Assignment 2 Results

Qiao Wang

1. Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?

Training a convnet from scratch with training sample of **1000**, a validation sample of 500, and a test sample of 500. Use augmentation and dropout(0.4), the test results are: loss: 0.5437 - accuracy: 0.8490.

2. Increase your training sample size. You may pick any amount. Keep the validation and test samples the same as above. Optimize your network (again training from scratch). What performance did you achieve?

Training a convnet from scratch with training sample of **3000**, a validation sample of 500, and a test sample of 500. Use augmentation and dropout(0.5), the test result is: loss: 0.3999 - accuracy: 0.8868

3. Now change your training sample so that you achieve better performance than those from Steps 1 and 2. This sample size may be larger, or smaller than those in the previous steps. The objective is to find the ideal training sample size to get best prediction results.

Training a convnet from scratch with training sample of **5000**, a validation sample of 500, and a test sample of 500. Use augmentation and dropout(0.5), the test results is: loss: 0.2071 - accuracy: 0.9150.

Training a convnet from scratch with training sample of **8000**, a validation sample of 500, and a test sample of 500. Use augmentation and dropout(0.5), the test results is: loss: 0.2239 - accuracy: 0.9070.

First, I increase the sample size to **5000**, the test result is: loss: 0.2071 - accuracy: 0.9150.

Then, I increase the sample size to **8000**, and we have a test result of loss: 0.2239 - accuracy: 0.9070, which is worse than when sample size is 5000.

Therefore, the best result I can get when training a convnet from scratch is: loss: 0.2071 - accuracy: 0.9150, which is when the training sample size is 5000.

4. Repeat Steps 1-3, but now using a pretrained network. The sample sizes you use in Steps 2 and 3 for the pretrained network may be the same or different from those using the network where you trained from scratch. Again, use any and all optimization techniques to get the best performance.

Using a pretrained network with training sample of **1000**, a validation sample of 500, and a test sample of 500. Use augmentation and dropout(0.5), the test results are: loss: 1.5191 - accuracy: 0.9810

Using a pretrained network with training sample of **3000**, a validation sample of 500, and a test sample of 500. Use augmentation and dropout(0.5), the test results are: loss: 0.9459 - accuracy: 0.9790

Using a pretrained network with training sample of **5000**, a validation sample of 500, and a test sample of 500. Use augmentation and dropout(0.5), the test results are: loss: 0.3557 - accuracy: 0.9810

Write a report summarizing your findings. What is the relationship between training sample size and choice of network?

Summary:

	training size	validation size	test size	augmentation	dropout	loss	accuracy
training from scratch	1000	500	500	yes	0.4	0.5437	0.849
	3000	500	500	yes	0.5	0.3999	0.8868
	5000	500	500	yes	0.5	0.2071	0.915
	8000	500	500	yes	0.5	0.2239	0.907
pretrained network	1000	500	500	yes	0.5	1.5191	0.981
	3000	500	500	yes	0.5	0.9459	0.979
	5000	500	500	yes	0.5	0.3557	0.981

As we see from the results above, using convnets and train from scratch could get a relatively good result for a small dataset. However, there are some overfitting problems. We can solve this problem to some degree by using techniques like dropout, augmentation, etc. When we set sample size to 1000, the accuracy is very low. As the sample size increases, the loss decreases and accuracy increases. When the sample size increases to 5000, the accuracy reaches peak and then decrease as sample size reach 8000.

When we use a pretrained network, the results of accuracy are all around 0.9800, although the loss reduces when the sample size increases. Therefore, in this case, the accuracy of the model doesn't change much, and the loss decreases as the sample size increases.

Training a convnet from scratch with training sample of 1000, a validation sample of 500, and a test sample of 500

Downloading the data

In [1]:

```
!unzip -qq '/fs/ess/PGS0333/BA_64061_KSU/data/dogs-vs-cats.zip'
```

In [2]:

```
!unzip -qq train.zip
```

Copying images to training, validation, and test directories

In [3]:

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index,
end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2000)
```

Building the model

Instantiating a small convnet for dogs vs. cats classification

In [4]:

```
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)
```

In [5]:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 43, 43, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 41, 41, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 12545
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

Configuring the model for training

In [6]:

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

Data preprocessing

Using image_dataset_from_directory to read images

In [7]:

```
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.
```

In [8]:

```
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)
```

In [9]:

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

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

In [10]:

```
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)
```

In [11]:

```
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

In [12]:

```
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

In [13]:

```
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 [==============================] - 5s 38ms/step - loss: 0.7281 - accura
cy: 0.5025 - val_loss: 0.7877 - val_accuracy: 0.5000
Epoch 2/30
63/63 [==============================] - 2s 30ms/step - loss: 0.7313 - accura
cy: 0.5295 - val_loss: 0.6778 - val_accuracy: 0.6290
Epoch 3/30
63/63 [==============================] - 2s 28ms/step - loss: 0.6879 - accura
cy: 0.6005 - val_loss: 0.7451 - val_accuracy: 0.5220
Epoch 4/30
63/63 [==============================] - 2s 28ms/step - loss: 0.6479 - accura
cy: 0.6325 - val_loss: 0.7421 - val_accuracy: 0.5580
Epoch 5/30
63/63 [==============================] - 2s 28ms/step - loss: 0.6098 - accura
cy: 0.6705 - val_loss: 0.6288 - val_accuracy: 0.6430
Epoch 6/30
63/63 [==============================] - 2s 28ms/step - loss: 0.5627 - accura
cy: 0.7020 - val_loss: 0.6872 - val_accuracy: 0.6230
Epoch 7/30
63/63 [==============================] - 2s 28ms/step - loss: 0.5525 - accura
cy: 0.7360 - val_loss: 0.9677 - val_accuracy: 0.6000
Epoch 8/30
63/63 [==============================] - 2s 27ms/step - loss: 0.4833 - accura
cy: 0.7745 - val_loss: 0.8117 - val_accuracy: 0.6070
Epoch 9/30
63/63 [==============================] - 2s 28ms/step - loss: 0.4395 - accura
cy: 0.7955 - val_loss: 0.6178 - val_accuracy: 0.6920
Epoch 10/30
63/63 [==============================] - 2s 28ms/step - loss: 0.3887 - accura
cy: 0.8390 - val_loss: 0.5901 - val_accuracy: 0.7150
Epoch 11/30
```

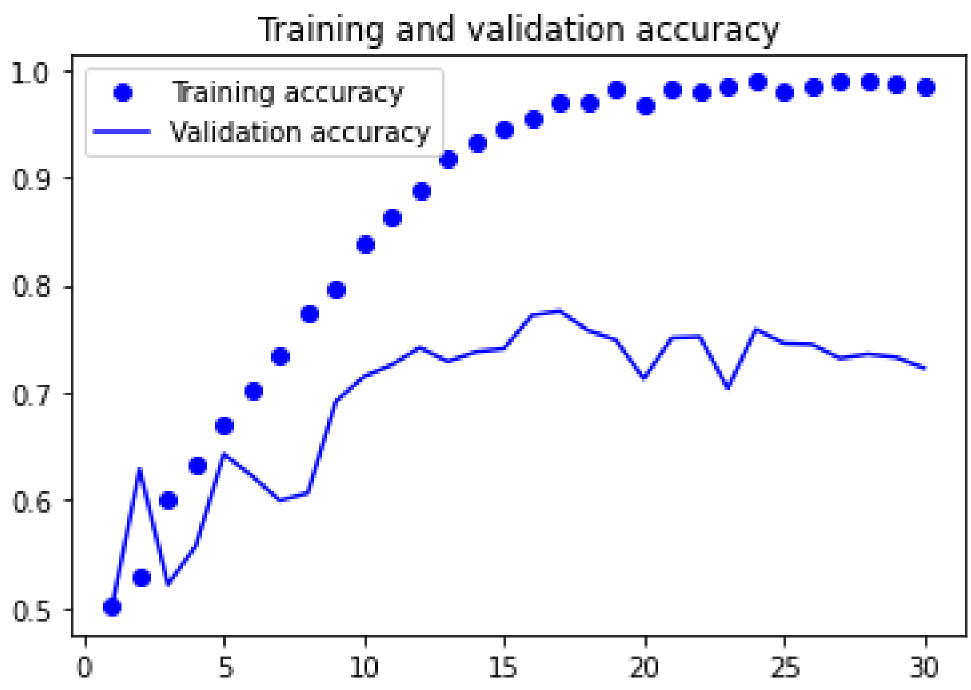
```
63/63 [==============================] - 2s 28ms/step - loss: 0.3209 - accura
cy: 0.8645 - val_loss: 0.6284 - val_accuracy: 0.7260
Epoch 12/30
63/63 [==============================] - 2s 28ms/step - loss: 0.2802 - accura
cy: 0.8885 - val_loss: 0.7380 - val_accuracy: 0.7420
Epoch 13/30
63/63 [==============================] - 2s 28ms/step - loss: 0.2119 - accura
cy: 0.9175 - val_loss: 0.6878 - val_accuracy: 0.7290
Epoch 14/30
63/63 [==============================] - 2s 28ms/step - loss: 0.1710 - accura
cy: 0.9315 - val_loss: 0.8529 - val_accuracy: 0.7380
Epoch 15/30
63/63 [==============================] - 2s 29ms/step - loss: 0.1394 - accura
cy: 0.9445 - val_loss: 0.9875 - val_accuracy: 0.7410
Epoch 16/30
63/63 [==============================] - 2s 28ms/step - loss: 0.1168 - accura
cy: 0.9540 - val_loss: 0.8968 - val_accuracy: 0.7720
Epoch 17/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0858 - accura
cy: 0.9690 - val_loss: 1.1203 - val_accuracy: 0.7760
Epoch 18/30
63/63 [==============================] - 2s 28ms/step - loss: 0.1041 - accura
cy: 0.9700 - val_loss: 1.1405 - val_accuracy: 0.7580
Epoch 19/30
63/63 [==============================] - 2s 29ms/step - loss: 0.0519 - accura
cy: 0.9825 - val_loss: 1.6991 - val_accuracy: 0.7490
Epoch 20/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0868 - accura
cy: 0.9675 - val_loss: 1.4378 - val_accuracy: 0.7130
Epoch 21/30
63/63 [==============================] - 2s 29ms/step - loss: 0.0478 - accura
cy: 0.9830 - val_loss: 1.4454 - val_accuracy: 0.7510
Epoch 22/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0584 - accura
cy: 0.9800 - val_loss: 1.3630 - val_accuracy: 0.7520
Epoch 23/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0501 - accura
cy: 0.9835 - val_loss: 1.8856 - val_accuracy: 0.7040
Epoch 24/30
63/63 [==============================] - 2s 29ms/step - loss: 0.0361 - accura
cy: 0.9890 - val_loss: 1.8424 - val_accuracy: 0.7590
Epoch 25/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0699 - accura
cy: 0.9800 - val_loss: 1.5784 - val_accuracy: 0.7460
Epoch 26/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0507 - accura
cy: 0.9835 - val_loss: 1.7306 - val_accuracy: 0.7450
Epoch 27/30
63/63 [==============================] - 2s 29ms/step - loss: 0.0408 - accura
cy: 0.9885 - val_loss: 1.7925 - val_accuracy: 0.7320
Epoch 28/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0331 - accura
cy: 0.9890 - val_loss: 1.9974 - val_accuracy: 0.7360
```

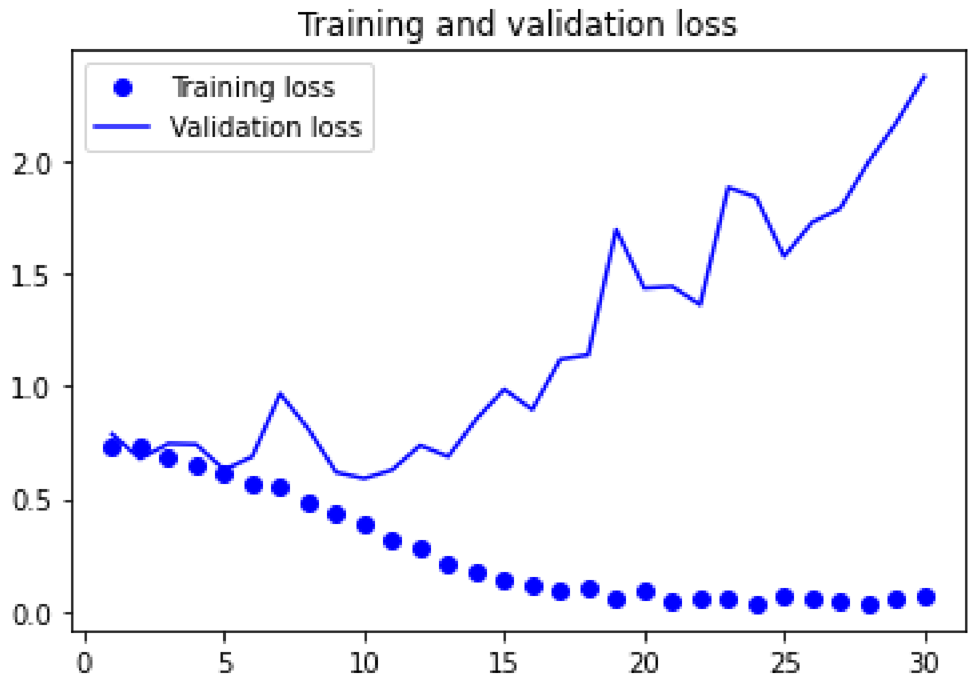
```
Epoch 29/30
63/63 [==============================] - 2s 27ms/step - loss: 0.0544 - accura
cy: 0.9860 - val_loss: 2.1735 - val_accuracy: 0.7330
Epoch 30/30
63/63 [==============================] - 2s 28ms/step - loss: 0.0622 - accura
cy: 0.9845 - val_loss: 2.3797 - val_accuracy: 0.7230
```

Displaying curves of loss and accuracy during training

In [14]:

```python
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

In [15]:

```
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 [==============================] - 1s 14ms/step - loss: 0.6620 - accura
cy: 0.6860
Test accuracy: 0.686
```

Using data augmentation

Define a data augmentation stage to add to an image model

In [16]:

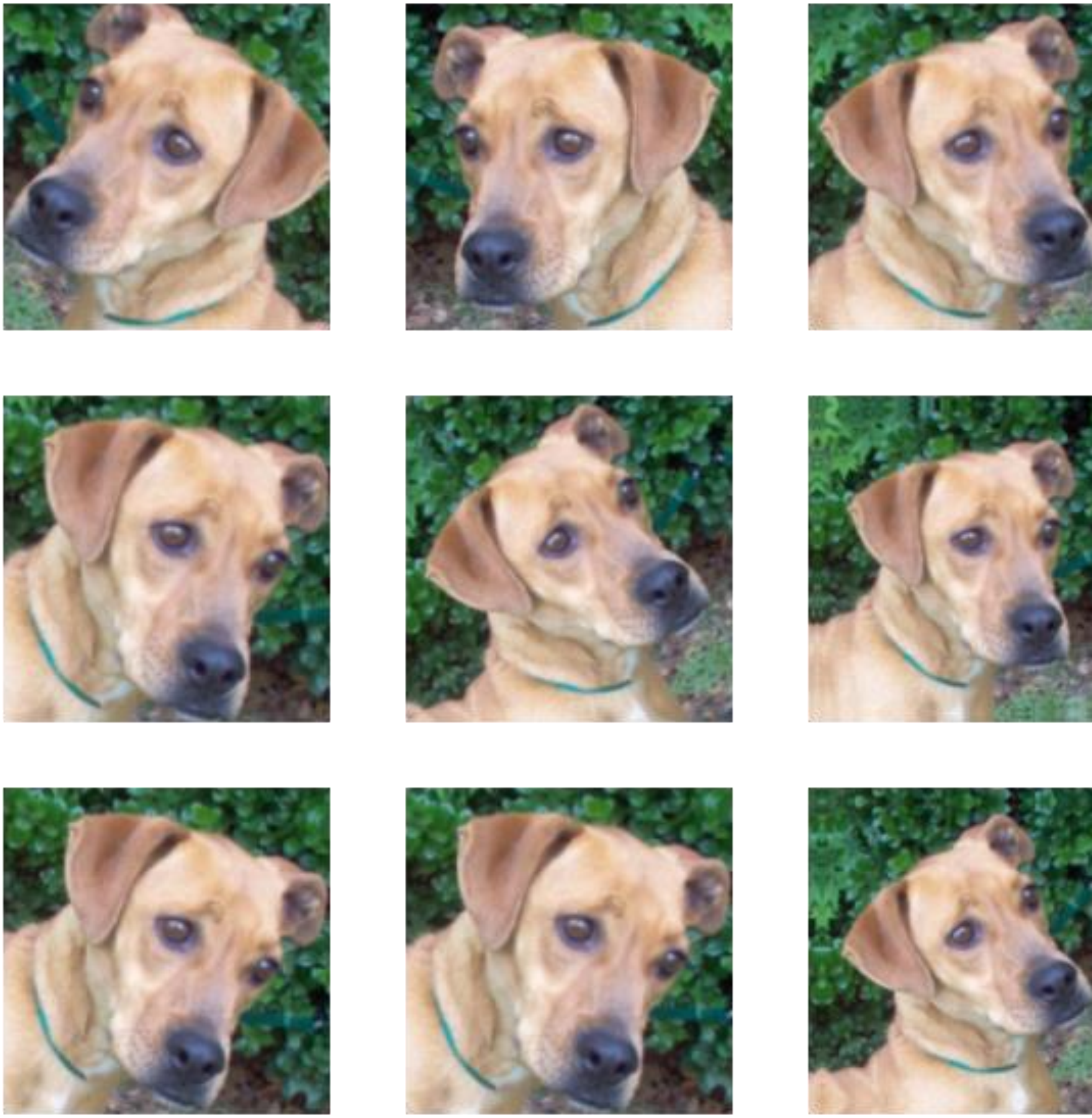
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

Displaying some randomly augmented training images

In [17]:

```
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")
```



Defining a new convnet that includes image augmentation, and dropout

In [18]:

```
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(0.4)(x)    # change the dropout from 0.5 to 0.4
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

In [19]:

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=80,  # change epochs from 100 to 80
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/80
63/63 [==============================] - 3s 31ms/step - loss: 0.7067 - accura
cy: 0.5165 - val_loss: 0.6852 - val_accuracy: 0.5560
Epoch 2/80
63/63 [==============================] - 2s 29ms/step - loss: 0.7041 - accura
cy: 0.5600 - val_loss: 0.6838 - val_accuracy: 0.5500
Epoch 3/80
63/63 [==============================] - 2s 29ms/step - loss: 0.6729 - accura
cy: 0.6110 - val_loss: 0.6920 - val_accuracy: 0.5460
Epoch 4/80
63/63 [==============================] - 2s 29ms/step - loss: 0.6557 - accura
cy: 0.6225 - val_loss: 0.6604 - val_accuracy: 0.5790
Epoch 5/80
63/63 [==============================] - 2s 29ms/step - loss: 0.6495 - accura
cy: 0.6425 - val_loss: 0.6610 - val_accuracy: 0.6140
Epoch 6/80
63/63 [==============================] - 2s 29ms/step - loss: 0.6109 - accura
cy: 0.6540 - val_loss: 1.1658 - val_accuracy: 0.5730
Epoch 7/80
63/63 [==============================] - 2s 29ms/step - loss: 0.6154 - accura
cy: 0.6795 - val_loss: 0.5736 - val_accuracy: 0.6950
```

```
Epoch 8/80
63/63 [==============================] - 2s 29ms/step - loss: 0.5897 - accura
cy: 0.7035 - val_loss: 0.6279 - val_accuracy: 0.6810
Epoch 9/80
63/63 [==============================] - 2s 30ms/step - loss: 0.5802 - accura
cy: 0.7040 - val_loss: 0.6419 - val_accuracy: 0.6910
Epoch 10/80
63/63 [==============================] - 2s 30ms/step - loss: 0.5710 - accura
cy: 0.7035 - val_loss: 0.5431 - val_accuracy: 0.7350
Epoch 11/80
63/63 [==============================] - 2s 29ms/step - loss: 0.5470 - accura
cy: 0.7240 - val_loss: 0.6522 - val_accuracy: 0.6880
Epoch 12/80
63/63 [==============================] - 2s 29ms/step - loss: 0.5214 - accura
cy: 0.7425 - val_loss: 0.5042 - val_accuracy: 0.7430
Epoch 13/80
63/63 [==============================] - 2s 28ms/step - loss: 0.5125 - accura
cy: 0.7565 - val_loss: 0.6554 - val_accuracy: 0.7340
Epoch 14/80
63/63 [==============================] - 2s 28ms/step - loss: 0.5010 - accura
cy: 0.7500 - val_loss: 0.5015 - val_accuracy: 0.7540
Epoch 15/80
63/63 [==============================] - 2s 28ms/step - loss: 0.4955 - accura
cy: 0.7630 - val_loss: 0.6987 - val_accuracy: 0.6910
Epoch 16/80
63/63 [==============================] - 2s 29ms/step - loss: 0.4789 - accura
cy: 0.7775 - val_loss: 0.5838 - val_accuracy: 0.7250
Epoch 17/80
63/63 [==============================] - 2s 29ms/step - loss: 0.4715 - accura
cy: 0.7775 - val_loss: 0.4991 - val_accuracy: 0.7650
Epoch 18/80
63/63 [==============================] - 2s 29ms/step - loss: 0.4562 - accura
cy: 0.7945 - val_loss: 0.5710 - val_accuracy: 0.7350
Epoch 19/80
63/63 [==============================] - 2s 28ms/step - loss: 0.4658 - accura
cy: 0.7915 - val_loss: 0.4418 - val_accuracy: 0.7900
Epoch 20/80
63/63 [==============================] - 2s 29ms/step - loss: 0.4599 - accura
cy: 0.7870 - val_loss: 0.4657 - val_accuracy: 0.7790
Epoch 21/80
63/63 [==============================] - 2s 29ms/step - loss: 0.4483 - accura
cy: 0.7930 - val_loss: 0.5815 - val_accuracy: 0.7030
Epoch 22/80
63/63 [==============================] - 2s 28ms/step - loss: 0.4382 - accura
cy: 0.7940 - val_loss: 0.6202 - val_accuracy: 0.7490
Epoch 23/80
63/63 [==============================] - 2s 29ms/step - loss: 0.4339 - accura
cy: 0.8005 - val_loss: 0.4995 - val_accuracy: 0.7800
Epoch 24/80
63/63 [==============================] - 2s 29ms/step - loss: 0.4112 - accura
cy: 0.8145 - val_loss: 0.4665 - val_accuracy: 0.7950
Epoch 25/80
```

```
63/63 [==============================] - 2s 29ms/step - loss: 0.3912 - accura
cy: 0.8330 - val_loss: 0.5159 - val_accuracy: 0.7550
Epoch 26/80
63/63 [==============================] - 2s 28ms/step - loss: 0.3978 - accura
cy: 0.8225 - val_loss: 0.4460 - val_accuracy: 0.8110
Epoch 27/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3964 - accura
cy: 0.8155 - val_loss: 0.4716 - val_accuracy: 0.7960
Epoch 28/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3859 - accura
cy: 0.8260 - val_loss: 0.6183 - val_accuracy: 0.7300
Epoch 29/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3803 - accura
cy: 0.8360 - val_loss: 0.5046 - val_accuracy: 0.7730
Epoch 30/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3758 - accura
cy: 0.8315 - val_loss: 0.7977 - val_accuracy: 0.7130
Epoch 31/80
63/63 [==============================] - 2s 28ms/step - loss: 0.3527 - accura
cy: 0.8375 - val_loss: 0.6236 - val_accuracy: 0.8000
Epoch 32/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3576 - accura
cy: 0.8425 - val_loss: 0.5660 - val_accuracy: 0.7640
Epoch 33/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3353 - accura
cy: 0.8540 - val_loss: 0.5750 - val_accuracy: 0.8180
Epoch 34/80
63/63 [==============================] - 2s 28ms/step - loss: 0.3402 - accura
cy: 0.8575 - val_loss: 0.5400 - val_accuracy: 0.7850
Epoch 35/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3140 - accura
cy: 0.8650 - val_loss: 0.5461 - val_accuracy: 0.7930
Epoch 36/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3298 - accura
cy: 0.8655 - val_loss: 0.4594 - val_accuracy: 0.7960
Epoch 37/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3310 - accura
cy: 0.8670 - val_loss: 0.6904 - val_accuracy: 0.7680
Epoch 38/80
63/63 [==============================] - 2s 28ms/step - loss: 0.3049 - accura
cy: 0.8745 - val_loss: 0.5247 - val_accuracy: 0.8190
Epoch 39/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3203 - accura
cy: 0.8685 - val_loss: 0.4621 - val_accuracy: 0.8020
Epoch 40/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2851 - accura
cy: 0.8790 - val_loss: 0.7653 - val_accuracy: 0.7940
Epoch 41/80
63/63 [==============================] - 2s 29ms/step - loss: 0.3136 - accura
cy: 0.8660 - val_loss: 0.4930 - val_accuracy: 0.8060
Epoch 42/80
63/63 [==============================] - 2s 28ms/step - loss: 0.3056 - accura
cy: 0.8685 - val_loss: 0.4781 - val_accuracy: 0.8120
```

```
Epoch 43/80
63/63 [==============================] - 2s 30ms/step - loss: 0.2810 - accura
cy: 0.8810 - val_loss: 0.4700 - val_accuracy: 0.8290
Epoch 44/80
63/63 [==============================] - 2s 30ms/step - loss: 0.2661 - accura
cy: 0.8930 - val_loss: 1.2875 - val_accuracy: 0.6740
Epoch 45/80
63/63 [==============================] - 2s 30ms/step - loss: 0.2815 - accura
cy: 0.8910 - val_loss: 0.5469 - val_accuracy: 0.7930
Epoch 46/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2696 - accura
cy: 0.8940 - val_loss: 0.8227 - val_accuracy: 0.7920
Epoch 47/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2462 - accura
cy: 0.8980 - val_loss: 0.5565 - val_accuracy: 0.8330
Epoch 48/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2627 - accura
cy: 0.8925 - val_loss: 0.5671 - val_accuracy: 0.7800
Epoch 49/80
63/63 [==============================] - 2s 30ms/step - loss: 0.2446 - accura
cy: 0.8915 - val_loss: 0.5453 - val_accuracy: 0.8010
Epoch 50/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2383 - accura
cy: 0.8990 - val_loss: 0.4764 - val_accuracy: 0.8470
Epoch 51/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2692 - accura
cy: 0.8920 - val_loss: 0.4775 - val_accuracy: 0.8160
Epoch 52/80
63/63 [==============================] - 2s 28ms/step - loss: 0.2244 - accura
cy: 0.9120 - val_loss: 0.7375 - val_accuracy: 0.8030
Epoch 53/80
63/63 [==============================] - 2s 28ms/step - loss: 0.2228 - accura
cy: 0.9065 - val_loss: 0.5973 - val_accuracy: 0.8110
Epoch 54/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2456 - accura
cy: 0.9010 - val_loss: 0.5144 - val_accuracy: 0.8180
Epoch 55/80
63/63 [==============================] - 2s 28ms/step - loss: 0.2151 - accura
cy: 0.9180 - val_loss: 0.7244 - val_accuracy: 0.8210
Epoch 56/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2265 - accura
cy: 0.9105 - val_loss: 0.5406 - val_accuracy: 0.8110
Epoch 57/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2274 - accura
cy: 0.9110 - val_loss: 0.5016 - val_accuracy: 0.8160
Epoch 58/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2249 - accura
cy: 0.9165 - val_loss: 0.4593 - val_accuracy: 0.8380
Epoch 59/80
63/63 [==============================] - 2s 29ms/step - loss: 0.2030 - accura
cy: 0.9265 - val_loss: 0.6098 - val_accuracy: 0.8050
Epoch 60/80
```

```
63/63 [==============================] - 3s 39ms/step - loss: 0.2300 - accura
cy: 0.9165 - val_loss: 0.7569 - val_accuracy: 0.8280
Epoch 61/80
63/63 [==============================] - 2s 31ms/step - loss: 0.2432 - accura
cy: 0.9115 - val_loss: 0.4634 - val_accuracy: 0.8470
Epoch 62/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1986 - accura
cy: 0.9270 - val_loss: 0.5855 - val_accuracy: 0.8290
Epoch 63/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1942 - accura
cy: 0.9320 - val_loss: 0.5501 - val_accuracy: 0.8410
Epoch 64/80
63/63 [==============================] - 2s 30ms/step - loss: 0.2051 - accura
cy: 0.9255 - val_loss: 0.5786 - val_accuracy: 0.8270
Epoch 65/80
63/63 [==============================] - 2s 30ms/step - loss: 0.1830 - accura
cy: 0.9305 - val_loss: 0.6350 - val_accuracy: 0.8230
Epoch 66/80
63/63 [==============================] - 2s 30ms/step - loss: 0.2005 - accura
cy: 0.9205 - val_loss: 0.5401 - val_accuracy: 0.8370
Epoch 67/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1982 - accura
cy: 0.9245 - val_loss: 0.6137 - val_accuracy: 0.8020
Epoch 68/80
63/63 [==============================] - 2s 31ms/step - loss: 0.2312 - accura
cy: 0.9130 - val_loss: 0.5874 - val_accuracy: 0.8300
Epoch 69/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1951 - accura
cy: 0.9260 - val_loss: 0.4314 - val_accuracy: 0.8550
Epoch 70/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1619 - accura
cy: 0.9320 - val_loss: 0.7138 - val_accuracy: 0.8430
Epoch 71/80
63/63 [==============================] - 2s 31ms/step - loss: 0.2078 - accura
cy: 0.9200 - val_loss: 0.4732 - val_accuracy: 0.8670
Epoch 72/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1862 - accura
cy: 0.9265 - val_loss: 1.2674 - val_accuracy: 0.7960
Epoch 73/80
63/63 [==============================] - 2s 30ms/step - loss: 0.1802 - accura
cy: 0.9345 - val_loss: 0.5652 - val_accuracy: 0.8480
Epoch 74/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1870 - accura
cy: 0.9320 - val_loss: 0.6469 - val_accuracy: 0.8540
Epoch 75/80
63/63 [==============================] - 2s 30ms/step - loss: 0.1809 - accura
cy: 0.9335 - val_loss: 0.7291 - val_accuracy: 0.8450
Epoch 76/80
63/63 [==============================] - 2s 32ms/step - loss: 0.1844 - accura
cy: 0.9235 - val_loss: 0.5942 - val_accuracy: 0.8370
Epoch 77/80
63/63 [==============================] - 2s 31ms/step - loss: 0.1893 - accura
cy: 0.9335 - val_loss: 0.5588 - val_accuracy: 0.8310
```

```
Epoch 78/80
63/63 [==============================] - 2s 30ms/step - loss: 0.1505 - accura
cy: 0.9455 - val_loss: 0.5778 - val_accuracy: 0.8330
Epoch 79/80
63/63 [==============================] - 2s 30ms/step - loss: 0.1783 - accura
cy: 0.9325 - val_loss: 0.8527 - val_accuracy: 0.8380
Epoch 80/80
63/63 [==============================] - 2s 30ms/step - loss: 0.1884 - accura
cy: 0.9310 - val_loss: 0.7090 - val_accuracy: 0.8270
```

Evaluating the model on the test set

In [20]:

```
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 [==============================] - 1s 15ms/step - loss: 0.5437 - accura
cy: 0.8490
Test accuracy: 0.849
```

Training a convnet from scratch with training sample of 3000, a validation sample of 500, and a test sample of 500

Downloading the data

In [1]:

```
#!unzip -qq '/fs/ess/PGS0333/BA_64061_KSU/data/dogs-vs-cats.zip'
```

In [2]:

```
#!unzip -qq train.zip
```

Copying images to training, validation, and test directories

In [3]:

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index,
end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=2999)
make_subset("validation", start_index=3000, end_index=3499)
make_subset("test", start_index=3500, end_index=3999)
```

Building the model

Instantiating a small convnet for dogs vs. cats classification

In [4]:

```
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)
```

In [5]:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 43, 43, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 41, 41, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 12545
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

Configuring the model for training

In [6]:

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

Data preprocessing

Using image_dataset_from_directory to read images

In [7]:

```
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 5998 files belonging to 2 classes.
Found 998 files belonging to 2 classes.
Found 998 files belonging to 2 classes.
```

In [8]:

```
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)
```

In [9]:

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

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

In [10]:

```
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)
```

In [11]:

```
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

In [12]:

```
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

In [13]:

```
"""
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)
"""
```

Out[13]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch.keras",\n        save_best_only=True,\n        monitor="val
_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    epochs=30,\n    val
idation_data=validation_dataset,\n    callbacks=callbacks)\n'
```

Displaying curves of loss and accuracy during training

In [14]:

```
"""
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()
"""
```

Out[14]:

```
'\nimport matplotlib.pyplot as plt\naccuracy = history.history["accuracy"]\nv
al_accuracy = history.history["val_accuracy"]\nloss = history.history["loss"]
```

```
\nval_loss = history.history["val_loss"]\nepochs = range(1, len(accuracy) + 1)
\nplt.plot(epochs, accuracy, "bo", label="Training accuracy")\nplt.plot(epoch
s, val_accuracy, "b", label="Validation accuracy")\nplt.title("Training and v
alidation accuracy")\nplt.legend()\nplt.figure()\nplt.plot(epochs, loss, "bo",
 label="Training loss")\nplt.plot(epochs, val_loss, "b", label="Validation lo
ss")\nplt.title("Training and validation loss")\nplt.legend()\nplt.show()\n'
```

Evaluating the model on the test set

In [15]:

```
"""
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}")
"""
```

Out[15]:

```
'\ntest_model = keras.models.load_model("convnet_from_scratch.keras")\ntest_l
oss, test_acc = test_model.evaluate(test_dataset)\nprint(f"Test accuracy: {te
st_acc:.3f}")\n'
```

Using data augmentation

Define a data augmentation stage to add to an image model

In [16]:

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

Displaying some randomly augmented training images

In [17]:

```
"""
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")
"""
```

Out[17]:

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

Defining a new convnet that includes image augmentation, regularization and dropout

In [18]:

```
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

In [19]:

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=80,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/80
188/188 [==============================] - 9s 28ms/step - loss: 0.7013 - accu
racy: 0.5367 - val_loss: 0.6745 - val_accuracy: 0.6222
Epoch 2/80
188/188 [==============================] - 5s 25ms/step - loss: 0.6622 - accu
racy: 0.6307 - val_loss: 0.6223 - val_accuracy: 0.6774
Epoch 3/80
188/188 [==============================] - 5s 25ms/step - loss: 0.6244 - accu
racy: 0.6557 - val_loss: 0.5671 - val_accuracy: 0.7194
Epoch 4/80
188/188 [==============================] - 5s 25ms/step - loss: 0.5998 - accu
racy: 0.6812 - val_loss: 0.6425 - val_accuracy: 0.6683
Epoch 5/80
188/188 [==============================] - 5s 24ms/step - loss: 0.5709 - accu
racy: 0.7062 - val_loss: 0.5270 - val_accuracy: 0.7545
Epoch 6/80
188/188 [==============================] - 5s 24ms/step - loss: 0.5397 - accu
racy: 0.7249 - val_loss: 0.4980 - val_accuracy: 0.7625
Epoch 7/80
```

```
188/188 [==============================] - 5s 24ms/step - loss: 0.5218 - accu
racy: 0.7487 - val_loss: 0.4215 - val_accuracy: 0.8146
Epoch 8/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4977 - accu
racy: 0.7598 - val_loss: 0.4423 - val_accuracy: 0.7976
Epoch 9/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4810 - accu
racy: 0.7749 - val_loss: 0.4341 - val_accuracy: 0.8096
Epoch 10/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4671 - accu
racy: 0.7798 - val_loss: 0.4010 - val_accuracy: 0.8196
Epoch 11/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4539 - accu
racy: 0.7893 - val_loss: 0.3904 - val_accuracy: 0.8096
Epoch 12/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4378 - accu
racy: 0.8018 - val_loss: 0.3867 - val_accuracy: 0.8196
Epoch 13/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4131 - accu
racy: 0.8129 - val_loss: 0.3734 - val_accuracy: 0.8297
Epoch 14/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4097 - accu
racy: 0.8184 - val_loss: 0.3403 - val_accuracy: 0.8637
Epoch 15/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4037 - accu
racy: 0.8268 - val_loss: 0.3120 - val_accuracy: 0.8637
Epoch 16/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3826 - accu
racy: 0.8336 - val_loss: 0.4983 - val_accuracy: 0.7936
Epoch 17/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3803 - accu
racy: 0.8348 - val_loss: 0.4090 - val_accuracy: 0.8537
Epoch 18/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3632 - accu
racy: 0.8408 - val_loss: 0.3277 - val_accuracy: 0.8848
Epoch 19/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3537 - accu
racy: 0.8503 - val_loss: 0.4235 - val_accuracy: 0.8317
Epoch 20/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3514 - accu
racy: 0.8546 - val_loss: 0.2837 - val_accuracy: 0.8818
Epoch 21/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3369 - accu
racy: 0.8550 - val_loss: 0.3131 - val_accuracy: 0.8828
Epoch 22/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3493 - accu
racy: 0.8590 - val_loss: 0.3317 - val_accuracy: 0.8557
Epoch 23/80
188/188 [==============================] - 5s 26ms/step - loss: 0.3231 - accu
racy: 0.8615 - val_loss: 0.3820 - val_accuracy: 0.8457
Epoch 24/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3298 - accu
racy: 0.8573 - val_loss: 0.4611 - val_accuracy: 0.7836
```

```
Epoch 25/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3255 - accu
racy: 0.8560 - val_loss: 0.3417 - val_accuracy: 0.8717
Epoch 26/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3262 - accu
racy: 0.8651 - val_loss: 0.2503 - val_accuracy: 0.9108
Epoch 27/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3250 - accu
racy: 0.8646 - val_loss: 0.2798 - val_accuracy: 0.9068
Epoch 28/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3126 - accu
racy: 0.8733 - val_loss: 0.3802 - val_accuracy: 0.8577
Epoch 29/80
188/188 [==============================] - 5s 24ms/step - loss: 0.2980 - accu
racy: 0.8746 - val_loss: 0.2611 - val_accuracy: 0.9098
Epoch 30/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3084 - accu
racy: 0.8713 - val_loss: 0.2998 - val_accuracy: 0.8888
Epoch 31/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3241 - accu
racy: 0.8648 - val_loss: 0.3702 - val_accuracy: 0.8527
Epoch 32/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3175 - accu
racy: 0.8705 - val_loss: 0.2866 - val_accuracy: 0.8938
Epoch 33/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3435 - accu
racy: 0.8593 - val_loss: 0.4798 - val_accuracy: 0.8838
Epoch 34/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3433 - accu
racy: 0.8573 - val_loss: 0.2443 - val_accuracy: 0.9048
Epoch 35/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3306 - accu
racy: 0.8686 - val_loss: 2.3848 - val_accuracy: 0.7144
Epoch 36/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3549 - accu
racy: 0.8630 - val_loss: 0.3681 - val_accuracy: 0.8798
Epoch 37/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3391 - accu
racy: 0.8623 - val_loss: 0.3490 - val_accuracy: 0.8868
Epoch 38/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3542 - accu
racy: 0.8596 - val_loss: 0.3876 - val_accuracy: 0.8397
Epoch 39/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3368 - accu
racy: 0.8620 - val_loss: 0.5940 - val_accuracy: 0.8717
Epoch 40/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3310 - accu
racy: 0.8706 - val_loss: 0.2680 - val_accuracy: 0.8958
Epoch 41/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3299 - accu
racy: 0.8665 - val_loss: 0.3128 - val_accuracy: 0.8928
Epoch 42/80
```

```
188/188 [==============================] - 5s 24ms/step - loss: 0.3455 - accu
racy: 0.8573 - val_loss: 0.3010 - val_accuracy: 0.8918
Epoch 43/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3463 - accu
racy: 0.8610 - val_loss: 0.4598 - val_accuracy: 0.8377
Epoch 44/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3580 - accu
racy: 0.8603 - val_loss: 0.2928 - val_accuracy: 0.9018
Epoch 45/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3565 - accu
racy: 0.8621 - val_loss: 0.3692 - val_accuracy: 0.8287
Epoch 46/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3547 - accu
racy: 0.8610 - val_loss: 0.3488 - val_accuracy: 0.8798
Epoch 47/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3523 - accu
racy: 0.8560 - val_loss: 0.5674 - val_accuracy: 0.7946
Epoch 48/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3737 - accu
racy: 0.8553 - val_loss: 0.6356 - val_accuracy: 0.8146
Epoch 49/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3734 - accu
racy: 0.8503 - val_loss: 0.9687 - val_accuracy: 0.7445
Epoch 50/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3659 - accu
racy: 0.8525 - val_loss: 0.3908 - val_accuracy: 0.8427
Epoch 51/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3566 - accu
racy: 0.8606 - val_loss: 0.9658 - val_accuracy: 0.7495
Epoch 52/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3662 - accu
racy: 0.8521 - val_loss: 0.3714 - val_accuracy: 0.8968
Epoch 53/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3781 - accu
racy: 0.8506 - val_loss: 0.2634 - val_accuracy: 0.9038
Epoch 54/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3798 - accu
racy: 0.8456 - val_loss: 1.2284 - val_accuracy: 0.8176
Epoch 55/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4102 - accu
racy: 0.8453 - val_loss: 0.2998 - val_accuracy: 0.9148
Epoch 56/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4404 - accu
racy: 0.8251 - val_loss: 0.3784 - val_accuracy: 0.8828
Epoch 57/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3989 - accu
racy: 0.8386 - val_loss: 0.2849 - val_accuracy: 0.8968
Epoch 58/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4147 - accu
racy: 0.8403 - val_loss: 0.2906 - val_accuracy: 0.9058
Epoch 59/80
188/188 [==============================] - 5s 25ms/step - loss: 0.3818 - accu
racy: 0.8479 - val_loss: 0.4894 - val_accuracy: 0.8798
```

```
Epoch 60/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3876 - accu
racy: 0.8358 - val_loss: 1.0239 - val_accuracy: 0.8267
Epoch 61/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4747 - accu
racy: 0.8213 - val_loss: 0.3941 - val_accuracy: 0.8798
Epoch 62/80
188/188 [==============================] - 5s 24ms/step - loss: 0.3936 - accu
racy: 0.8324 - val_loss: 0.5380 - val_accuracy: 0.8257
Epoch 63/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4257 - accu
racy: 0.8293 - val_loss: 0.4603 - val_accuracy: 0.8547
Epoch 64/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4307 - accu
racy: 0.8311 - val_loss: 0.3651 - val_accuracy: 0.8206
Epoch 65/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4546 - accu
racy: 0.8168 - val_loss: 0.2663 - val_accuracy: 0.8818
Epoch 66/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4361 - accu
racy: 0.8286 - val_loss: 0.3358 - val_accuracy: 0.8868
Epoch 67/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4496 - accu
racy: 0.8188 - val_loss: 0.3353 - val_accuracy: 0.8808
Epoch 68/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4687 - accu
racy: 0.8269 - val_loss: 0.4740 - val_accuracy: 0.8667
Epoch 69/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4311 - accu
racy: 0.8231 - val_loss: 0.4173 - val_accuracy: 0.8707
Epoch 70/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4982 - accu
racy: 0.8074 - val_loss: 0.5288 - val_accuracy: 0.8236
Epoch 71/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4758 - accu
racy: 0.8173 - val_loss: 0.4178 - val_accuracy: 0.8216
Epoch 72/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4796 - accu
racy: 0.8183 - val_loss: 0.3006 - val_accuracy: 0.8888
Epoch 73/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4372 - accu
racy: 0.8184 - val_loss: 0.4484 - val_accuracy: 0.8487
Epoch 74/80
188/188 [==============================] - 5s 24ms/step - loss: 0.6609 - accu
racy: 0.8118 - val_loss: 2.0788 - val_accuracy: 0.6703
Epoch 75/80
188/188 [==============================] - 5s 25ms/step - loss: 0.5195 - accu
racy: 0.8229 - val_loss: 0.7086 - val_accuracy: 0.8156
Epoch 76/80
188/188 [==============================] - 5s 25ms/step - loss: 0.5815 - accu
racy: 0.8069 - val_loss: 0.3858 - val_accuracy: 0.8858
Epoch 77/80
```

```
188/188 [==============================] - 5s 25ms/step - loss: 0.5060 - accu
racy: 0.8174 - val_loss: 0.3619 - val_accuracy: 0.8367
Epoch 78/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4919 - accu
racy: 0.8014 - val_loss: 0.3146 - val_accuracy: 0.8727
Epoch 79/80
188/188 [==============================] - 5s 25ms/step - loss: 0.4880 - accu
racy: 0.8144 - val_loss: 2.6971 - val_accuracy: 0.7094
Epoch 80/80
188/188 [==============================] - 5s 24ms/step - loss: 0.4626 - accu
racy: 0.8159 - val_loss: 0.3114 - val_accuracy: 0.8657
```

Evaluating the model on the test set

In [20]:

```
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 [==============================] - 1s 14ms/step - loss: 0.3999 - accura
cy: 0.8868
Test accuracy: 0.887
```

Training a convnet from scratch with training sample of 5000, a validation sample of 500, and a test sample of 500

Downloading the data

In [1]:

```
!unzip -qq '/fs/ess/PGS0333/BA_64061_KSU/data/dogs-vs-cats.zip'
```

In [2]:

```
!unzip -qq train.zip
```

Copying images to training, validation, and test directories

In [3]:

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, 
end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=5000)
make_subset("validation", start_index=5000, end_index=5500)
make_subset("test", start_index=5500, end_index=6000)
```

Building the model

Instantiating a small convnet for dogs vs. cats classification

In [4]:

```
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)
```

In [5]:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 43, 43, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 41, 41, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 12545
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

Configuring the model for training

In [6]:

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

Data preprocessing

Using image_dataset_from_directory to read images

In [7]:

```
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 10000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

In [8]:

```
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)
```

In [9]:

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

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

In [10]:

```
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)
```

In [11]:

```
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

In [12]:

```
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

In [13]:

```
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
313/313 [==============================] - 13s 25ms/step - loss: 0.7100 - acc
uracy: 0.5490 - val_loss: 0.6479 - val_accuracy: 0.6480
Epoch 2/30
313/313 [==============================] - 7s 22ms/step - loss: 0.6220 - accu
racy: 0.6644 - val_loss: 0.5241 - val_accuracy: 0.7440
Epoch 3/30
313/313 [==============================] - 7s 22ms/step - loss: 0.5212 - accu
racy: 0.7444 - val_loss: 0.4661 - val_accuracy: 0.7810
Epoch 4/30
313/313 [==============================] - 7s 22ms/step - loss: 0.4452 - accu
racy: 0.7956 - val_loss: 0.4746 - val_accuracy: 0.7870
Epoch 5/30
313/313 [==============================] - 7s 23ms/step - loss: 0.3822 - accu
racy: 0.8326 - val_loss: 0.5290 - val_accuracy: 0.7740
Epoch 6/30
313/313 [==============================] - 7s 22ms/step - loss: 0.3272 - accu
racy: 0.8573 - val_loss: 0.5761 - val_accuracy: 0.7860
Epoch 7/30
313/313 [==============================] - 7s 22ms/step - loss: 0.2765 - accu
racy: 0.8835 - val_loss: 0.4929 - val_accuracy: 0.7930
Epoch 8/30
313/313 [==============================] - 7s 22ms/step - loss: 0.2166 - accu
racy: 0.9101 - val_loss: 0.4669 - val_accuracy: 0.8280
Epoch 9/30
313/313 [==============================] - 7s 22ms/step - loss: 0.1701 - accu
racy: 0.9297 - val_loss: 0.6306 - val_accuracy: 0.8150
Epoch 10/30
313/313 [==============================] - 7s 22ms/step - loss: 0.1321 - accu
racy: 0.9482 - val_loss: 0.6038 - val_accuracy: 0.8430
Epoch 11/30
```

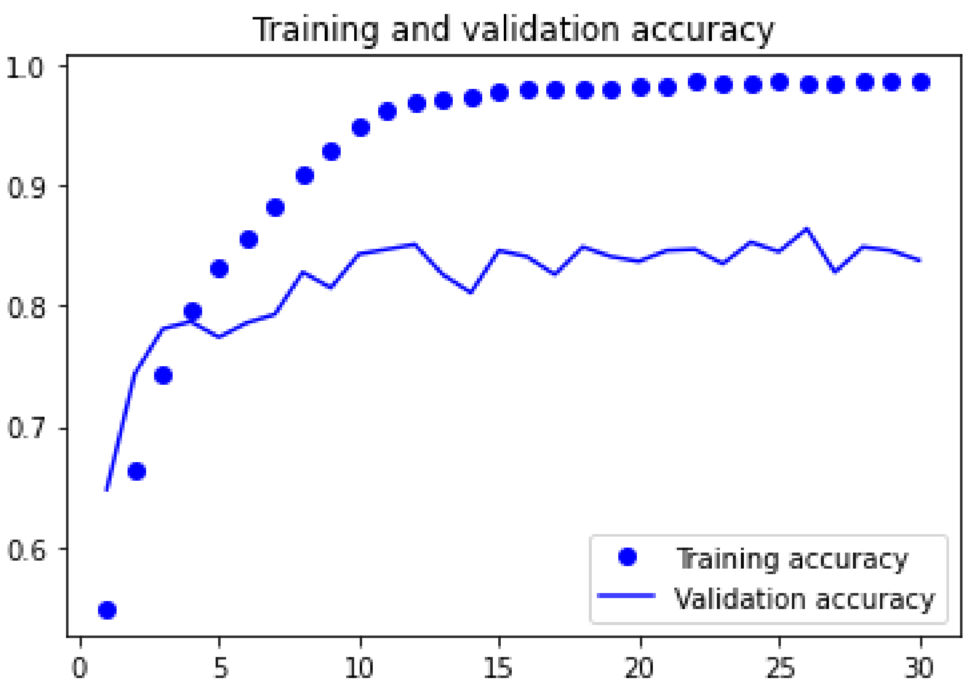
```
313/313 [==============================] - 7s 23ms/step - loss: 0.1071 - accu
racy: 0.9617 - val_loss: 0.6322 - val_accuracy: 0.8470
Epoch 12/30
313/313 [==============================] - 7s 23ms/step - loss: 0.0902 - accu
racy: 0.9680 - val_loss: 0.6687 - val_accuracy: 0.8510
Epoch 13/30
313/313 [==============================] - 7s 23ms/step - loss: 0.0833 - accu
racy: 0.9704 - val_loss: 0.7229 - val_accuracy: 0.8260
Epoch 14/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0759 - accu
racy: 0.9729 - val_loss: 0.7790 - val_accuracy: 0.8110
Epoch 15/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0653 - accu
racy: 0.9785 - val_loss: 0.7850 - val_accuracy: 0.8460
Epoch 16/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0643 - accu
racy: 0.9800 - val_loss: 0.8671 - val_accuracy: 0.8410
Epoch 17/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0626 - accu
racy: 0.9803 - val_loss: 1.5262 - val_accuracy: 0.8260
Epoch 18/30
313/313 [==============================] - 7s 23ms/step - loss: 0.0674 - accu
racy: 0.9806 - val_loss: 1.0057 - val_accuracy: 0.8490
Epoch 19/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0605 - accu
racy: 0.9800 - val_loss: 1.0948 - val_accuracy: 0.8410
Epoch 20/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0641 - accu
racy: 0.9825 - val_loss: 1.2610 - val_accuracy: 0.8370
Epoch 21/30
313/313 [==============================] - 8s 24ms/step - loss: 0.0590 - accu
racy: 0.9823 - val_loss: 1.3441 - val_accuracy: 0.8460
Epoch 22/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0557 - accu
racy: 0.9862 - val_loss: 1.4184 - val_accuracy: 0.8470
Epoch 23/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0610 - accu
racy: 0.9842 - val_loss: 1.3167 - val_accuracy: 0.8350
Epoch 24/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0581 - accu
racy: 0.9847 - val_loss: 1.1253 - val_accuracy: 0.8530
Epoch 25/30
313/313 [==============================] - 7s 23ms/step - loss: 0.0504 - accu
racy: 0.9858 - val_loss: 1.3824 - val_accuracy: 0.8450
Epoch 26/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0646 - accu
racy: 0.9854 - val_loss: 1.2968 - val_accuracy: 0.8640
Epoch 27/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0657 - accu
racy: 0.9851 - val_loss: 1.4639 - val_accuracy: 0.8280
Epoch 28/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0579 - accu
racy: 0.9862 - val_loss: 1.4657 - val_accuracy: 0.8490
```

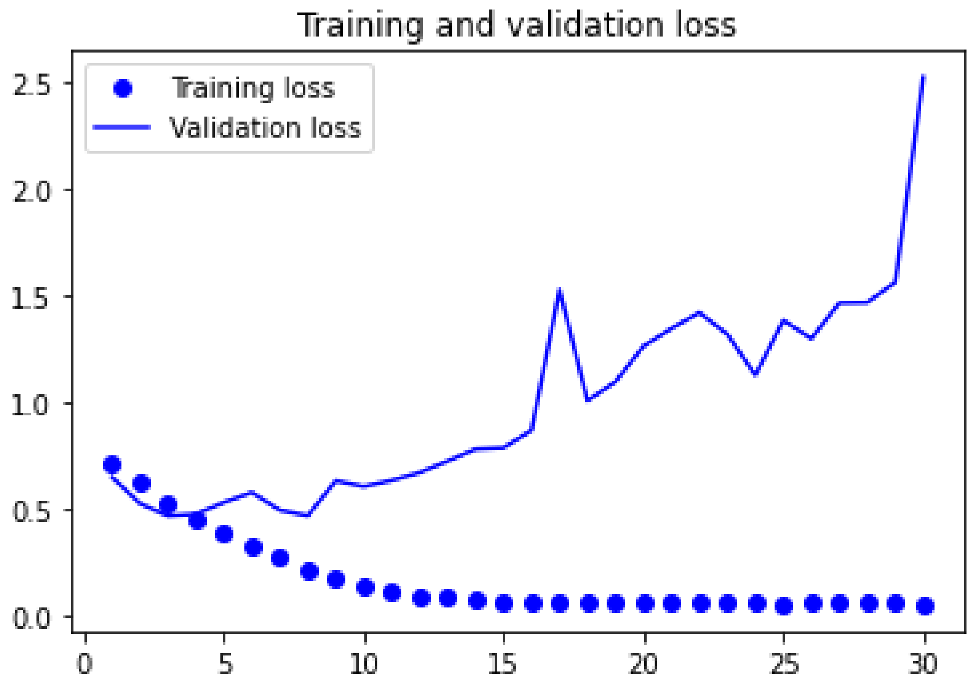
```
Epoch 29/30
313/313 [==============================] - 7s 22ms/step - loss: 0.0597 - accu
racy: 0.9860 - val_loss: 1.5611 - val_accuracy: 0.8460
Epoch 30/30
313/313 [==============================] - 7s 23ms/step - loss: 0.0515 - accu
racy: 0.9861 - val_loss: 2.5216 - val_accuracy: 0.8380
```

Displaying curves of loss and accuracy during training

In [14]:

```
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

In [15]:

```
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 [==============================] - 1s 15ms/step - loss: 0.4459 - accura
cy: 0.7990
Test accuracy: 0.799
```

Using data augmentation

Define a data augmentation stage to add to an image model

In [16]:

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

Displaying some randomly augmented training images

In [17]:

```
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")
```

Defining a new convnet that includes image augmentation, regularization and dropout

In [18]:

```
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

In [19]:

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=80,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/80
313/313 [==============================] - 8s 23ms/step - loss: 0.7085 - accu
racy: 0.5472 - val_loss: 0.6473 - val_accuracy: 0.6280
Epoch 2/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6436 - accu
racy: 0.6348 - val_loss: 0.5860 - val_accuracy: 0.6870
Epoch 3/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6026 - accu
racy: 0.6773 - val_loss: 0.5413 - val_accuracy: 0.7280
Epoch 4/80
313/313 [==============================] - 8s 25ms/step - loss: 0.5605 - accu
racy: 0.7158 - val_loss: 0.6777 - val_accuracy: 0.6710
Epoch 5/80
313/313 [==============================] - 7s 23ms/step - loss: 0.5249 - accu
racy: 0.7430 - val_loss: 0.5795 - val_accuracy: 0.7050
Epoch 6/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4967 - accu
racy: 0.7601 - val_loss: 0.4453 - val_accuracy: 0.7990
Epoch 7/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4714 - accu
racy: 0.7775 - val_loss: 0.4263 - val_accuracy: 0.8020
```

```
Epoch 8/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4315 - accu
racy: 0.8005 - val_loss: 0.3825 - val_accuracy: 0.8320
Epoch 9/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4126 - accu
racy: 0.8136 - val_loss: 0.3821 - val_accuracy: 0.8300
Epoch 10/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3923 - accu
racy: 0.8244 - val_loss: 0.9279 - val_accuracy: 0.6840
Epoch 11/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3709 - accu
racy: 0.8406 - val_loss: 0.6279 - val_accuracy: 0.7800
Epoch 12/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3564 - accu
racy: 0.8476 - val_loss: 0.5450 - val_accuracy: 0.8020
Epoch 13/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3380 - accu
racy: 0.8495 - val_loss: 0.2726 - val_accuracy: 0.8870
Epoch 14/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3357 - accu
racy: 0.8587 - val_loss: 0.2468 - val_accuracy: 0.9020
Epoch 15/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3135 - accu
racy: 0.8667 - val_loss: 0.2766 - val_accuracy: 0.8790
Epoch 16/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3212 - accu
racy: 0.8664 - val_loss: 1.1412 - val_accuracy: 0.6280
Epoch 17/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3134 - accu
racy: 0.8695 - val_loss: 0.2261 - val_accuracy: 0.9020
Epoch 18/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3096 - accu
racy: 0.8720 - val_loss: 0.2283 - val_accuracy: 0.9130
Epoch 19/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3088 - accu
racy: 0.8674 - val_loss: 0.4922 - val_accuracy: 0.8490
Epoch 20/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3031 - accu
racy: 0.8750 - val_loss: 0.2459 - val_accuracy: 0.8970
Epoch 21/80
313/313 [==============================] - 7s 24ms/step - loss: 0.3020 - accu
racy: 0.8723 - val_loss: 0.2751 - val_accuracy: 0.8980
Epoch 22/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3065 - accu
racy: 0.8706 - val_loss: 0.2204 - val_accuracy: 0.9210
Epoch 23/80
313/313 [==============================] - 7s 23ms/step - loss: 0.2973 - accu
racy: 0.8759 - val_loss: 0.2631 - val_accuracy: 0.8910
Epoch 24/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3077 - accu
racy: 0.8753 - val_loss: 0.2650 - val_accuracy: 0.8980
Epoch 25/80
```

```
313/313 [==============================] - 7s 23ms/step - loss: 0.3067 - accu
racy: 0.8758 - val_loss: 0.2240 - val_accuracy: 0.9170
Epoch 26/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3196 - accu
racy: 0.8688 - val_loss: 0.2485 - val_accuracy: 0.8890
Epoch 27/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3124 - accu
racy: 0.8727 - val_loss: 0.3176 - val_accuracy: 0.8990
Epoch 28/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3297 - accu
racy: 0.8622 - val_loss: 0.2380 - val_accuracy: 0.8970
Epoch 29/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3271 - accu
racy: 0.8665 - val_loss: 0.3226 - val_accuracy: 0.8960
Epoch 30/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3322 - accu
racy: 0.8668 - val_loss: 0.4391 - val_accuracy: 0.8790
Epoch 31/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3331 - accu
racy: 0.8623 - val_loss: 0.2398 - val_accuracy: 0.9170
Epoch 32/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3690 - accu
racy: 0.8528 - val_loss: 0.3074 - val_accuracy: 0.8740
Epoch 33/80
313/313 [==============================] - 8s 24ms/step - loss: 0.3580 - accu
racy: 0.8530 - val_loss: 0.2445 - val_accuracy: 0.8970
Epoch 34/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3646 - accu
racy: 0.8568 - val_loss: 0.3815 - val_accuracy: 0.8040
Epoch 35/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3949 - accu
racy: 0.8419 - val_loss: 0.3172 - val_accuracy: 0.8670
Epoch 36/80
313/313 [==============================] - 7s 23ms/step - loss: 0.3834 - accu
racy: 0.8471 - val_loss: 0.2677 - val_accuracy: 0.8950
Epoch 37/80
313/313 [==============================] - 8s 24ms/step - loss: 0.4062 - accu
racy: 0.8369 - val_loss: 0.9175 - val_accuracy: 0.8380
Epoch 38/80
313/313 [==============================] - 9s 27ms/step - loss: 0.3934 - accu
racy: 0.8404 - val_loss: 0.3398 - val_accuracy: 0.9140
Epoch 39/80
313/313 [==============================] - 9s 29ms/step - loss: 0.4115 - accu
racy: 0.8388 - val_loss: 0.3155 - val_accuracy: 0.8780
Epoch 40/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4306 - accu
racy: 0.8331 - val_loss: 0.6346 - val_accuracy: 0.7710
Epoch 41/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4223 - accu
racy: 0.8352 - val_loss: 0.3256 - val_accuracy: 0.8680
Epoch 42/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4123 - accu
racy: 0.8353 - val_loss: 0.2849 - val_accuracy: 0.8780
```

```
Epoch 43/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4292 - accu
racy: 0.8374 - val_loss: 0.4640 - val_accuracy: 0.7860
Epoch 44/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4406 - accu
racy: 0.8268 - val_loss: 0.3164 - val_accuracy: 0.8670
Epoch 45/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4382 - accu
racy: 0.8288 - val_loss: 0.6347 - val_accuracy: 0.8510
Epoch 46/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4506 - accu
racy: 0.8228 - val_loss: 0.4618 - val_accuracy: 0.8030
Epoch 47/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4790 - accu
racy: 0.8111 - val_loss: 0.3703 - val_accuracy: 0.8700
Epoch 48/80
313/313 [==============================] - 8s 24ms/step - loss: 0.4737 - accu
racy: 0.8153 - val_loss: 0.4769 - val_accuracy: 0.8420
Epoch 49/80
313/313 [==============================] - 8s 24ms/step - loss: 0.5042 - accu
racy: 0.8108 - val_loss: 0.4332 - val_accuracy: 0.8170
Epoch 50/80
313/313 [==============================] - 8s 24ms/step - loss: 0.4661 - accu
racy: 0.8106 - val_loss: 0.3641 - val_accuracy: 0.8390
Epoch 51/80
313/313 [==============================] - 7s 23ms/step - loss: 0.5111 - accu
racy: 0.8038 - val_loss: 0.5490 - val_accuracy: 0.7600
Epoch 52/80
313/313 [==============================] - 7s 23ms/step - loss: 0.4933 - accu
racy: 0.8100 - val_loss: 0.4601 - val_accuracy: 0.8140
Epoch 53/80
313/313 [==============================] - 7s 23ms/step - loss: 0.5665 - accu
racy: 0.7935 - val_loss: 0.5204 - val_accuracy: 0.8180
Epoch 54/80
313/313 [==============================] - 8s 24ms/step - loss: 0.8305 - accu
racy: 0.7883 - val_loss: 0.4358 - val_accuracy: 0.8030
Epoch 55/80
313/313 [==============================] - 8s 24ms/step - loss: 0.5653 - accu
racy: 0.7699 - val_loss: 1.1045 - val_accuracy: 0.7030
Epoch 56/80
313/313 [==============================] - 7s 23ms/step - loss: 0.5717 - accu
racy: 0.7654 - val_loss: 0.6066 - val_accuracy: 0.6160
Epoch 57/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6231 - accu
racy: 0.7561 - val_loss: 0.7035 - val_accuracy: 0.8050
Epoch 58/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6366 - accu
racy: 0.7521 - val_loss: 0.3822 - val_accuracy: 0.8320
Epoch 59/80
313/313 [==============================] - 7s 23ms/step - loss: 0.7207 - accu
racy: 0.7526 - val_loss: 0.4978 - val_accuracy: 0.7790
Epoch 60/80
```

```
313/313 [==============================] - 7s 23ms/step - loss: 0.5988 - accu
racy: 0.7567 - val_loss: 0.7791 - val_accuracy: 0.8630
Epoch 61/80
313/313 [==============================] - 8s 24ms/step - loss: 0.7173 - accu
racy: 0.7496 - val_loss: 0.3506 - val_accuracy: 0.8240
Epoch 62/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6413 - accu
racy: 0.7624 - val_loss: 0.5071 - val_accuracy: 0.7890
Epoch 63/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6937 - accu
racy: 0.7424 - val_loss: 0.5318 - val_accuracy: 0.7410
Epoch 64/80
313/313 [==============================] - 7s 23ms/step - loss: 0.5900 - accu
racy: 0.7461 - val_loss: 0.4062 - val_accuracy: 0.8290
Epoch 65/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6657 - accu
racy: 0.7406 - val_loss: 0.4436 - val_accuracy: 0.7760
Epoch 66/80
313/313 [==============================] - 7s 24ms/step - loss: 0.6120 - accu
racy: 0.7284 - val_loss: 0.3400 - val_accuracy: 0.8570
Epoch 67/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6894 - accu
racy: 0.7491 - val_loss: 0.5961 - val_accuracy: 0.7390
Epoch 68/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6624 - accu
racy: 0.7276 - val_loss: 0.4832 - val_accuracy: 0.7910
Epoch 69/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6145 - accu
racy: 0.7428 - val_loss: 0.4742 - val_accuracy: 0.7550
Epoch 70/80
313/313 [==============================] - 8s 26ms/step - loss: 0.7200 - accu
racy: 0.7363 - val_loss: 0.5561 - val_accuracy: 0.7350
Epoch 71/80
313/313 [==============================] - 8s 24ms/step - loss: 0.6692 - accu
racy: 0.7420 - val_loss: 0.5122 - val_accuracy: 0.7760
Epoch 72/80
313/313 [==============================] - 8s 24ms/step - loss: 0.6900 - accu
racy: 0.7354 - val_loss: 2.8342 - val_accuracy: 0.5990
Epoch 73/80
313/313 [==============================] - 8s 24ms/step - loss: 0.6887 - accu
racy: 0.7213 - val_loss: 0.5601 - val_accuracy: 0.7590
Epoch 74/80
313/313 [==============================] - 7s 23ms/step - loss: 0.6664 - accu
racy: 0.7248 - val_loss: 1.0051 - val_accuracy: 0.5290
Epoch 75/80
313/313 [==============================] - 8s 24ms/step - loss: 0.6987 - accu
racy: 0.7268 - val_loss: 0.5395 - val_accuracy: 0.7330
Epoch 76/80
313/313 [==============================] - 8s 24ms/step - loss: 0.7089 - accu
racy: 0.7017 - val_loss: 0.5606 - val_accuracy: 0.6710
Epoch 77/80
313/313 [==============================] - 8s 26ms/step - loss: 0.6801 - accu
racy: 0.7002 - val_loss: 0.4867 - val_accuracy: 0.7900
```

```
Epoch 78/80
313/313 [==============================] - 9s 27ms/step - loss: 0.6834 - accu
racy: 0.7238 - val_loss: 0.6524 - val_accuracy: 0.7180
Epoch 79/80
313/313 [==============================] - 8s 24ms/step - loss: 0.7559 - accu
racy: 0.7261 - val_loss: 0.5264 - val_accuracy: 0.7770
Epoch 80/80
313/313 [==============================] - 8s 24ms/step - loss: 0.6554 - accu
racy: 0.7236 - val_loss: 0.5299 - val_accuracy: 0.7480
```

Evaluating the model on the test set

In [20]:

```
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 [==============================] - 1s 15ms/step - loss: 0.2071 - accura
cy: 0.9150
Test accuracy: 0.915
```

Training a convnet from scratch with training sample of 8000, a validation sample of 500, and a test sample of 500

Downloading the data

In [1]:

```
#!unzip -qq '/fs/ess/PGS0333/BA_64061_KSU/data/dogs-vs-cats.zip'
```

In [2]:

```
#!unzip -qq train.zip
```

Copying images to training, validation, and test directories

In [3]:

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index,
end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=7999)
make_subset("validation", start_index=8000, end_index=8499)
make_subset("test", start_index=8499, end_index=8999)
```

Building the model

Instantiating a small convnet for dogs vs. cats classification

In [4]:

```
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)
```

In [5]:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 180, 180, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 43, 43, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 41, 41, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 12545
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
_________________________________________________________________
```

Configuring the model for training

In [6]:

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

Data preprocessing

Using image_dataset_from_directory to read images

In [7]:

```
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 15998 files belonging to 2 classes.
Found 998 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

In [8]:

```
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)
```

In [9]:

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

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

In [10]:

```
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)
```

In [11]:

```
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

In [12]:

```
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

In [13]:

```
"""
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)
"""
```

Out[13]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch.keras",\n        save_best_only=True,\n        monitor="val
_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    epochs=30,\n    val
idation_data=validation_dataset,\n    callbacks=callbacks)\n'
```

Displaying curves of loss and accuracy during training

In [14]:

```
"""
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()
"""
```

Out[14]:

```
'\nimport matplotlib.pyplot as plt\naccuracy = history.history["accuracy"]\nv
al_accuracy = history.history["val_accuracy"]\nloss = history.history["loss"]
```

```
\nval_loss = history.history["val_loss"]\nepochs = range(1, len(accuracy) + 1)
\nplt.plot(epochs, accuracy, "bo", label="Training accuracy")\nplt.plot(epoch
s, val_accuracy, "b", label="Validation accuracy")\nplt.title("Training and v
alidation accuracy")\nplt.legend()\nplt.figure()\nplt.plot(epochs, loss, "bo",
 label="Training loss")\nplt.plot(epochs, val_loss, "b", label="Validation lo
ss")\nplt.title("Training and validation loss")\nplt.legend()\nplt.show()\n'
```

Evaluating the model on the test set

In [15]:

```
"""
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}")
"""
```

Out[15]:

```
'\ntest_model = keras.models.load_model("convnet_from_scratch.keras")\ntest_l
oss, test_acc = test_model.evaluate(test_dataset)\nprint(f"Test accuracy: {te
st_acc:.3f}")\n'
```

Using data augmentation

Define a data augmentation stage to add to an image model

In [16]:

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

Displaying some randomly augmented training images

In [17]:

```
"""
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")
"""
```

Out[17]:

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

Defining a new convnet that includes image augmentation, regularization and dropout

In [18]:

```
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

In [19]:

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=80,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/80
500/500 [==============================] - 16s 24ms/step - loss: 0.6698 - acc
uracy: 0.5857 - val_loss: 0.5948 - val_accuracy: 0.6974
Epoch 2/80
500/500 [==============================] - 12s 23ms/step - loss: 0.5821 - acc
uracy: 0.6964 - val_loss: 0.5041 - val_accuracy: 0.7465
Epoch 3/80
500/500 [==============================] - 11s 23ms/step - loss: 0.5173 - acc
uracy: 0.7536 - val_loss: 0.4001 - val_accuracy: 0.8196
Epoch 4/80
500/500 [==============================] - 12s 23ms/step - loss: 0.4723 - acc
uracy: 0.7782 - val_loss: 0.4987 - val_accuracy: 0.7836
Epoch 5/80
500/500 [==============================] - 12s 23ms/step - loss: 0.4270 - acc
uracy: 0.8062 - val_loss: 0.3770 - val_accuracy: 0.8367
Epoch 6/80
500/500 [==============================] - 12s 23ms/step - loss: 0.3997 - acc
uracy: 0.8220 - val_loss: 0.2858 - val_accuracy: 0.8818
Epoch 7/80
```

```
500/500 [==============================] - 12s 23ms/step - loss: 0.3660 - acc
uracy: 0.8396 - val_loss: 0.3034 - val_accuracy: 0.8758
Epoch 8/80
500/500 [==============================] - 11s 22ms/step - loss: 0.3541 - acc
uracy: 0.8462 - val_loss: 0.2472 - val_accuracy: 0.8988
Epoch 9/80
500/500 [==============================] - 12s 23ms/step - loss: 0.3404 - acc
uracy: 0.8548 - val_loss: 0.3367 - val_accuracy: 0.8567
Epoch 10/80
500/500 [==============================] - 12s 23ms/step - loss: 0.3285 - acc
uracy: 0.8629 - val_loss: 0.2327 - val_accuracy: 0.8998
Epoch 11/80
500/500 [==============================] - 11s 23ms/step - loss: 0.3237 - acc
uracy: 0.8645 - val_loss: 0.2286 - val_accuracy: 0.9058
Epoch 12/80
500/500 [==============================] - 12s 23ms/step - loss: 0.3222 - acc
uracy: 0.8661 - val_loss: 0.2522 - val_accuracy: 0.9038
Epoch 13/80
500/500 [==============================] - 12s 23ms/step - loss: 0.3229 - acc
uracy: 0.8694 - val_loss: 0.2510 - val_accuracy: 0.9128
Epoch 14/80
500/500 [==============================] - 12s 23ms/step - loss: 0.3242 - acc
uracy: 0.8648 - val_loss: 0.2271 - val_accuracy: 0.9098
Epoch 15/80
500/500 [==============================] - 11s 21ms/step - loss: 0.3219 - acc
uracy: 0.8669 - val_loss: 0.4112 - val_accuracy: 0.8246
Epoch 16/80
500/500 [==============================] - 10s 21ms/step - loss: 0.3310 - acc
uracy: 0.8598 - val_loss: 0.1919 - val_accuracy: 0.9228
Epoch 17/80
500/500 [==============================] - 11s 22ms/step - loss: 0.3342 - acc
uracy: 0.8607 - val_loss: 0.2056 - val_accuracy: 0.9188
Epoch 18/80
500/500 [==============================] - 10s 21ms/step - loss: 0.3450 - acc
uracy: 0.8559 - val_loss: 0.2414 - val_accuracy: 0.8908
Epoch 19/80
500/500 [==============================] - 10s 21ms/step - loss: 0.3657 - acc
uracy: 0.8515 - val_loss: 0.3468 - val_accuracy: 0.8307
Epoch 20/80
500/500 [==============================] - 10s 21ms/step - loss: 0.3517 - acc
uracy: 0.8532 - val_loss: 0.2270 - val_accuracy: 0.9208
Epoch 21/80
500/500 [==============================] - 11s 21ms/step - loss: 0.3681 - acc
uracy: 0.8481 - val_loss: 0.3836 - val_accuracy: 0.8888
Epoch 22/80
500/500 [==============================] - 10s 21ms/step - loss: 0.3722 - acc
uracy: 0.8447 - val_loss: 0.6300 - val_accuracy: 0.7285
Epoch 23/80
500/500 [==============================] - 10s 20ms/step - loss: 0.4038 - acc
uracy: 0.8415 - val_loss: 0.5272 - val_accuracy: 0.8717
Epoch 24/80
500/500 [==============================] - 10s 21ms/step - loss: 0.4020 - acc
uracy: 0.8377 - val_loss: 0.2686 - val_accuracy: 0.8988
```

```
Epoch 25/80
500/500 [==============================] - 10s 21ms/step - loss: 0.4430 - acc
uracy: 0.8336 - val_loss: 1.5721 - val_accuracy: 0.7806
Epoch 26/80
500/500 [==============================] - 10s 21ms/step - loss: 0.4454 - acc
uracy: 0.8182 - val_loss: 0.3221 - val_accuracy: 0.8707
Epoch 27/80
500/500 [==============================] - 11s 21ms/step - loss: 0.4203 - acc
uracy: 0.8275 - val_loss: 0.2136 - val_accuracy: 0.9118
Epoch 28/80
500/500 [==============================] - 11s 21ms/step - loss: 0.4422 - acc
uracy: 0.8264 - val_loss: 0.2804 - val_accuracy: 0.8788
Epoch 29/80
500/500 [==============================] - 10s 21ms/step - loss: 0.4528 - acc
uracy: 0.8213 - val_loss: 0.3431 - val_accuracy: 0.8677
Epoch 30/80
500/500 [==============================] - 11s 21ms/step - loss: 0.4511 - acc
uracy: 0.8179 - val_loss: 0.3673 - val_accuracy: 0.8878
Epoch 31/80
500/500 [==============================] - 10s 20ms/step - loss: 0.4706 - acc
uracy: 0.8105 - val_loss: 0.4326 - val_accuracy: 0.8427
Epoch 32/80
500/500 [==============================] - 10s 20ms/step - loss: 0.4646 - acc
uracy: 0.8095 - val_loss: 0.3251 - val_accuracy: 0.8707
Epoch 33/80
500/500 [==============================] - 10s 21ms/step - loss: 0.5123 - acc
uracy: 0.8054 - val_loss: 1.0964 - val_accuracy: 0.6393
Epoch 34/80
500/500 [==============================] - 10s 20ms/step - loss: 0.5418 - acc
uracy: 0.7726 - val_loss: 0.3325 - val_accuracy: 0.8497
Epoch 35/80
500/500 [==============================] - 10s 20ms/step - loss: 0.4987 - acc
uracy: 0.8029 - val_loss: 0.3348 - val_accuracy: 0.8617
Epoch 36/80
500/500 [==============================] - 11s 21ms/step - loss: 0.5345 - acc
uracy: 0.8021 - val_loss: 0.4660 - val_accuracy: 0.8136
Epoch 37/80
500/500 [==============================] - 10s 21ms/step - loss: 0.5108 - acc
uracy: 0.7906 - val_loss: 0.4385 - val_accuracy: 0.8126
Epoch 38/80
500/500 [==============================] - 10s 21ms/step - loss: 0.5487 - acc
uracy: 0.7854 - val_loss: 0.3410 - val_accuracy: 0.8507
Epoch 39/80
500/500 [==============================] - 10s 21ms/step - loss: 0.5248 - acc
uracy: 0.7893 - val_loss: 0.3030 - val_accuracy: 0.8697
Epoch 40/80
500/500 [==============================] - 10s 21ms/step - loss: 0.5730 - acc
uracy: 0.7750 - val_loss: 0.3053 - val_accuracy: 0.8687
Epoch 41/80
500/500 [==============================] - 10s 21ms/step - loss: 0.5474 - acc
uracy: 0.7735 - val_loss: 0.4701 - val_accuracy: 0.7595
Epoch 42/80
```

```
500/500 [==============================] - 10s 20ms/step - loss: 0.5688 - acc
uracy: 0.7653 - val_loss: 0.5299 - val_accuracy: 0.8327
Epoch 43/80
500/500 [==============================] - 10s 20ms/step - loss: 0.5633 - acc
uracy: 0.7732 - val_loss: 0.5697 - val_accuracy: 0.7154
Epoch 44/80
500/500 [==============================] - 10s 20ms/step - loss: 0.6075 - acc
uracy: 0.7611 - val_loss: 0.3907 - val_accuracy: 0.8267
Epoch 45/80
500/500 [==============================] - 10s 20ms/step - loss: 0.6059 - acc
uracy: 0.7528 - val_loss: 0.4196 - val_accuracy: 0.8317
Epoch 46/80
500/500 [==============================] - 10s 20ms/step - loss: 0.7011 - acc
uracy: 0.7299 - val_loss: 0.4721 - val_accuracy: 0.8006
Epoch 47/80
500/500 [==============================] - 11s 21ms/step - loss: 0.7024 - acc
uracy: 0.7442 - val_loss: 0.4047 - val_accuracy: 0.8086
Epoch 48/80
500/500 [==============================] - 10s 21ms/step - loss: 0.7302 - acc
uracy: 0.7452 - val_loss: 0.5936 - val_accuracy: 0.7004
Epoch 49/80
500/500 [==============================] - 10s 20ms/step - loss: 0.7334 - acc
uracy: 0.7185 - val_loss: 0.3979 - val_accuracy: 0.8347
Epoch 50/80
500/500 [==============================] - 10s 21ms/step - loss: 0.6484 - acc
uracy: 0.7150 - val_loss: 0.4708 - val_accuracy: 0.8046
Epoch 51/80
500/500 [==============================] - 10s 20ms/step - loss: 0.6544 - acc
uracy: 0.7255 - val_loss: 0.4457 - val_accuracy: 0.8036
Epoch 52/80
500/500 [==============================] - 10s 20ms/step - loss: 0.6426 - acc
uracy: 0.7240 - val_loss: 0.4706 - val_accuracy: 0.8196
Epoch 53/80
500/500 [==============================] - 10s 21ms/step - loss: 0.6671 - acc
uracy: 0.7307 - val_loss: 0.5251 - val_accuracy: 0.7926
Epoch 54/80
500/500 [==============================] - 10s 21ms/step - loss: 0.6281 - acc
uracy: 0.7216 - val_loss: 0.4577 - val_accuracy: 0.7966
Epoch 55/80
500/500 [==============================] - 10s 21ms/step - loss: 0.7185 - acc
uracy: 0.6918 - val_loss: 0.5353 - val_accuracy: 0.7405
Epoch 56/80
500/500 [==============================] - 10s 21ms/step - loss: 0.7301 - acc
uracy: 0.7176 - val_loss: 0.4469 - val_accuracy: 0.7866
Epoch 57/80
500/500 [==============================] - 10s 20ms/step - loss: 0.6787 - acc
uracy: 0.6987 - val_loss: 0.7591 - val_accuracy: 0.8377
Epoch 58/80
500/500 [==============================] - 10s 21ms/step - loss: 0.8962 - acc
uracy: 0.6985 - val_loss: 0.4884 - val_accuracy: 0.7936
Epoch 59/80
500/500 [==============================] - 10s 20ms/step - loss: 0.8222 - acc
uracy: 0.6948 - val_loss: 0.5391 - val_accuracy: 0.8026
```

```
Epoch 60/80
500/500 [==============================] - 10s 20ms/step - loss: 0.7255 - acc
uracy: 0.6885 - val_loss: 0.5430 - val_accuracy: 0.7315
Epoch 61/80
500/500 [==============================] - 10s 21ms/step - loss: 0.7795 - acc
uracy: 0.6950 - val_loss: 0.4455 - val_accuracy: 0.8056
Epoch 62/80
500/500 [==============================] - 11s 21ms/step - loss: 0.7661 - acc
uracy: 0.6893 - val_loss: 0.4932 - val_accuracy: 0.7996
Epoch 63/80
500/500 [==============================] - 10s 21ms/step - loss: 0.6789 - acc
uracy: 0.6858 - val_loss: 0.7269 - val_accuracy: 0.7325
Epoch 64/80
500/500 [==============================] - 10s 21ms/step - loss: 0.6541 - acc
uracy: 0.7032 - val_loss: 0.4906 - val_accuracy: 0.7575
Epoch 65/80
500/500 [==============================] - 10s 20ms/step - loss: 0.7988 - acc
uracy: 0.7289 - val_loss: 0.4875 - val_accuracy: 0.7766
Epoch 66/80
500/500 [==============================] - 10s 20ms/step - loss: 0.7619 - acc
uracy: 0.7186 - val_loss: 0.5024 - val_accuracy: 0.7856
Epoch 67/80
500/500 [==============================] - 10s 21ms/step - loss: 0.6605 - acc
uracy: 0.7142 - val_loss: 0.5712 - val_accuracy: 0.8126
Epoch 68/80
500/500 [==============================] - 10s 21ms/step - loss: 0.7300 - acc
uracy: 0.6925 - val_loss: 0.8057 - val_accuracy: 0.8146
Epoch 69/80
500/500 [==============================] - 10s 20ms/step - loss: 0.7359 - acc
uracy: 0.7048 - val_loss: 0.4732 - val_accuracy: 0.7826
Epoch 70/80
500/500 [==============================] - 11s 21ms/step - loss: 0.6933 - acc
uracy: 0.7053 - val_loss: 0.5170 - val_accuracy: 0.7485
Epoch 71/80
500/500 [==============================] - 10s 20ms/step - loss: 0.8406 - acc
uracy: 0.7012 - val_loss: 0.9676 - val_accuracy: 0.6253
Epoch 72/80
500/500 [==============================] - 10s 20ms/step - loss: 0.6573 - acc
uracy: 0.7063 - val_loss: 1.1291 - val_accuracy: 0.6914
Epoch 73/80
500/500 [==============================] - 10s 20ms/step - loss: 0.8186 - acc
uracy: 0.6640 - val_loss: 0.6766 - val_accuracy: 0.5561
Epoch 74/80
500/500 [==============================] - 11s 21ms/step - loss: 0.9503 - acc
uracy: 0.6762 - val_loss: 0.5581 - val_accuracy: 0.7375
Epoch 75/80
500/500 [==============================] - 10s 21ms/step - loss: 0.7765 - acc
uracy: 0.6905 - val_loss: 0.7470 - val_accuracy: 0.6443
Epoch 76/80
500/500 [==============================] - 10s 21ms/step - loss: 0.7347 - acc
uracy: 0.6849 - val_loss: 0.4996 - val_accuracy: 0.8126
Epoch 77/80
```

```
500/500 [==============================] - 10s 20ms/step - loss: 0.7122 - acc
uracy: 0.6692 - val_loss: 0.9294 - val_accuracy: 0.7615
Epoch 78/80
500/500 [==============================] - 11s 22ms/step - loss: 0.7860 - acc
uracy: 0.6724 - val_loss: 0.5466 - val_accuracy: 0.7365
Epoch 79/80
500/500 [==============================] - 10s 21ms/step - loss: 0.9206 - acc
uracy: 0.6649 - val_loss: 0.5738 - val_accuracy: 0.6834
Epoch 80/80
500/500 [==============================] - 10s 20ms/step - loss: 0.8316 - acc
uracy: 0.6814 - val_loss: 0.5858 - val_accuracy: 0.7555
```

Evaluating the model on the test set

In [20]:

```
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 [==============================] - 1s 15ms/step - loss: 0.2239 - accura
cy: 0.9070
Test accuracy: 0.907
```

Using a pretrained network with training sample of 1000, a validation sample of 500, and a test sample of 500

Downloading the data

In [1]:

```
#!unzip -qq '/fs/ess/PGS0333/BA_64061_KSU/data/dogs-vs-cats.zip'
```

In [2]:

```
#!unzip -qq train.zip
```

Copying images to training, validation, and test directories

In [3]:

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index,
end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2000)
```

Building the model

Instantiating a small convnet for dogs vs. cats classification

In [4]:

```
"""
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)
"""
```

Out[4]:

```
'\nfrom tensorflow import keras\nfrom tensorflow.keras import layers\ninputs
= keras.Input(shape=(180, 180, 3))\nx = layers.Rescaling(1./255)(inputs)\nx =
 layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)\nx = layers.M
axPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=64, kernel_size=3, act
ivation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2
D(filters=128, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D
(pool_size=2)(x)\nx = layers.Conv2D(filters=256, kernel_size=3, activation="r
elu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=
256, kernel_size=3, activation="relu")(x)\nx = layers.Flatten()(x)\noutputs =
 layers.Dense(1, activation="sigmoid")(x)\nmodel = keras.Model(inputs=inputs,
 outputs=outputs)\n'
```

In [5]:

```
# model.summary()
```

Configuring the model for training

In [6]:

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

Out[6]:

```
'\nmodel.compile(loss="binary_crossentropy",\n              optimizer="rmspro
p",\n              metrics=["accuracy"])\n'
```

Data preprocessing

Using image_dataset_from_directory to read images

In [7]:

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

In [8]:

```
"""
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)
"""
```

Out[8]:

```
'\nimport numpy as np\nimport tensorflow as tf\nrandom_numbers = np.random.no
rmal(size=(1000, 16))\ndataset = tf.data.Dataset.from_tensor_slices(random_nu
mbers)\n'
```

In [9]:

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

Out[9]:

```
'\nfor i, element in enumerate(dataset):\n    print(element.shape)\n    if i >
= 2:\n        break\n'
```

In [10]:

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

Out[10]:

```
'\nbatched_dataset = dataset.batch(32)\nfor i, element in enumerate(batched_d
ataset):\n    print(element.shape)\n    if i >= 2:\n        break\n'
```

In [11]:

```
"""
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
"""
```

Out[11]:

```
'\nreshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))\nfor i, el
ement in enumerate(reshaped_dataset):\n    print(element.shape)\n    if i >=
2:\n        break\n'
```

Displaying the shapes of the data and labels yielded by the Dataset

In [12]:

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

Out[12]:

```
'\nfor data_batch, labels_batch in train_dataset:\n    print("data batch shap
e:", data_batch.shape)\n    print("labels batch shape:", labels_batch.shape)\
n    break\n'
```

Fitting the model using a Dataset

In [13]:

```
"""
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)
"""
```

Out[13]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch.keras",\n        save_best_only=True,\n        monitor="val
_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    epochs=30,\n    val
idation_data=validation_dataset,\n    callbacks=callbacks)\n'
```

Displaying curves of loss and accuracy during training

In [14]:

```
"""
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()
"""
```

Out[14]:

```
'\nimport matplotlib.pyplot as plt\naccuracy = history.history["accuracy"]\nv
al_accuracy = history.history["val_accuracy"]\nloss = history.history["loss"]
\nval_loss = history.history["val_loss"]\nepochs = range(1, len(accuracy) + 1)
\nplt.plot(epochs, accuracy, "bo", label="Training accuracy")\nplt.plot(epoch
s, val_accuracy, "b", label="Validation accuracy")\nplt.title("Training and v
alidation accuracy")\nplt.legend()\nplt.figure()\nplt.plot(epochs, loss, "bo",
 label="Training loss")\nplt.plot(epochs, val_loss, "b", label="Validation lo
ss")\nplt.title("Training and validation loss")\nplt.legend()\nplt.show()\n'
```

Evaluating the model on the test set

In [15]:

```
"""
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}")
"""
```

Out[15]:

```
'\ntest_model = keras.models.load_model("convnet_from_scratch.keras")\ntest_l
oss, test_acc = test_model.evaluate(test_dataset)\nprint(f"Test accuracy: {te
st_acc:.3f}")\n'
```

Using data augmentation

Define a data augmentation stage to add to an image model

In [16]:

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

Out[16]:

```
'\ndata_augmentation = keras.Sequential(\n    [\n        layers.RandomFlip("h
orizontal"),\n        layers.RandomRotation(0.1),\n        layers.RandomZoom
(0.2),\n    ]\n)\n'
```

Displaying some randomly augmented training images

In [17]:

```
"""
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")
"""
```

Out[17]:

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

Defining a new convnet that includes image augmentation and dropout

In [18]:

```
"""
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"])
"""
```

Out[18]:

```
'\ninputs = keras.Input(shape=(180, 180, 3))\nx = data_augmentation(inputs)\n
x = layers.Rescaling(1./255)(x)\nx = layers.Conv2D(filters=32, kernel_size=3,
 activation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.C
onv2D(filters=64, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling
2D(pool_size=2)(x)\nx = layers.Conv2D(filters=128, kernel_size=3, activation=
"relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filter
s=256, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D(pool_siz
e=2)(x)\nx = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)\
nx = layers.Flatten()(x)\nx = layers.Dropout(0.5)(x)\noutputs = layers.Dense
(1, activation="sigmoid")(x)\nmodel = keras.Model(inputs=inputs, outputs=outp
uts)\n\nmodel.compile(loss="binary_crossentropy",\n              optimizer="r
msprop",\n              metrics=["accuracy"])\n'
```

Training the regularized convnet

In [19]:

```
"""
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)
"""
```

Out[19]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch_with_augmentation.keras",\n        save_best_only=True,\n
      monitor="val_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    e
pochs=100,\n    validation_data=validation_dataset,\n    callbacks=callbacks)
\n'
```

Evaluating the model on the test set

In [20]:

```
"""
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}")
"""
```

Out[20]:

```
'\ntest_model = keras.models.load_model(\n    "convnet_from_scratch_with_augm
entation.keras")\ntest_loss, test_acc = test_model.evaluate(test_dataset)\npr
int(f"Test accuracy: {test_acc:.3f}")\n'
```

Leveraging a pretrained model

Feature extraction with a pretrained model

Instantiating the VGG16 convolutional base

In [21]:

```
from tensorflow import keras # import keras
from tensorflow.keras import layers
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

In [22]:

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (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
_________________________________________________________________
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
_________________________________________________________________
```

Fast feature extraction without data augmentation

Extracting the VGG16 features and corresponding labels

In [23]:

```
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)
```

In [24]:

```
train_features.shape
```

Out[24]:

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

Defining and training the densely connected classifier

In [25]:

```
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 [==============================] - 1s 6ms/step - loss: 13.5073 - accura
cy: 0.9210 - val_loss: 4.7467 - val_accuracy: 0.9660
Epoch 2/20
63/63 [==============================] - 0s 4ms/step - loss: 3.0386 - accurac
y: 0.9770 - val_loss: 12.0533 - val_accuracy: 0.9430
Epoch 3/20
63/63 [==============================] - 0s 3ms/step - loss: 2.1036 - accurac
y: 0.9850 - val_loss: 3.2843 - val_accuracy: 0.9750
Epoch 4/20
63/63 [==============================] - 0s 4ms/step - loss: 1.9179 - accurac
y: 0.9875 - val_loss: 7.3282 - val_accuracy: 0.9700
Epoch 5/20
```

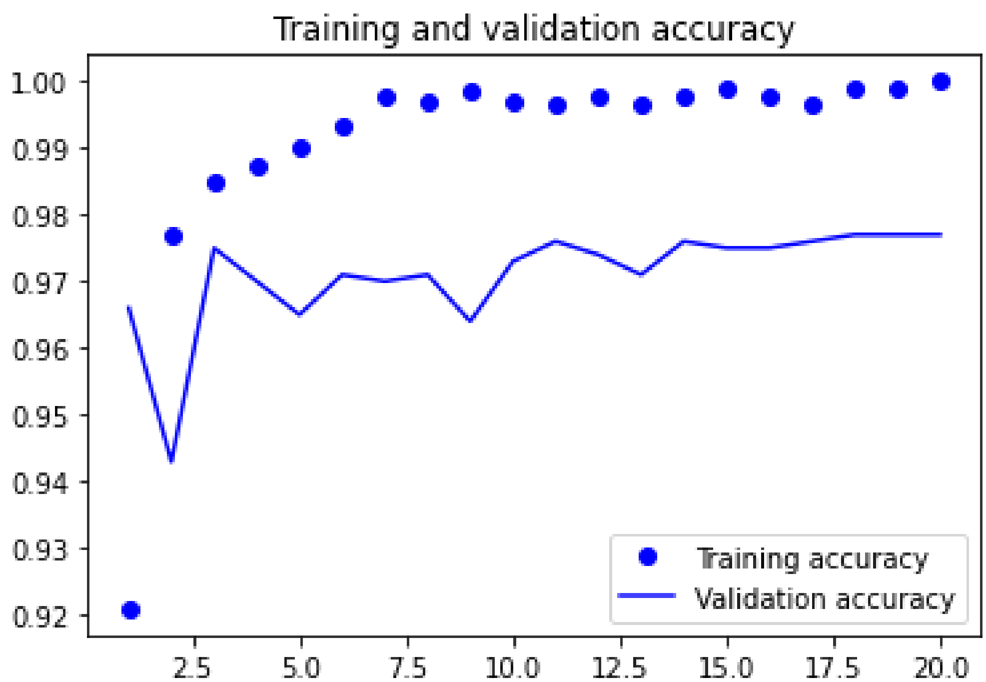
```
63/63 [==============================] - 0s 3ms/step - loss: 0.9677 - accurac
y: 0.9900 - val_loss: 6.0118 - val_accuracy: 0.9650
Epoch 6/20
63/63 [==============================] - 0s 3ms/step - loss: 1.2342 - accurac
y: 0.9935 - val_loss: 6.0537 - val_accuracy: 0.9710
Epoch 7/20
63/63 [==============================] - 0s 3ms/step - loss: 0.2122 - accurac
y: 0.9975 - val_loss: 6.1084 - val_accuracy: 0.9700
Epoch 8/20
63/63 [==============================] - 0s 3ms/step - loss: 0.4881 - accurac
y: 0.9970 - val_loss: 5.7784 - val_accuracy: 0.9710
Epoch 9/20
63/63 [==============================] - 0s 3ms/step - loss: 0.0937 - accurac
y: 0.9985 - val_loss: 8.9953 - val_accuracy: 0.9640
Epoch 10/20
63/63 [==============================] - 0s 3ms/step - loss: 0.4623 - accurac
y: 0.9970 - val_loss: 5.5047 - val_accuracy: 0.9730
Epoch 11/20
63/63 [==============================] - 0s 3ms/step - loss: 0.2748 - accurac
y: 0.9965 - val_loss: 5.4949 - val_accuracy: 0.9760
Epoch 12/20
63/63 [==============================] - 0s 3ms/step - loss: 0.4063 - accurac
y: 0.9975 - val_loss: 5.0313 - val_accuracy: 0.9740
Epoch 13/20
63/63 [==============================] - 0s 3ms/step - loss: 0.2805 - accurac
y: 0.9965 - val_loss: 6.3703 - val_accuracy: 0.9710
Epoch 14/20
63/63 [==============================] - 0s 3ms/step - loss: 0.5061 - accurac
y: 0.9975 - val_loss: 5.6572 - val_accuracy: 0.9760
Epoch 15/20
63/63 [==============================] - 0s 3ms/step - loss: 0.0828 - accurac
y: 0.9990 - val_loss: 5.7342 - val_accuracy: 0.9750
Epoch 16/20
63/63 [==============================] - 0s 3ms/step - loss: 0.2083 - accurac
y: 0.9975 - val_loss: 5.3191 - val_accuracy: 0.9750
Epoch 17/20
63/63 [==============================] - 0s 3ms/step - loss: 0.2813 - accurac
y: 0.9965 - val_loss: 5.0407 - val_accuracy: 0.9760
Epoch 18/20
63/63 [==============================] - 0s 3ms/step - loss: 0.0779 - accurac
y: 0.9990 - val_loss: 5.0738 - val_accuracy: 0.9770
Epoch 19/20
63/63 [==============================] - 0s 3ms/step - loss: 0.0072 - accurac
y: 0.9990 - val_loss: 5.6584 - val_accuracy: 0.9770
Epoch 20/20
63/63 [==============================] - 0s 3ms/step - loss: 3.5128e-17 - acc
uracy: 1.0000 - val_loss: 5.6584 - val_accuracy: 0.9770
```

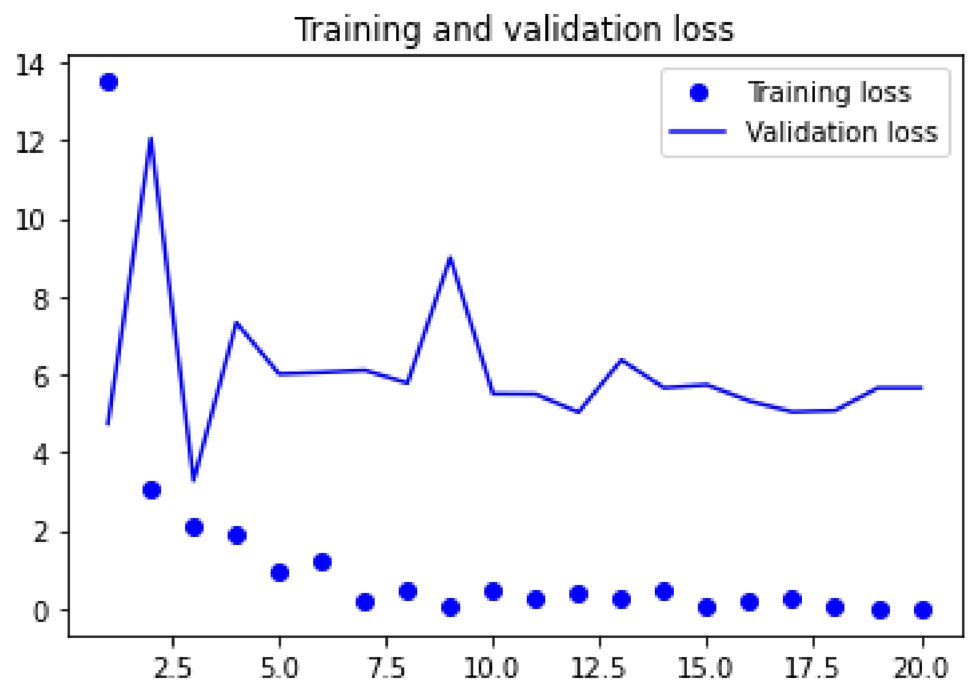
Plotting the results

In [26]:

```
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

In [27]:

```
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

In [28]:

```
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
```

In [29]:

```
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

In [30]:

```
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"])
```

In [31]:

```
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 [==============================] - 5s 56ms/step - loss: 21.2061 - accur
acy: 0.8910 - val_loss: 22.9828 - val_accuracy: 0.8940
Epoch 2/50
63/63 [==============================] - 3s 52ms/step - loss: 7.0033 - accura
cy: 0.9445 - val_loss: 5.7581 - val_accuracy: 0.9620
Epoch 3/50
63/63 [==============================] - 3s 53ms/step - loss: 6.5587 - accura
cy: 0.9570 - val_loss: 5.6807 - val_accuracy: 0.9590
Epoch 4/50
63/63 [==============================] - 4s 56ms/step - loss: 4.4935 - accura
cy: 0.9605 - val_loss: 6.0700 - val_accuracy: 0.9640
Epoch 5/50
63/63 [==============================] - 4s 56ms/step - loss: 3.2817 - accura
cy: 0.9675 - val_loss: 6.1735 - val_accuracy: 0.9650
Epoch 6/50
63/63 [==============================] - 3s 52ms/step - loss: 5.4043 - accura
cy: 0.9590 - val_loss: 7.8655 - val_accuracy: 0.9640
Epoch 7/50
63/63 [==============================] - 3s 53ms/step - loss: 4.8845 - accura
cy: 0.9620 - val_loss: 2.8868 - val_accuracy: 0.9760
Epoch 8/50
63/63 [==============================] - 3s 53ms/step - loss: 2.2545 - accura
cy: 0.9765 - val_loss: 5.1264 - val_accuracy: 0.9700
Epoch 9/50
```

```
63/63 [==============================] - 3s 52ms/step - loss: 2.3987 - accura
cy: 0.9770 - val_loss: 4.8956 - val_accuracy: 0.9740
Epoch 10/50
63/63 [==============================] - 4s 57ms/step - loss: 2.7642 - accura
cy: 0.9730 - val_loss: 4.4262 - val_accuracy: 0.9770
Epoch 11/50
63/63 [==============================] - 3s 52ms/step - loss: 2.7902 - accura
cy: 0.9755 - val_loss: 3.1060 - val_accuracy: 0.9770
Epoch 12/50
63/63 [==============================] - 3s 53ms/step - loss: 2.5042 - accura
cy: 0.9755 - val_loss: 4.9537 - val_accuracy: 0.9680
Epoch 13/50
63/63 [==============================] - 3s 52ms/step - loss: 1.4807 - accura
cy: 0.9795 - val_loss: 3.4375 - val_accuracy: 0.9790
Epoch 14/50
63/63 [==============================] - 3s 52ms/step - loss: 1.5025 - accura
cy: 0.9840 - val_loss: 3.9441 - val_accuracy: 0.9740
Epoch 15/50
63/63 [==============================] - 3s 52ms/step - loss: 1.9034 - accura
cy: 0.9805 - val_loss: 3.6990 - val_accuracy: 0.9730
Epoch 16/50
63/63 [==============================] - 3s 52ms/step - loss: 2.0529 - accura
cy: 0.9770 - val_loss: 4.0766 - val_accuracy: 0.9780
Epoch 17/50
63/63 [==============================] - 3s 52ms/step - loss: 1.4646 - accura
cy: 0.9790 - val_loss: 3.3099 - val_accuracy: 0.9770
Epoch 18/50
63/63 [==============================] - 3s 52ms/step - loss: 1.4809 - accura
cy: 0.9820 - val_loss: 3.2118 - val_accuracy: 0.9720
Epoch 19/50
63/63 [==============================] - 3s 52ms/step - loss: 1.1507 - accura
cy: 0.9815 - val_loss: 2.6941 - val_accuracy: 0.9750
Epoch 20/50
63/63 [==============================] - 3s 53ms/step - loss: 1.6299 - accura
cy: 0.9805 - val_loss: 2.7439 - val_accuracy: 0.9760
Epoch 21/50
63/63 [==============================] - 3s 53ms/step - loss: 1.1083 - accura
cy: 0.9845 - val_loss: 3.9618 - val_accuracy: 0.9790
Epoch 22/50
63/63 [==============================] - 3s 52ms/step - loss: 1.3424 - accura
cy: 0.9835 - val_loss: 3.0976 - val_accuracy: 0.9810
Epoch 23/50
63/63 [==============================] - 3s 52ms/step - loss: 1.3167 - accura
cy: 0.9795 - val_loss: 3.8232 - val_accuracy: 0.9720
Epoch 24/50
63/63 [==============================] - 3s 53ms/step - loss: 1.3782 - accura
cy: 0.9810 - val_loss: 2.3537 - val_accuracy: 0.9780
Epoch 25/50
63/63 [==============================] - 3s 53ms/step - loss: 0.9867 - accura
cy: 0.9845 - val_loss: 2.9392 - val_accuracy: 0.9770
Epoch 26/50
63/63 [==============================] - 3s 53ms/step - loss: 0.9766 - accura
cy: 0.9825 - val_loss: 2.4336 - val_accuracy: 0.9780
```

```
Epoch 27/50
63/63 [==============================] - 3s 52ms/step - loss: 1.0308 - accura
cy: 0.9830 - val_loss: 3.6002 - val_accuracy: 0.9720
Epoch 28/50
63/63 [==============================] - 3s 52ms/step - loss: 0.6114 - accura
cy: 0.9850 - val_loss: 2.4190 - val_accuracy: 0.9810
Epoch 29/50
63/63 [==============================] - 3s 52ms/step - loss: 0.7387 - accura
cy: 0.9855 - val_loss: 4.1295 - val_accuracy: 0.9660
Epoch 30/50
63/63 [==============================] - 3s 52ms/step - loss: 0.5670 - accura
cy: 0.9845 - val_loss: 3.4808 - val_accuracy: 0.9740
Epoch 31/50
63/63 [==============================] - 3s 53ms/step - loss: 0.7438 - accura
cy: 0.9855 - val_loss: 2.7265 - val_accuracy: 0.9750
Epoch 32/50
63/63 [==============================] - 3s 52ms/step - loss: 0.6358 - accura
cy: 0.9880 - val_loss: 4.9777 - val_accuracy: 0.9630
Epoch 33/50
63/63 [==============================] - 3s 52ms/step - loss: 1.0329 - accura
cy: 0.9845 - val_loss: 2.0601 - val_accuracy: 0.9770
Epoch 34/50
63/63 [==============================] - 3s 52ms/step - loss: 0.5896 - accura
cy: 0.9895 - val_loss: 2.1925 - val_accuracy: 0.9800
Epoch 35/50
63/63 [==============================] - 3s 52ms/step - loss: 0.9808 - accura
cy: 0.9855 - val_loss: 2.6108 - val_accuracy: 0.9790
Epoch 36/50
63/63 [==============================] - 4s 53ms/step - loss: 0.8450 - accura
cy: 0.9850 - val_loss: 2.7040 - val_accuracy: 0.9710
Epoch 37/50
63/63 [==============================] - 3s 53ms/step - loss: 0.5562 - accura
cy: 0.9875 - val_loss: 2.3025 - val_accuracy: 0.9730
Epoch 38/50
63/63 [==============================] - 3s 51ms/step - loss: 1.2197 - accura
cy: 0.9820 - val_loss: 2.5752 - val_accuracy: 0.9760
Epoch 39/50
63/63 [==============================] - 3s 52ms/step - loss: 0.6530 - accura
cy: 0.9890 - val_loss: 3.0409 - val_accuracy: 0.9690
Epoch 40/50
63/63 [==============================] - 3s 52ms/step - loss: 0.3301 - accura
cy: 0.9940 - val_loss: 2.8597 - val_accuracy: 0.9800
Epoch 41/50
63/63 [==============================] - 3s 52ms/step - loss: 0.5592 - accura
cy: 0.9895 - val_loss: 2.3041 - val_accuracy: 0.9750
Epoch 42/50
63/63 [==============================] - 3s 52ms/step - loss: 0.5874 - accura
cy: 0.9920 - val_loss: 2.4542 - val_accuracy: 0.9760
Epoch 43/50
63/63 [==============================] - 3s 52ms/step - loss: 0.3692 - accura
cy: 0.9905 - val_loss: 2.3337 - val_accuracy: 0.9780
Epoch 44/50
```

```
63/63 [==============================] - 3s 53ms/step - loss: 0.9109 - accura
cy: 0.9860 - val_loss: 2.7692 - val_accuracy: 0.9780
Epoch 45/50
63/63 [==============================] - 3s 52ms/step - loss: 0.8140 - accura
cy: 0.9860 - val_loss: 2.4054 - val_accuracy: 0.9750
Epoch 46/50
63/63 [==============================] - 3s 53ms/step - loss: 0.7053 - accura
cy: 0.9885 - val_loss: 2.5727 - val_accuracy: 0.9790
Epoch 47/50
63/63 [==============================] - 3s 53ms/step - loss: 0.6289 - accura
cy: 0.9865 - val_loss: 2.2830 - val_accuracy: 0.9760
Epoch 48/50
63/63 [==============================] - 3s 52ms/step - loss: 0.5192 - accura
cy: 0.9885 - val_loss: 2.3128 - val_accuracy: 0.9780
Epoch 49/50
63/63 [==============================] - 3s 52ms/step - loss: 0.5046 - accura
cy: 0.9905 - val_loss: 2.6822 - val_accuracy: 0.9770
Epoch 50/50
63/63 [==============================] - 3s 53ms/step - loss: 0.8204 - accura
cy: 0.9860 - val_loss: 2.0805 - val_accuracy: 0.9790
```

Evaluating the model on the test set

In [32]:

```
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 [==============================] - 1s 32ms/step - loss: 2.4537 - accura
cy: 0.9770
Test accuracy: 0.977
```

Fine-tuning a pretrained model

In [33]:

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None, None, 3)]   0

_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792

_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928

_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0

_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856

_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584

_________________________________________________________________
```

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

Freezing all layers until the fourth from the last

In [34]:

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

Fine-tuning the model

In [35]:

```
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 [==============================] - 5s 61ms/step - loss: 0.3013 - accura
cy: 0.9920 - val_loss: 2.3283 - val_accuracy: 0.9720
Epoch 2/30
63/63 [==============================] - 4s 56ms/step - loss: 0.4117 - accura
cy: 0.9850 - val_loss: 2.0034 - val_accuracy: 0.9790
Epoch 3/30
63/63 [==============================] - 4s 56ms/step - loss: 0.4557 - accura
cy: 0.9890 - val_loss: 2.1317 - val_accuracy: 0.9780
Epoch 4/30
63/63 [==============================] - 4s 56ms/step - loss: 0.3864 - accura
cy: 0.9875 - val_loss: 2.2103 - val_accuracy: 0.9800
Epoch 5/30
63/63 [==============================] - 4s 57ms/step - loss: 0.2877 - accura
cy: 0.9920 - val_loss: 2.3774 - val_accuracy: 0.9760
Epoch 6/30
63/63 [==============================] - 4s 57ms/step - loss: 0.2910 - accura
cy: 0.9890 - val_loss: 2.4382 - val_accuracy: 0.9740
Epoch 7/30
63/63 [==============================] - 4s 56ms/step - loss: 0.1776 - accura
cy: 0.9930 - val_loss: 2.0192 - val_accuracy: 0.9780
Epoch 8/30
63/63 [==============================] - 4s 56ms/step - loss: 0.3179 - accura
cy: 0.9915 - val_loss: 2.1185 - val_accuracy: 0.9750
Epoch 9/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2422 - accura
cy: 0.9930 - val_loss: 1.8964 - val_accuracy: 0.9770
Epoch 10/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2623 - accura
cy: 0.9940 - val_loss: 1.5055 - val_accuracy: 0.9820
Epoch 11/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2996 - accura
cy: 0.9920 - val_loss: 1.2986 - val_accuracy: 0.9820
Epoch 12/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0817 - accura
cy: 0.9975 - val_loss: 1.7039 - val_accuracy: 0.9800
Epoch 13/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2091 - accura
cy: 0.9940 - val_loss: 1.7767 - val_accuracy: 0.9820
Epoch 14/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2408 - accura
cy: 0.9940 - val_loss: 1.6753 - val_accuracy: 0.9810
Epoch 15/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2389 - accura
cy: 0.9950 - val_loss: 1.4084 - val_accuracy: 0.9800
Epoch 16/30
63/63 [==============================] - 4s 56ms/step - loss: 0.1064 - accura
cy: 0.9955 - val_loss: 1.6390 - val_accuracy: 0.9790
Epoch 17/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0449 - accura
cy: 0.9990 - val_loss: 1.6157 - val_accuracy: 0.9790
```

```
Epoch 18/30
63/63 [==============================] - 4s 56ms/step - loss: 0.1464 - accura
cy: 0.9930 - val_loss: 1.7780 - val_accuracy: 0.9770
Epoch 19/30
63/63 [==============================] - 4s 56ms/step - loss: 0.2795 - accura
cy: 0.9955 - val_loss: 1.5480 - val_accuracy: 0.9780
Epoch 20/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0916 - accura
cy: 0.9970 - val_loss: 1.7212 - val_accuracy: 0.9760
Epoch 21/30
63/63 [==============================] - 4s 57ms/step - loss: 0.0743 - accura
cy: 0.9965 - val_loss: 1.1029 - val_accuracy: 0.9800
Epoch 22/30
63/63 [==============================] - 4s 56ms/step - loss: 0.1037 - accura
cy: 0.9960 - val_loss: 1.8676 - val_accuracy: 0.9760
Epoch 23/30
63/63 [==============================] - 4s 57ms/step - loss: 0.1488 - accura
cy: 0.9940 - val_loss: 1.5035 - val_accuracy: 0.9810
Epoch 24/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0420 - accura
cy: 0.9970 - val_loss: 1.4502 - val_accuracy: 0.9810
Epoch 25/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0781 - accura
cy: 0.9965 - val_loss: 1.5331 - val_accuracy: 0.9810
Epoch 26/30
63/63 [==============================] - 4s 57ms/step - loss: 0.1049 - accura
cy: 0.9955 - val_loss: 1.3614 - val_accuracy: 0.9800
Epoch 27/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0205 - accura
cy: 0.9985 - val_loss: 1.2473 - val_accuracy: 0.9820
Epoch 28/30
63/63 [==============================] - 4s 56ms/step - loss: 0.1799 - accura
cy: 0.9940 - val_loss: 1.7278 - val_accuracy: 0.9790
Epoch 29/30
63/63 [==============================] - 4s 56ms/step - loss: 0.0545 - accura
cy: 0.9980 - val_loss: 1.5502 - val_accuracy: 0.9820
Epoch 30/30
63/63 [==============================] - 4s 57ms/step - loss: 0.0587 - accura
cy: 0.9950 - val_loss: 2.0134 - val_accuracy: 0.9760
```

In [36]:

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

```
32/32 [==============================] - 1s 33ms/step - loss: 1.5191 - accura
cy: 0.9810
Test accuracy: 0.981
```

Summary

Using a pretrained network with training sample of 3000, a validation sample of 500, and a test sample of 500

Downloading the data

In [1]:

```
#!unzip -qq '/fs/ess/PGS0333/BA_64061_KSU/data/dogs-vs-cats.zip'
```

In [2]:

```
#!unzip -qq train.zip
```

Copying images to training, validation, and test directories

In [3]:

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index,
end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=2999)
make_subset("validation", start_index=3000, end_index=3499)
make_subset("test", start_index=3500, end_index=3999)
```

Building the model

Instantiating a small convnet for dogs vs. cats classification

In [4]:

```
"""
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)
"""
```

Out[4]:

```
'\nfrom tensorflow import keras\nfrom tensorflow.keras import layers\ninputs = keras.Input(shape=(180, 180, 3))\nx = layers.Rescaling(1./255)(inputs)\nx = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)\nx = layers.Flatten()(x)\noutputs = layers.Dense(1, activation="sigmoid")(x)\nmodel = keras.Model(inputs=inputs, outputs=outputs)\n'
```

In [5]:

```
# model.summary()
```

Configuring the model for training

In [6]:

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

Out[6]:

```
'\nmodel.compile(loss="binary_crossentropy",\n              optimizer="rmsprop",\n              metrics=["accuracy"])\n'
```

Data preprocessing

Using image_dataset_from_directory to read images

In [7]:

```
from tensorflow import keras
from tensorflow.keras import layers
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 5998 files belonging to 2 classes.
Found 998 files belonging to 2 classes.
Found 998 files belonging to 2 classes.
```

In [8]:

```
"""
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)
"""
```

Out[8]:

```
'\nimport numpy as np\nimport tensorflow as tf\nrandom_numbers = np.random.no
rmal(size=(1000, 16))\ndataset = tf.data.Dataset.from_tensor_slices(random_nu
mbers)\n'
```

In [9]:

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

Out[9]:

```
'\nfor i, element in enumerate(dataset):\n    print(element.shape)\n    if i >
= 2:\n        break\n'
```

In [10]:

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

Out[10]:

```
'\nbatched_dataset = dataset.batch(32)\nfor i, element in enumerate(batched_d
ataset):\n    print(element.shape)\n    if i >= 2:\n        break\n'
```

In [11]:

```
"""
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
"""
```

Out[11]:

```
'\nreshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))\nfor i, el
ement in enumerate(reshaped_dataset):\n    print(element.shape)\n    if i >=
2:\n        break\n'
```

Displaying the shapes of the data and labels yielded by the Dataset

In [12]:

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

Out[12]:

```
'\nfor data_batch, labels_batch in train_dataset:\n    print("data batch shap
e:", data_batch.shape)\n    print("labels batch shape:", labels_batch.shape)\
n    break\n'
```

Fitting the model using a Dataset

In [13]:

```
"""
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)
"""
```

Out[13]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch.keras",\n        save_best_only=True,\n        monitor="val
_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    epochs=30,\n    val
idation_data=validation_dataset,\n    callbacks=callbacks)\n'
```

Displaying curves of loss and accuracy during training

In [14]:

```
"""
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()
"""
```

Out[14]:

```
'\nimport matplotlib.pyplot as plt\naccuracy = history.history["accuracy"]\nv
al_accuracy = history.history["val_accuracy"]\nloss = history.history["loss"]
\nval_loss = history.history["val_loss"]\nepochs = range(1, len(accuracy) + 1)
\nplt.plot(epochs, accuracy, "bo", label="Training accuracy")\nplt.plot(epoch
s, val_accuracy, "b", label="Validation accuracy")\nplt.title("Training and v
alidation accuracy")\nplt.legend()\nplt.figure()\nplt.plot(epochs, loss, "bo",
 label="Training loss")\nplt.plot(epochs, val_loss, "b", label="Validation lo
ss")\nplt.title("Training and validation loss")\nplt.legend()\nplt.show()\n'
```

Evaluating the model on the test set

In [15]:

```
"""
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}")
"""
```

Out[15]:

```
'\ntest_model = keras.models.load_model("convnet_from_scratch.keras")\ntest_l
oss, test_acc = test_model.evaluate(test_dataset)\nprint(f"Test accuracy: {te
st_acc:.3f}")\n'
```

Using data augmentation

Define a data augmentation stage to add to an image model

In [16]:

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

Out[16]:

```
'\ndata_augmentation = keras.Sequential(\n    [\n        layers.RandomFlip("h
orizontal"),\n        layers.RandomRotation(0.1),\n        layers.RandomZoom
(0.2),\n    ]\n)\n'
```

Displaying some randomly augmented training images

In [17]:

```
"""
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")
"""
```

Out[17]:

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

Defining a new convnet that includes image augmentation and dropout

In [18]:

```
"""
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"])
"""
```

Out[18]:

```
'\ninputs = keras.Input(shape=(180, 180, 3))\nx = data_augmentation(inputs)\n
x = layers.Rescaling(1./255)(x)\nx = layers.Conv2D(filters=32, kernel_size=3,
 activation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.C
onv2D(filters=64, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling
2D(pool_size=2)(x)\nx = layers.Conv2D(filters=128, kernel_size=3, activation=
"relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filter
s=256, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D(pool_siz
e=2)(x)\nx = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)\
nx = layers.Flatten()(x)\nx = layers.Dropout(0.5)(x)\noutputs = layers.Dense
(1, activation="sigmoid")(x)\nmodel = keras.Model(inputs=inputs, outputs=outp
uts)\n\nmodel.compile(loss="binary_crossentropy",\n              optimizer="r
msprop",\n              metrics=["accuracy"])\n'
```

Training the regularized convnet

In [19]:

```
"""
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)
"""
```

Out[19]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch_with_augmentation.keras",\n        save_best_only=True,\n
      monitor="val_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    e
pochs=100,\n    validation_data=validation_dataset,\n    callbacks=callbacks)
\n'
```

Evaluating the model on the test set

In [20]:

```
"""
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}")
"""
```

Out[20]:

```
'\ntest_model = keras.models.load_model(\n    "convnet_from_scratch_with_augm
entation.keras")\ntest_loss, test_acc = test_model.evaluate(test_dataset)\npr
int(f"Test accuracy: {test_acc:.3f}")\n'
```

Leveraging a pretrained model

Feature extraction with a pretrained model

Instantiating the VGG16 convolutional base

In [21]:

```
from tensorflow import keras # import keras
from tensorflow.keras import layers
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

In [22]:

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (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     
_________________________________________________________________
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
_________________________________________________________________
```

Fast feature extraction without data augmentation

Extracting the VGG16 features and corresponding labels

In [23]:

```
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)
```

In [24]:

```
train_features.shape
```

Out[24]:

```
(5998, 5, 5, 512)
```

Defining and training the densely connected classifier

In [25]:

```
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
188/188 [==============================] - 1s 4ms/step - loss: 8.3404 - accur
acy: 0.9505 - val_loss: 5.5009 - val_accuracy: 0.9739
Epoch 2/20
188/188 [==============================] - 1s 3ms/step - loss: 3.3456 - accur
acy: 0.9800 - val_loss: 5.1963 - val_accuracy: 0.9800
Epoch 3/20
188/188 [==============================] - 0s 3ms/step - loss: 2.2134 - accur
acy: 0.9860 - val_loss: 4.7894 - val_accuracy: 0.9760
Epoch 4/20
188/188 [==============================] - 0s 2ms/step - loss: 1.6404 - accur
acy: 0.9892 - val_loss: 3.9754 - val_accuracy: 0.9810
Epoch 5/20
```

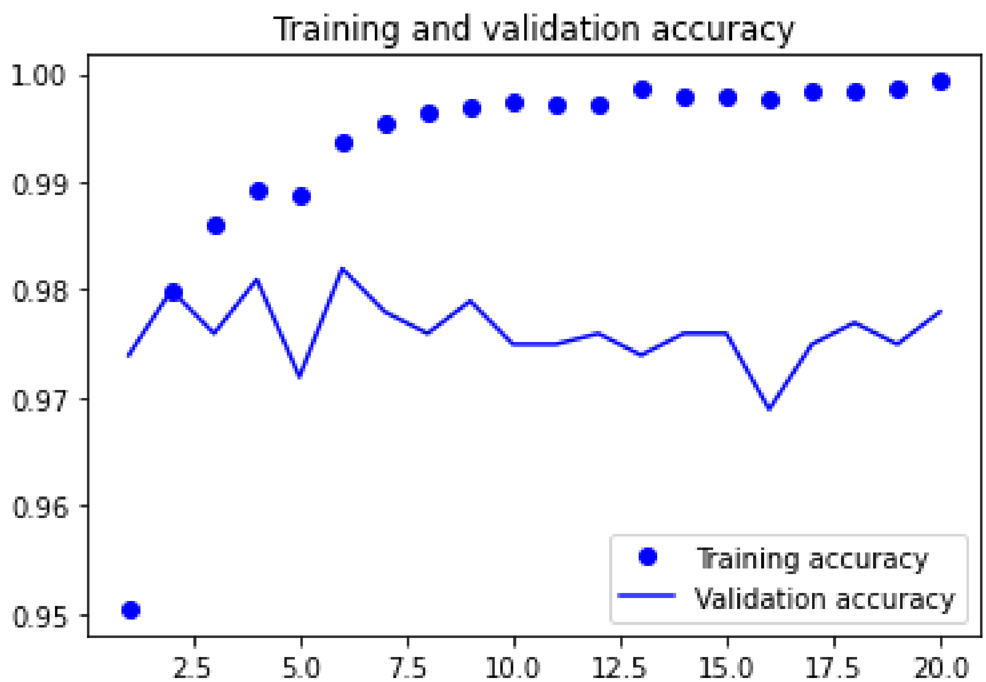
```
188/188 [==============================] - 1s 3ms/step - loss: 1.4981 - accur
acy: 0.9888 - val_loss: 5.3861 - val_accuracy: 0.9719
Epoch 6/20
188/188 [==============================] - 0s 2ms/step - loss: 0.8559 - accur
acy: 0.9938 - val_loss: 3.9264 - val_accuracy: 0.9820
Epoch 7/20
188/188 [==============================] - 0s 2ms/step - loss: 0.6068 - accur
acy: 0.9955 - val_loss: 6.1616 - val_accuracy: 0.9780
Epoch 8/20
188/188 [==============================] - 0s 2ms/step - loss: 0.2739 - accur
acy: 0.9965 - val_loss: 4.5638 - val_accuracy: 0.9760
Epoch 9/20
188/188 [==============================] - 0s 2ms/step - loss: 0.2804 - accur
acy: 0.9970 - val_loss: 4.9678 - val_accuracy: 0.9790
Epoch 10/20
188/188 [==============================] - 0s 2ms/step - loss: 0.2552 - accur
acy: 0.9973 - val_loss: 7.0815 - val_accuracy: 0.9749
Epoch 11/20
188/188 [==============================] - 0s 2ms/step - loss: 0.2234 - accur
acy: 0.9972 - val_loss: 5.6555 - val_accuracy: 0.9749
Epoch 12/20
188/188 [==============================] - 0s 2ms/step - loss: 0.2629 - accur
acy: 0.9972 - val_loss: 6.1986 - val_accuracy: 0.9760
Epoch 13/20
188/188 [==============================] - 0s 2ms/step - loss: 0.1071 - accur
acy: 0.9987 - val_loss: 5.3456 - val_accuracy: 0.9739
Epoch 14/20
188/188 [==============================] - 0s 2ms/step - loss: 0.2354 - accur
acy: 0.9978 - val_loss: 4.6037 - val_accuracy: 0.9760
Epoch 15/20
188/188 [==============================] - 0s 2ms/step - loss: 0.3182 - accur
acy: 0.9980 - val_loss: 5.5216 - val_accuracy: 0.9760
Epoch 16/20
188/188 [==============================] - 0s 2ms/step - loss: 0.1429 - accur
acy: 0.9977 - val_loss: 5.7715 - val_accuracy: 0.9689
Epoch 17/20
188/188 [==============================] - 0s 2ms/step - loss: 0.0636 - accur
acy: 0.9985 - val_loss: 5.8755 - val_accuracy: 0.9749
Epoch 18/20
188/188 [==============================] - 0s 2ms/step - loss: 0.1503 - accur
acy: 0.9985 - val_loss: 6.7080 - val_accuracy: 0.9770
Epoch 19/20
188/188 [==============================] - 0s 2ms/step - loss: 0.0935 - accur
acy: 0.9987 - val_loss: 5.6859 - val_accuracy: 0.9749
Epoch 20/20
188/188 [==============================] - 0s 2ms/step - loss: 0.0171 - accur
acy: 0.9993 - val_loss: 5.6052 - val_accuracy: 0.9780
```

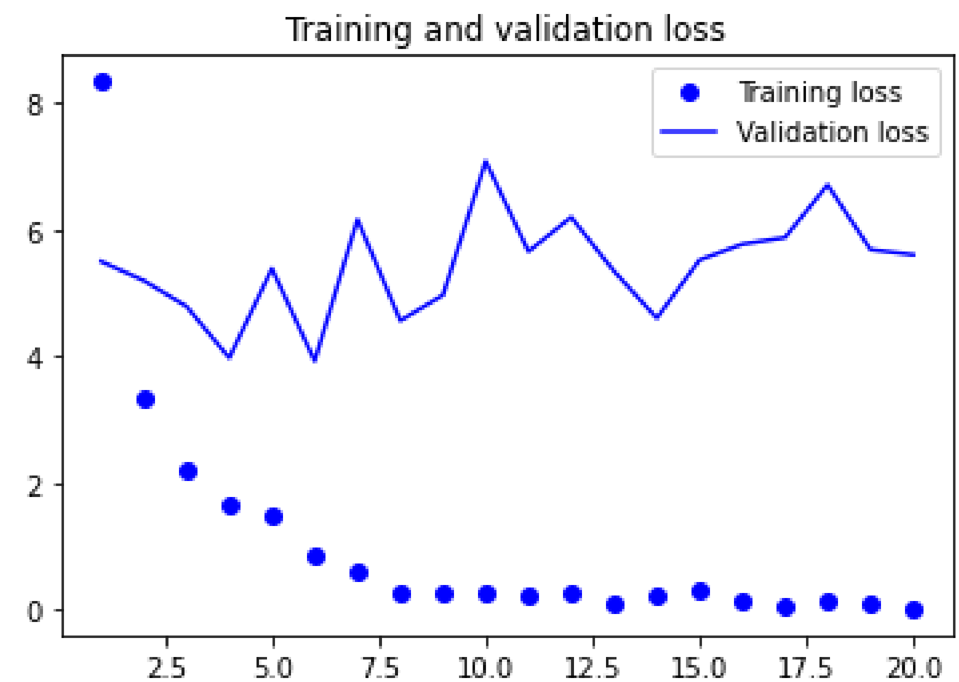
Plotting the results

In [26]:

```
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

In [27]:

```
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

In [28]:

```
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
```

In [29]:

```
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

In [30]:

```
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"])
```

In [31]:

```
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
188/188 [==============================] - 9s 42ms/step - loss: 11.1415 - acc
uracy: 0.9263 - val_loss: 2.8871 - val_accuracy: 0.9739
Epoch 2/50
188/188 [==============================] - 8s 39ms/step - loss: 6.5339 - accu
racy: 0.9498 - val_loss: 4.5935 - val_accuracy: 0.9669
Epoch 3/50
188/188 [==============================] - 8s 40ms/step - loss: 4.3118 - accu
racy: 0.9548 - val_loss: 1.8875 - val_accuracy: 0.9780
Epoch 4/50
188/188 [==============================] - 8s 40ms/step - loss: 3.2748 - accu
racy: 0.9592 - val_loss: 3.2645 - val_accuracy: 0.9709
Epoch 5/50
188/188 [==============================] - 8s 40ms/step - loss: 2.1827 - accu
racy: 0.9608 - val_loss: 1.2957 - val_accuracy: 0.9780
Epoch 6/50
188/188 [==============================] - 8s 40ms/step - loss: 1.7407 - accu
racy: 0.9615 - val_loss: 0.8285 - val_accuracy: 0.9810
Epoch 7/50
188/188 [==============================] - 8s 40ms/step - loss: 1.1840 - accu
racy: 0.9627 - val_loss: 0.6427 - val_accuracy: 0.9820
Epoch 8/50
188/188 [==============================] - 8s 40ms/step - loss: 0.8556 - accu
racy: 0.9668 - val_loss: 0.3899 - val_accuracy: 0.9830
Epoch 9/50
```

```
188/188 [==============================] - 8s 40ms/step - loss: 0.7557 - accu
racy: 0.9648 - val_loss: 0.6245 - val_accuracy: 0.9800
Epoch 10/50
188/188 [==============================] - 8s 39ms/step - loss: 0.6924 - accu
racy: 0.9673 - val_loss: 0.6261 - val_accuracy: 0.9810
Epoch 11/50
188/188 [==============================] - 8s 39ms/step - loss: 0.6886 - accu
racy: 0.9693 - val_loss: 0.5560 - val_accuracy: 0.9830
Epoch 12/50
188/188 [==============================] - 8s 39ms/step - loss: 0.6191 - accu
racy: 0.9708 - val_loss: 0.8211 - val_accuracy: 0.9770
Epoch 13/50
188/188 [==============================] - 8s 39ms/step - loss: 0.6507 - accu
racy: 0.9732 - val_loss: 1.7536 - val_accuracy: 0.9559
Epoch 14/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6279 - accu
racy: 0.9713 - val_loss: 0.8081 - val_accuracy: 0.9729
Epoch 15/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6727 - accu
racy: 0.9737 - val_loss: 0.5705 - val_accuracy: 0.9820
Epoch 16/50
188/188 [==============================] - 8s 39ms/step - loss: 0.6008 - accu
racy: 0.9743 - val_loss: 0.7119 - val_accuracy: 0.9850
Epoch 17/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6891 - accu
racy: 0.9707 - val_loss: 0.9312 - val_accuracy: 0.9800
Epoch 18/50
188/188 [==============================] - 8s 39ms/step - loss: 0.6361 - accu
racy: 0.9748 - val_loss: 0.7632 - val_accuracy: 0.9770
Epoch 19/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6982 - accu
racy: 0.9713 - val_loss: 0.7708 - val_accuracy: 0.9810
Epoch 20/50
188/188 [==============================] - 8s 40ms/step - loss: 0.5491 - accu
racy: 0.9773 - val_loss: 0.6233 - val_accuracy: 0.9830
Epoch 21/50
188/188 [==============================] - 8s 39ms/step - loss: 0.6702 - accu
racy: 0.9740 - val_loss: 1.2653 - val_accuracy: 0.9749
Epoch 22/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6501 - accu
racy: 0.9743 - val_loss: 0.6579 - val_accuracy: 0.9860
Epoch 23/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7387 - accu
racy: 0.9732 - val_loss: 0.8399 - val_accuracy: 0.9840
Epoch 24/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7319 - accu
racy: 0.9733 - val_loss: 0.6483 - val_accuracy: 0.9850
Epoch 25/50
188/188 [==============================] - 8s 39ms/step - loss: 0.5826 - accu
racy: 0.9788 - val_loss: 1.0020 - val_accuracy: 0.9810
Epoch 26/50
188/188 [==============================] - 8s 41ms/step - loss: 0.5837 - accu
racy: 0.9802 - val_loss: 0.7137 - val_accuracy: 0.9840
```

```
Epoch 27/50
188/188 [==============================] - 8s 41ms/step - loss: 0.6349 - accu
racy: 0.9785 - val_loss: 0.6639 - val_accuracy: 0.9810
Epoch 28/50
188/188 [==============================] - 8s 41ms/step - loss: 0.7112 - accu
racy: 0.9770 - val_loss: 0.8520 - val_accuracy: 0.9860
Epoch 29/50
188/188 [==============================] - 8s 40ms/step - loss: 0.5568 - accu
racy: 0.9805 - val_loss: 1.0098 - val_accuracy: 0.9850
Epoch 30/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6739 - accu
racy: 0.9785 - val_loss: 1.0824 - val_accuracy: 0.9760
Epoch 31/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6461 - accu
racy: 0.9795 - val_loss: 0.8808 - val_accuracy: 0.9850
Epoch 32/50
188/188 [==============================] - 8s 40ms/step - loss: 0.8002 - accu
racy: 0.9750 - val_loss: 0.9471 - val_accuracy: 0.9820
Epoch 33/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7060 - accu
racy: 0.9770 - val_loss: 0.7227 - val_accuracy: 0.9840
Epoch 34/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7301 - accu
racy: 0.9782 - val_loss: 0.8728 - val_accuracy: 0.9820
Epoch 35/50
188/188 [==============================] - 8s 40ms/step - loss: 0.5756 - accu
racy: 0.9798 - val_loss: 0.8718 - val_accuracy: 0.9820
Epoch 36/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6341 - accu
racy: 0.9790 - val_loss: 0.8387 - val_accuracy: 0.9830
Epoch 37/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7626 - accu
racy: 0.9782 - val_loss: 0.8807 - val_accuracy: 0.9830
Epoch 38/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7786 - accu
racy: 0.9767 - val_loss: 0.7706 - val_accuracy: 0.9840
Epoch 39/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7105 - accu
racy: 0.9780 - val_loss: 0.9285 - val_accuracy: 0.9820
Epoch 40/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7216 - accu
racy: 0.9783 - val_loss: 0.9317 - val_accuracy: 0.9830
Epoch 41/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7098 - accu
racy: 0.9780 - val_loss: 0.9825 - val_accuracy: 0.9810
Epoch 42/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6639 - accu
racy: 0.9813 - val_loss: 0.9120 - val_accuracy: 0.9850
Epoch 43/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6375 - accu
racy: 0.9815 - val_loss: 1.2511 - val_accuracy: 0.9840
Epoch 44/50
```

```
188/188 [==============================] - 8s 40ms/step - loss: 0.7082 - accu
racy: 0.9785 - val_loss: 0.9283 - val_accuracy: 0.9870
Epoch 45/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6106 - accu
racy: 0.9845 - val_loss: 0.8911 - val_accuracy: 0.9850
Epoch 46/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6737 - accu
racy: 0.9808 - val_loss: 1.0990 - val_accuracy: 0.9810
Epoch 47/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7285 - accu
racy: 0.9787 - val_loss: 1.3349 - val_accuracy: 0.9830
Epoch 48/50
188/188 [==============================] - 8s 40ms/step - loss: 0.6577 - accu
racy: 0.9830 - val_loss: 1.3496 - val_accuracy: 0.9830
Epoch 49/50
188/188 [==============================] - 8s 40ms/step - loss: 0.9656 - accu
racy: 0.9787 - val_loss: 1.0118 - val_accuracy: 0.9800
Epoch 50/50
188/188 [==============================] - 8s 40ms/step - loss: 0.7398 - accu
racy: 0.9823 - val_loss: 1.8748 - val_accuracy: 0.9780
```

Evaluating the model on the test set

In [32]:

```
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 [==============================] - 1s 33ms/step - loss: 0.7367 - accura
cy: 0.9790
Test accuracy: 0.979
```

Fine-tuning a pretrained model

In [33]:

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
```

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

Freezing all layers until the fourth from the last

In [34]:

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

Fine-tuning the model

In [35]:

```
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
188/188 [==============================] - 10s 45ms/step - loss: 0.6928 - acc
uracy: 0.9780 - val_loss: 0.7297 - val_accuracy: 0.9830
Epoch 2/30
188/188 [==============================] - 8s 44ms/step - loss: 0.5555 - accu
racy: 0.9808 - val_loss: 0.9200 - val_accuracy: 0.9800
Epoch 3/30
188/188 [==============================] - 8s 44ms/step - loss: 0.3402 - accu
racy: 0.9852 - val_loss: 0.7360 - val_accuracy: 0.9800
Epoch 4/30
188/188 [==============================] - 9s 45ms/step - loss: 0.3804 - accu
racy: 0.9852 - val_loss: 0.7474 - val_accuracy: 0.9830
Epoch 5/30
188/188 [==============================] - 8s 43ms/step - loss: 0.3299 - accu
racy: 0.9845 - val_loss: 0.7136 - val_accuracy: 0.9850
Epoch 6/30
188/188 [==============================] - 8s 43ms/step - loss: 0.3167 - accu
racy: 0.9865 - val_loss: 0.6155 - val_accuracy: 0.9840
Epoch 7/30
188/188 [==============================] - 8s 43ms/step - loss: 0.2677 - accu
racy: 0.9855 - val_loss: 0.5729 - val_accuracy: 0.9850
Epoch 8/30
188/188 [==============================] - 8s 43ms/step - loss: 0.1810 - accu
racy: 0.9900 - val_loss: 0.7391 - val_accuracy: 0.9780
Epoch 9/30
188/188 [==============================] - 8s 43ms/step - loss: 0.1700 - accu
racy: 0.9895 - val_loss: 0.5627 - val_accuracy: 0.9800
Epoch 10/30
188/188 [==============================] - 8s 44ms/step - loss: 0.1326 - accu
racy: 0.9900 - val_loss: 0.5365 - val_accuracy: 0.9830
Epoch 11/30
188/188 [==============================] - 8s 43ms/step - loss: 0.1140 - accu
racy: 0.9918 - val_loss: 0.6783 - val_accuracy: 0.9780
Epoch 12/30
188/188 [==============================] - 8s 44ms/step - loss: 0.1389 - accu
racy: 0.9902 - val_loss: 0.4407 - val_accuracy: 0.9860
Epoch 13/30
188/188 [==============================] - 8s 43ms/step - loss: 0.1394 - accu
racy: 0.9915 - val_loss: 0.4445 - val_accuracy: 0.9860
Epoch 14/30
188/188 [==============================] - 8s 43ms/step - loss: 0.1565 - accu
racy: 0.9918 - val_loss: 0.4780 - val_accuracy: 0.9840
Epoch 15/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0792 - accu
racy: 0.9940 - val_loss: 0.5984 - val_accuracy: 0.9810
Epoch 16/30
188/188 [==============================] - 8s 43ms/step - loss: 0.1312 - accu
racy: 0.9907 - val_loss: 0.7834 - val_accuracy: 0.9820
Epoch 17/30
188/188 [==============================] - 8s 44ms/step - loss: 0.0776 - accu
racy: 0.9950 - val_loss: 0.6558 - val_accuracy: 0.9840
```

```
Epoch 18/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0899 - accu
racy: 0.9920 - val_loss: 0.6427 - val_accuracy: 0.9840
Epoch 19/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0774 - accu
racy: 0.9933 - val_loss: 0.5150 - val_accuracy: 0.9870
Epoch 20/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0772 - accu
racy: 0.9940 - val_loss: 0.5283 - val_accuracy: 0.9860
Epoch 21/30
188/188 [==============================] - 8s 44ms/step - loss: 0.0716 - accu
racy: 0.9943 - val_loss: 0.5145 - val_accuracy: 0.9830
Epoch 22/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0568 - accu
racy: 0.9953 - val_loss: 0.5406 - val_accuracy: 0.9880
Epoch 23/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0569 - accu
racy: 0.9957 - val_loss: 0.5038 - val_accuracy: 0.9860
Epoch 24/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0556 - accu
racy: 0.9948 - val_loss: 0.6287 - val_accuracy: 0.9860
Epoch 25/30
188/188 [==============================] - 8s 43ms/step - loss: 0.0645 - accu
racy: 0.9947 - val_loss: 0.4839 - val_accuracy: 0.9880
Epoch 26/30
188/188 [==============================] - 8s 44ms/step - loss: 0.0593 - accu
racy: 0.9953 - val_loss: 0.5203 - val_accuracy: 0.9890
Epoch 27/30
188/188 [==============================] - 9s 45ms/step - loss: 0.0666 - accu
racy: 0.9952 - val_loss: 0.4642 - val_accuracy: 0.9850
Epoch 28/30
188/188 [==============================] - 8s 44ms/step - loss: 0.0503 - accu
racy: 0.9950 - val_loss: 0.4715 - val_accuracy: 0.9860
Epoch 29/30
188/188 [==============================] - 8s 44ms/step - loss: 0.0286 - accu
racy: 0.9975 - val_loss: 0.5109 - val_accuracy: 0.9890
Epoch 30/30
188/188 [==============================] - 8s 44ms/step - loss: 0.0466 - accu
racy: 0.9965 - val_loss: 0.6845 - val_accuracy: 0.9810
```

In [36]:

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

```
32/32 [==============================] - 1s 33ms/step - loss: 0.9459 - accura
cy: 0.9790
Test accuracy: 0.979
```

Summary

Using a pretrained network with training sample of 5000, a validation sample of 500, and a test sample of 500

Downloading the data

In [1]:

```
#!unzip -qq '/fs/ess/PGS0333/BA_64061_KSU/data/dogs-vs-cats.zip'
```

In [2]:

```
#!unzip -qq train.zip
```

Copying images to training, validation, and test directories

In [3]:

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index,
end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=4999)
make_subset("validation", start_index=5000, end_index=5499)
make_subset("test", start_index=5500, end_index=5999)
```

Building the model

Instantiating a small convnet for dogs vs. cats classification

In [4]:

```
"""
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)
"""
```

Out[4]:

```
'\nfrom tensorflow import keras\nfrom tensorflow.keras import layers\ninputs
= keras.Input(shape=(180, 180, 3))\nx = layers.Rescaling(1./255)(inputs)\nx =
 layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)\nx = layers.M
axPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=64, kernel_size=3, act
ivation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2
D(filters=128, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D
(pool_size=2)(x)\nx = layers.Conv2D(filters=256, kernel_size=3, activation="r
elu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filters=
256, kernel_size=3, activation="relu")(x)\nx = layers.Flatten()(x)\noutputs =
 layers.Dense(1, activation="sigmoid")(x)\nmodel = keras.Model(inputs=inputs,
 outputs=outputs)\n'
```

In [5]:

```
# model.summary()
```

Configuring the model for training

In [6]:

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

Out[6]:

```
'\nmodel.compile(loss="binary_crossentropy",\n              optimizer="rmspro
p",\n              metrics=["accuracy"])\n'
```

Data preprocessing

Using image_dataset_from_directory to read images

In [7]:

```
from tensorflow import keras
from tensorflow.keras import layers
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 9998 files belonging to 2 classes.
Found 998 files belonging to 2 classes.
Found 998 files belonging to 2 classes.
```

In [8]:

```
"""
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)
"""
```

Out[8]:

```
'\nimport numpy as np\nimport tensorflow as tf\nrandom_numbers = np.random.no
rmal(size=(1000, 16))\ndataset = tf.data.Dataset.from_tensor_slices(random_nu
mbers)\n'
```

In [9]:

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

Out[9]:

```
'\nfor i, element in enumerate(dataset):\n    print(element.shape)\n    if i >
= 2:\n        break\n'
```

In [10]:

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

Out[10]:

```
'\nbatched_dataset = dataset.batch(32)\nfor i, element in enumerate(batched_d
ataset):\n    print(element.shape)\n    if i >= 2:\n        break\n'
```

In [11]:

```
"""
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
"""
```

Out[11]:

```
'\nreshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))\nfor i, el
ement in enumerate(reshaped_dataset):\n    print(element.shape)\n    if i >=
2:\n        break\n'
```

Displaying the shapes of the data and labels yielded by the Dataset

In [12]:

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

Out[12]:

```
'\nfor data_batch, labels_batch in train_dataset:\n    print("data batch shap
e:", data_batch.shape)\n    print("labels batch shape:", labels_batch.shape)\
n    break\n'
```

Fitting the model using a Dataset

In [13]:

```
"""
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)
"""
```

Out[13]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch.keras",\n        save_best_only=True,\n        monitor="val
_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    epochs=30,\n    val
idation_data=validation_dataset,\n    callbacks=callbacks)\n'
```

Displaying curves of loss and accuracy during training

In [14]:

```
"""
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()
"""
```

Out[14]:

```
'\nimport matplotlib.pyplot as plt\naccuracy = history.history["accuracy"]\nv
al_accuracy = history.history["val_accuracy"]\nloss = history.history["loss"]
\nval_loss = history.history["val_loss"]\nepochs = range(1, len(accuracy) + 1)
\nplt.plot(epochs, accuracy, "bo", label="Training accuracy")\nplt.plot(epoch
s, val_accuracy, "b", label="Validation accuracy")\nplt.title("Training and v
alidation accuracy")\nplt.legend()\nplt.figure()\nplt.plot(epochs, loss, "bo",
 label="Training loss")\nplt.plot(epochs, val_loss, "b", label="Validation lo
ss")\nplt.title("Training and validation loss")\nplt.legend()\nplt.show()\n'
```

Evaluating the model on the test set

In [15]:

```
"""
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}")
"""
```

Out[15]:

```
'\ntest_model = keras.models.load_model("convnet_from_scratch.keras")\ntest_l
oss, test_acc = test_model.evaluate(test_dataset)\nprint(f"Test accuracy: {te
st_acc:.3f}")\n'
```

Using data augmentation

Define a data augmentation stage to add to an image model

In [16]:

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

Out[16]:

```
'\ndata_augmentation = keras.Sequential(\n    [\n        layers.RandomFlip("h
orizontal"),\n        layers.RandomRotation(0.1),\n        layers.RandomZoom
(0.2),\n    ]\n)\n'
```

Displaying some randomly augmented training images

In [17]:

```
"""
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")
"""
```

Out[17]:

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

Defining a new convnet that includes image augmentation and dropout

In [18]:

```
"""
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"])
"""
```

Out[18]:

```
'\ninputs = keras.Input(shape=(180, 180, 3))\nx = data_augmentation(inputs)\n
x = layers.Rescaling(1./255)(x)\nx = layers.Conv2D(filters=32, kernel_size=3,
 activation="relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.C
onv2D(filters=64, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling
2D(pool_size=2)(x)\nx = layers.Conv2D(filters=128, kernel_size=3, activation=
"relu")(x)\nx = layers.MaxPooling2D(pool_size=2)(x)\nx = layers.Conv2D(filter
s=256, kernel_size=3, activation="relu")(x)\nx = layers.MaxPooling2D(pool_siz
e=2)(x)\nx = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)\
nx = layers.Flatten()(x)\nx = layers.Dropout(0.5)(x)\noutputs = layers.Dense
(1, activation="sigmoid")(x)\nmodel = keras.Model(inputs=inputs, outputs=outp
uts)\n\nmodel.compile(loss="binary_crossentropy",\n              optimizer="r
msprop",\n              metrics=["accuracy"])\n'
```

Training the regularized convnet

In [19]:

```
"""
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)
"""
```

Out[19]:

```
'\ncallbacks = [\n    keras.callbacks.ModelCheckpoint(\n        filepath="con
vnet_from_scratch_with_augmentation.keras",\n        save_best_only=True,\n
      monitor="val_loss")\n]\nhistory = model.fit(\n    train_dataset,\n    e
pochs=100,\n    validation_data=validation_dataset,\n    callbacks=callbacks)
\n'
```

Evaluating the model on the test set

In [20]:

```
"""
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}")
"""
```

Out[20]:

```
'\ntest_model = keras.models.load_model(\n    "convnet_from_scratch_with_augm
entation.keras")\ntest_loss, test_acc = test_model.evaluate(test_dataset)\npr
int(f"Test accuracy: {test_acc:.3f}")\n'
```

Leveraging a pretrained model

Feature extraction with a pretrained model

Instantiating the VGG16 convolutional base

In [21]:

```
from tensorflow import keras # import keras
from tensorflow.keras import layers
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

In [22]:

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (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
_________________________________________________________________
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
_________________________________________________________________
```

Fast feature extraction without data augmentation

Extracting the VGG16 features and corresponding labels

In [23]:

```
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)
```

In [24]:

```
train_features.shape
```

Out[24]:

```
(9998, 5, 5, 512)
```

Defining and training the densely connected classifier

In [25]:

```
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
313/313 [==============================] - 1s 3ms/step - loss: 7.6790 - accur
acy: 0.9564 - val_loss: 5.2808 - val_accuracy: 0.9719
Epoch 2/20
313/313 [==============================] - 1s 2ms/step - loss: 3.6543 - accur
acy: 0.9779 - val_loss: 5.5756 - val_accuracy: 0.9699
Epoch 3/20
313/313 [==============================] - 1s 2ms/step - loss: 1.9265 - accur
acy: 0.9849 - val_loss: 5.5961 - val_accuracy: 0.9749
Epoch 4/20
313/313 [==============================] - 1s 2ms/step - loss: 1.5110 - accur
acy: 0.9883 - val_loss: 4.9416 - val_accuracy: 0.9749
Epoch 5/20
```

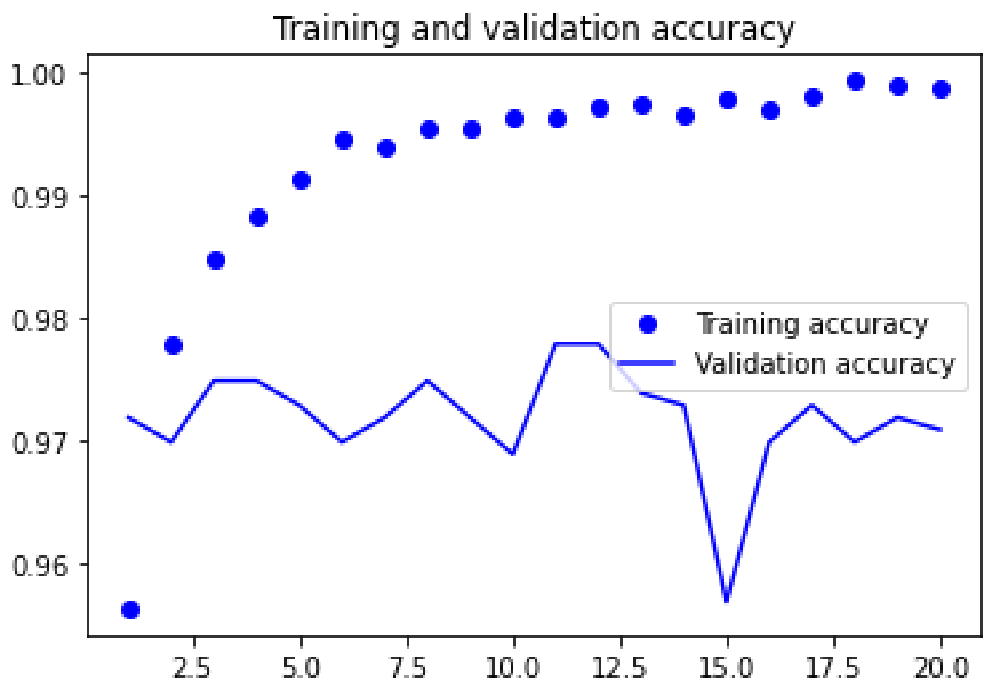
```
313/313 [==============================] - 1s 2ms/step - loss: 0.9044 - accur
acy: 0.9915 - val_loss: 5.3424 - val_accuracy: 0.9729
Epoch 6/20
313/313 [==============================] - 1s 2ms/step - loss: 0.7547 - accur
acy: 0.9947 - val_loss: 4.7005 - val_accuracy: 0.9699
Epoch 7/20
313/313 [==============================] - 1s 2ms/step - loss: 0.4912 - accur
acy: 0.9940 - val_loss: 5.3495 - val_accuracy: 0.9719
Epoch 8/20
313/313 [==============================] - 1s 2ms/step - loss: 0.3394 - accur
acy: 0.9956 - val_loss: 5.9827 - val_accuracy: 0.9749
Epoch 9/20
313/313 [==============================] - 1s 2ms/step - loss: 0.3428 - accur
acy: 0.9956 - val_loss: 6.8560 - val_accuracy: 0.9719
Epoch 10/20
313/313 [==============================] - 1s 2ms/step - loss: 0.2664 - accur
acy: 0.9964 - val_loss: 7.2837 - val_accuracy: 0.9689
Epoch 11/20
313/313 [==============================] - 1s 2ms/step - loss: 0.3664 - accur
acy: 0.9964 - val_loss: 5.8309 - val_accuracy: 0.9780
Epoch 12/20
313/313 [==============================] - 1s 2ms/step - loss: 0.2396 - accur
acy: 0.9973 - val_loss: 5.6674 - val_accuracy: 0.9780
Epoch 13/20
313/313 [==============================] - 1s 2ms/step - loss: 0.2081 - accur
acy: 0.9976 - val_loss: 5.8744 - val_accuracy: 0.9739
Epoch 14/20
313/313 [==============================] - 1s 2ms/step - loss: 0.2907 - accur
acy: 0.9966 - val_loss: 5.7072 - val_accuracy: 0.9729
Epoch 15/20
313/313 [==============================] - 1s 2ms/step - loss: 0.1788 - accur
acy: 0.9980 - val_loss: 9.6363 - val_accuracy: 0.9569
Epoch 16/20
313/313 [==============================] - 1s 2ms/step - loss: 0.1805 - accur
acy: 0.9971 - val_loss: 7.4781 - val_accuracy: 0.9699
Epoch 17/20
313/313 [==============================] - 1s 2ms/step - loss: 0.1058 - accur
acy: 0.9982 - val_loss: 6.7146 - val_accuracy: 0.9729
Epoch 18/20
313/313 [==============================] - 1s 2ms/step - loss: 0.0558 - accur
acy: 0.9994 - val_loss: 6.8548 - val_accuracy: 0.9699
Epoch 19/20
313/313 [==============================] - 1s 2ms/step - loss: 0.0696 - accur
acy: 0.9990 - val_loss: 7.1342 - val_accuracy: 0.9719
Epoch 20/20
313/313 [==============================] - 1s 2ms/step - loss: 0.0474 - accur
acy: 0.9989 - val_loss: 7.2712 - val_accuracy: 0.9709
```

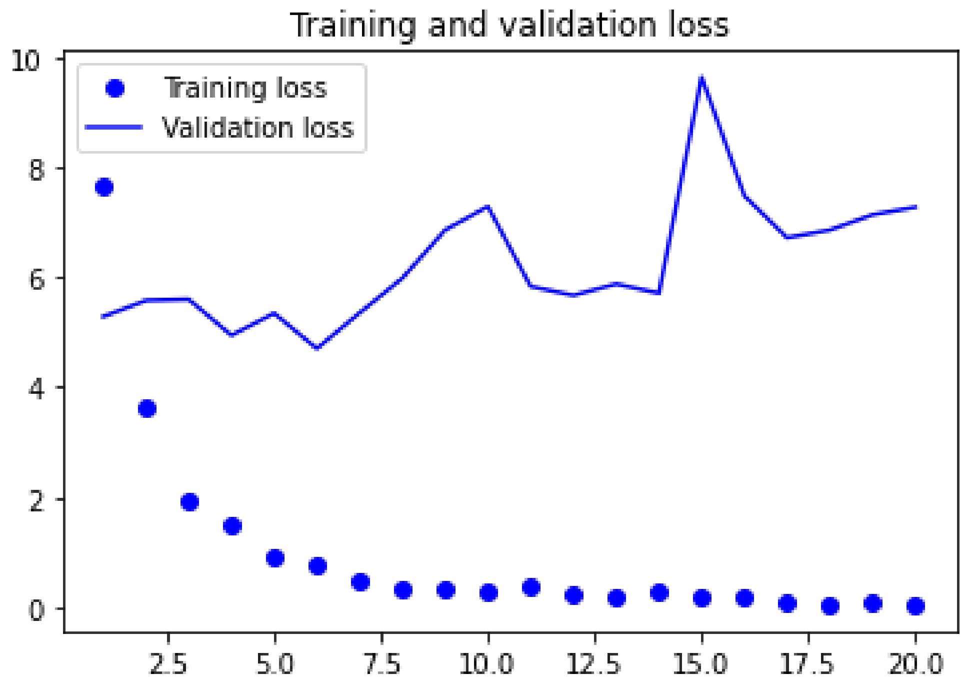
Plotting the results

In [26]:

```
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

In [27]:

```
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

In [28]:

```
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
```

In [29]:

```
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

In [30]:

```
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"])
```

In [31]:

```
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
313/313 [==============================] - 12s 35ms/step - loss: 9.6923 - acc
uracy: 0.9346 - val_loss: 5.2397 - val_accuracy: 0.9679
Epoch 2/50
313/313 [==============================] - 11s 34ms/step - loss: 4.9260 - acc
uracy: 0.9525 - val_loss: 2.3895 - val_accuracy: 0.9749
Epoch 3/50
313/313 [==============================] - 11s 34ms/step - loss: 2.4564 - acc
uracy: 0.9564 - val_loss: 1.5804 - val_accuracy: 0.9729
Epoch 4/50
313/313 [==============================] - 11s 35ms/step - loss: 1.1197 - acc
uracy: 0.9556 - val_loss: 0.9224 - val_accuracy: 0.9689
Epoch 5/50
313/313 [==============================] - 11s 35ms/step - loss: 0.6696 - acc
uracy: 0.9590 - val_loss: 0.8900 - val_accuracy: 0.9639
Epoch 6/50
313/313 [==============================] - 11s 35ms/step - loss: 0.6269 - acc
uracy: 0.9626 - val_loss: 0.7655 - val_accuracy: 0.9699
Epoch 7/50
313/313 [==============================] - 11s 35ms/step - loss: 0.6374 - acc
uracy: 0.9633 - val_loss: 0.7202 - val_accuracy: 0.9719
Epoch 8/50
313/313 [==============================] - 11s 35ms/step - loss: 0.6553 - acc
uracy: 0.9650 - val_loss: 1.2151 - val_accuracy: 0.9629
Epoch 9/50
```

```
313/313 [==============================] - 11s 35ms/step - loss: 0.6400 - acc
uracy: 0.9677 - val_loss: 1.3147 - val_accuracy: 0.9689
Epoch 10/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7644 - acc
uracy: 0.9653 - val_loss: 1.2588 - val_accuracy: 0.9709
Epoch 11/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7015 - acc
uracy: 0.9664 - val_loss: 1.0466 - val_accuracy: 0.9739
Epoch 12/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7859 - acc
uracy: 0.9679 - val_loss: 0.8831 - val_accuracy: 0.9770
Epoch 13/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7914 - acc
uracy: 0.9659 - val_loss: 1.3361 - val_accuracy: 0.9689
Epoch 14/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7501 - acc
uracy: 0.9682 - val_loss: 1.0085 - val_accuracy: 0.9699
Epoch 15/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7335 - acc
uracy: 0.9687 - val_loss: 1.1531 - val_accuracy: 0.9729
Epoch 16/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7070 - acc
uracy: 0.9715 - val_loss: 1.2293 - val_accuracy: 0.9689
Epoch 17/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8132 - acc
uracy: 0.9698 - val_loss: 1.2263 - val_accuracy: 0.9719
Epoch 18/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8423 - acc
uracy: 0.9700 - val_loss: 2.0989 - val_accuracy: 0.9589
Epoch 19/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7331 - acc
uracy: 0.9714 - val_loss: 2.0393 - val_accuracy: 0.9609
Epoch 20/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7890 - acc
uracy: 0.9707 - val_loss: 1.2456 - val_accuracy: 0.9760
Epoch 21/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7665 - acc
uracy: 0.9715 - val_loss: 1.3908 - val_accuracy: 0.9719
Epoch 22/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7519 - acc
uracy: 0.9734 - val_loss: 1.5127 - val_accuracy: 0.9719
Epoch 23/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7151 - acc
uracy: 0.9741 - val_loss: 1.7963 - val_accuracy: 0.9639
Epoch 24/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8171 - acc
uracy: 0.9725 - val_loss: 1.5686 - val_accuracy: 0.9739
Epoch 25/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8799 - acc
uracy: 0.9714 - val_loss: 1.5058 - val_accuracy: 0.9719
Epoch 26/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8031 - acc
uracy: 0.9740 - val_loss: 1.7395 - val_accuracy: 0.9679
```

```
Epoch 27/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8709 - acc
uracy: 0.9710 - val_loss: 1.6984 - val_accuracy: 0.9689
Epoch 28/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7026 - acc
uracy: 0.9765 - val_loss: 2.1059 - val_accuracy: 0.9739
Epoch 29/50
313/313 [==============================] - 11s 35ms/step - loss: 0.9013 - acc
uracy: 0.9723 - val_loss: 3.1759 - val_accuracy: 0.9549
Epoch 30/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8879 - acc
uracy: 0.9715 - val_loss: 2.2850 - val_accuracy: 0.9669
Epoch 31/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7657 - acc
uracy: 0.9762 - val_loss: 2.2964 - val_accuracy: 0.9619
Epoch 32/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8014 - acc
uracy: 0.9751 - val_loss: 1.7148 - val_accuracy: 0.9699
Epoch 33/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8062 - acc
uracy: 0.9731 - val_loss: 1.8675 - val_accuracy: 0.9739
Epoch 34/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8036 - acc
uracy: 0.9761 - val_loss: 2.8071 - val_accuracy: 0.9609
Epoch 35/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8583 - acc
uracy: 0.9745 - val_loss: 1.7494 - val_accuracy: 0.9709
Epoch 36/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8375 - acc
uracy: 0.9755 - val_loss: 2.0629 - val_accuracy: 0.9729
Epoch 37/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7910 - acc
uracy: 0.9764 - val_loss: 2.2334 - val_accuracy: 0.9679
Epoch 38/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8255 - acc
uracy: 0.9767 - val_loss: 1.9966 - val_accuracy: 0.9689
Epoch 39/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8527 - acc
uracy: 0.9763 - val_loss: 2.7698 - val_accuracy: 0.9609
Epoch 40/50
313/313 [==============================] - 11s 35ms/step - loss: 0.7939 - acc
uracy: 0.9778 - val_loss: 2.7091 - val_accuracy: 0.9679
Epoch 41/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8483 - acc
uracy: 0.9771 - val_loss: 2.7788 - val_accuracy: 0.9679
Epoch 42/50
313/313 [==============================] - 11s 35ms/step - loss: 0.9186 - acc
uracy: 0.9747 - val_loss: 1.8424 - val_accuracy: 0.9719
Epoch 43/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8484 - acc
uracy: 0.9773 - val_loss: 3.0520 - val_accuracy: 0.9589
Epoch 44/50
```

```
313/313 [==============================] - 11s 35ms/step - loss: 0.8122 - acc
uracy: 0.9754 - val_loss: 1.7815 - val_accuracy: 0.9709
Epoch 45/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8498 - acc
uracy: 0.9762 - val_loss: 1.9218 - val_accuracy: 0.9699
Epoch 46/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8169 - acc
uracy: 0.9775 - val_loss: 2.2869 - val_accuracy: 0.9689
Epoch 47/50
313/313 [==============================] - 11s 35ms/step - loss: 0.9346 - acc
uracy: 0.9753 - val_loss: 2.1793 - val_accuracy: 0.9699
Epoch 48/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8180 - acc
uracy: 0.9790 - val_loss: 4.1362 - val_accuracy: 0.9539
Epoch 49/50
313/313 [==============================] - 11s 35ms/step - loss: 0.8713 - acc
uracy: 0.9769 - val_loss: 2.7774 - val_accuracy: 0.9619
Epoch 50/50
313/313 [==============================] - 11s 35ms/step - loss: 0.9165 - acc
uracy: 0.9767 - val_loss: 2.1626 - val_accuracy: 0.9739
```

Evaluating the model on the test set

In [32]:

```
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 [==============================] - 1s 30ms/step - loss: 0.4685 - accura
cy: 0.9850
Test accuracy: 0.985
```

Fine-tuning a pretrained model

In [33]:

```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                Output Shape              Param #   
=================================================================
input_3 (InputLayer)        [(None, None, None, 3)]   0         
                                                                 
_________________________________________________________________
block1_conv1 (Conv2D)       (None, None, None, 64)    1792      
                                                                 
_________________________________________________________________
block1_conv2 (Conv2D)       (None, None, None, 64)    36928     
                                                                 
_________________________________________________________________
block1_pool (MaxPooling2D)  (None, None, None, 64)    0         
                                                                 
_________________________________________________________________
block2_conv1 (Conv2D)       (None, None, None, 128)   73856     
                                                                 
_________________________________________________________________
block2_conv2 (Conv2D)       (None, None, None, 128)   147584    
                                                                 
_________________________________________________________________
```

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

Freezing all layers until the fourth from the last

In [34]:

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

Fine-tuning the model

In [35]:

```
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
313/313 [==============================] - 14s 39ms/step - loss: 0.7166 - acc
uracy: 0.9767 - val_loss: 1.6411 - val_accuracy: 0.9749
Epoch 2/30
313/313 [==============================] - 12s 38ms/step - loss: 0.4809 - acc
uracy: 0.9794 - val_loss: 1.8219 - val_accuracy: 0.9729
Epoch 3/30
313/313 [==============================] - 12s 38ms/step - loss: 0.4577 - acc
uracy: 0.9802 - val_loss: 1.4528 - val_accuracy: 0.9709
Epoch 4/30
313/313 [==============================] - 12s 38ms/step - loss: 0.2875 - acc
uracy: 0.9846 - val_loss: 1.2672 - val_accuracy: 0.9719
Epoch 5/30
313/313 [==============================] - 12s 38ms/step - loss: 0.2989 - acc
uracy: 0.9837 - val_loss: 0.9301 - val_accuracy: 0.9780
Epoch 6/30
313/313 [==============================] - 12s 38ms/step - loss: 0.2356 - acc
uracy: 0.9844 - val_loss: 0.9989 - val_accuracy: 0.9719
Epoch 7/30
313/313 [==============================] - 12s 39ms/step - loss: 0.2032 - acc
uracy: 0.9842 - val_loss: 1.1434 - val_accuracy: 0.9739
Epoch 8/30
313/313 [==============================] - 12s 38ms/step - loss: 0.1712 - acc
uracy: 0.9864 - val_loss: 1.0423 - val_accuracy: 0.9729
Epoch 9/30
313/313 [==============================] - 12s 39ms/step - loss: 0.1669 - acc
uracy: 0.9870 - val_loss: 0.6818 - val_accuracy: 0.9719
Epoch 10/30
313/313 [==============================] - 12s 38ms/step - loss: 0.1539 - acc
uracy: 0.9859 - val_loss: 0.8058 - val_accuracy: 0.9729
Epoch 11/30
313/313 [==============================] - 12s 39ms/step - loss: 0.1168 - acc
uracy: 0.9881 - val_loss: 0.7075 - val_accuracy: 0.9790
Epoch 12/30
313/313 [==============================] - 12s 38ms/step - loss: 0.1092 - acc
uracy: 0.9900 - val_loss: 0.7504 - val_accuracy: 0.9770
Epoch 13/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0905 - acc
uracy: 0.9914 - val_loss: 0.9088 - val_accuracy: 0.9719
Epoch 14/30
313/313 [==============================] - 12s 39ms/step - loss: 0.1122 - acc
uracy: 0.9895 - val_loss: 0.5597 - val_accuracy: 0.9760
Epoch 15/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0837 - acc
uracy: 0.9897 - val_loss: 0.9406 - val_accuracy: 0.9709
Epoch 16/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0726 - acc
uracy: 0.9915 - val_loss: 0.9343 - val_accuracy: 0.9669
Epoch 17/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0771 - acc
uracy: 0.9914 - val_loss: 1.0177 - val_accuracy: 0.9689
```

```
Epoch 18/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0694 - acc
uracy: 0.9932 - val_loss: 0.7923 - val_accuracy: 0.9749
Epoch 19/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0497 - acc
uracy: 0.9949 - val_loss: 0.7644 - val_accuracy: 0.9749
Epoch 20/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0601 - acc
uracy: 0.9934 - val_loss: 0.7196 - val_accuracy: 0.9749
Epoch 21/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0532 - acc
uracy: 0.9943 - val_loss: 0.6580 - val_accuracy: 0.9760
Epoch 22/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0531 - acc
uracy: 0.9933 - val_loss: 0.9021 - val_accuracy: 0.9739
Epoch 23/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0663 - acc
uracy: 0.9918 - val_loss: 0.6082 - val_accuracy: 0.9770
Epoch 24/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0533 - acc
uracy: 0.9943 - val_loss: 0.6947 - val_accuracy: 0.9770
Epoch 25/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0404 - acc
uracy: 0.9949 - val_loss: 0.5410 - val_accuracy: 0.9760
Epoch 26/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0479 - acc
uracy: 0.9942 - val_loss: 0.7172 - val_accuracy: 0.9760
Epoch 27/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0333 - acc
uracy: 0.9953 - val_loss: 0.5674 - val_accuracy: 0.9820
Epoch 28/30
313/313 [==============================] - 12s 38ms/step - loss: 0.0354 - acc
uracy: 0.9957 - val_loss: 0.5682 - val_accuracy: 0.9770
Epoch 29/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0309 - acc
uracy: 0.9961 - val_loss: 0.6533 - val_accuracy: 0.9790
Epoch 30/30
313/313 [==============================] - 12s 39ms/step - loss: 0.0487 - acc
uracy: 0.9945 - val_loss: 0.6744 - val_accuracy: 0.9770
```

In [36]:

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

```
32/32 [==============================] - 1s 30ms/step - loss: 0.3557 - accura
cy: 0.9810
Test accuracy: 0.981
```

Summary