This is a companion notebook for the book Deep Learning with Python, Second Edition. For readability, it only contains runnable code blocks and section titles, and omits everything else in the book: text paragraphs, figures, and pseudocode.

If you want to be able to follow what's going on, I recommend reading the notebook side by side with your copy of the book.

This notebook was generated for TensorFlow 2.6.

+ Code + Text

```
Start coding or generate with AI.
```

Introduction to deep learning for computer vision

Introduction to convnets

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)
```

Displaying the model's summary

```
model.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73,856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 10)	11,530

```
Total params: 104,202 (407.04 KB)
```

Training the convnet on MNIST images

```
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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8835 - loss: 0.3664
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9866 - loss: 0.0442
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9911 - loss: 0.0296
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9937 - loss: 0.0203
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9946 - loss: 0.0170
<keras.src.callbacks.history.History at 0x7bc33dbb0a50>
```

Evaluating the convnet

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

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9906 - loss: 0.0317
Test accuracy: 0.993
```

The convolution operation

Understanding border effects and padding

Understanding convolution strides

The max-pooling operation

An incorrectly structured convnet missing its max-pooling layers

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model_no_max_pool = keras.Model(inputs=inputs, outputs=outputs)
```

```
model_no_max_pool.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
conv2d_4 (Conv2D)	(None, 24, 24, 64)	18,496
conv2d_5 (Conv2D)	(None, 22, 22, 128)	73,856
flatten_1 (Flatten)	(None, 61952)	0
dense_1 (Dense)	(None, 10)	619,530

```
Total params: 712,202 (2.72 MB)
```

Training a convnet from scratch on a small dataset

The relevance of deep learning for small-data problems

Downloading the data

Reading in training, validation, test datasets

- Step 1: Download the compressed dataset cats_vs_dogs_small.zip from Canvas. Note: this is a slightly different dataset as the book as it contains 2000 pictures for training, 1000 for validation and 1000 (as opposed to 2000) for testing.
- Step 2: Unzip the file onto your local drive.
- Step 3: Upload your unzipped files (all folders and files) to your Google Drive

Instructions: log into your Google Drive using the same google account of your Google Colab. Find the "Colab Notebooks" folder. Drag the cats_vs_dogs_small folder into the "Colab Notebooks" folder. This should upload all subfolders and pictures onto your Google Drive.

- Step 4: mount your Google Drive within Colab using the following code

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

- Step 5: find the path of your datafiles in Google Drive. On the left panel of Colab, Click File -> click folder content -> drive -> ...

For example, my path would look like something in the following: "/content/drive/MyDrive/Colab Notebooks/cats_vs_dogs_small"

Then you can set the path to your files with the following code:

```
import os, shutil, pathlib

new_base_dir = pathlib.Path("/content/drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
```

Make sure you change my code to your specific machine to have the correct folder.

Building the model

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)
```

```
model.summary()
```

```
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d_6 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_7 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_8 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_4 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_9 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_5 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_10 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_2 (Flatten)	(None, 12544)	0
dense_2 (Dense)	(None, 1)	12,545

```
Total params: 991,041 (3.78 MB)
```

Configuring the model for training

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

Data preprocessing

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)
```

```
Found 2000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

Note the difference in the number of files for the testing dataset.

```
import numpy as np
import tensorflow as tf
random_numbers = np.random.normal(size=(1000, 16))
dataset = tf.data.Dataset.from_tensor_slices(random_numbers)


for i, element in enumerate(dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(16,)
(16,)
(16,)
```

```
batched_dataset = dataset.batch(32)
for i, element in enumerate(batched_dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(32, 16)
(32, 16)
(32, 16)
```

```
reshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))
for i, element in enumerate(reshaped_dataset):
    print(element.shape)
    if i >= 2:
        break
```

```
(4, 4)
(4, 4)
(4, 4)
```

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,)
```

Fitting the model using a `Dataset`

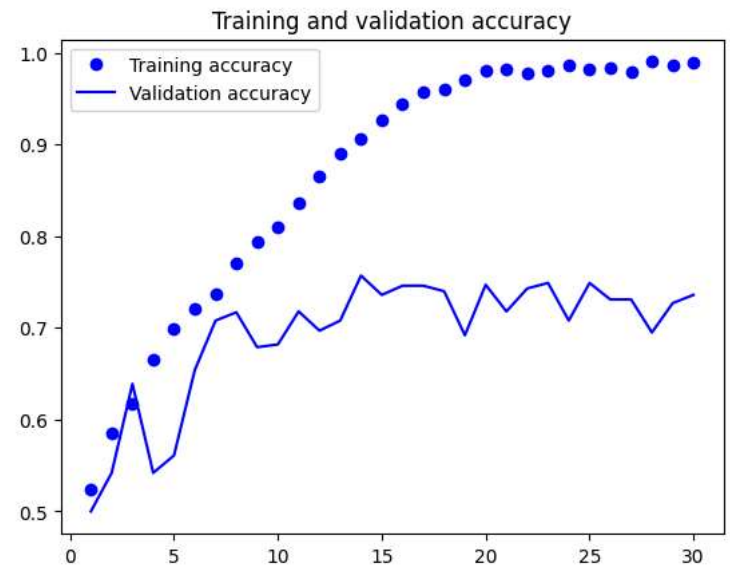
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

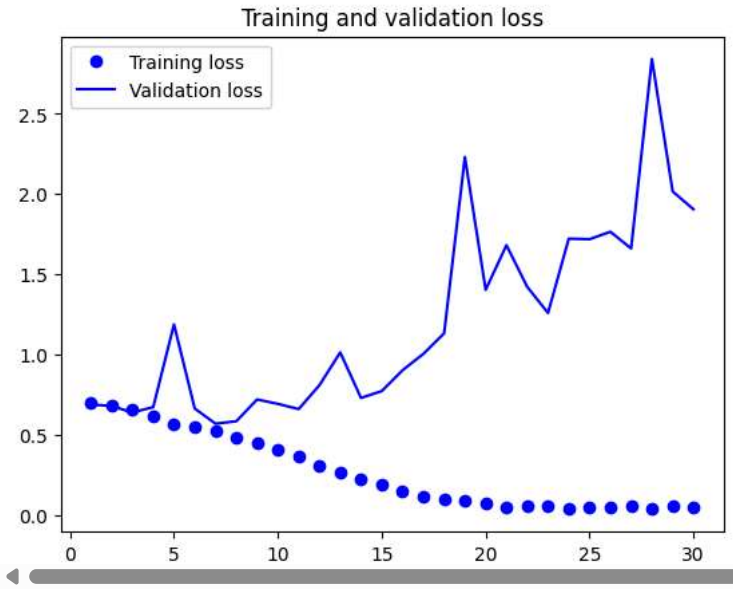
```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 389s 6s/step - accuracy: 0.4915 - loss: 0.6963 - val_accuracy: 0.5000 - val_loss: 0.6910
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 66s 174ms/step - accuracy: 0.5793 - loss: 0.6833 - val_accuracy: 0.5420 - val_loss: 0.6803
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 180ms/step - accuracy: 0.6210 - loss: 0.6582 - val_accuracy: 0.6390 - val_loss: 0.6410
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 195ms/step - accuracy: 0.6549 - loss: 0.6192 - val_accuracy: 0.5420 - val_loss: 0.6738
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.6827 - loss: 0.5718 - val_accuracy: 0.5610 - val_loss: 1.1888
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.7040 - loss: 0.5995 - val_accuracy: 0.6540 - val_loss: 0.6648
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 143ms/step - accuracy: 0.7251 - loss: 0.5430 - val_accuracy: 0.7080 - val_loss: 0.5709
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 193ms/step - accuracy: 0.7643 - loss: 0.4932 - val_accuracy: 0.7170 - val_loss: 0.5862
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 201ms/step - accuracy: 0.7928 - loss: 0.4468 - val_accuracy: 0.6790 - val_loss: 0.7216
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 17s 140ms/step - accuracy: 0.7846 - loss: 0.4470 - val_accuracy: 0.6820 - val_loss: 0.6942
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 152ms/step - accuracy: 0.8169 - loss: 0.3823 - val_accuracy: 0.7180 - val_loss: 0.6616
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 151ms/step - accuracy: 0.8604 - loss: 0.3200 - val_accuracy: 0.6970 - val_loss: 0.8106
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 171ms/step - accuracy: 0.8855 - loss: 0.2869 - val_accuracy: 0.7080 - val_loss: 1.0143
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 175ms/step - accuracy: 0.8838 - loss: 0.2771 - val_accuracy: 0.7570 - val_loss: 0.7312
```

```
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 194ms/step - accuracy: 0.9325 - loss: 0.1780 - val_accuracy: 0.7360 - val_loss: 0.7739
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 155ms/step - accuracy: 0.9439 - loss: 0.1456 - val_accuracy: 0.7460 - val_loss: 0.9034
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 137ms/step - accuracy: 0.9500 - loss: 0.1323 - val_accuracy: 0.7460 - val_loss: 1.0053
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 141ms/step - accuracy: 0.9550 - loss: 0.1088 - val_accuracy: 0.7400 - val_loss: 1.1333
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 152ms/step - accuracy: 0.9716 - loss: 0.0964 - val_accuracy: 0.6920 - val_loss: 2.2302
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 187ms/step - accuracy: 0.9667 - loss: 0.1330 - val_accuracy: 0.7470 - val_loss: 1.4044
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 171ms/step - accuracy: 0.9825 - loss: 0.0557 - val_accuracy: 0.7180 - val_loss: 1.6818
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 191ms/step - accuracy: 0.9601 - loss: 0.1091 - val_accuracy: 0.7430 - val_loss: 1.4225
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 171ms/step - accuracy: 0.9763 - loss: 0.0668 - val_accuracy: 0.7490 - val_loss: 1.2600
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 172ms/step - accuracy: 0.9885 - loss: 0.0444 - val_accuracy: 0.7080 - val_loss: 1.7225
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 181ms/step - accuracy: 0.9727 - loss: 0.0813 - val_accuracy: 0.7490 - val_loss: 1.7192
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 150ms/step - accuracy: 0.9789 - loss: 0.0676 - val_accuracy: 0.7310 - val_loss: 1.7654
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 158ms/step - accuracy: 0.9766 - loss: 0.0559 - val_accuracy: 0.7310 - val_loss: 1.6616
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 181ms/step - accuracy: 0.9916 - loss: 0.0307 - val_accuracy: 0.6950 - val_loss: 2.8410
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 183ms/step - accuracy: 0.9802 - loss: 0.0668 - val_accuracy: 0.7270 - val_loss: 2.0161
```

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()
```





Evaluating the model on the test set

```
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}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 193s 6s/step - accuracy: 0.7006 - loss: 0.5986
Test accuracy: 0.690
```

Using data augmentation

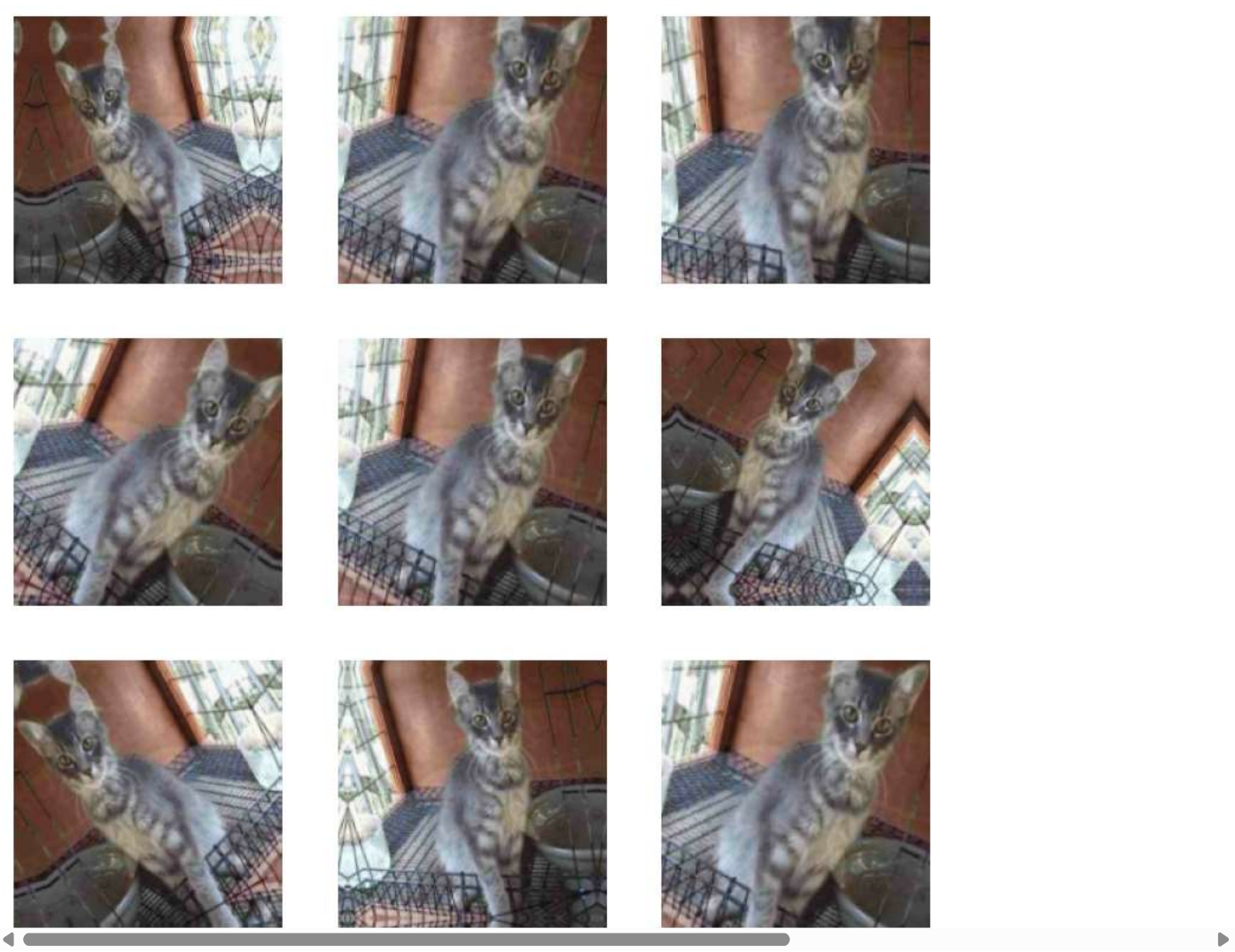
Define a data augmentation stage to add to an image model

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

Displaying some randomly augmented training images

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
```

```
    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")
```



Defining a new convnet that includes image augmentation and dropout

```
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"])
```

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)
```

```
Epoch 1/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 162ms/step - accuracy: 0.4989 - loss: 0.7093 - val_accuracy: 0.5000 - val_loss: 0.6928
Epoch 2/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 195ms/step - accuracy: 0.5401 - loss: 0.7113 - val_accuracy: 0.5720 - val_loss: 0.6851
Epoch 3/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 178ms/step - accuracy: 0.5710 - loss: 0.6860 - val_accuracy: 0.5120 - val_loss: 0.6852
Epoch 4/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 149ms/step - accuracy: 0.6238 - loss: 0.6667 - val_accuracy: 0.5010 - val_loss: 0.9716
Epoch 5/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 162ms/step - accuracy: 0.6232 - loss: 0.6736 - val_accuracy: 0.5860 - val_loss: 0.6560
Epoch 6/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 218ms/step - accuracy: 0.6542 - loss: 0.6126 - val_accuracy: 0.6220 - val_loss: 0.6356
Epoch 7/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 16s 138ms/step - accuracy: 0.6839 - loss: 0.6191 - val_accuracy: 0.6840 - val_loss: 0.5960
Epoch 8/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 193ms/step - accuracy: 0.6657 - loss: 0.6028 - val_accuracy: 0.6520 - val_loss: 0.6259
Epoch 9/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.6985 - loss: 0.5793 - val_accuracy: 0.6750 - val_loss: 0.6149
Epoch 10/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.6966 - loss: 0.5756 - val_accuracy: 0.6660 - val_loss: 0.6197
Epoch 11/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 179ms/step - accuracy: 0.7247 - loss: 0.5749 - val_accuracy: 0.7040 - val_loss: 0.5609
Epoch 12/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 180ms/step - accuracy: 0.7337 - loss: 0.5501 - val_accuracy: 0.7060 - val_loss: 0.5563
Epoch 13/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 179ms/step - accuracy: 0.7204 - loss: 0.5556 - val_accuracy: 0.6700 - val_loss: 0.6466
Epoch 14/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 194ms/step - accuracy: 0.7274 - loss: 0.5391 - val_accuracy: 0.7000 - val_loss: 0.5755
Epoch 15/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 175ms/step - accuracy: 0.7311 - loss: 0.5407 - val_accuracy: 0.7260 - val_loss: 0.5454
Epoch 16/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 135ms/step - accuracy: 0.7574 - loss: 0.5191 - val_accuracy: 0.7110 - val_loss: 0.5612
Epoch 17/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 196ms/step - accuracy: 0.7516 - loss: 0.5117 - val_accuracy: 0.7350 - val_loss: 0.5337
Epoch 18/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 176ms/step - accuracy: 0.7530 - loss: 0.5169 - val_accuracy: 0.7060 - val_loss: 0.5634
Epoch 19/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 180ms/step - accuracy: 0.7578 - loss: 0.5176 - val_accuracy: 0.7280 - val_loss: 0.5402
Epoch 20/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 156ms/step - accuracy: 0.7787 - loss: 0.4660 - val_accuracy: 0.6870 - val_loss: 0.5855
Epoch 21/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 198ms/step - accuracy: 0.7718 - loss: 0.4770 - val_accuracy: 0.7140 - val_loss: 0.5616
Epoch 22/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.7692 - loss: 0.4806 - val_accuracy: 0.7630 - val_loss: 0.4994
Epoch 23/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 152ms/step - accuracy: 0.7842 - loss: 0.4647 - val_accuracy: 0.7750 - val_loss: 0.4907
Epoch 24/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 174ms/step - accuracy: 0.8039 - loss: 0.4473 - val_accuracy: 0.7430 - val_loss: 0.5296
Epoch 25/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 155ms/step - accuracy: 0.7654 - loss: 0.4643 - val_accuracy: 0.7480 - val_loss: 0.5400
Epoch 26/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 152ms/step - accuracy: 0.8113 - loss: 0.4102 - val_accuracy: 0.6960 - val_loss: 0.6684
Epoch 27/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 145ms/step - accuracy: 0.7974 - loss: 0.4395 - val_accuracy: 0.7930 - val_loss: 0.4573
Epoch 28/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 153ms/step - accuracy: 0.8138 - loss: 0.4207 - val_accuracy: 0.7630 - val_loss: 0.5077
Epoch 29/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 167ms/step - accuracy: 0.8026 - loss: 0.4118 - val_accuracy: 0.7680 - val_loss: 0.5784
```

Evaluating the model on the test set

```
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}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 91ms/step - accuracy: 0.8041 - loss: 0.4397
Test accuracy: 0.796
```

Leveraging a pretrained model

Feature extraction with a pretrained model

Instantiating the VGG16 convolutional base

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
conv_base.summary()
```

```
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, 180, 180, 3)	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1,792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36,928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73,856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147,584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295,168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590,080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590,080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

Fast feature extraction without data augmentation

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)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 9s 9s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 196ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 184ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 172ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 193ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 210ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 192ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 175ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 189ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 175ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 194ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
```

```
train_features.shape
```

```
(2000, 5, 5, 512)
```

Defining and training the densely connected classifier

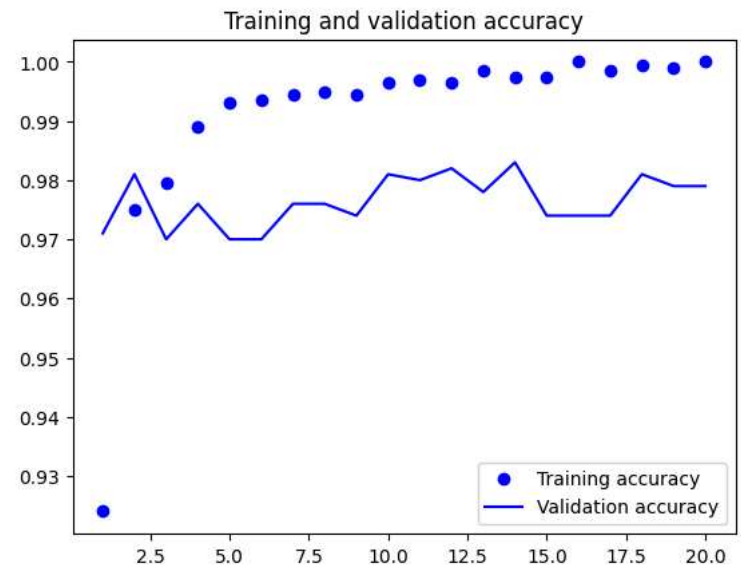
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
```

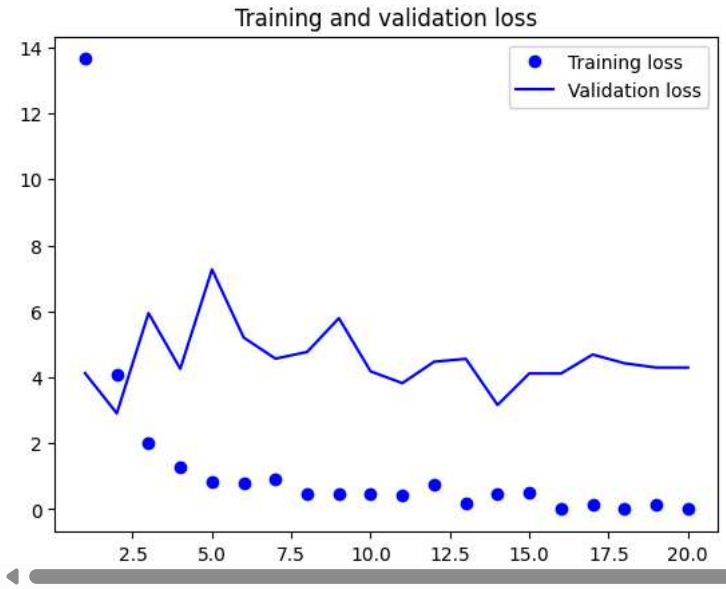
```
                metrics=["accuracy"])

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)
```

```
Epoch 1/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 31ms/step - accuracy: 0.8457 - loss: 29.8741 - val_accuracy: 0.9710 - val_loss: 4.1239
Epoch 2/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9787 - loss: 3.3324 - val_accuracy: 0.9810 - val_loss: 2.9090
Epoch 3/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9755 - loss: 2.3330 - val_accuracy: 0.9700 - val_loss: 5.9434
Epoch 4/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9919 - loss: 0.7044 - val_accuracy: 0.9760 - val_loss: 4.2590
Epoch 5/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9948 - loss: 0.6364 - val_accuracy: 0.9700 - val_loss: 7.2682
Epoch 6/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9922 - loss: 1.0086 - val_accuracy: 0.9700 - val_loss: 5.2047
Epoch 7/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9969 - loss: 0.3292 - val_accuracy: 0.9760 - val_loss: 4.5657
Epoch 8/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9921 - loss: 0.7668 - val_accuracy: 0.9760 - val_loss: 4.7689
Epoch 9/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9947 - loss: 0.3983 - val_accuracy: 0.9740 - val_loss: 5.7925
Epoch 10/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9955 - loss: 0.5592 - val_accuracy: 0.9810 - val_loss: 4.1799
Epoch 11/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9969 - loss: 0.4829 - val_accuracy: 0.9800 - val_loss: 3.8228
Epoch 12/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9954 - loss: 0.8219 - val_accuracy: 0.9820 - val_loss: 4.4715
Epoch 13/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9969 - loss: 0.3025 - val_accuracy: 0.9780 - val_loss: 4.5608
Epoch 14/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9962 - loss: 0.4776 - val_accuracy: 0.9830 - val_loss: 3.1625
Epoch 15/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9955 - loss: 0.5925 - val_accuracy: 0.9740 - val_loss: 4.1168
Epoch 16/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 5.9595e-09 - val_accuracy: 0.9740 - val_loss: 4.1151
Epoch 17/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9982 - loss: 0.2145 - val_accuracy: 0.9740 - val_loss: 4.6902
Epoch 18/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9999 - loss: 2.8307e-04 - val_accuracy: 0.9810 - val_loss: 4.4263
Epoch 19/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9996 - loss: 0.0564 - val_accuracy: 0.9790 - val_loss: 4.2964
Epoch 20/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 1.0127e-18 - val_accuracy: 0.9790 - val_loss: 4.2964
```

Plotting the results

```
import matplotlib.pyplot as plt
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "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()
```





Feature extraction together with data augmentation

Instantiating and freezing the VGG16 convolutional base

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

Printing the list of trainable weights before and after freezing

```
conv_base.trainable = True
print("This is the number of trainable weights "
      "before freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 26
```

```
conv_base.trainable = False
print("This is the number of trainable weights "
      "after freezing the conv base:", len(conv_base.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 0
```

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"])


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)
```

```
Epoch 1/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 17s 207ms/step - accuracy: 0.8171 - loss: 35.9510 - val_accuracy: 0.9670 - val_loss: 5.0563
Epoch 2/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 187ms/step - accuracy: 0.9561 - loss: 4.7635 - val_accuracy: 0.9640 - val_loss: 4.7975
Epoch 3/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 171ms/step - accuracy: 0.9531 - loss: 5.5895 - val_accuracy: 0.9690 - val_loss: 3.5244
Epoch 4/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 193ms/step - accuracy: 0.9621 - loss: 5.2091 - val_accuracy: 0.9780 - val_loss: 2.7564
Epoch 5/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 190ms/step - accuracy: 0.9694 - loss: 3.4878 - val_accuracy: 0.9770 - val_loss: 3.3927
Epoch 6/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 189ms/step - accuracy: 0.9664 - loss: 4.2338 - val_accuracy: 0.9630 - val_loss: 8.2009
Epoch 7/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - accuracy: 0.9585 - loss: 4.5031 - val_accuracy: 0.9580 - val_loss: 9.6900
Epoch 8/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 172ms/step - accuracy: 0.9710 - loss: 3.1823 - val_accuracy: 0.9750 - val_loss: 3.3284
Epoch 9/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 199ms/step - accuracy: 0.9693 - loss: 2.8788 - val_accuracy: 0.9710 - val_loss: 4.0782
Epoch 10/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 197ms/step - accuracy: 0.9861 - loss: 1.3116 - val_accuracy: 0.9780 - val_loss: 3.6350
Epoch 11/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step - accuracy: 0.9789 - loss: 1.9874 - val_accuracy: 0.9740 - val_loss: 3.5514
Epoch 12/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 186ms/step - accuracy: 0.9735 - loss: 2.3973 - val_accuracy: 0.9790 - val_loss: 2.7980
Epoch 13/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 189ms/step - accuracy: 0.9791 - loss: 1.4935 - val_accuracy: 0.9770 - val_loss: 2.7978
Epoch 14/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 162ms/step - accuracy: 0.9783 - loss: 1.4962 - val_accuracy: 0.9810 - val_loss: 2.8299
Epoch 15/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 169ms/step - accuracy: 0.9838 - loss: 1.3517 - val_accuracy: 0.9650 - val_loss: 6.1465
Epoch 16/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 178ms/step - accuracy: 0.9758 - loss: 2.4975 - val_accuracy: 0.9770 - val_loss: 3.2949
Epoch 17/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 185ms/step - accuracy: 0.9752 - loss: 2.0481 - val_accuracy: 0.9740 - val_loss: 3.6018
Epoch 18/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 196ms/step - accuracy: 0.9769 - loss: 1.6120 - val_accuracy: 0.9780 - val_loss: 3.5974
Epoch 19/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 170ms/step - accuracy: 0.9781 - loss: 1.3722 - val_accuracy: 0.9770 - val_loss: 2.5767
Epoch 20/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 166ms/step - accuracy: 0.9814 - loss: 1.5392 - val_accuracy: 0.9770 - val_loss: 2.3657
Epoch 21/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 198ms/step - accuracy: 0.9825 - loss: 1.4050 - val_accuracy: 0.9750 - val_loss: 2.6764
Epoch 22/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 0.9816 - loss: 1.2772 - val_accuracy: 0.9750 - val_loss: 2.7991
Epoch 23/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 189ms/step - accuracy: 0.9816 - loss: 1.0828 - val_accuracy: 0.9730 - val_loss: 3.0479
```

```
Epoch 24/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 200ms/step - accuracy: 0.9861 - loss: 0.6581 - val_accuracy: 0.9740 - val_loss: 2.4844
Epoch 25/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 199ms/step - accuracy: 0.9917 - loss: 0.3826 - val_accuracy: 0.9740 - val_loss: 3.0861
Epoch 26/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 198ms/step - accuracy: 0.9876 - loss: 0.7380 - val_accuracy: 0.9730 - val_loss: 3.0990
Epoch 27/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 191ms/step - accuracy: 0.9852 - loss: 0.5755 - val_accuracy: 0.9760 - val_loss: 2.3208
Epoch 28/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 173ms/step - accuracy: 0.9937 - loss: 0.3371 - val_accuracy: 0.9720 - val_loss: 2.8027
Epoch 29/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 206ms/step - accuracy: 0.9876 - loss: 1.1537 - val_accuracy: 0.9780 - val_loss: 2.0988
```

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}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 108ms/step - accuracy: 0.9864 - loss: 1.0768
Test accuracy: 0.981
```

Fine-tuning a pretrained model

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1,792
block1_conv2 (Conv2D)	(None, None, None, 64)	36,928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73,856
block2_conv2 (Conv2D)	(None, None, None, 128)	147,584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295,168
block3_conv2 (Conv2D)	(None, None, None, 256)	590,080
block3_conv3 (Conv2D)	(None, None, None, 256)	590,080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1,180,160
block4_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block4_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

Freezing all layers until the fourth from the last

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

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)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 15s 189ms/step - accuracy: 0.9853 - loss: 0.5457 - val_accuracy: 0.9770 - val_loss: 2.1714
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 186ms/step - accuracy: 0.9922 - loss: 0.3308 - val_accuracy: 0.9750 - val_loss: 1.5463
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 176ms/step - accuracy: 0.9951 - loss: 0.1741 - val_accuracy: 0.9760 - val_loss: 1.6165
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 193ms/step - accuracy: 0.9980 - loss: 0.0552 - val_accuracy: 0.9800 - val_loss: 1.5615
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 202ms/step - accuracy: 0.9969 - loss: 0.0839 - val_accuracy: 0.9770 - val_loss: 1.6337
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 211ms/step - accuracy: 0.9901 - loss: 0.3716 - val_accuracy: 0.9820 - val_loss: 1.4993
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 199ms/step - accuracy: 0.9954 - loss: 0.1618 - val_accuracy: 0.9820 - val_loss: 1.4173
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 196ms/step - accuracy: 0.9948 - loss: 0.1301 - val_accuracy: 0.9780 - val_loss: 1.7092
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 173ms/step - accuracy: 0.9940 - loss: 0.3197 - val_accuracy: 0.9820 - val_loss: 1.8194
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 170ms/step - accuracy: 0.9938 - loss: 0.1986 - val_accuracy: 0.9800 - val_loss: 1.4601
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 199ms/step - accuracy: 0.9954 - loss: 0.0702 - val_accuracy: 0.9800 - val_loss: 1.4071
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 205ms/step - accuracy: 0.9977 - loss: 0.0988 - val_accuracy: 0.9800 - val_loss: 1.4708
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 213ms/step - accuracy: 0.9960 - loss: 0.1332 - val_accuracy: 0.9780 - val_loss: 1.3614
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 170ms/step - accuracy: 0.9950 - loss: 0.1625 - val_accuracy: 0.9780 - val_loss: 1.6802
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 207ms/step - accuracy: 0.9904 - loss: 0.3968 - val_accuracy: 0.9800 - val_loss: 1.1674
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 217ms/step - accuracy: 0.9931 - loss: 0.1778 - val_accuracy: 0.9810 - val_loss: 1.1505
```