Deep Learning

Lecture Topic:
Introduction to Deep Learning for Computer Vision

Anxiao (Andrew) Jiang

Learning Objectives:

1. Understand how a convolutional neural network works.

2. Understand how to train a convolutional neural network using regularization, data augmentation, feature extraction, and fine tuning.

Roadmap of this lecture:

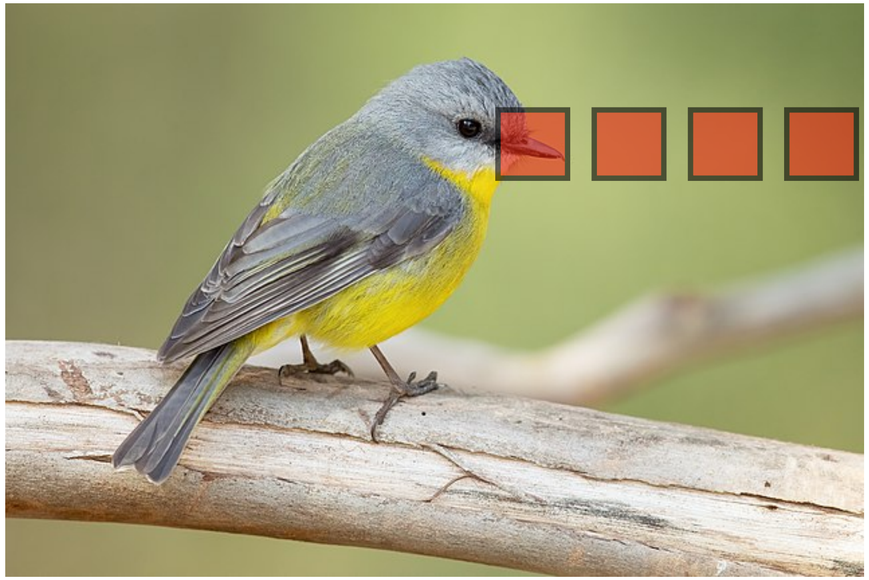
1. How a convolutional neural network (CNN) works
   - 1.1 CNN architecture
   - 1.2 How a convolution layer works
   - 1.3 How max-pooling and flatten layers work
2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

CNN for Images (or image-like data)

CNN for Images (or image-like data)

1. Some patterns are much smaller than the image.

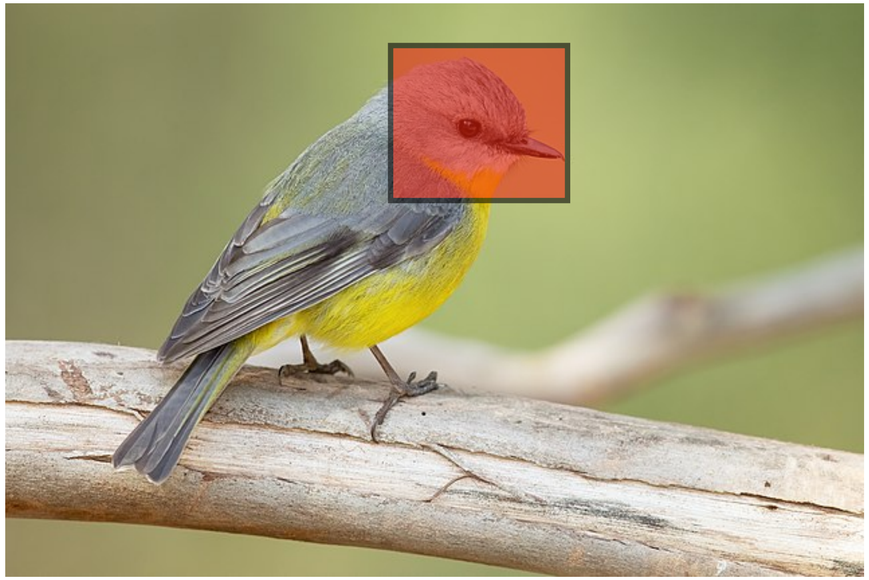
CNN for Images (or image-like data)



1. Some patterns are much smaller than the image.

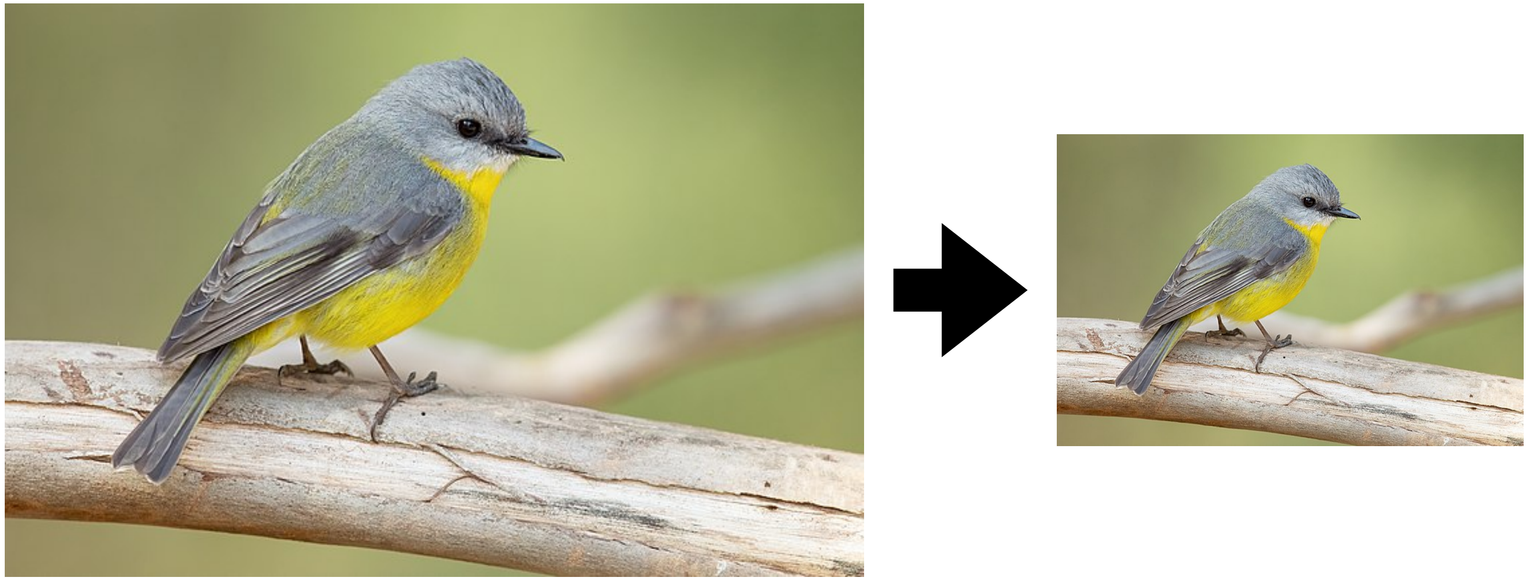
2. We can check different parts of the image to detect a pattern using the same “filter”.

CNN for Images (or image-like data)



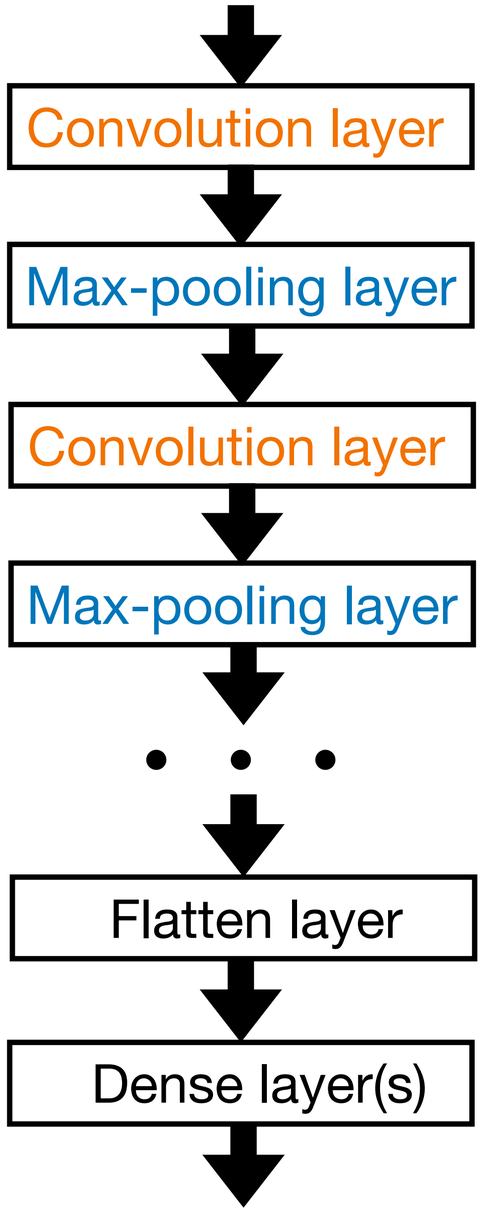
1. Some patterns are much smaller than the image.
2. We can check different parts of the image to detect a pattern using the same “filter”.
3. Small local patterns are combined to become a larger local pattern.

CNN for Images (or image-like data)



1. Some patterns are much smaller than the image.
2. We can check different parts of the image to detect a pattern using the same “filter”.
3. Small local patterns are combined to become a larger local pattern.
4. Subsampling an image can still enable us to detect a relatively large pattern.

CNN Architecture



Detect local features

Subsample image

Detect (larger) local features

Subsample image

Turn image to a vector

Get final result

Quiz questions:

1. What properties of images inspired the CNN architecture?

2. What is the basic architecture of CNN?

Roadmap of this lecture:

1. How a convolutional neural network (CNN) works
    - 1.1 CNN architecture
    - 1.2 How a convolution layer works
    - 1.3 How max-pooling and flatten layers work
2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

How a convolution layer works

Image

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

How a convolution layer works

Image

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

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

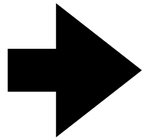
Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

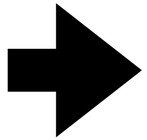
Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

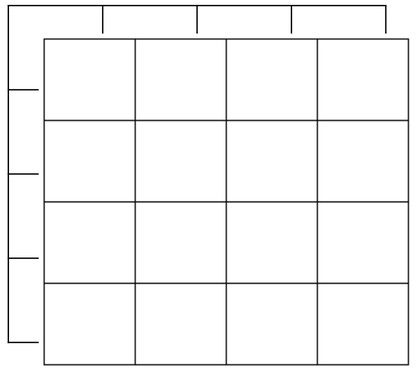
How a convolution layer works

Image

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



Feature map for filter 1



Feature map for filter 2

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

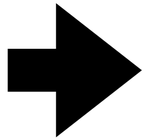
Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

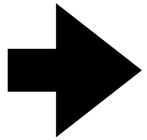
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1			

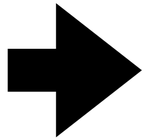
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0		

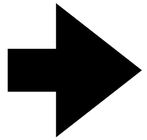
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	

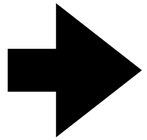
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3

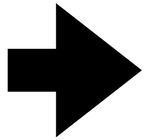
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2			

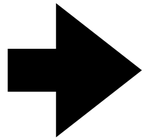
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3		

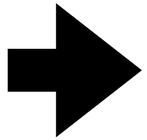
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	

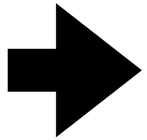
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5

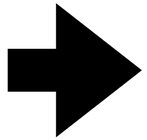
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1			

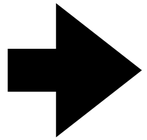
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0		

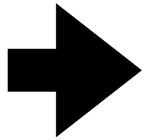
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	

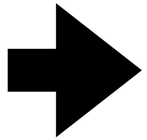
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1

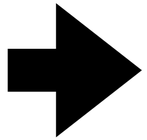
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1			

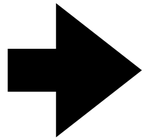
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2		

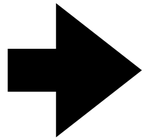
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	

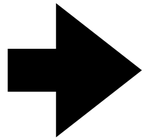
Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

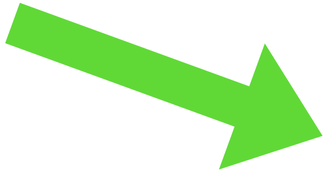
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3			

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

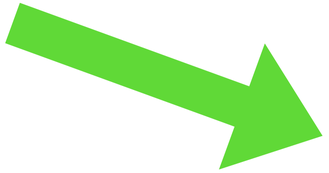
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2		

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

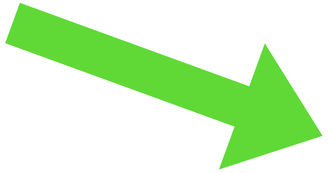
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

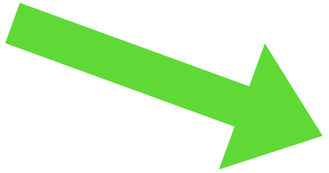
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

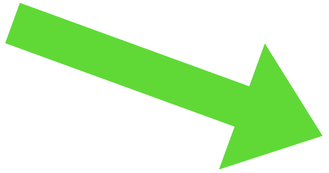
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2			

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

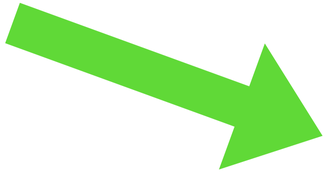
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3		

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

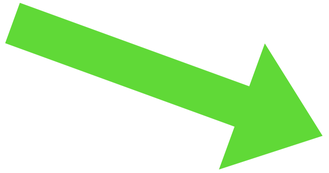
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

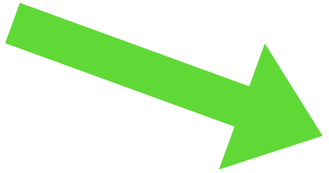
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

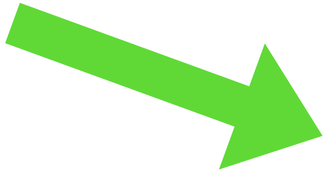
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3			

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

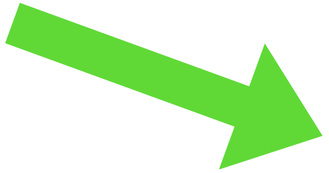
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0		

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

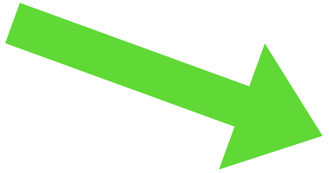
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

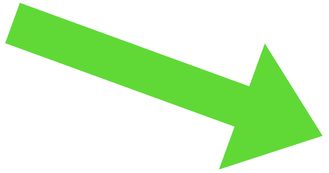
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	-3

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

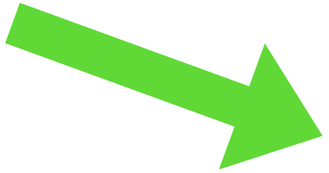
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	-3
3			

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

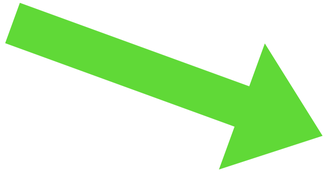
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	-3
3	-2		

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

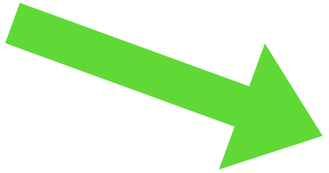
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	-3
3	-2	-4	

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

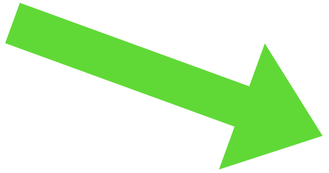
How a convolution layer works

Image

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

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3



Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	-3
3	-2	-4	1

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

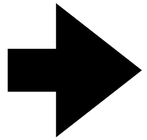
Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

How a convolution layer works

Image

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



Feature map for filter 1

-1	0	2	3	
-2	-3	-2	0	-1
-1	-2	-3	0	-1
-1	-3	0	-1	-3
	3	-2	-4	1

Feature map for filter 2

Filter 1

-1	-1	-1
1	**1**	**1**
-1	-1	-1

Filter 2

-1	-1	**1**
-1	**1**	-1
1	-1	-1

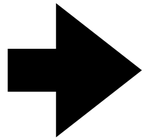
Each filter has 3x3 = 9 parameters, which are shared by 4x4 = 16 neurons.

For large images, the saving is big compared to full-connected networks.

How a convolution layer works

Image

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



Feature map for filter 1

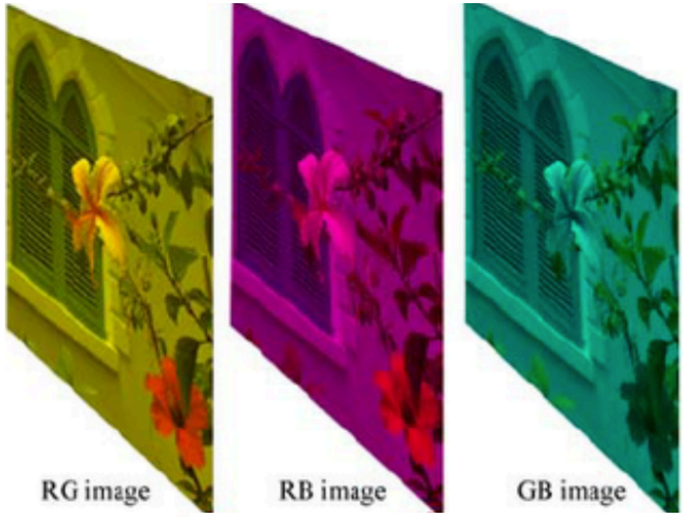
-1	0	2	3
-2	-3	-2	0	-1
-1	-2	-3	0	-1
-1	-3	0	-1	-3
	3	-2	-4	1

Feature map for filter 2

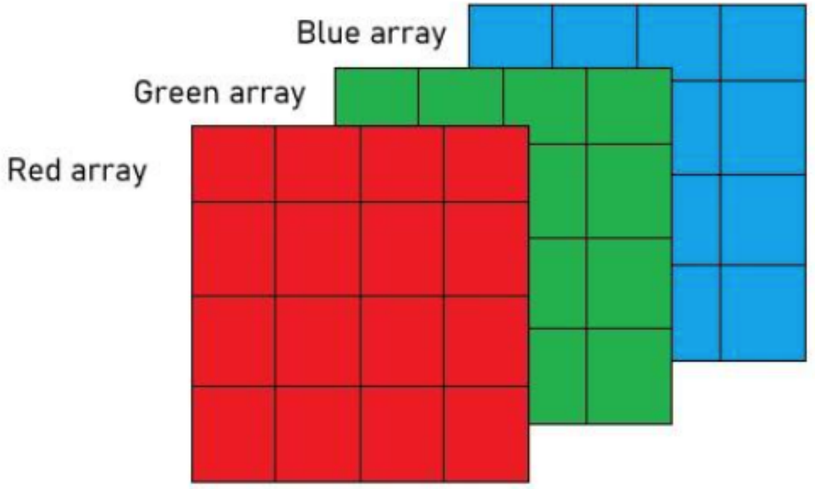
An image of depth 2, also called “an image of 2 channels”.

Number of channels equals the number of convolution filters.

A colored image is an image of 3 channels

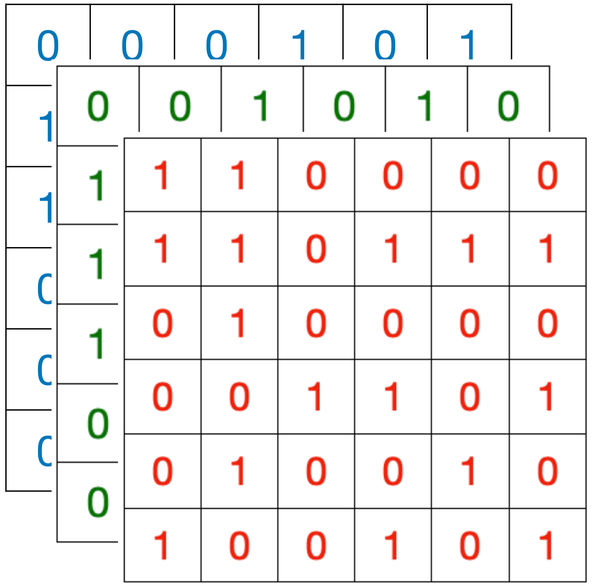


RGB Image

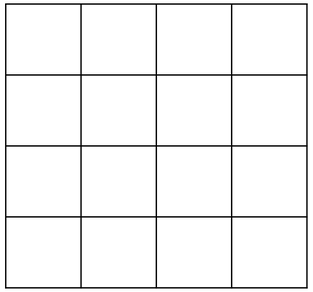


Depth of a convolution filter = # of channels in the image

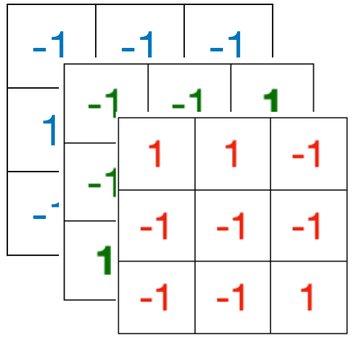
Image



Feature map



Convolution Filter



Quiz questions:

1. How does convolution work in a convolution layer?

2. Why does a convolution layer have fewer parameters to train than a fully-connected layer?

3. What is the relation between the number of channels in an image (or feature map) and the depth of a convolution filter?

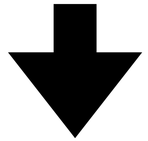
Roadmap of this lecture:

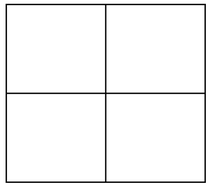
1. How a convolutional neural network (CNN) works
    - 1.1 CNN architecture
    - 1.2 How a convolution layer works
    - 1.3 How max-pooling and flatten layers work
2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

How a max-pooling layer works

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3

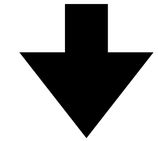


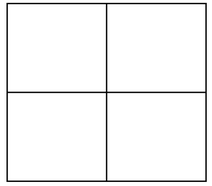


Subsampled feature map for filter 1

Feature map for filter 2

-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	-3
3	-2	-4	1



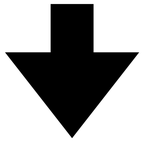


Subsampled feature map for filter 2

How a max-pooling layer works

Feature map for filter 1

-1	0	2	3
-2	-3	-4	-5
-1	0	1	1
-1	-2	-2	-3

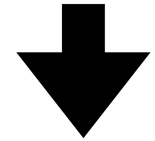


0	3
0	1

Subsampled feature map for filter 1

Feature map for filter 2

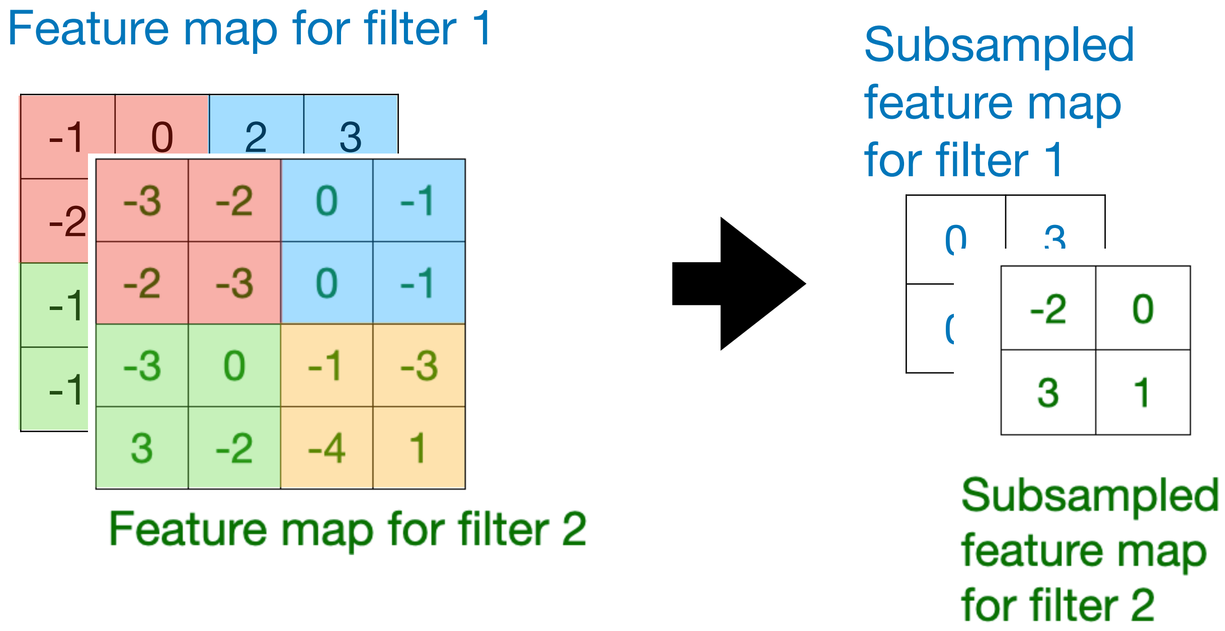
-3	-2	0	-1
-2	-3	0	-1
-3	0	-1	-3
3	-2	-4	1



-2	0
3	1

Subsampled feature map for filter 2

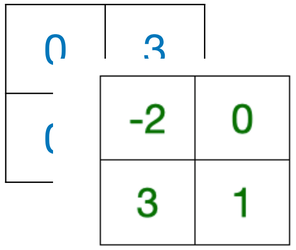
How a max-pooling layer works



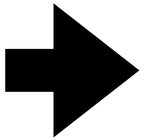
The “image” becomes smaller after max-pooling, but the number of channels is unchanged.

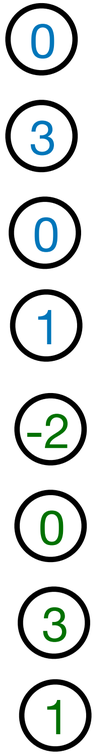
How a flatten layer works

Subsampled feature map for filter 1



Subsampled feature map for filter 2





Quiz questions:

1. How does a max-pooling layer work?

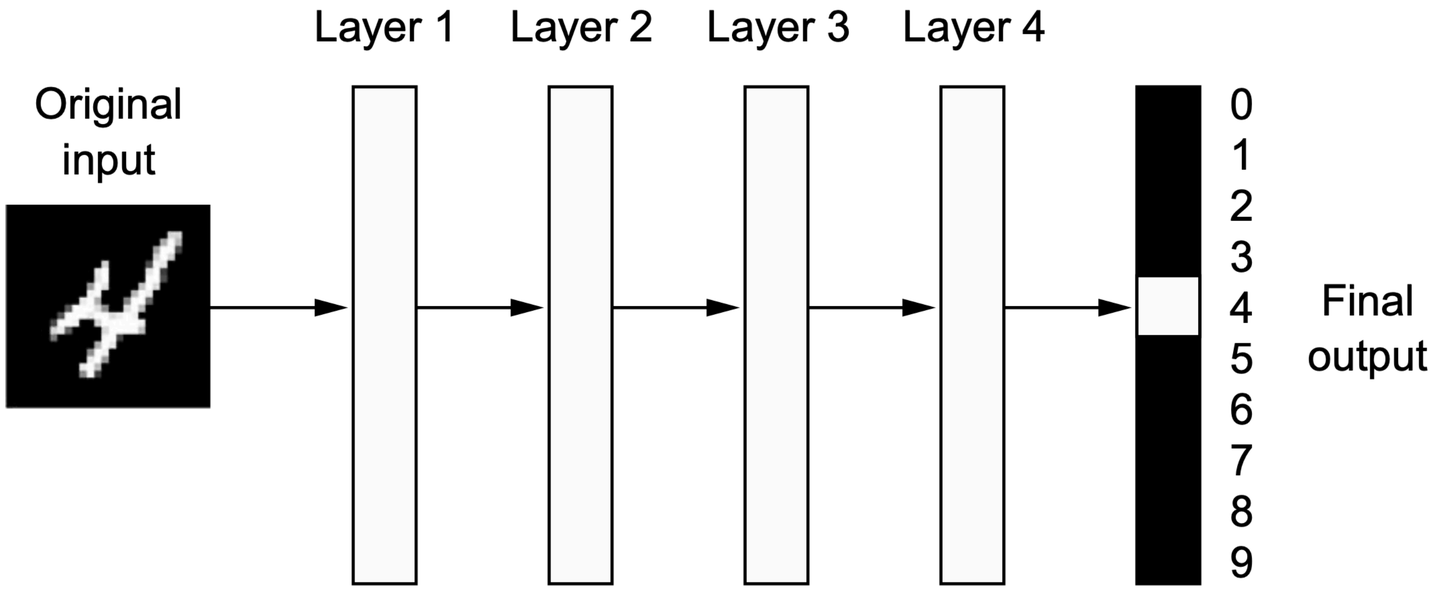
2. How does a flatten layer work?

Roadmap of this lecture:

1. How a convolutional neural network (CNN) works
    - 1.1 CNN architecture
    - 1.2 How a convolution layer works
    - 1.3 How max-pooling and flatten layers work

2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

Example of CNN: for MNIST



Example of CNN: for MNIST

Listing 8.1 Instantiating a small convnet

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

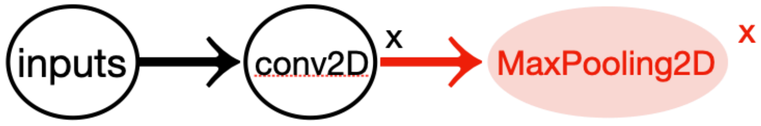
inputs

Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

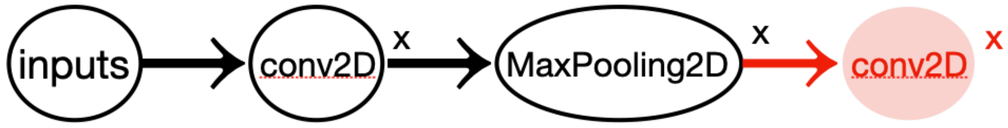
Example of CNN: for MNIST

```
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

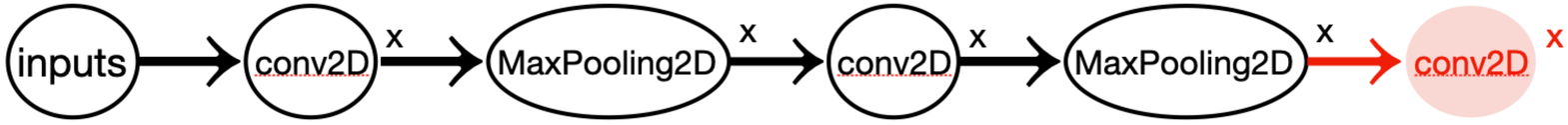


Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

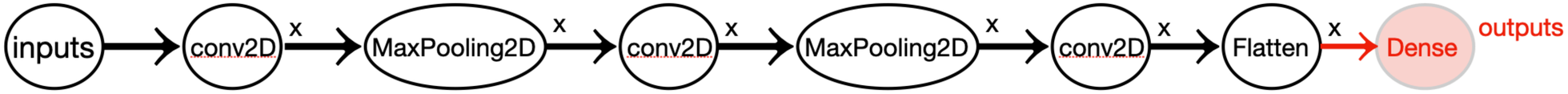


Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

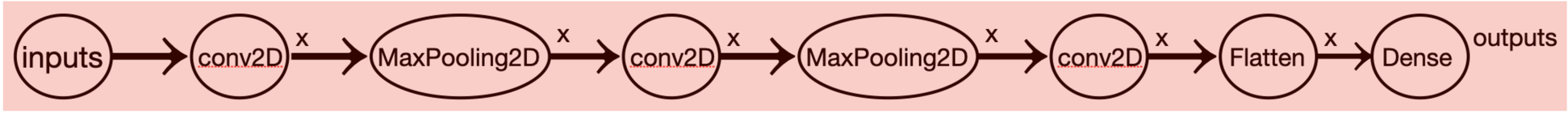
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



Example of CNN: for MNIST

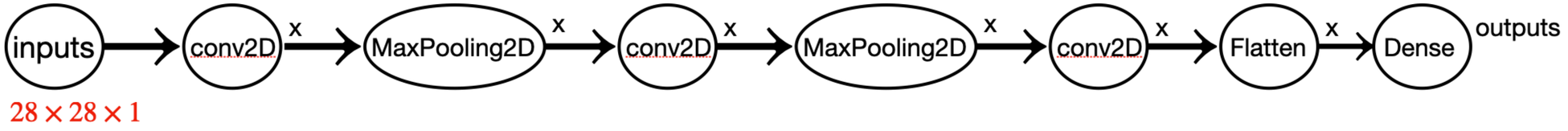
```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



model

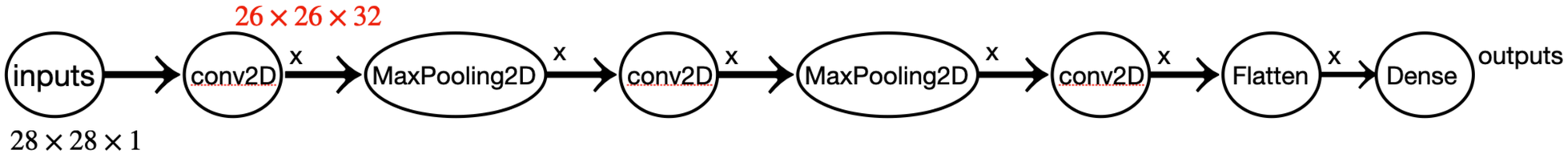
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



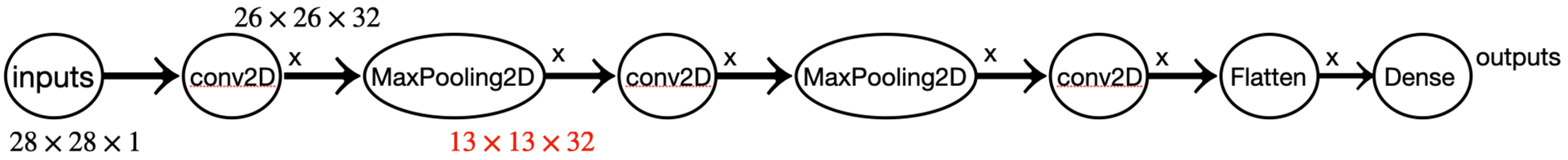
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



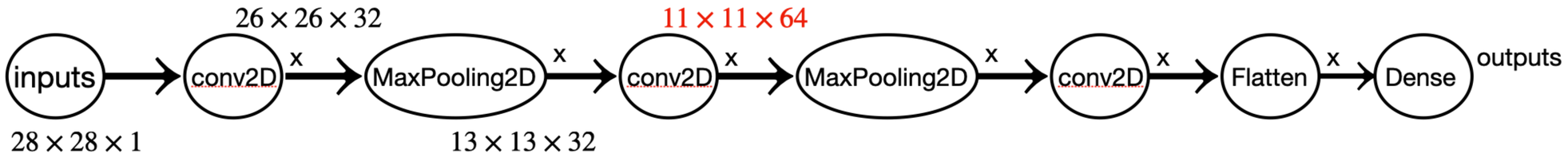
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



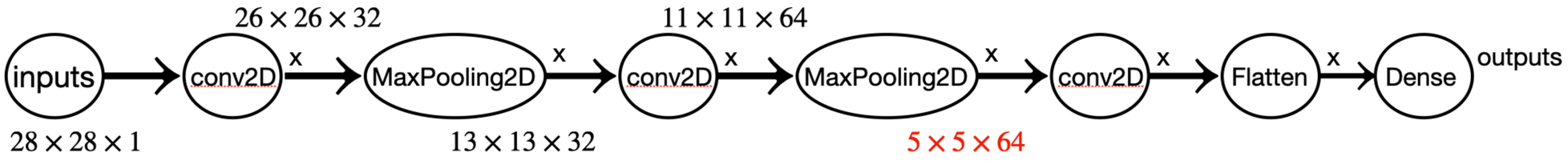
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



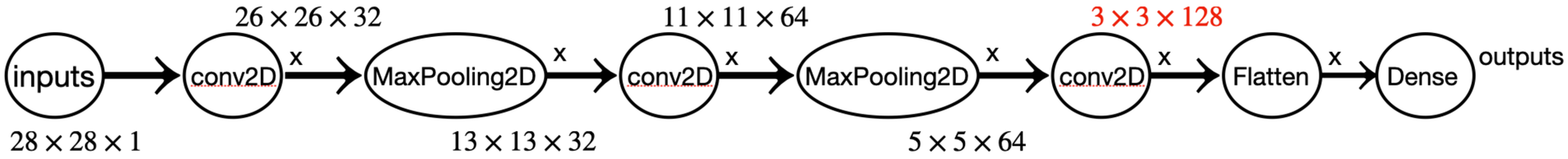
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



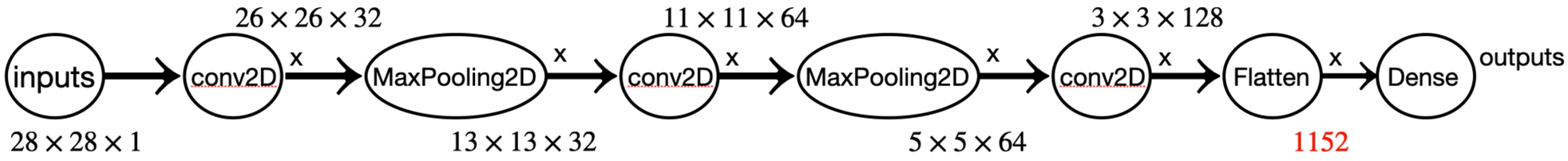
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



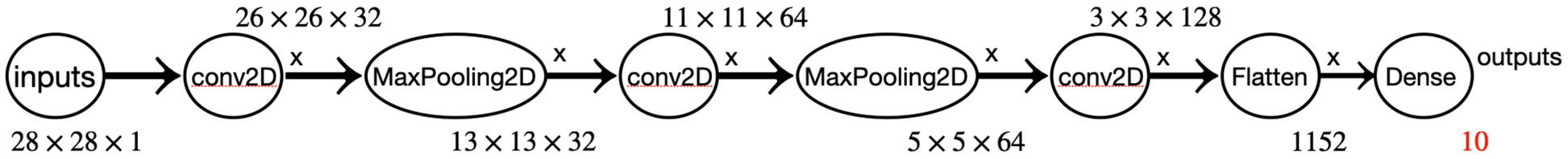
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



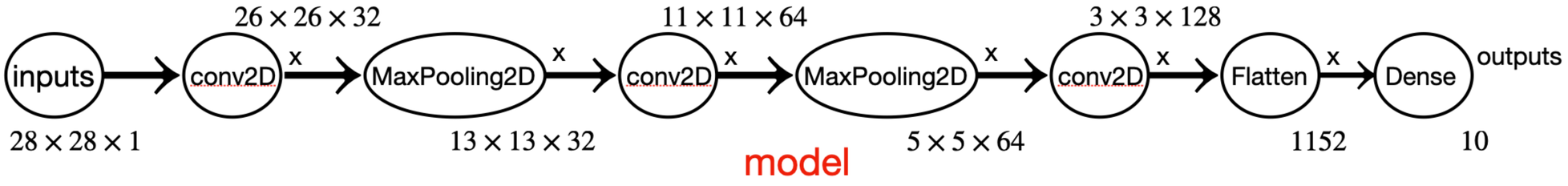
Example of CNN: for MNIST

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```



Example of CNN: for MNIST

```
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```



Example of CNN: for MNIST

Listing 8.4 Evaluating the convnet

```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"Test accuracy: {test_acc:.3f}")
Test accuracy: 0.991
```

Good performance

Understanding boarder effects and padding

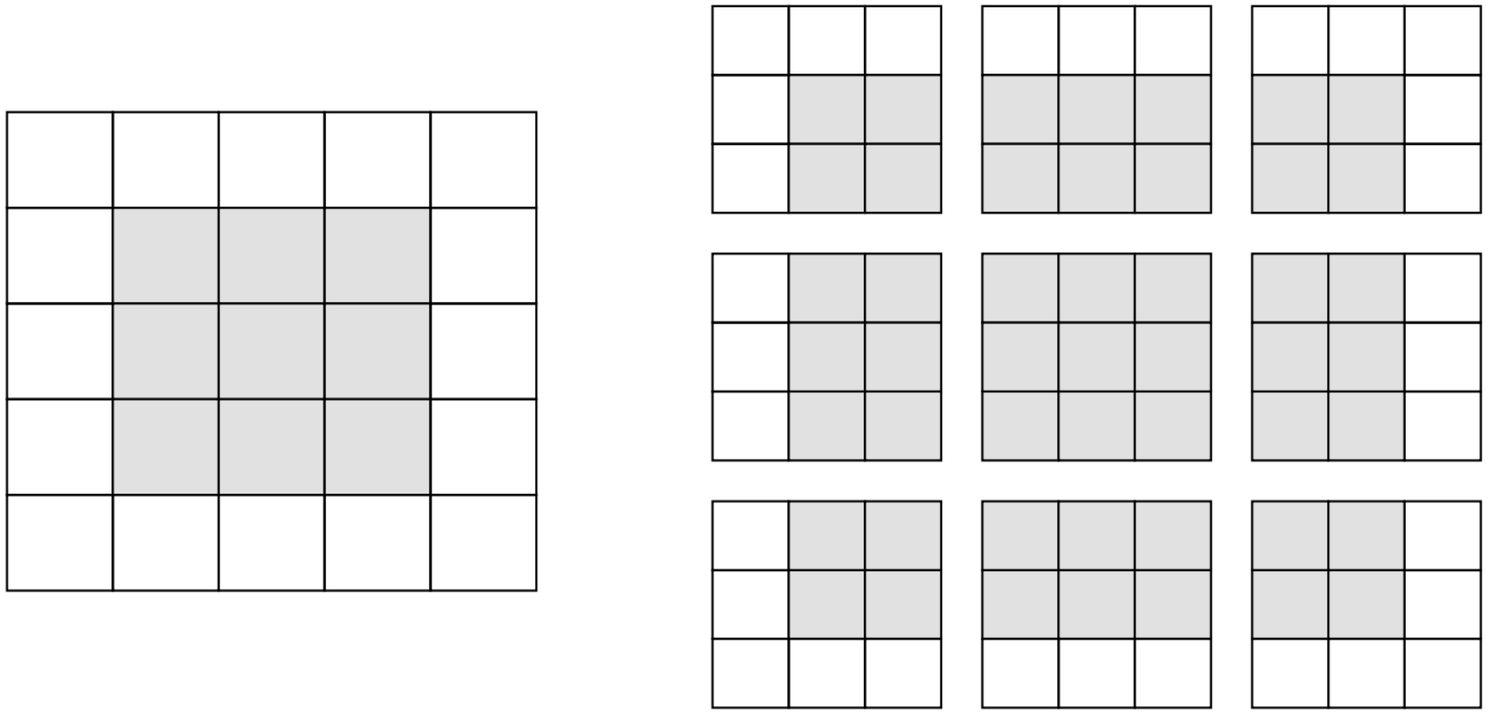


Figure 8.5 Valid locations of 3 × 3 patches in a 5 × 5 input feature map

Understanding boarder effects and padding

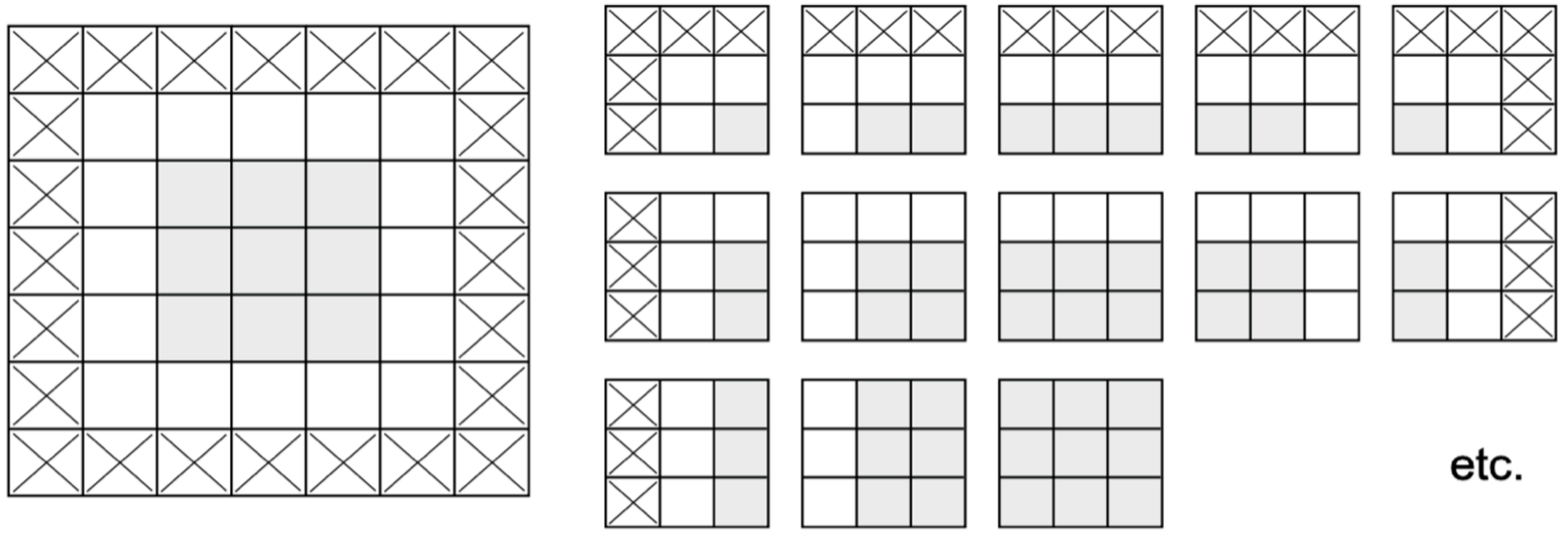


Figure 8.6 Padding a 5 × 5 input in order to be able to extract 25 3 × 3 patches

In `Conv2D` layers, padding is configurable via the `padding` argument, which takes two values: `"valid"`, which means no padding (only valid window locations will be used), and `"same"`, which means "pad in such a way as to have an output with the same width and height as the input." The `padding` argument defaults to `"valid"`.

Understanding convolution strides

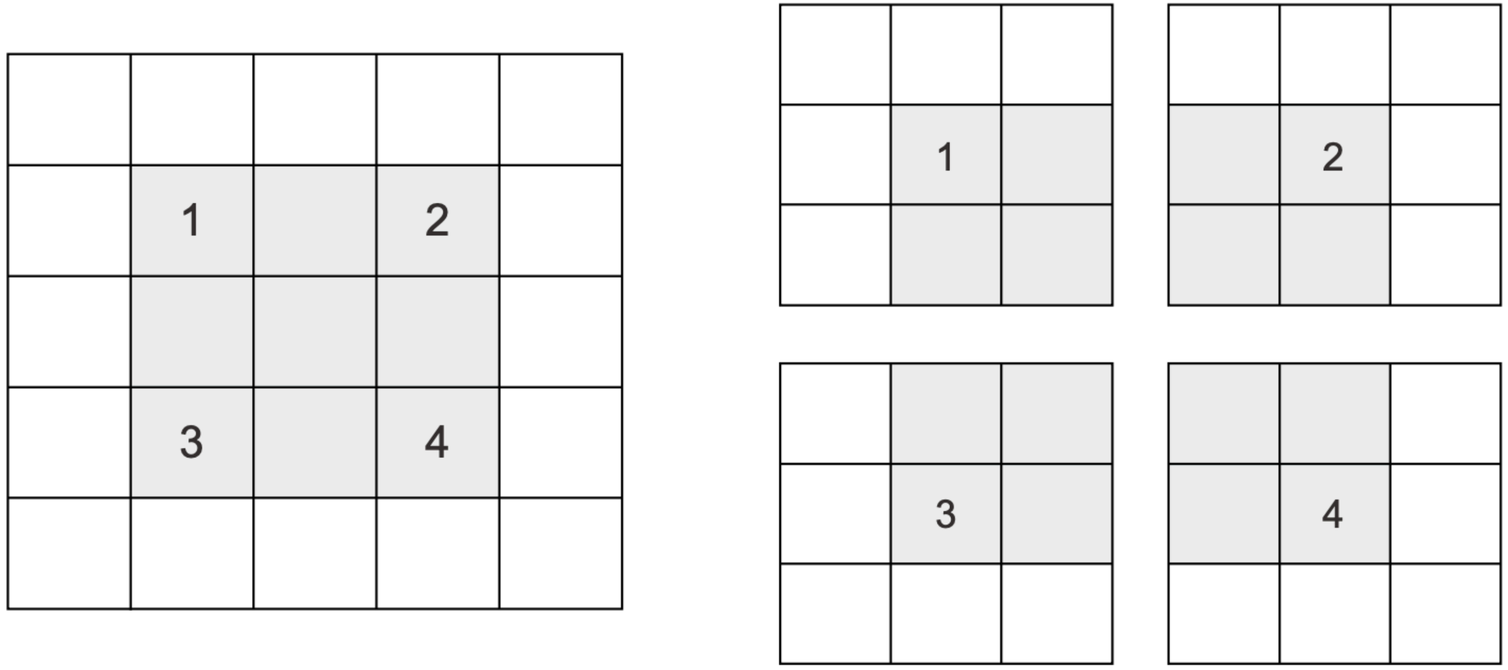


Figure 8.7 3 × 3 convolution patches with 2 × 2 strides

Quiz questions:

1. How to design a CNN using Keras?

2. What is the border effect for a convolution layer?

3. What is stride for convolution?

Roadmap of this lecture:

1. How a convolutional neural network (CNN) works
    - 1.1 CNN architecture
    - 1.2 How a convolution layer works
    - 1.3 How max-pooling and flatten layers work
2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

Task: dog-versus-cat image recognition

model

cat

model

dog

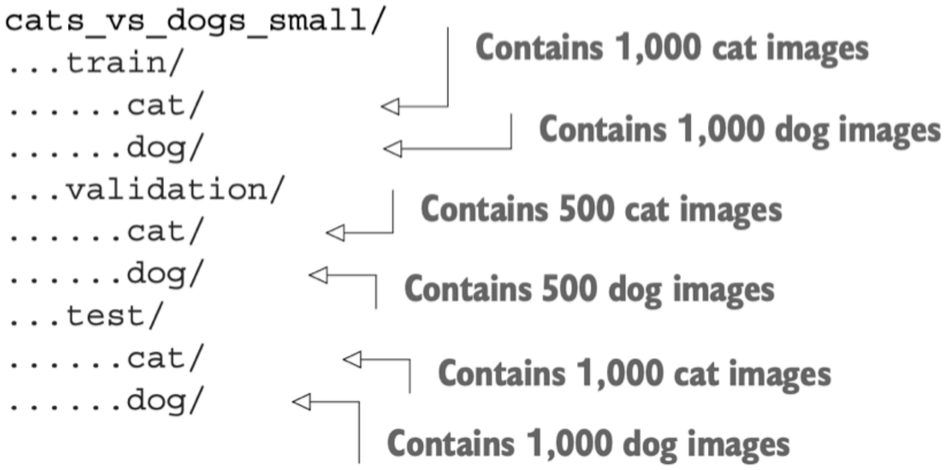
Task: dog-versus-cat image recognition

We can download the dataset in Kaggle.

Figure 8.8 Samples from the Dogs vs. Cats dataset. Sizes weren't modified: the samples come in different sizes, colors, backgrounds, etc.

This dataset contains 25,000 images of dogs and cats (12,500 from each class) and is 543 MB (compressed). After downloading and uncompressing the data, we'll create a new dataset containing three subsets: a training set with 1,000 samples of each class, a validation set with 500 samples of each class, and a test set with 1,000 samples of each class. Why do this? Because many of the image datasets you'll encounter in your career only contain a few thousand samples, not tens of thousands. Having more data available would make the problem easier, so it's good practice to learn with a small dataset.

The subsampled dataset we will work with will have the following directory structure:



Build the model

Listing 8.7 Instantiating a small convnet for dogs vs. cats classification

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

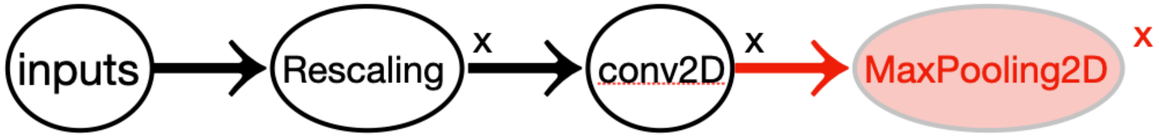
Rescale inputs to the [0, 1] range by dividing them by 255.

Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.



Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

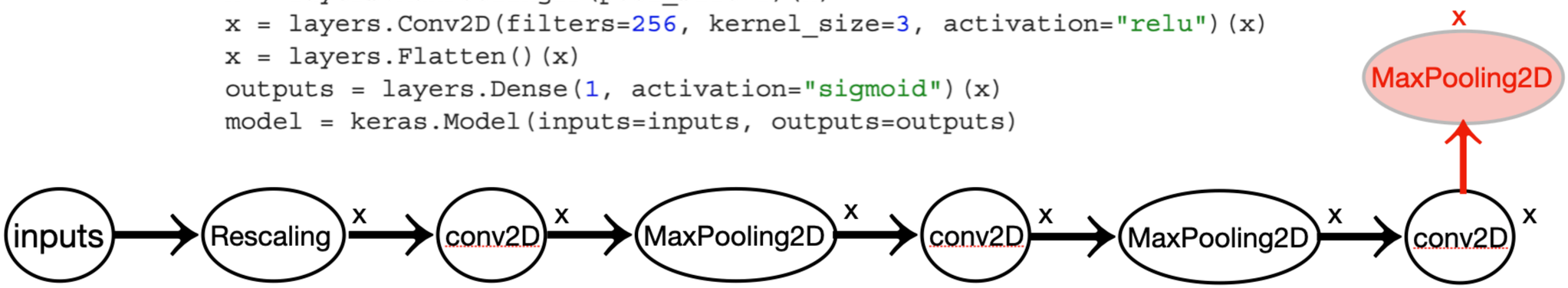
Rescale inputs to the [0, 1] range by dividing them by 255.

Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

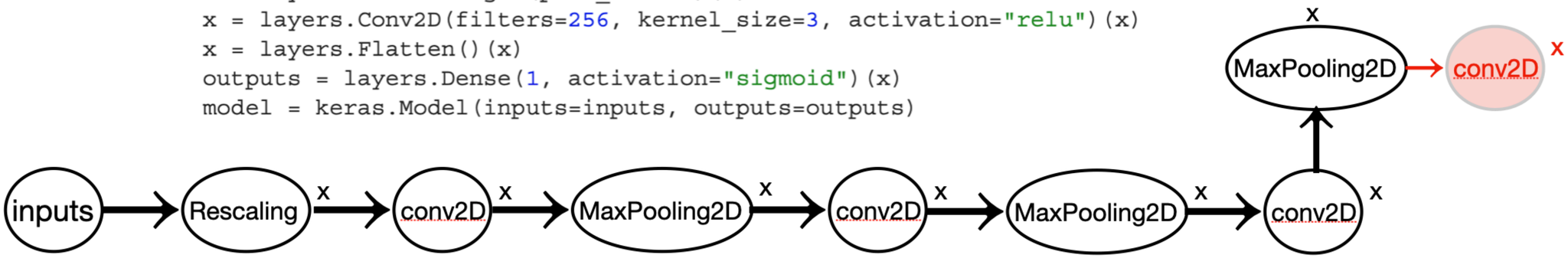


Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

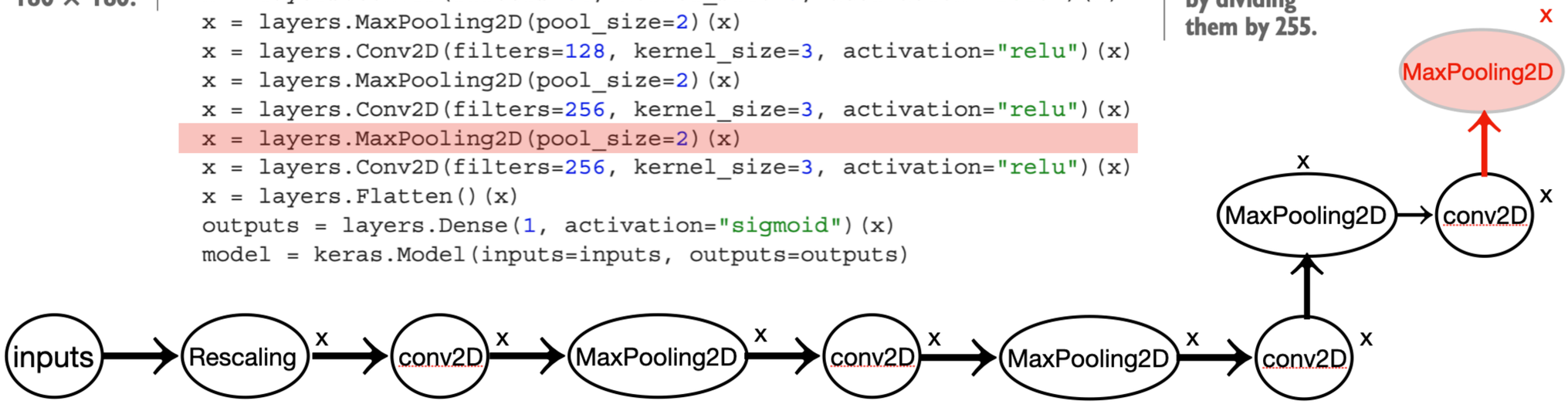


Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

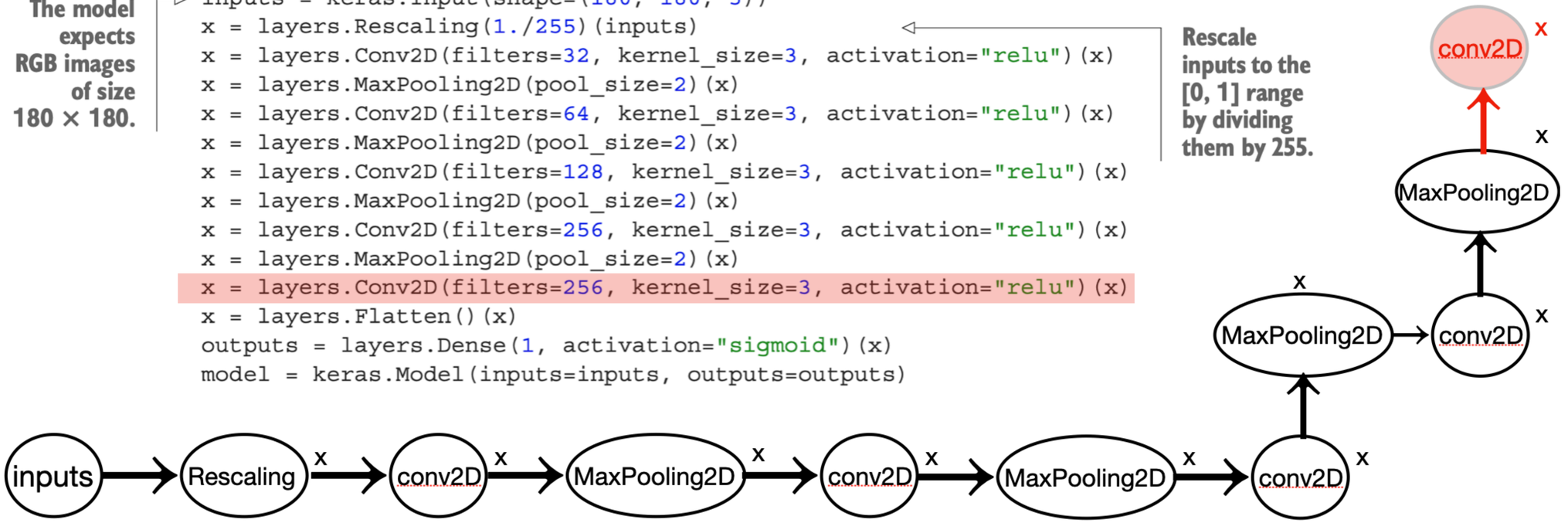


Build the model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.



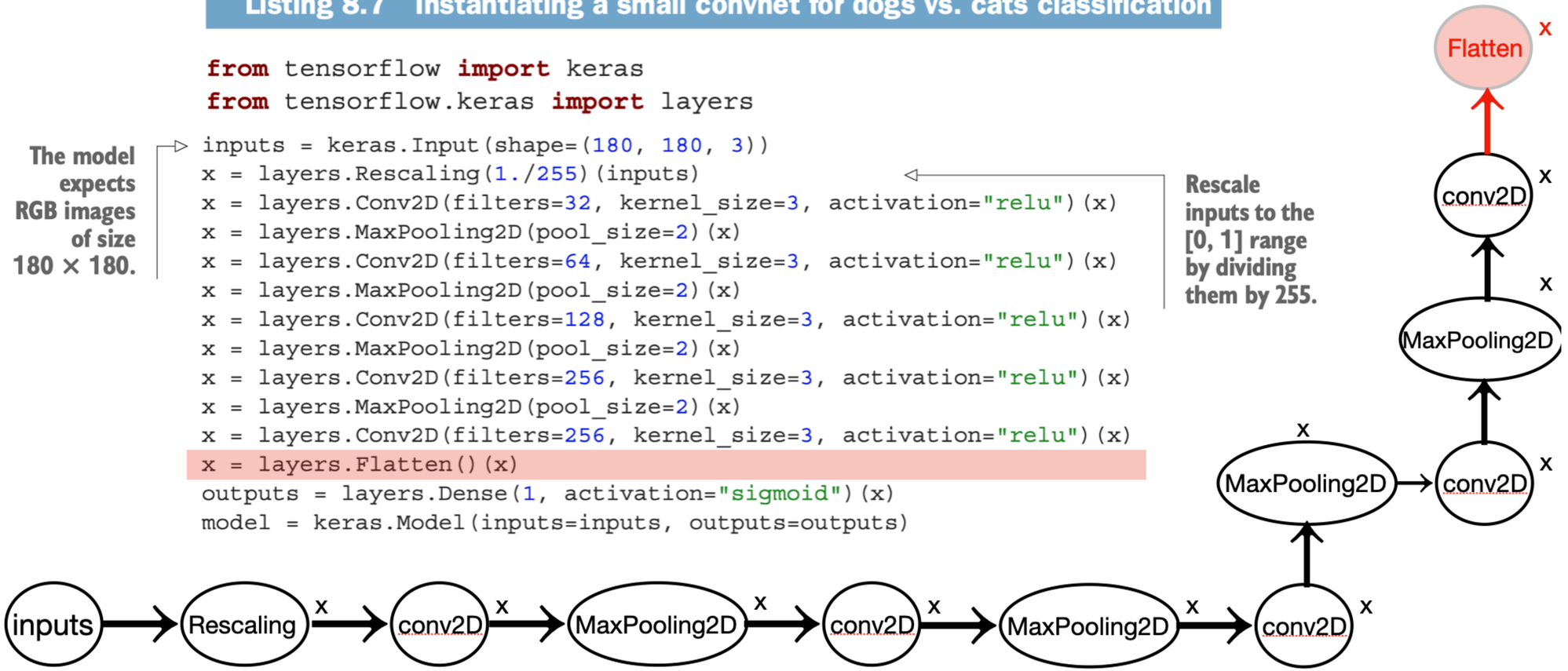
Build the model

Listing 8.7 Instantiating a small convnet for dogs vs. cats classification

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.



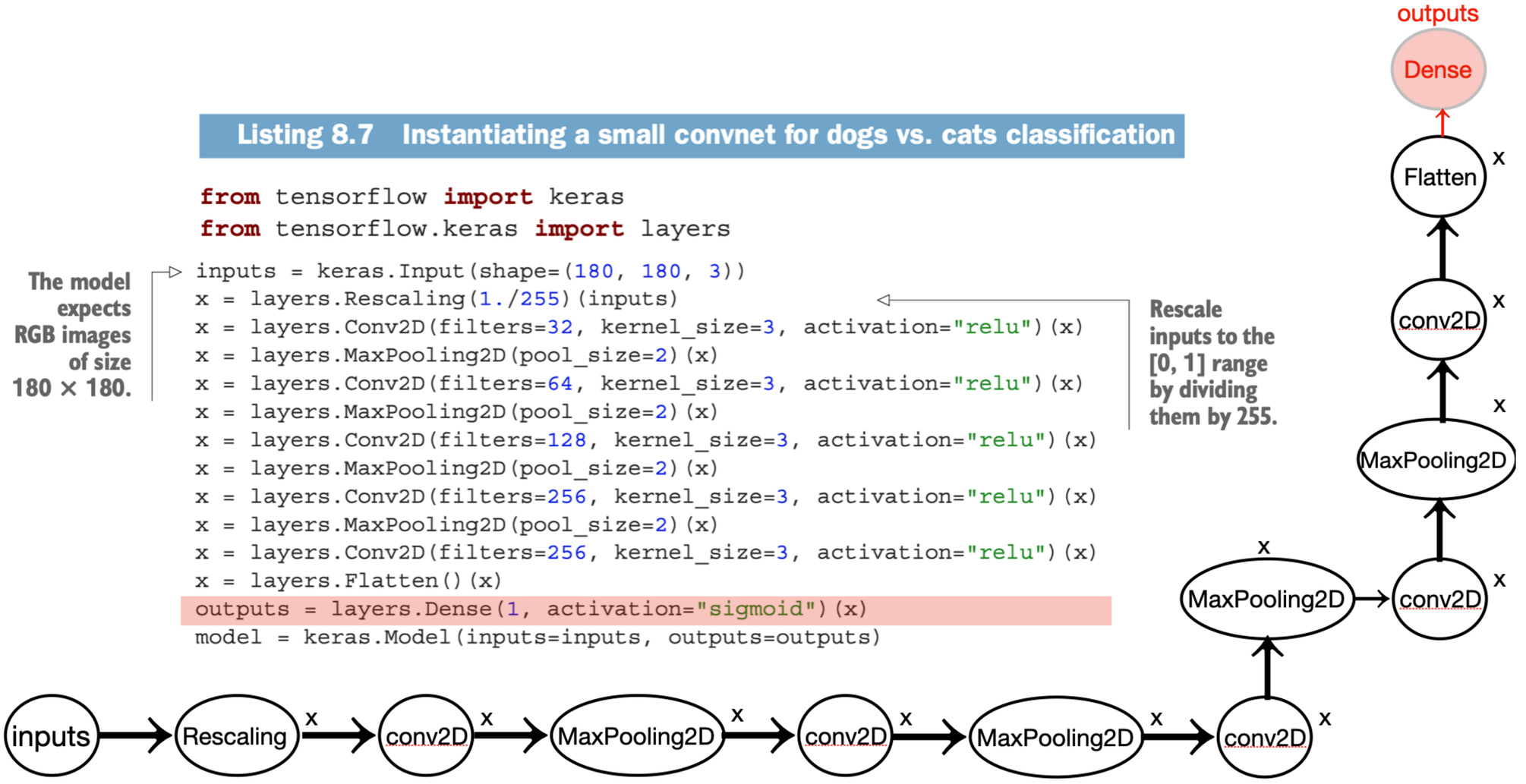
Build the model

Listing 8.7 Instantiating a small convnet for dogs vs. cats classification

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.



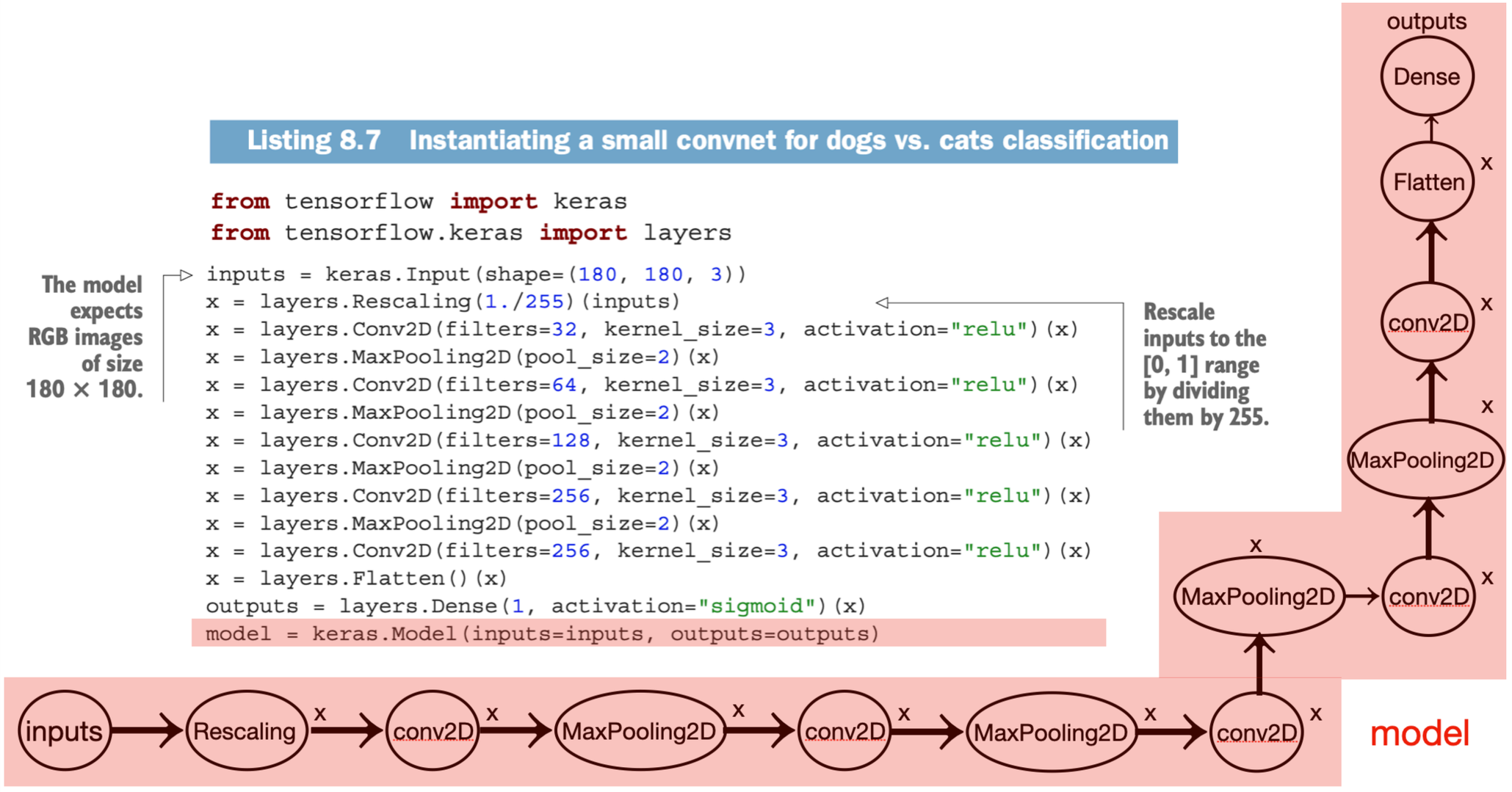
Build the model

Listing 8.7 Instantiating a small convnet for dogs vs. cats classification

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.



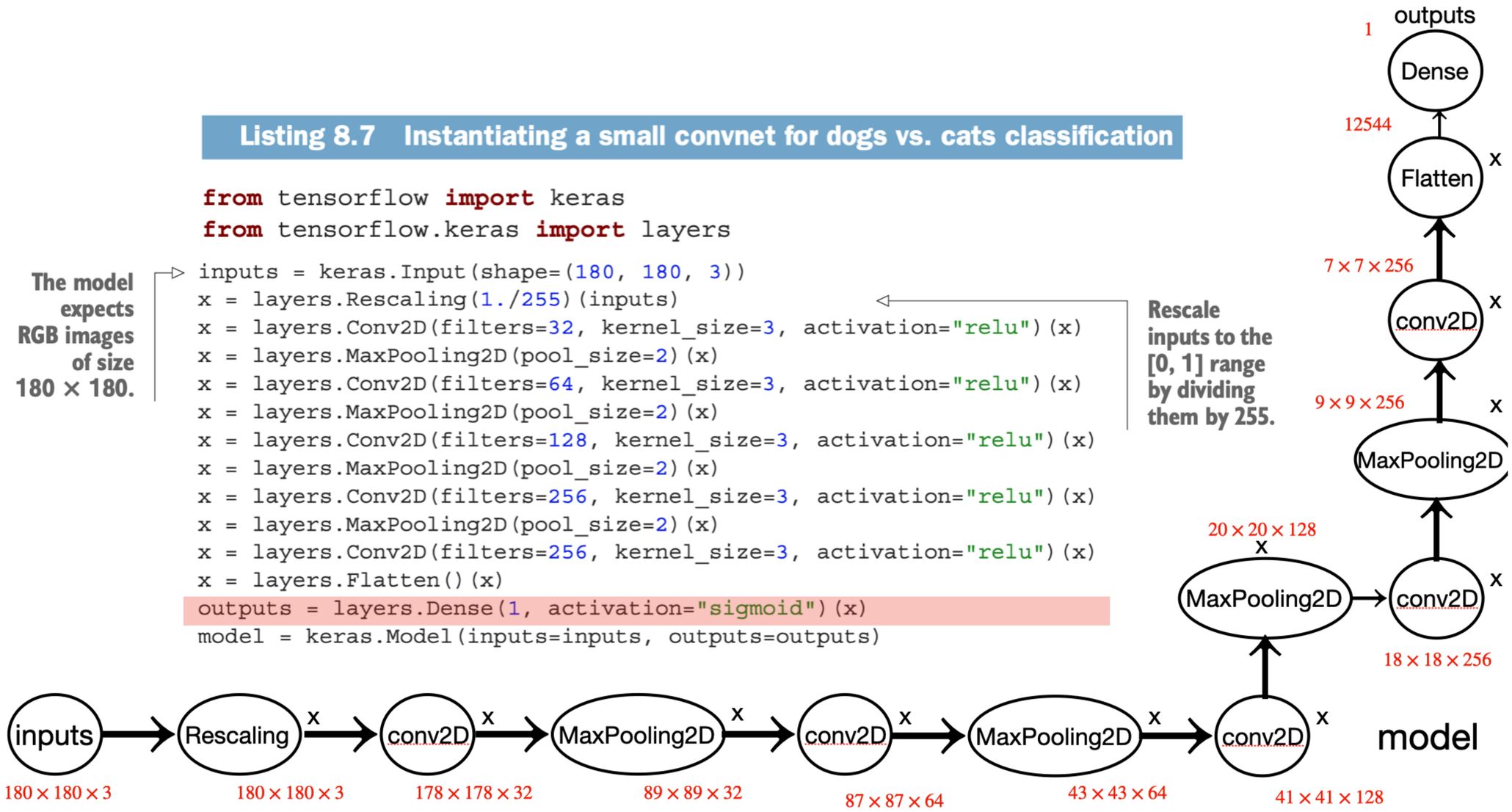
Build the model

Listing 8.7 Instantiating a small convnet for dogs vs. cats classification

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.



Compile the model

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Data preprocessing

As you know by now, data should be formatted into appropriately preprocessed floating-point tensors before being fed into the model. Currently, the data sits on a drive as JPEG files, so the steps for getting it into the model are roughly as follows:

1. Read the picture files.
2. Decode the JPEG content to RGB grids of pixels.
3. Convert these into floating-point tensors.
4. Resize them to a shared size (we'll use 180 × 180).
5. Pack them into batches (we'll use batches of 32 images).

It may seem a bit daunting, but fortunately Keras has utilities to take care of these steps automatically. In particular, Keras features the utility function `image_dataset_from_directory()`, which lets you quickly set up a data pipeline that can automatically turn image files on disk into batches of preprocessed tensors. This is what we'll use here.

Data preprocessing

Calling `image_dataset_from_directory(directory)` will first list the subdirectories of `directory` and assume each one contains images from one of our classes. It will then index the image files in each subdirectory. Finally, it will create and return a `tf.data.Dataset` object configured to read these files, shuffle them, decode them to tensors, resize them to a shared size, and pack them into batches.

Listing 8.9 Using `image_dataset_from_directory` to read images

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

Data preprocessing

Let's look at the output of one of these `Dataset` objects: it yields batches of 180 × 180 RGB images (shape `(32, 180, 180, 3)`) and integer labels (shape `(32,)`). There are 32 samples in each batch (the batch size).

Listing 8.10 Displaying the shapes of the data and labels yielded by the `Dataset`

```
>>> for data_batch, labels_batch in train_dataset:
>>>     print("data batch shape:", data_batch.shape)
>>>     print("labels batch shape:", labels_batch.shape)
>>>     break
data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
```

Train the model

Listing 8.11 Fitting the model using a `Dataset`

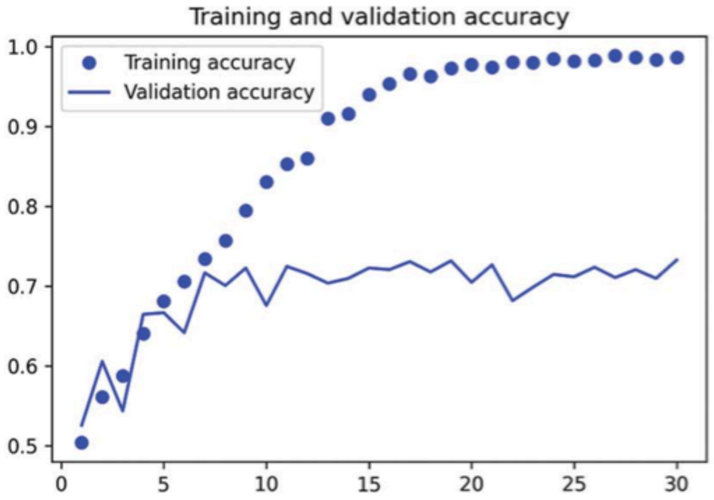
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]


history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Train the model:
plot loss and accuracy during training

Listing 8.12 Displaying curves of loss and accuracy during training

```
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```

plot loss and accuracy during training



Test performance

```
test_model = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

We get a test accuracy of 69.5%.

Quiz question:

1. How to prepare data for a general image dataset?

Roadmap of this lecture:

1. How a convolutional neural network (CNN) works
    - 1.1 CNN architecture
    - 1.2 How a convolution layer works
    - 1.3 How max-pooling and flatten layers work
2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

Use data augmentation

data augmentation can be done by adding a number of *data augmentation layers* at the start of your model. Let's get started with an example: the following Sequential model chains several random image transformations. In our model, we'd include it right before the `Rescaling` layer.

Listing 8.14 Define a data augmentation stage to add to an image model

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

Use data augmentation

These are just a few of the layers available (for more, see the Keras documentation). Let's quickly go over this code:

- `RandomFlip("horizontal")`—Applies horizontal flipping to a random 50% of the images that go through it
- `RandomRotation(0.1)`—Rotates the input images by a random value in the range [–10%, +10%] (these are fractions of a full circle—in degrees, the range would be [–36 degrees, +36 degrees])
- `RandomZoom(0.2)`—Zooms in or out of the image by a random factor in the range [-20%, +20%]

Use data augmentation

Listing 8.15 Displaying some randomly augmented training images

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

We can use take(N) to only sample *N* batches from the dataset. This is equivalent to inserting a break in the loop after the *N*th batch.

Apply the augmentation stage to the batch of images.

Display the first image in the output batch. For each of the nine iterations, this is a different augmentation of the same image.

Figure 8.10 Generating variations of a very good boy via random data augmentation

Use data augmentation

One last thing you should know about random image augmentation layers: just like `Dropout`, they're inactive during inference (when we call `predict()` or `evaluate()`). During evaluation, our model will behave just the same as when it did not include data augmentation and dropout.

Use data augmentation

Fight overfitting

```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Train the model

Listing 8.17 Training the regularized convnet

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Train the model

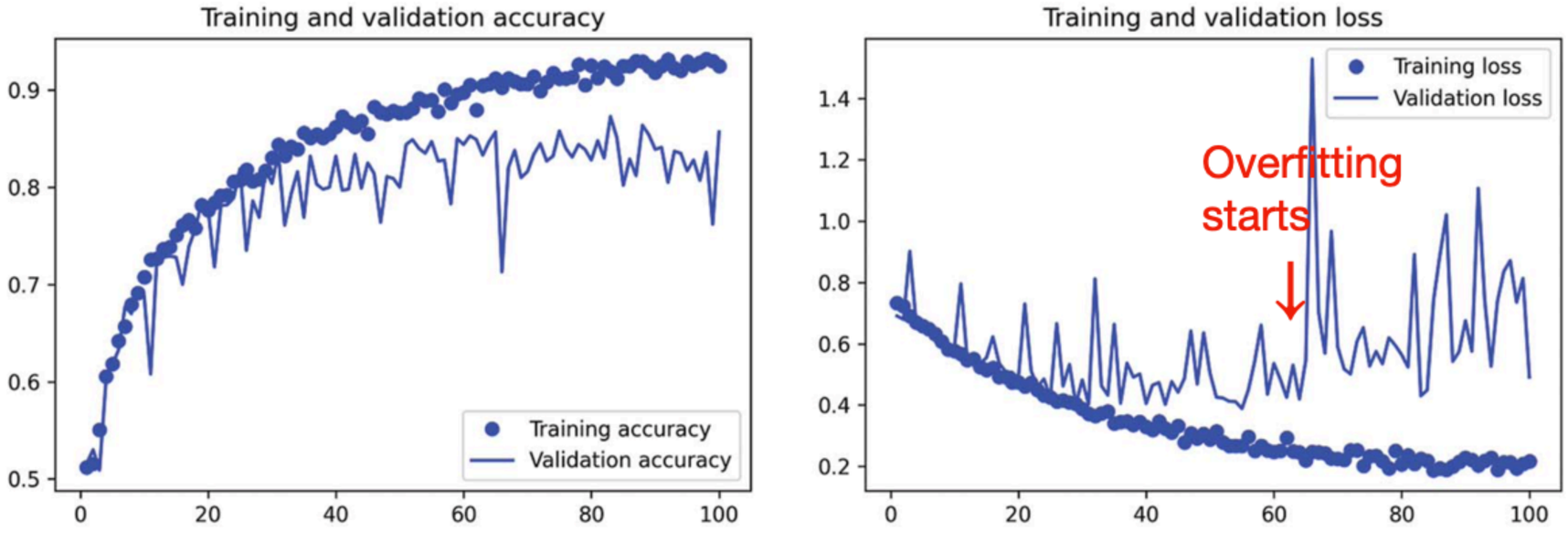


Figure 8.11 Training and validation metrics with data augmentation

Test performance

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

We get a test accuracy of 83.5%. It's starting to look good!

Quiz questions:

1. When is data augmentation needed?

2. How to use data augmentation for a CNN?

Roadmap of this lecture:

1. How a convolutional neural network (CNN) works
    - 1.1 CNN architecture
    - 1.2 How a convolution layer works
    - 1.3 How max-pooling and flatten layers work
2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

Leveraging a pre-trained model

Use a good (and usually large) model trained on a large (and related) dataset.

Such a pre-trained model can extract good features from input data.

Two ways to use a pre-trained model:

1) feature extraction

2) fine-tuning

Feature extraction with a pre-trained model

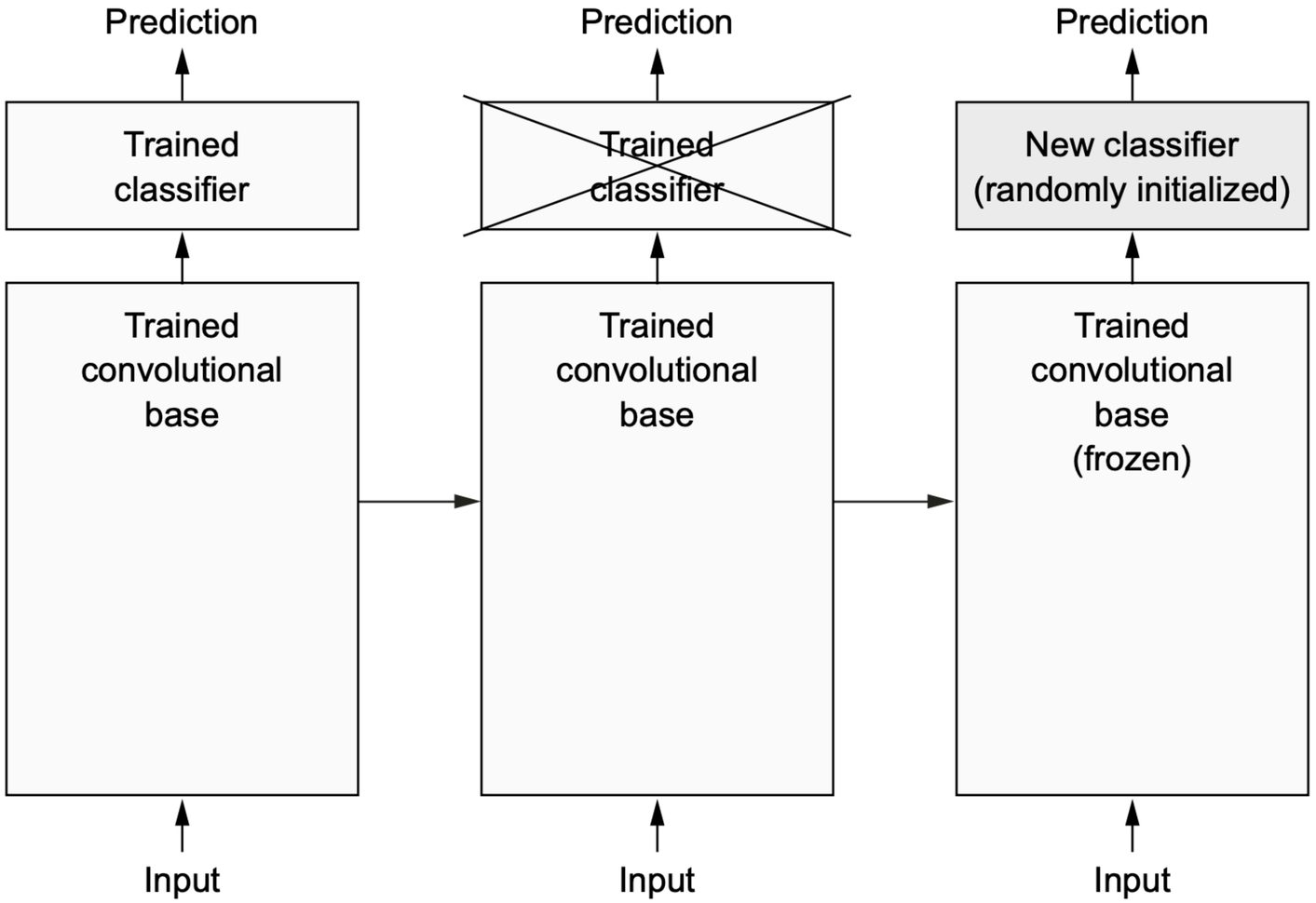


Figure 8.12 Swapping classifiers while keeping the same convolutional base

Feature extraction with a pre-trained model

Let's instantiate the VGG16 model.

Listing 8.19 Instantiating the VGG16 convolutional base

```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

We pass three arguments to the constructor:

- `weights` specifies the weight checkpoint from which to initialize the model.
- `include_top` refers to including (or not) the densely connected classifier on top of the network. By default, this densely connected classifier corresponds to the 1,000 classes from ImageNet. Because we intend to use our own densely connected classifier (with only two classes: `cat` and `dog`), we don't need to include it.
- `input_shape` is the shape of the image tensors that we'll feed to the network. This argument is purely optional: if we don't pass it, the network will be able to process inputs of any size. Here we pass it so that we can visualize (in the following summary) how the size of the feature maps shrinks with each new convolution and pooling layer.

Feature extraction with a pre-trained model

Here's the detail of the architecture of the VGG16 convolutional base. It's similar to the simple convnets you're already familiar with:

```
>>> conv_base.summary()
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_19 (InputLayer)        [(None, 180, 180, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 180, 180, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 180, 180, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 90, 90, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 90, 90, 128)       73856
_________________________________________________________________
```

Feature extraction with a pre-trained model

```
block2_conv2 (Conv2D)         (None, 90, 90, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)    (None, 45, 45, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)         (None, 45, 45, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)         (None, 45, 45, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)         (None, 45, 45, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)    (None, 22, 22, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)         (None, 22, 22, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)         (None, 22, 22, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)         (None, 22, 22, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)    (None, 11, 11, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)         (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)         (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)         (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)    (None, 5, 5, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Feature extraction with a pre-trained model

At this point, there are two ways we could proceed:

- Run the convolutional base over our dataset, record its output to a NumPy array on disk, and then use this data as input to a standalone, densely connected classifier similar to those you saw in chapter 4 of this book. This solution is fast and cheap to run, because it only requires running the convolutional base once for every input image, and the convolutional base is by far the most expensive part of the pipeline. But for the same reason, this technique won't allow us to use data augmentation.
- Extend the model we have (`conv_base`) by adding `Dense` layers on top, and run the whole thing from end to end on the input data. This will allow us to use data augmentation, because every input image goes through the convolutional base every time it's seen by the model. But for the same reason, this technique is far more expensive than the first.

Feature extraction with a pre-trained model

Listing 8.20 Extracting the VGG16 features and corresponding labels

```
import numpy as np

def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels =  get_features_and_labels(train_dataset)
val_features, val_labels =  get_features_and_labels(validation_dataset)
test_features, test_labels =  get_features_and_labels(test_dataset)
```

Feature extraction with a pre-trained model

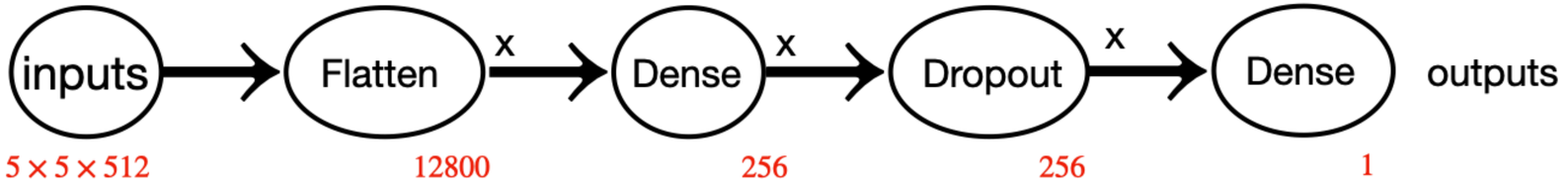
The extracted features are currently of shape `(samples, 5, 5, 512)`:

```
>>> train_features.shape
(2000, 5, 5, 512)
```

Listing 8.21 Defining and training the densely connected classifier

```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
```

Note the use of the Flatten layer before passing the features to a Dense layer.



Feature extraction with a pre-trained model

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_features, train_labels,
    epochs=20,
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

Feature extraction with a pre-trained model

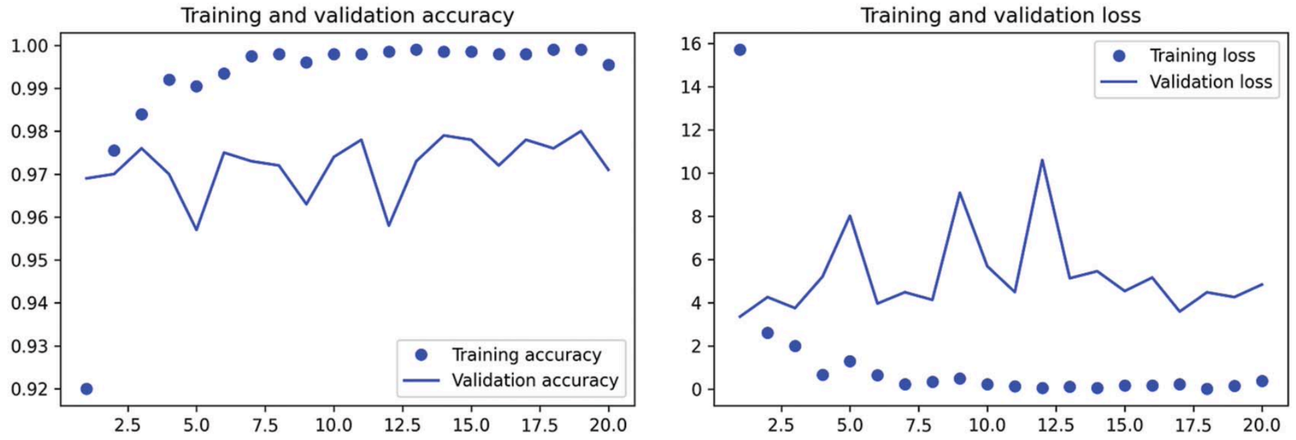


Figure 8.13 Training and validation metrics for plain feature extraction

Validation accuracy: 97%

Feature extraction with a pre-trained model

At this point, there are two ways we could proceed:

- Run the convolutional base over our dataset, record its output to a NumPy array on disk, and then use this data as input to a standalone, densely connected classifier similar to those you saw in chapter 4 of this book. This solution is fast and cheap to run, because it only requires running the convolutional base once for every input image, and the convolutional base is by far the most expensive part of the pipeline. But for the same reason, this technique won't allow us to use data augmentation.
- Extend the model we have (`conv_base`) by adding `Dense` layers on top, and run the whole thing from end to end on the input data. This will allow us to use data augmentation, because every input image goes through the convolutional base every time it's seen by the model. But for the same reason, this technique is far more expensive than the first.

Feature extraction together with data augmentation

In Keras, we freeze a layer or model by setting its `trainable` attribute to `False`.

Listing 8.23 Instantiating and freezing the VGG16 convolutional base

```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

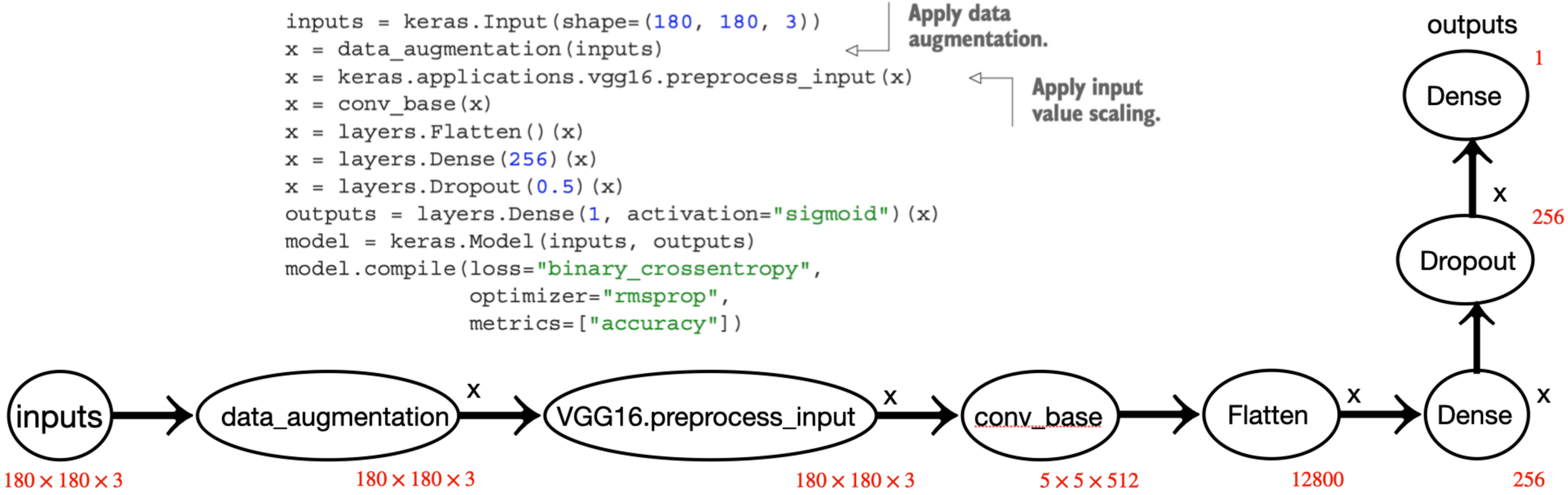
Feature extraction together with data augmentation

Listing 8.25 Adding a data augmentation stage and a classifier to the convolutional base

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Apply data augmentation.

Apply input value scaling.



Feature extraction together with data augmentation

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
```

```
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Feature extraction together with data augmentation

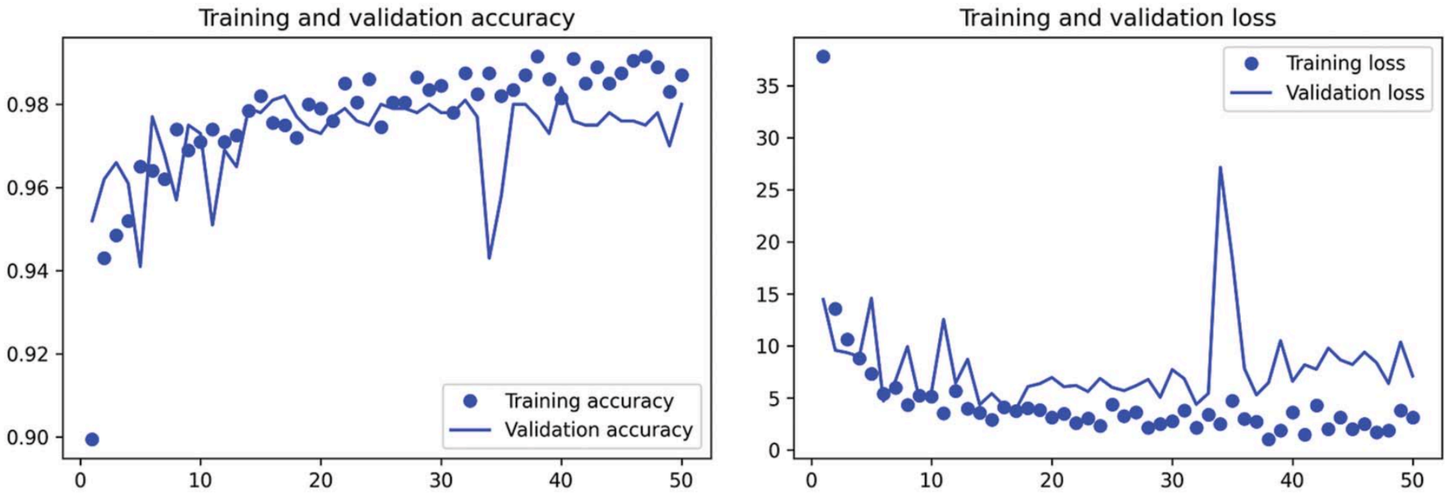


Figure 8.14 Training and validation metrics for feature extraction with data augmentation

Validation accuracy: 98%

Feature extraction together with data augmentation

Let's check the test accuracy.

Listing 8.26 Evaluating the model on the test set

```
test_model = keras.models.load_model(
    "feature_extraction_with_data_augmentation.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

We get a test accuracy of 97.5%.

Quiz questions:

1. How to use a pretrained model?

2. When can feature extraction from a pretrained model be useful?

Roadmap of this lecture:

1. How a convolutional neural network (CNN) works
    - 1.1 CNN architecture
    - 1.2 How a convolution layer works
    - 1.3 How max-pooling and flatten layers work
2. Example of CNN for MNIST
3. Train a CNN without regularization
4. Train a CNN with data augmentation
5. Use feature extraction from a trained model
6. Fine-tune a pre-trained model

Fine-tuning a pre-trained model

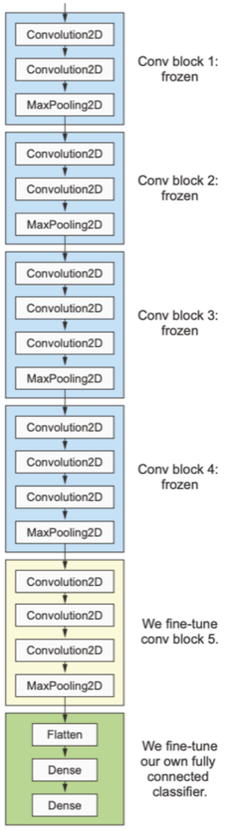
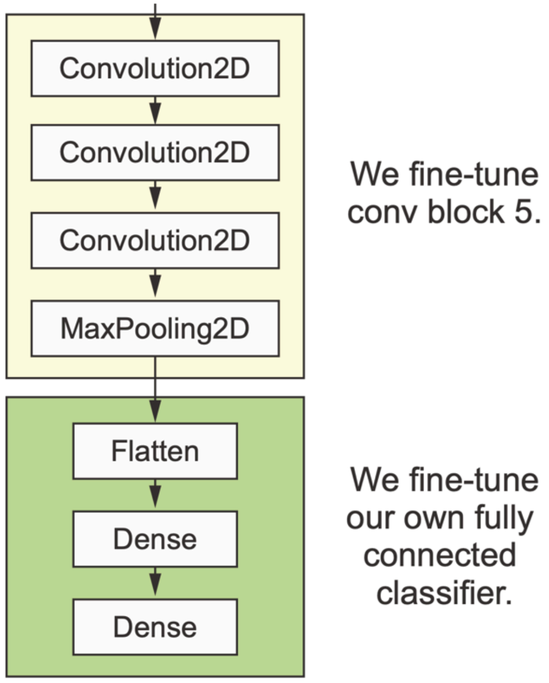


Figure 8.15 Fine-tuning the last convolutional block of the VGG16 network

Another widely used technique for model reuse, complementary to feature extraction, is *fine-tuning* (see figure 8.15). Fine-tuning consists of unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the newly added part of the model (in this case, the fully connected classifier) and these top layers. This is called *fine-tuning* because it slightly adjusts the more abstract representations of the model being reused in order to make them more relevant for the problem at hand.



Fine-tuning a pre-trained model

the steps for fine-tuning a network are as follows:

1. Add our custom network on top of an already-trained base network.
2. Freeze the base network.
3. Train the part we added.
4. Unfreeze some layers in the base network. (Note that you should not unfreeze "batch normalization" layers, which are not relevant here since there are no such layers in VGG16. Batch normalization and its impact on fine-tuning is explained in the next chapter.)
5. Jointly train both these layers and the part we added.

You already completed the first three steps when doing feature extraction. Let's proceed with step 4: we'll unfreeze our `conv_base` and then freeze individual layers inside it.

Fine-tuning a pre-trained model

Listing 8.27 Freezing all layers until the fourth from the last

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

Now we can begin fine-tuning the model. We'll do this with the RMSprop optimizer, using a very low learning rate. The reason for using a low learning rate is that we want to limit the magnitude of the modifications we make to the representations of the three layers we're fine-tuning. Updates that are too large may harm these representations.

Fine-tuning a pre-trained model

Listing 8.28 Fine-tuning the model

```
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Fine-tuning a pre-trained model

We can finally evaluate this model on the test data:

```
model = keras.models.load_model("fine_tuning.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

Test accuracy: 98.5%

Quiz questions:

1. What is fine tuning?

2. When is fine tuning most useful?