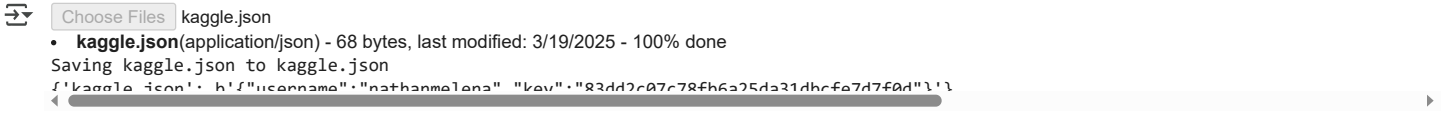
```
from google.colab import files
files.upload()
```




```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle competitions download -c dogs-vs-cats
```

```
!unzip -qq dogs-vs-cats.zip
```

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

Above downloads and unzips all the info we need from Kaggle

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import image_dataset_from_directory
```

Imports needed librarys

```
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)
```

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

```
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (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)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_3 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_4 (Conv2D)	(None, 7, 7, 256)	590,080
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 1)	12,545

Total params: 991,041 (3.78 MB)
Trainable params: 991,041 (3.78 MB)
Non-trainable params: 0 (0.00 B)

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

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

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

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

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

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

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

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

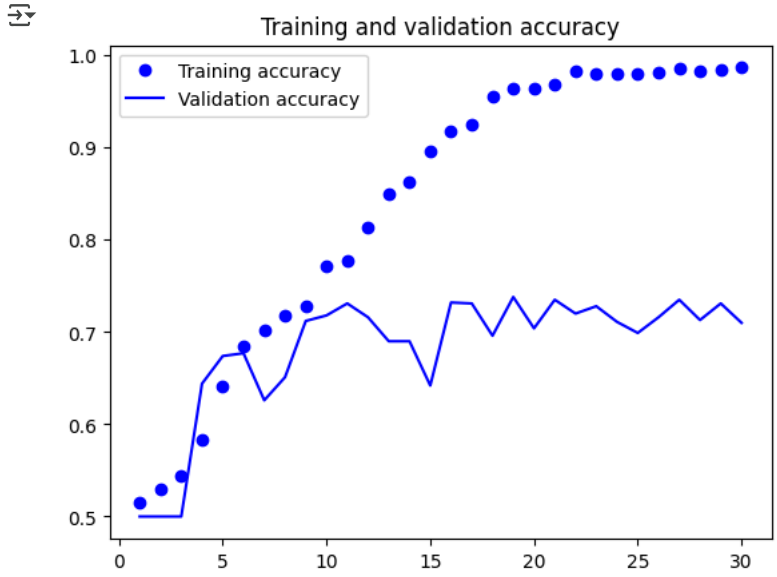
```
(4, 4)
(4, 4)
(4, 4)
```

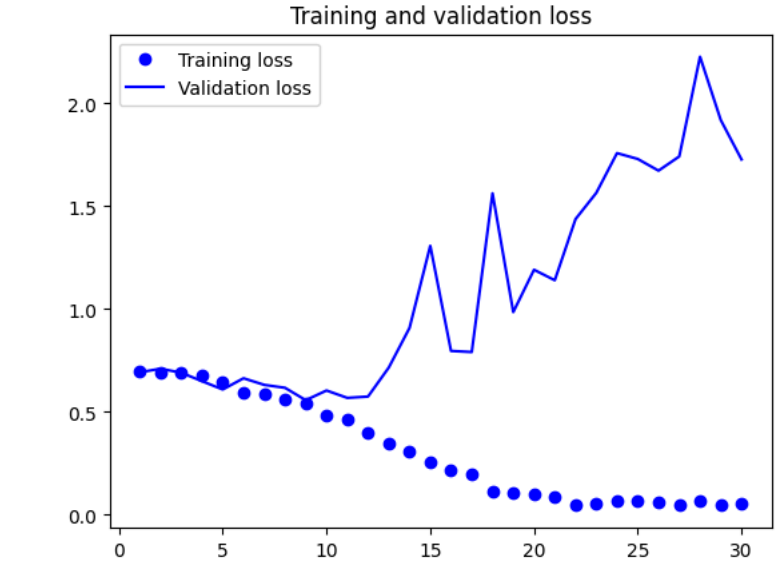
```
callbacks = [
 keras.callbacks.ModelCheckpoint(
 filepath="convnet_from_scratch.keras",
 save_best_only=True,
 monitor="val_loss")
]

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

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 15s 132ms/step - accuracy: 0.5079 - loss: 0.7076 - val_accuracy: 0.5000 - val_loss: 0.6927
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 53ms/step - accuracy: 0.5413 - loss: 0.6921 - val_accuracy: 0.5000 - val_loss: 0.7101
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step - accuracy: 0.5293 - loss: 0.6954 - val_accuracy: 0.5000 - val_loss: 0.6907
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.5681 - loss: 0.6890 - val_accuracy: 0.6440 - val_loss: 0.6496
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - accuracy: 0.6371 - loss: 0.6439 - val_accuracy: 0.6740 - val_loss: 0.6091
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step - accuracy: 0.6694 - loss: 0.6092 - val_accuracy: 0.6770 - val_loss: 0.6637
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 67ms/step - accuracy: 0.7096 - loss: 0.6117 - val_accuracy: 0.6260 - val_loss: 0.6316
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.7133 - loss: 0.5801 - val_accuracy: 0.6510 - val_loss: 0.6173
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.7156 - loss: 0.5496 - val_accuracy: 0.7120 - val_loss: 0.5571
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step - accuracy: 0.7655 - loss: 0.4929 - val_accuracy: 0.7180 - val_loss: 0.6042
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.7653 - loss: 0.4831 - val_accuracy: 0.7310 - val_loss: 0.5683
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.8042 - loss: 0.4177 - val_accuracy: 0.7160 - val_loss: 0.5747
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.8397 - loss: 0.3626 - val_accuracy: 0.6900 - val_loss: 0.7157
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.8505 - loss: 0.3262 - val_accuracy: 0.6900 - val_loss: 0.9085
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 46ms/step - accuracy: 0.8832 - loss: 0.2709 - val_accuracy: 0.6420 - val_loss: 1.3076
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.8968 - loss: 0.2602 - val_accuracy: 0.7320 - val_loss: 0.7965
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.9248 - loss: 0.2030 - val_accuracy: 0.7310 - val_loss: 0.7913
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 53ms/step - accuracy: 0.9598 - loss: 0.1067 - val_accuracy: 0.6960 - val_loss: 1.5629
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.9510 - loss: 0.1481 - val_accuracy: 0.7380 - val_loss: 0.9855
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.9658 - loss: 0.0970 - val_accuracy: 0.7040 - val_loss: 1.1912
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.9766 - loss: 0.0666 - val_accuracy: 0.7350 - val_loss: 1.1403
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 67ms/step - accuracy: 0.9879 - loss: 0.0385 - val_accuracy: 0.7200 - val_loss: 1.4375
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.9757 - loss: 0.0639 - val_accuracy: 0.7280 - val_loss: 1.5649
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.9782 - loss: 0.0725 - val_accuracy: 0.7110 - val_loss: 1.7583
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 81ms/step - accuracy: 0.9821 - loss: 0.0492 - val_accuracy: 0.6990 - val_loss: 1.7299
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.9885 - loss: 0.0345 - val_accuracy: 0.7160 - val_loss: 1.6731
Epoch 27/30
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.9884 - loss: 0.0406 - val_accuracy: 0.7350 - val_loss: 1.7422
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - accuracy: 0.9757 - loss: 0.0990 - val_accuracy: 0.7130 - val_loss: 2.2270
Epoch 29/30
```

```
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()
```






```
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}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.7019 - loss: 0.5825
Test accuracy: 0.705
```

```
rm -rf cats_vs_dogs_small
```

```
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)
```

```
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.
```

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

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

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

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

```
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)
```

```
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)
```

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


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")
```

```
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"])


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

```
Epoch 1/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 0.5166 - loss: 0.8083 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 2/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 91ms/step - accuracy: 0.5236 - loss: 0.6961 - val_accuracy: 0.5000 - val_loss: 0.6921
Epoch 3/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 51ms/step - accuracy: 0.5168 - loss: 0.6919 - val_accuracy: 0.5090 - val_loss: 0.6908
Epoch 4/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 72ms/step - accuracy: 0.5507 - loss: 0.7042 - val_accuracy: 0.5600 - val_loss: 0.6843
Epoch 5/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.5553 - loss: 0.6907 - val_accuracy: 0.6120 - val_loss: 0.6510
Epoch 6/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.5922 - loss: 0.6972 - val_accuracy: 0.5910 - val_loss: 0.6983
Epoch 7/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 82ms/step - accuracy: 0.6305 - loss: 0.6497 - val_accuracy: 0.6140 - val_loss: 0.6421
Epoch 8/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 56ms/step - accuracy: 0.6543 - loss: 0.6358 - val_accuracy: 0.5790 - val_loss: 0.7167
Epoch 9/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.6619 - loss: 0.6169 - val_accuracy: 0.5610 - val_loss: 0.6832
Epoch 10/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.6871 - loss: 0.6084 - val_accuracy: 0.6420 - val_loss: 0.6142
Epoch 11/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 60ms/step - accuracy: 0.6946 - loss: 0.5932 - val_accuracy: 0.6730 - val_loss: 0.6094
Epoch 12/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 85ms/step - accuracy: 0.7123 - loss: 0.5746 - val_accuracy: 0.7120 - val_loss: 0.5488
Epoch 13/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.7059 - loss: 0.5969 - val_accuracy: 0.7010 - val_loss: 0.5718
Epoch 14/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 92ms/step - accuracy: 0.7327 - loss: 0.5542 - val_accuracy: 0.7120 - val_loss: 0.5521
Epoch 15/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 57ms/step - accuracy: 0.7384 - loss: 0.5295 - val_accuracy: 0.7390 - val_loss: 0.5278
Epoch 16/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 81ms/step - accuracy: 0.7443 - loss: 0.5427 - val_accuracy: 0.7000 - val_loss: 0.5705
Epoch 17/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.7486 - loss: 0.5100 - val_accuracy: 0.6990 - val_loss: 0.6090
Epoch 18/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 70ms/step - accuracy: 0.7476 - loss: 0.5225 - val_accuracy: 0.7250 - val_loss: 0.5196
Epoch 19/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.7621 - loss: 0.5152 - val_accuracy: 0.7130 - val_loss: 0.5542
Epoch 20/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 75ms/step - accuracy: 0.7602 - loss: 0.5137 - val_accuracy: 0.7510 - val_loss: 0.5032
Epoch 21/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.7846 - loss: 0.4861 - val_accuracy: 0.7450 - val_loss: 0.5075
Epoch 22/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.7555 - loss: 0.5041 - val_accuracy: 0.7250 - val_loss: 0.5598
Epoch 23/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 83ms/step - accuracy: 0.7790 - loss: 0.4886 - val_accuracy: 0.7550 - val_loss: 0.5025
Epoch 24/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 50ms/step - accuracy: 0.7800 - loss: 0.4693 - val_accuracy: 0.7320 - val_loss: 0.5607
Epoch 25/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 83ms/step - accuracy: 0.7917 - loss: 0.4768 - val_accuracy: 0.7630 - val_loss: 0.4910
Epoch 26/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - accuracy: 0.8103 - loss: 0.4510 - val_accuracy: 0.7670 - val_loss: 0.4892
Epoch 27/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 59ms/step - accuracy: 0.8028 - loss: 0.4421 - val_accuracy: 0.7520 - val_loss: 0.5207
Epoch 28/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - accuracy: 0.8071 - loss: 0.4190 - val_accuracy: 0.7560 - val_loss: 0.5099
Epoch 29/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.8044 - loss: 0.4139 - val_accuracy: 0.7830 - val_loss: 0.4369
```

```
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}")
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.8384 - loss: 0.4777
Test accuracy: 0.840
```

above is for question 1

Below is for question 4 redoing step 1

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

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_38 (InputLayer)	(None, 180, 180, 3)	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1,792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36,928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73,856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147,584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295,168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590,080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590,080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

Total params: 14,714,688 (56.13 MB)
Trainable params: 14,714,688 (56.13 MB)
Non-trainable params: 0 (0.00 B)

```
import numpy as np

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

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 8s 8s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 148ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 234ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 161ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 175ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
```

```
train_features.shape
```

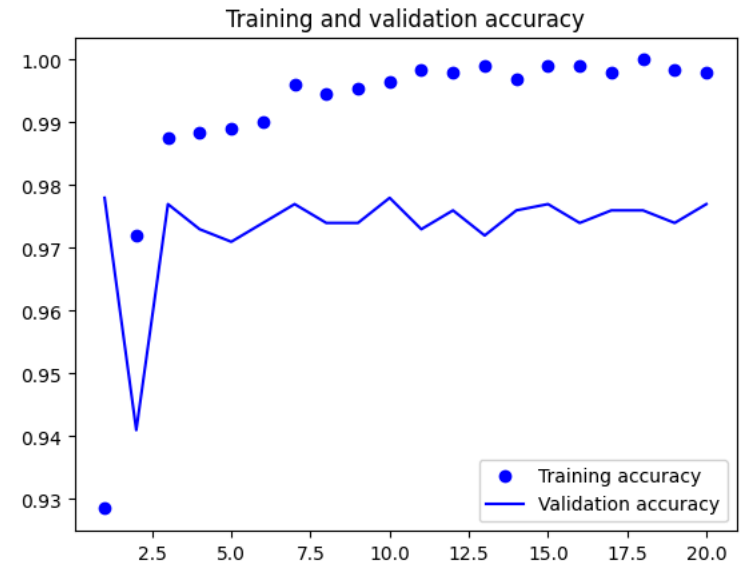
```
(2000, 5, 5, 512)
```

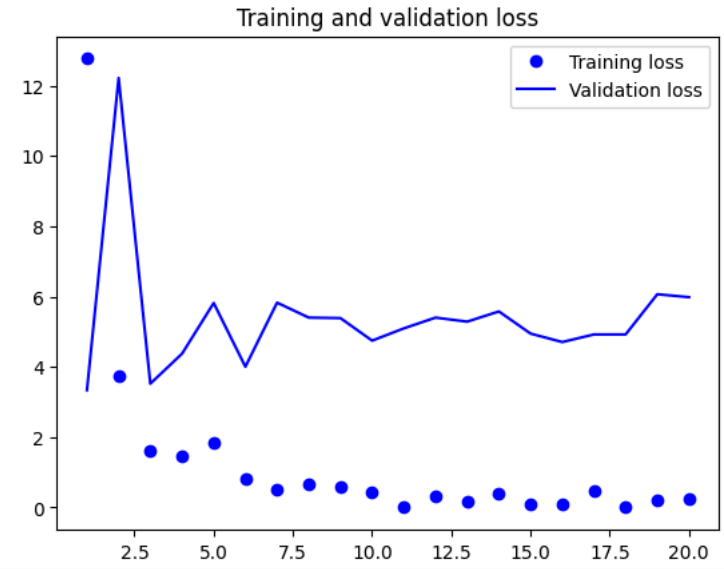
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              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 ━━━━━━━━━━━━━━━━━━━━ 4s 33ms/step - accuracy: 0.8626 - loss: 22.5310 - val_accuracy: 0.9780 - val_loss: 3.3374
Epoch 2/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9708 - loss: 3.5702 - val_accuracy: 0.9410 - val_loss: 12.2140
Epoch 3/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9885 - loss: 1.8692 - val_accuracy: 0.9770 - val_loss: 3.5224
Epoch 4/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9931 - loss: 1.0189 - val_accuracy: 0.9730 - val_loss: 4.3735
Epoch 5/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9911 - loss: 1.6565 - val_accuracy: 0.9710 - val_loss: 5.8167
Epoch 6/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9882 - loss: 0.8334 - val_accuracy: 0.9740 - val_loss: 4.0038
Epoch 7/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9953 - loss: 0.5813 - val_accuracy: 0.9770 - val_loss: 5.8267
Epoch 8/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9944 - loss: 0.7347 - val_accuracy: 0.9740 - val_loss: 5.4006
Epoch 9/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9959 - loss: 0.4938 - val_accuracy: 0.9740 - val_loss: 5.3867
Epoch 10/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9986 - loss: 0.1199 - val_accuracy: 0.9780 - val_loss: 4.7462
Epoch 11/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9985 - loss: 0.0170 - val_accuracy: 0.9730 - val_loss: 5.0943
Epoch 12/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9991 - loss: 0.1335 - val_accuracy: 0.9760 - val_loss: 5.4006
Epoch 13/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9997 - loss: 0.0488 - val_accuracy: 0.9720 - val_loss: 5.2862
Epoch 14/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9981 - loss: 0.2314 - val_accuracy: 0.9760 - val_loss: 5.5767
Epoch 15/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9976 - loss: 0.2392 - val_accuracy: 0.9770 - val_loss: 4.9462
Epoch 16/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9997 - loss: 0.0131 - val_accuracy: 0.9740 - val_loss: 4.7058
Epoch 17/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9986 - loss: 0.3227 - val_accuracy: 0.9760 - val_loss: 4.9224
Epoch 18/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 3.1596e-28 - val_accuracy: 0.9760 - val_loss: 4.9224
Epoch 19/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.1722 - val_accuracy: 0.9740 - val_loss: 6.0658
Epoch 20/20
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9957 - loss: 0.3179 - val_accuracy: 0.9770 - val_loss: 5.9820
```

```
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()
```






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

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

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

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

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

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

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
```

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


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 16/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 172ms/step - accuracy: 0.9816 - loss: 1.1352 - val_accuracy: 0.9750 - val_loss: 2.5306
Epoch 17/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 135ms/step - accuracy: 0.9878 - loss: 0.7203 - val_accuracy: 0.9770 - val_loss: 3.1745
Epoch 18/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 138ms/step - accuracy: 0.9845 - loss: 0.6789 - val_accuracy: 0.9820 - val_loss: 1.9322
Epoch 19/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 138ms/step - accuracy: 0.9865 - loss: 1.2751 - val_accuracy: 0.9760 - val_loss: 2.3938
Epoch 20/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 176ms/step - accuracy: 0.9821 - loss: 1.0337 - val_accuracy: 0.9800 - val_loss: 2.6144
Epoch 21/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 173ms/step - accuracy: 0.9871 - loss: 0.9049 - val_accuracy: 0.9770 - val_loss: 2.7922
Epoch 22/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 173ms/step - accuracy: 0.9831 - loss: 0.9627 - val_accuracy: 0.9770 - val_loss: 2.4767
Epoch 23/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 174ms/step - accuracy: 0.9817 - loss: 1.1605 - val_accuracy: 0.9740 - val_loss: 2.4791
Epoch 24/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 174ms/step - accuracy: 0.9812 - loss: 1.1367 - val_accuracy: 0.9850 - val_loss: 1.5766
Epoch 25/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 145ms/step - accuracy: 0.9790 - loss: 1.5253 - val_accuracy: 0.9810 - val_loss: 1.3721
Epoch 26/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - accuracy: 0.9861 - loss: 0.6116 - val_accuracy: 0.9750 - val_loss: 1.8961
Epoch 27/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 140ms/step - accuracy: 0.9866 - loss: 0.6450 - val_accuracy: 0.9770 - val_loss: 2.1270
Epoch 28/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 176ms/step - accuracy: 0.9850 - loss: 0.7699 - val_accuracy: 0.9790 - val_loss: 2.1379
Epoch 29/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 134ms/step - accuracy: 0.9849 - loss: 0.6594 - val_accuracy: 0.9800 - val_loss: 1.5682
Epoch 30/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 135ms/step - accuracy: 0.9918 - loss: 0.4199 - val_accuracy: 0.9790 - val_loss: 1.9112
Epoch 31/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 137ms/step - accuracy: 0.9875 - loss: 0.6735 - val_accuracy: 0.9800 - val_loss: 2.1212
Epoch 32/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - accuracy: 0.9899 - loss: 0.5115 - val_accuracy: 0.9810 - val_loss: 1.9078
Epoch 33/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 176ms/step - accuracy: 0.9836 - loss: 0.7320 - val_accuracy: 0.9770 - val_loss: 1.8529
Epoch 34/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 135ms/step - accuracy: 0.9910 - loss: 0.3938 - val_accuracy: 0.9790 - val_loss: 1.7992
Epoch 35/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 137ms/step - accuracy: 0.9913 - loss: 0.2530 - val_accuracy: 0.9790 - val_loss: 2.0142
Epoch 36/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 138ms/step - accuracy: 0.9845 - loss: 0.9509 - val_accuracy: 0.9760 - val_loss: 2.1196
Epoch 37/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 139ms/step - accuracy: 0.9889 - loss: 0.3643 - val_accuracy: 0.9720 - val_loss: 3.2446
Epoch 38/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 176ms/step - accuracy: 0.9812 - loss: 0.9601 - val_accuracy: 0.9780 - val_loss: 2.4012
Epoch 39/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 135ms/step - accuracy: 0.9895 - loss: 0.6623 - val_accuracy: 0.9770 - val_loss: 2.2018
Epoch 40/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 136ms/step - accuracy: 0.9907 - loss: 0.3499 - val_accuracy: 0.9780 - val_loss: 2.3506
Epoch 41/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 138ms/step - accuracy: 0.9895 - loss: 0.5191 - val_accuracy: 0.9740 - val_loss: 2.3173
Epoch 42/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 176ms/step - accuracy: 0.9914 - loss: 0.2876 - val_accuracy: 0.9790 - val_loss: 2.1934
Epoch 43/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 174ms/step - accuracy: 0.9906 - loss: 0.4793 - val_accuracy: 0.9750 - val_loss: 2.8926
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 174ms/step - accuracy: 0.9863 - loss: 1.2494 - val_accuracy: 0.9750 - val_loss: 2.5237
Epoch 45/50
```

```
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 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - accuracy: 0.9803 - loss: 1.9460
Test accuracy: 0.981
```

Above is for question for step 1

```
rm -rf cats_vs_dogs_small
```

```
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=1500)
make_subset("validation", start_index=1500, end_index=2000)
make_subset("test", start_index=2000, end_index=2500)
```

```
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 3000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

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

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

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

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

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

```
    if i >= 2:
        break
```

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

```
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)


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


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")
```

```
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"])


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation_1500.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 7s 52ms/step - accuracy: 0.5230 - loss: 0.7030 - val_accuracy: 0.6320 - val_loss: 0.6861
Epoch 2/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 0.5558 - loss: 0.6920 - val_accuracy: 0.6130 - val_loss: 0.6658
Epoch 3/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.6178 - loss: 0.6687 - val_accuracy: 0.6160 - val_loss: 0.6541
Epoch 4/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - accuracy: 0.6364 - loss: 0.6321 - val_accuracy: 0.5180 - val_loss: 0.8655
Epoch 5/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 58ms/step - accuracy: 0.6582 - loss: 0.6372 - val_accuracy: 0.6640 - val_loss: 0.6053
Epoch 6/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 10s 58ms/step - accuracy: 0.6654 - loss: 0.6029 - val_accuracy: 0.6570 - val_loss: 0.6267
Epoch 7/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 63ms/step - accuracy: 0.6777 - loss: 0.5823 - val_accuracy: 0.6370 - val_loss: 0.6508
Epoch 8/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 10s 57ms/step - accuracy: 0.7043 - loss: 0.5722 - val_accuracy: 0.6190 - val_loss: 1.0304
Epoch 9/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 45ms/step - accuracy: 0.6972 - loss: 0.6025 - val_accuracy: 0.7060 - val_loss: 0.5757
Epoch 10/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 7s 61ms/step - accuracy: 0.7203 - loss: 0.5403 - val_accuracy: 0.6250 - val_loss: 0.7044
Epoch 11/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 11s 67ms/step - accuracy: 0.7220 - loss: 0.5397 - val_accuracy: 0.7260 - val_loss: 0.5489
Epoch 12/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 56ms/step - accuracy: 0.7470 - loss: 0.5089 - val_accuracy: 0.6970 - val_loss: 0.6230
Epoch 13/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 11s 63ms/step - accuracy: 0.7440 - loss: 0.5085 - val_accuracy: 0.7300 - val_loss: 0.5256
Epoch 14/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 4s 46ms/step - accuracy: 0.7607 - loss: 0.4959 - val_accuracy: 0.7500 - val_loss: 0.5039
Epoch 15/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.7794 - loss: 0.4727 - val_accuracy: 0.7540 - val_loss: 0.5338
Epoch 16/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 55ms/step - accuracy: 0.7795 - loss: 0.4725 - val_accuracy: 0.7490 - val_loss: 0.5639
Epoch 17/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - accuracy: 0.7685 - loss: 0.4660 - val_accuracy: 0.7540 - val_loss: 0.5100
Epoch 18/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 9s 49ms/step - accuracy: 0.7826 - loss: 0.4689 - val_accuracy: 0.7240 - val_loss: 0.5922
Epoch 19/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - accuracy: 0.7966 - loss: 0.4424 - val_accuracy: 0.7590 - val_loss: 0.5484
Epoch 20/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.8059 - loss: 0.4327 - val_accuracy: 0.7780 - val_loss: 0.5386
Epoch 21/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 12s 77ms/step - accuracy: 0.8189 - loss: 0.4062 - val_accuracy: 0.7840 - val_loss: 0.4637
Epoch 22/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 8s 50ms/step - accuracy: 0.8194 - loss: 0.3970 - val_accuracy: 0.7780 - val_loss: 0.4873
Epoch 23/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 57ms/step - accuracy: 0.8256 - loss: 0.3941 - val_accuracy: 0.7960 - val_loss: 0.4799
Epoch 24/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 11s 66ms/step - accuracy: 0.8187 - loss: 0.3864 - val_accuracy: 0.7950 - val_loss: 0.5042
Epoch 25/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.8273 - loss: 0.3803 - val_accuracy: 0.8180 - val_loss: 0.4255
Epoch 26/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 68ms/step - accuracy: 0.8444 - loss: 0.3545 - val_accuracy: 0.8180 - val_loss: 0.4734
```

```
Epoch 27/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 8s 45ms/step - accuracy: 0.8298 - loss: 0.3652 - val_accuracy: 0.7670 - val_loss: 0.5578
Epoch 28/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 0.8444 - loss: 0.3424 - val_accuracy: 0.8200 - val_loss: 0.4158
Epoch 29/100
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.8582 - loss: 0.3316 - val_accuracy: 0.8220 - val_loss: 0.4518
```

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation_1500.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.8944 - loss: 0.2659
Test accuracy: 0.886
```

Above is for question 2

Below is for question 4 step 2

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

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_45 (InputLayer)	(None, 180, 180, 3)	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1,792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36,928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73,856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147,584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295,168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590,080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590,080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

Total params: 14,714,688 (56.13 MB)
Trainable params: 14,714,688 (56.13 MB)
Non-trainable params: 0 (0.00 B)

```
import numpy as np

def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
```

```
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 636ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 173ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 156ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
```

```
train_features.shape
```

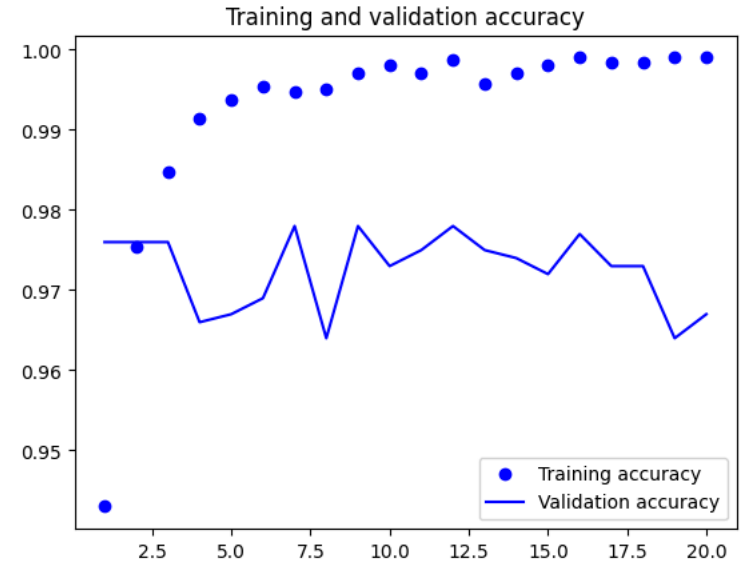
```
(3000, 5, 5, 512)
```

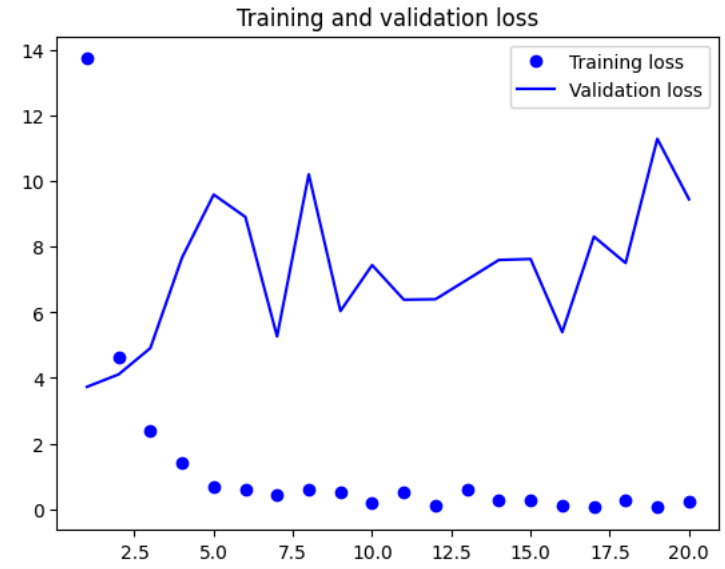
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
```

```
              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
94/94 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9021 - loss: 30.1099 - val_accuracy: 0.9760 - val_loss: 3.7281
Epoch 2/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9726 - loss: 4.4222 - val_accuracy: 0.9760 - val_loss: 4.1090
Epoch 3/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9860 - loss: 2.1425 - val_accuracy: 0.9760 - val_loss: 4.9099
Epoch 4/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9917 - loss: 1.4599 - val_accuracy: 0.9660 - val_loss: 7.6702
Epoch 5/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9941 - loss: 0.5411 - val_accuracy: 0.9670 - val_loss: 9.5961
Epoch 6/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9958 - loss: 0.5123 - val_accuracy: 0.9690 - val_loss: 8.9138
Epoch 7/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9944 - loss: 0.4816 - val_accuracy: 0.9780 - val_loss: 5.2673
Epoch 8/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9946 - loss: 0.8522 - val_accuracy: 0.9640 - val_loss: 10.2143
Epoch 9/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9978 - loss: 0.3367 - val_accuracy: 0.9780 - val_loss: 6.0425
Epoch 10/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9970 - loss: 0.2150 - val_accuracy: 0.9730 - val_loss: 7.4443
Epoch 11/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9964 - loss: 0.6876 - val_accuracy: 0.9750 - val_loss: 6.3891
Epoch 12/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9995 - loss: 0.0451 - val_accuracy: 0.9780 - val_loss: 6.4026
Epoch 13/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9949 - loss: 0.8662 - val_accuracy: 0.9750 - val_loss: 7.0005
Epoch 14/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9973 - loss: 0.2533 - val_accuracy: 0.9740 - val_loss: 7.6012
Epoch 15/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9990 - loss: 0.1189 - val_accuracy: 0.9720 - val_loss: 7.6312
Epoch 16/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9991 - loss: 0.1419 - val_accuracy: 0.9770 - val_loss: 5.4000
Epoch 17/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9988 - loss: 0.0309 - val_accuracy: 0.9730 - val_loss: 8.3138
Epoch 18/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9972 - loss: 0.2967 - val_accuracy: 0.9730 - val_loss: 7.5081
Epoch 19/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9995 - loss: 0.0227 - val_accuracy: 0.9640 - val_loss: 11.2991
Epoch 20/20
94/94 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9980 - loss: 0.4956 - val_accuracy: 0.9670 - val_loss: 9.4539
```

```
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()
```






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

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

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

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

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

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

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
```

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


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=50,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 15s 143ms/step - accuracy: 0.8534 - loss: 44.6440 - val_accuracy: 0.9750 - val_loss: 4.6916
Epoch 2/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 14s 152ms/step - accuracy: 0.9493 - loss: 5.6058 - val_accuracy: 0.9770 - val_loss: 3.8332
Epoch 3/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 14s 147ms/step - accuracy: 0.9582 - loss: 4.6335 - val_accuracy: 0.9720 - val_loss: 4.9421
Epoch 4/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 12s 130ms/step - accuracy: 0.9569 - loss: 5.9197 - val_accuracy: 0.9730 - val_loss: 3.7860
Epoch 5/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 20s 123ms/step - accuracy: 0.9746 - loss: 2.7633 - val_accuracy: 0.9750 - val_loss: 3.9176
Epoch 6/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 14s 148ms/step - accuracy: 0.9684 - loss: 3.3787 - val_accuracy: 0.9710 - val_loss: 5.1082
Epoch 7/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 20s 147ms/step - accuracy: 0.9648 - loss: 3.0932 - val_accuracy: 0.9740 - val_loss: 3.8209
Epoch 8/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 21s 149ms/step - accuracy: 0.9732 - loss: 2.8678 - val_accuracy: 0.9640 - val_loss: 4.5819
Epoch 9/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 21s 153ms/step - accuracy: 0.9650 - loss: 2.7495 - val_accuracy: 0.9780 - val_loss: 2.4742
Epoch 10/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 20s 153ms/step - accuracy: 0.9771 - loss: 1.3847 - val_accuracy: 0.9760 - val_loss: 2.3005
Epoch 11/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 18s 125ms/step - accuracy: 0.9825 - loss: 1.4519 - val_accuracy: 0.9710 - val_loss: 2.3285
Epoch 12/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 23s 152ms/step - accuracy: 0.9774 - loss: 1.6300 - val_accuracy: 0.9770 - val_loss: 2.1239
Epoch 13/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 12s 130ms/step - accuracy: 0.9767 - loss: 1.3600 - val_accuracy: 0.9820 - val_loss: 1.6313
Epoch 14/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 14s 149ms/step - accuracy: 0.9799 - loss: 1.2234 - val_accuracy: 0.9750 - val_loss: 1.8780
Epoch 15/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 21s 152ms/step - accuracy: 0.9795 - loss: 1.3453 - val_accuracy: 0.9760 - val_loss: 1.4883
Epoch 16/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 20s 147ms/step - accuracy: 0.9713 - loss: 1.3756 - val_accuracy: 0.9760 - val_loss: 1.8498
Epoch 17/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 14s 150ms/step - accuracy: 0.9757 - loss: 1.1574 - val_accuracy: 0.9760 - val_loss: 1.5798
Epoch 18/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 14s 154ms/step - accuracy: 0.9755 - loss: 1.1684 - val_accuracy: 0.9780 - val_loss: 0.9427
Epoch 19/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 12s 123ms/step - accuracy: 0.9800 - loss: 0.7703 - val_accuracy: 0.9830 - val_loss: 1.0001
Epoch 20/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 15s 155ms/step - accuracy: 0.9803 - loss: 0.6352 - val_accuracy: 0.9840 - val_loss: 0.7855
Epoch 21/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 17s 121ms/step - accuracy: 0.9879 - loss: 0.6512 - val_accuracy: 0.9810 - val_loss: 0.8701
Epoch 22/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 21s 122ms/step - accuracy: 0.9829 - loss: 0.6089 - val_accuracy: 0.9790 - val_loss: 0.9867
Epoch 23/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 23s 149ms/step - accuracy: 0.9825 - loss: 0.5542 - val_accuracy: 0.9830 - val_loss: 0.9701
Epoch 24/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 18s 124ms/step - accuracy: 0.9865 - loss: 0.4402 - val_accuracy: 0.9830 - val_loss: 0.9258
Epoch 25/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 23s 149ms/step - accuracy: 0.9797 - loss: 0.6937 - val_accuracy: 0.9810 - val_loss: 0.9715
Epoch 26/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 18s 123ms/step - accuracy: 0.9840 - loss: 0.5994 - val_accuracy: 0.9820 - val_loss: 0.8538
Epoch 27/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 21s 129ms/step - accuracy: 0.9819 - loss: 0.6778 - val_accuracy: 0.9800 - val_loss: 0.6222
Epoch 28/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 12s 124ms/step - accuracy: 0.9867 - loss: 0.3861 - val_accuracy: 0.9820 - val_loss: 0.7969
Epoch 29/50
94/94 ━━━━━━━━━━━━━━━━━━━━ 23s 148ms/step - accuracy: 0.9885 - loss: 0.3448 - val_accuracy: 0.9730 - val_loss: 1.2507
```

```
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 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.9762 - loss: 1.0900
Test accuracy: 0.975
```

Double-click (or enter) to edit

```
rm -rf cats_vs_dogs_small
```

```
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=2000)
make_subset("validation", start_index=2000, end_index=2500)
make_subset("test", start_index=2500, end_index=3000)
```

```
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 4000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

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

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

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

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

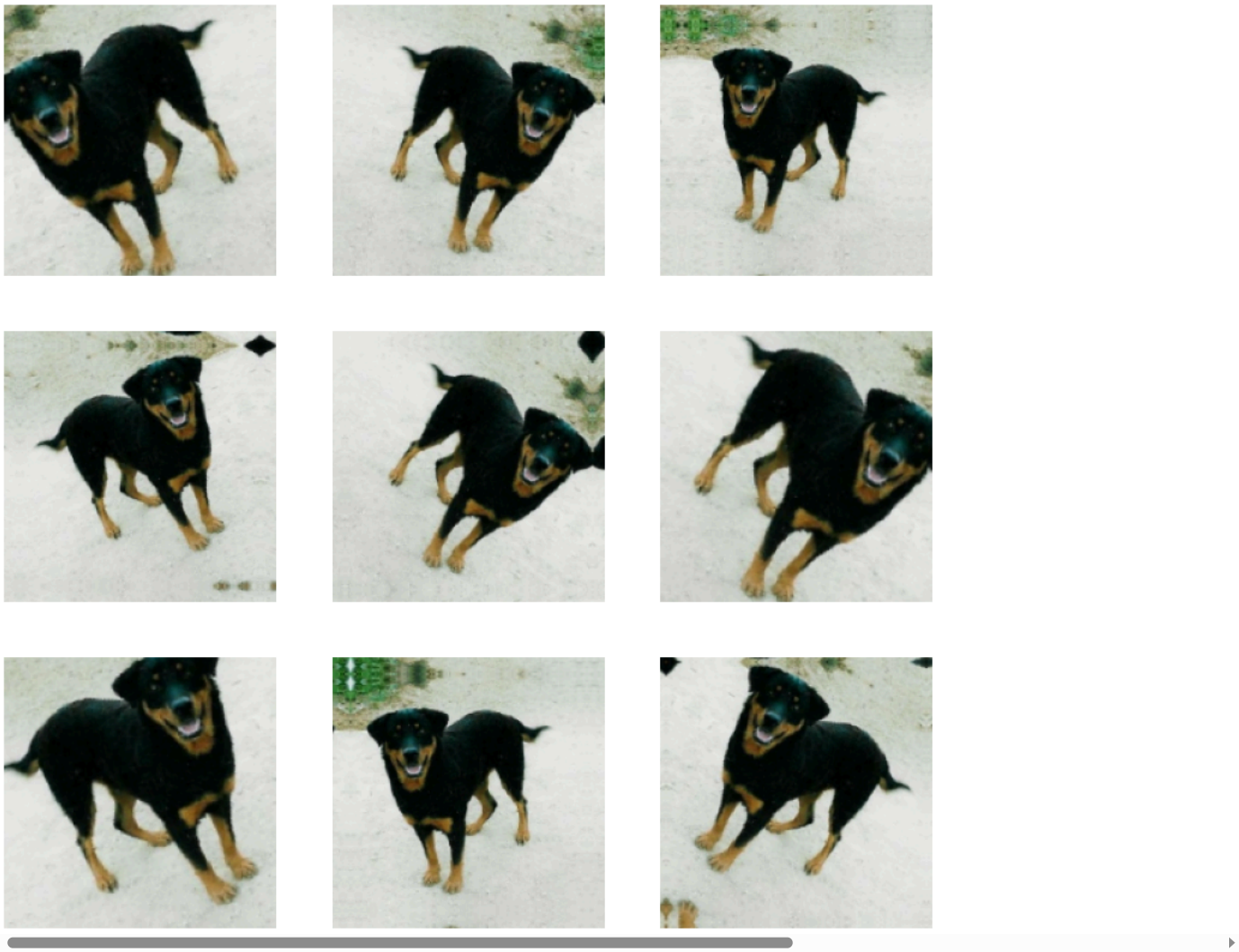
```
reshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))
for i, element in enumerate(reshaped_dataset):
    print(element.shape)
    if i >= 2:
        break
```

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

```
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)
```

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

```
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")
```



```
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"])


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation_2000.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 8s 45ms/step - accuracy: 0.5236 - loss: 0.7030 - val_accuracy: 0.5000 - val_loss: 0.6930
Epoch 2/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - accuracy: 0.5146 - loss: 0.6909 - val_accuracy: 0.5010 - val_loss: 0.7732
Epoch 3/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 42ms/step - accuracy: 0.5789 - loss: 0.6812 - val_accuracy: 0.5910 - val_loss: 0.6551
Epoch 4/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 46ms/step - accuracy: 0.6224 - loss: 0.6514 - val_accuracy: 0.6210 - val_loss: 0.6382
Epoch 5/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - accuracy: 0.6346 - loss: 0.6479 - val_accuracy: 0.6520 - val_loss: 0.6068
Epoch 6/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 52ms/step - accuracy: 0.6771 - loss: 0.6328 - val_accuracy: 0.6510 - val_loss: 0.6107
Epoch 7/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 10s 48ms/step - accuracy: 0.6837 - loss: 0.5850 - val_accuracy: 0.7080 - val_loss: 0.5693
Epoch 8/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 42ms/step - accuracy: 0.6959 - loss: 0.5782 - val_accuracy: 0.7250 - val_loss: 0.5438
Epoch 9/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - accuracy: 0.7162 - loss: 0.5558 - val_accuracy: 0.7450 - val_loss: 0.5318
Epoch 10/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 48ms/step - accuracy: 0.7283 - loss: 0.5508 - val_accuracy: 0.7350 - val_loss: 0.5194
Epoch 11/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 42ms/step - accuracy: 0.7353 - loss: 0.5256 - val_accuracy: 0.7370 - val_loss: 0.5014
Epoch 12/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 54ms/step - accuracy: 0.7604 - loss: 0.4995 - val_accuracy: 0.7520 - val_loss: 0.5259
Epoch 13/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 49ms/step - accuracy: 0.7603 - loss: 0.5016 - val_accuracy: 0.7830 - val_loss: 0.4672
Epoch 14/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 60ms/step - accuracy: 0.7664 - loss: 0.4720 - val_accuracy: 0.7060 - val_loss: 0.5595
Epoch 15/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 42ms/step - accuracy: 0.7866 - loss: 0.4674 - val_accuracy: 0.7180 - val_loss: 0.5682
Epoch 16/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 10s 43ms/step - accuracy: 0.7754 - loss: 0.4551 - val_accuracy: 0.7850 - val_loss: 0.4373
Epoch 17/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - accuracy: 0.7898 - loss: 0.4468 - val_accuracy: 0.7910 - val_loss: 0.4655
Epoch 18/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 42ms/step - accuracy: 0.8020 - loss: 0.4249 - val_accuracy: 0.8220 - val_loss: 0.4038
Epoch 19/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - accuracy: 0.8136 - loss: 0.4218 - val_accuracy: 0.7860 - val_loss: 0.4535
Epoch 20/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 42ms/step - accuracy: 0.8115 - loss: 0.4042 - val_accuracy: 0.8000 - val_loss: 0.4252
Epoch 21/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 52ms/step - accuracy: 0.8312 - loss: 0.4037 - val_accuracy: 0.8250 - val_loss: 0.3788
Epoch 22/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 42ms/step - accuracy: 0.8316 - loss: 0.3816 - val_accuracy: 0.8180 - val_loss: 0.3916
Epoch 23/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 11s 47ms/step - accuracy: 0.8358 - loss: 0.3839 - val_accuracy: 0.8320 - val_loss: 0.3736
Epoch 24/100
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 55ms/step - accuracy: 0.8379 - loss: 0.3708 - val_accuracy: 0.8230 - val_loss: 0.4020
Epoch 25/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 46ms/step - accuracy: 0.8432 - loss: 0.3637 - val_accuracy: 0.8370 - val_loss: 0.3692
Epoch 26/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 56ms/step - accuracy: 0.8466 - loss: 0.3553 - val_accuracy: 0.8310 - val_loss: 0.3876
Epoch 27/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 45ms/step - accuracy: 0.8390 - loss: 0.3602 - val_accuracy: 0.8220 - val_loss: 0.3963
Epoch 28/100
125/125 ━━━━━━━━━━━━━━━━━━━━ 10s 42ms/step - accuracy: 0.8643 - loss: 0.3221 - val_accuracy: 0.8340 - val_loss: 0.3735
Epoch 29/100
```

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation_2000.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.8955 - loss: 0.3354
Test accuracy: 0.895
```

Above is for question 3

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

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_52 (InputLayer)	(None, 180, 180, 3)	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1,792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36,928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73,856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147,584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295,168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590,080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590,080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

Total params: 14,714,688 (56.13 MB)
Trainable params: 14,714,688 (56.13 MB)
Non-trainable params: 0 (0.00 B)

```
import numpy as np

def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
```

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

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 209ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 148ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 149ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 164ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 150ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 152ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 157ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 154ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 155ms/step
```

```
train_features.shape
```

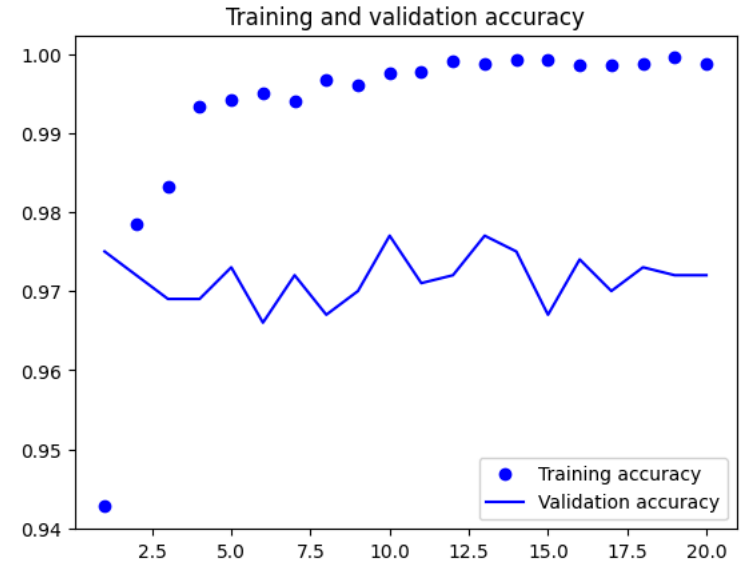
```
(4000, 5, 5, 512)
```

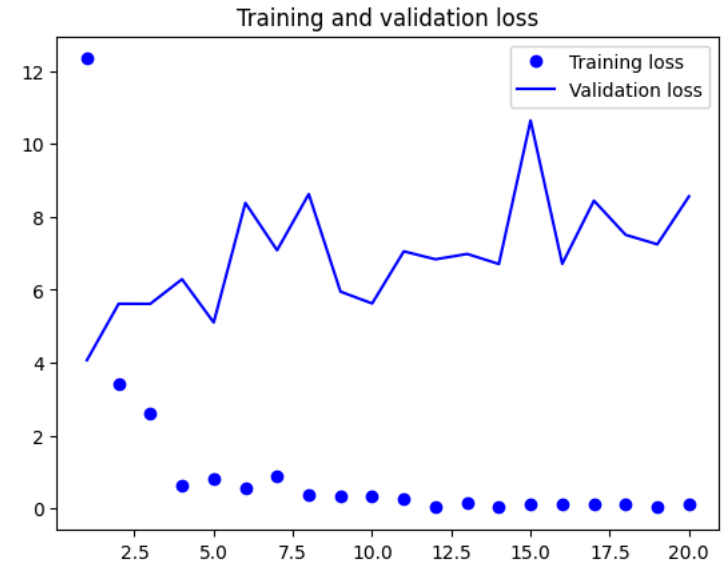
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
```

```
              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
125/125 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.9007 - loss: 21.4446 - val_accuracy: 0.9750 - val_loss: 4.0685
Epoch 2/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9795 - loss: 3.4604 - val_accuracy: 0.9720 - val_loss: 5.6135
Epoch 3/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9813 - loss: 2.2920 - val_accuracy: 0.9690 - val_loss: 5.6127
Epoch 4/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9941 - loss: 0.6511 - val_accuracy: 0.9690 - val_loss: 6.2866
Epoch 5/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9947 - loss: 0.5955 - val_accuracy: 0.9730 - val_loss: 5.1012
Epoch 6/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9944 - loss: 0.7611 - val_accuracy: 0.9660 - val_loss: 8.3788
Epoch 7/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9963 - loss: 0.4830 - val_accuracy: 0.9720 - val_loss: 7.0809
Epoch 8/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9969 - loss: 0.2037 - val_accuracy: 0.9670 - val_loss: 8.6212
Epoch 9/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9980 - loss: 0.1177 - val_accuracy: 0.9700 - val_loss: 5.9478
Epoch 10/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9989 - loss: 0.1348 - val_accuracy: 0.9770 - val_loss: 5.6219
Epoch 11/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9979 - loss: 0.2081 - val_accuracy: 0.9710 - val_loss: 7.0508
Epoch 12/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9995 - loss: 0.0371 - val_accuracy: 0.9720 - val_loss: 6.8327
Epoch 13/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9985 - loss: 0.1497 - val_accuracy: 0.9770 - val_loss: 6.9797
Epoch 14/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9986 - loss: 0.1305 - val_accuracy: 0.9750 - val_loss: 6.7042
Epoch 15/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9999 - loss: 0.0082 - val_accuracy: 0.9670 - val_loss: 10.6375
Epoch 16/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9974 - loss: 0.1760 - val_accuracy: 0.9740 - val_loss: 6.7086
Epoch 17/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9990 - loss: 0.1087 - val_accuracy: 0.9700 - val_loss: 8.4397
Epoch 18/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9981 - loss: 0.0778 - val_accuracy: 0.9730 - val_loss: 7.5034
Epoch 19/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9996 - loss: 0.0290 - val_accuracy: 0.9720 - val_loss: 7.2443
Epoch 20/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9993 - loss: 0.0871 - val_accuracy: 0.9720 - val_loss: 8.5591
```

```
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()
```






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

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

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

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

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

```
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",


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
125/125 ━━━━━━━━━━━━━━━━━━━━ 20s 141ms/step - accuracy: 0.8581 - loss: 25.7548 - val_accuracy: 0.9730 - val_loss: 3.3680
Epoch 2/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 17s 136ms/step - accuracy: 0.9462 - loss: 5.6874 - val_accuracy: 0.9760 - val_loss: 3.5800
Epoch 3/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 19s 121ms/step - accuracy: 0.9655 - loss: 3.3855 - val_accuracy: 0.9810 - val_loss: 3.1159
Epoch 4/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 22s 137ms/step - accuracy: 0.9696 - loss: 3.4288 - val_accuracy: 0.9780 - val_loss: 2.7733
Epoch 5/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 15s 119ms/step - accuracy: 0.9641 - loss: 3.0986 - val_accuracy: 0.9790 - val_loss: 2.6326
Epoch 6/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 14s 115ms/step - accuracy: 0.9709 - loss: 2.8627 - val_accuracy: 0.9670 - val_loss: 4.5874
Epoch 7/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 14s 115ms/step - accuracy: 0.9711 - loss: 1.8790 - val_accuracy: 0.9740 - val_loss: 2.6995
Epoch 8/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 23s 136ms/step - accuracy: 0.9680 - loss: 1.7781 - val_accuracy: 0.9780 - val_loss: 2.4443
Epoch 9/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 18s 120ms/step - accuracy: 0.9745 - loss: 1.3516 - val_accuracy: 0.9780 - val_loss: 1.8944
Epoch 10/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 15s 120ms/step - accuracy: 0.9668 - loss: 1.6475 - val_accuracy: 0.9780 - val_loss: 1.6934
Epoch 11/50
125/125 ━━━━━━━━━━━━━━━━━━━━ 15s 120ms/step - accuracy: 0.9722 - loss: 1.2452 - val_accuracy: 0.9760 - val_loss: 1.3550
Epoch 12/50
```