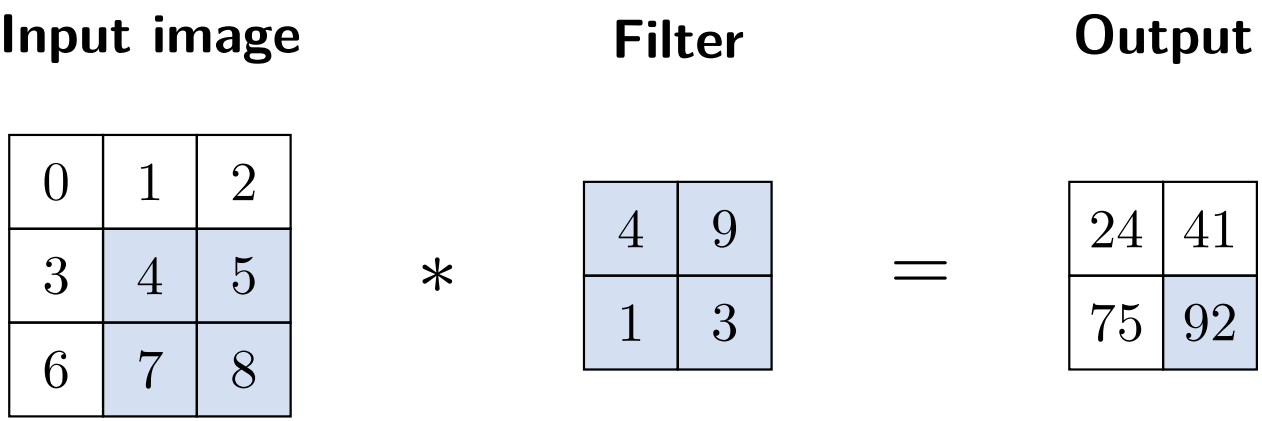
Output

24	41
75	92

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

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

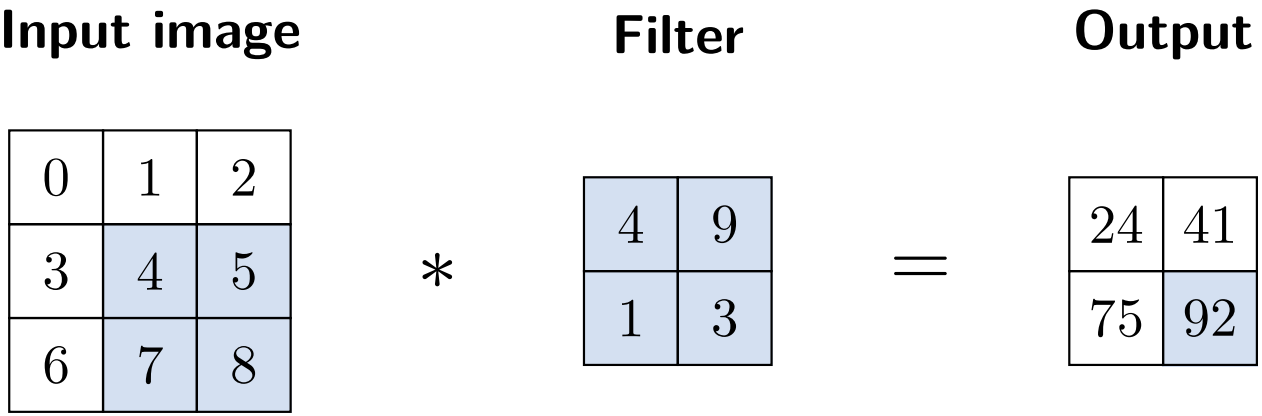
Convolutional Layers for Grayscale Images



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

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

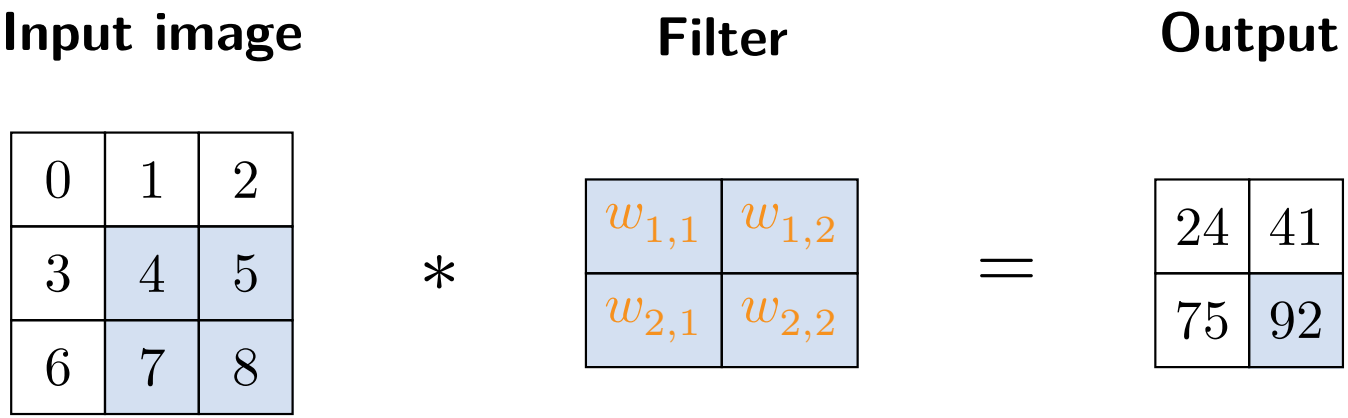


Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

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

Convolutional Layers for Grayscale Images

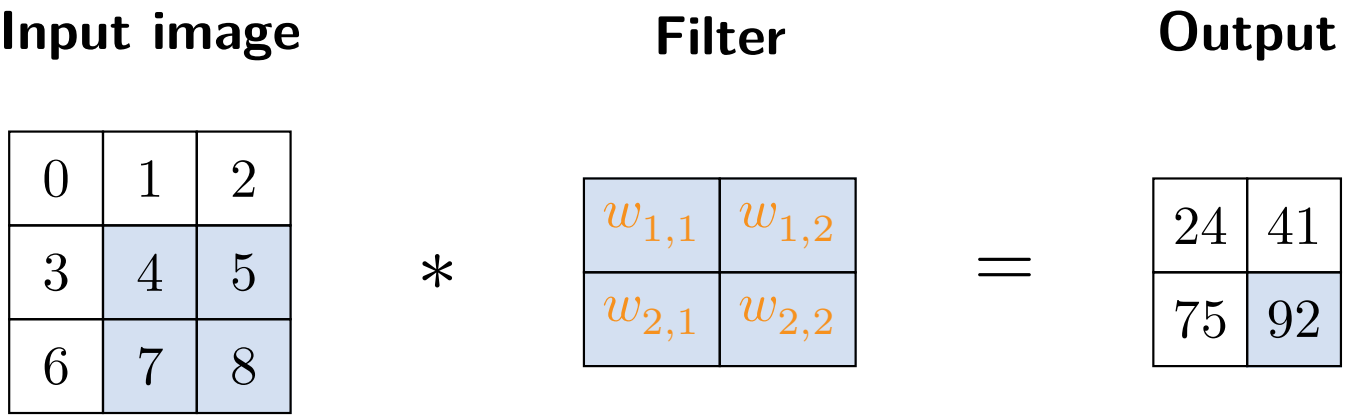


Convolutional layer: every output unit extracts information from a **local square region** in the input image, using weights from a **filter** matrix.

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

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						

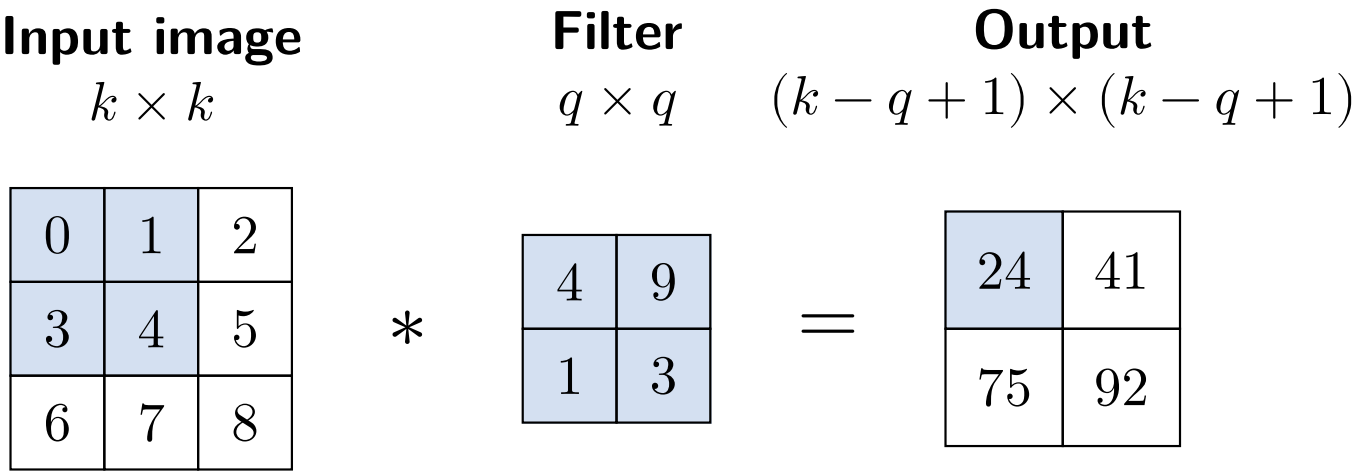
- **Very small** q, much smaller than k. For instance, $q = 2$ or $q = 3$. So output nearly same dimensions as input.

Dimensions and Padding

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.
- 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

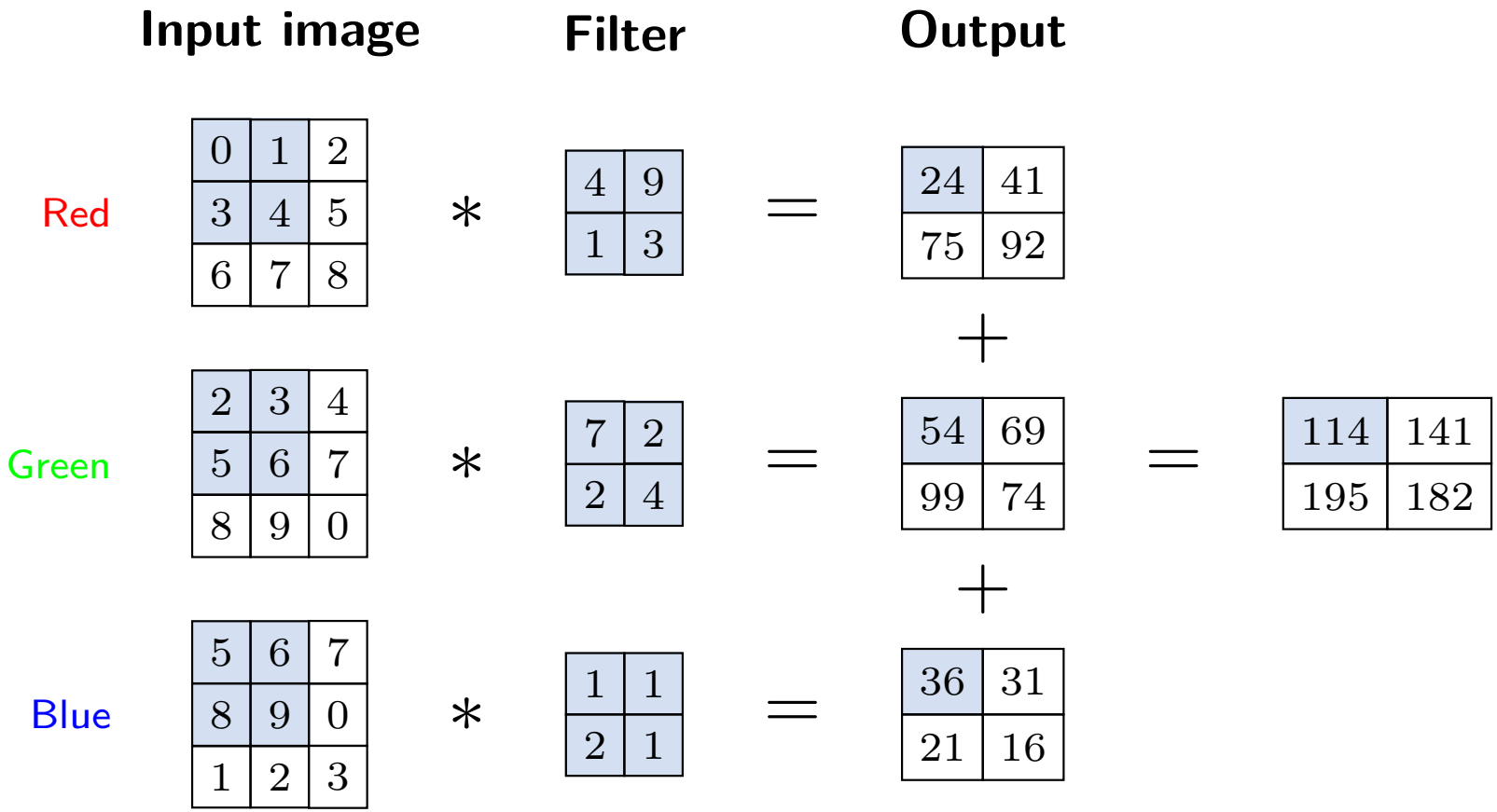
$=$

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

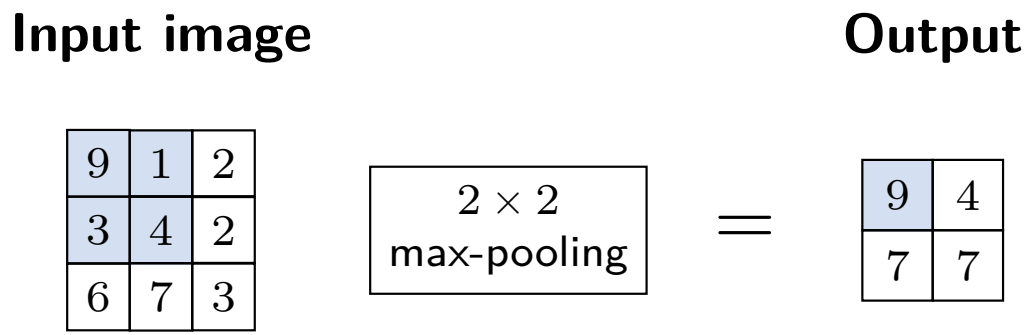
- Often want to extract different aspects from inputs. For instance, detect different features in image.
- Solution: duplicate the operation from the previous slide m times with **separate filters** to get m **output channels**
- SGD then optimizes filters to be jointly useful.

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**

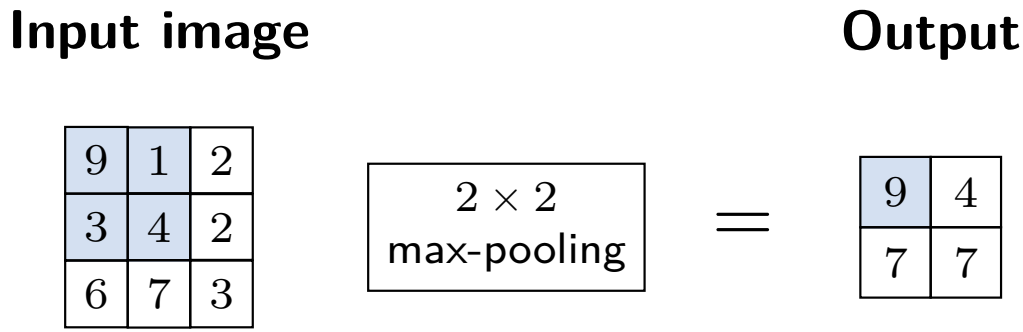
Pooling

- 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**



- **No parameters!**
- It is also possible, but less common nowadays, to take the average instead of the max: average pooling

High-level Network Structure

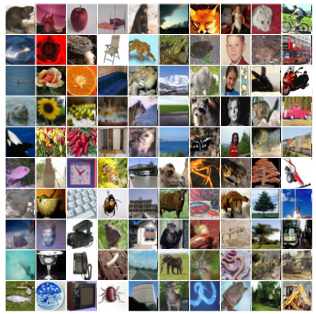


Figure 18.8 Examples of natural images. The `CIFAR-100` database consists of 100 color image classes, with 600 examples in each class (500 train, 100 test). Each image is $32 \times 32 \times 3$ (red, green, blue). Here we display a randomly chosen image from each class. The classes are organized by 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`.

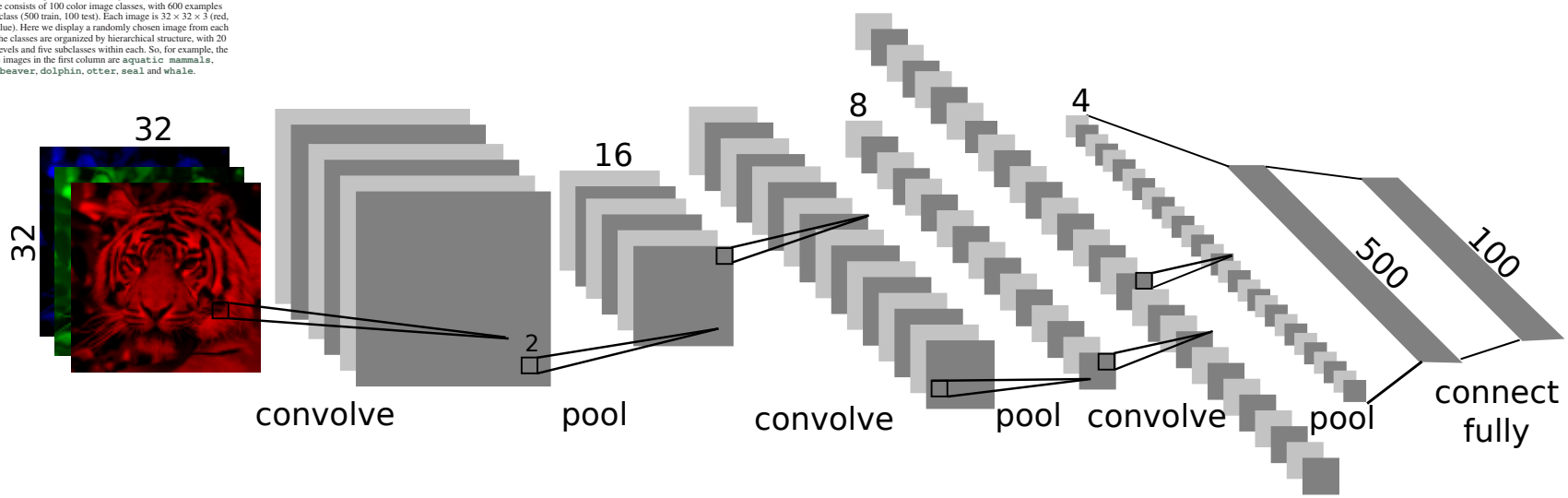


Figure 18.9 Architecture of a deep-learning network for the `CIFAR-100` image classification task. The input layer and

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

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

Outline

Optimization of Neural Networks

1. Backpropagation

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

Final Remarks

7. Remarks about ChatGPT

Q&A

Remarks about ChatGPT

```
Predict the next word in a [...]
```

Classification Task

- **Repeatedly predict the next word** to generate a long text
- **No reasoning, no notion of 'truth'**, just produce text with the same statistical patterns as human text
- Trained on essentially the whole internet
- Additional fine-tuning after training to make it be non-toxic, helpful to the user, etc.

Remarks about ChatGPT

```
Predict the next word in a [...]
```

Classification Task

- **Repeatedly predict the next word** to generate a long text
- **No reasoning, no notion of 'truth'**, just produce text with the same statistical patterns as human text
- Trained on essentially the whole internet
- Additional fine-tuning after training to make it be non-toxic, helpful to the user, etc.

Network Structure

- Based on special layer structure called a **transformer**.[1]
- Inputs: sequences of text of variable length
- Details of network not public, but estimated to have 1.7 trillion parameters

[1] Unfortunately, there was not enough time in this lecture to cover transformers

Outline

Optimization of Neural Networks

1. Backpropagation

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

Final Remarks

7. Remarks about ChatGPT

Q&A

Questions

Questions about any part of the course?