Assignment-2

**Convolution **

Fetching the dataset

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

```
Choose files kaggle.json
kaggle.json(application/json) - 72 bytes, last modified: 18/10/2025 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"tahmirachowdhury","key":"bea1a2d306355d0f01014083b33883a7"}'}
```

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

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

```
Downloading dogs-vs-cats.zip to /content
 92% 751M/812M [00:04<00:01, 43.7MB/s]
100% 812M/812M [00:04<00:00, 202MB/s]
```

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

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

Transferring images into training, validation, and testing folders

```
import os
import shutil
from pathlib import Path

raw_data_dir = Path("train")
new_base_dir = 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, exist_ok=True)  # prevents FileExistsError
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=raw_data_dir / fname, dst=dir / fname)
```

Building and training the CNN from scratch

Model 1:The dataset was divided into 1,000 training samples, 500 validation samples, and 500 test samples

```
make_subset("test", start_index=0, end_index=500)
make_subset("validation", start_index=500, end_index=1000)
make_subset("train", start_index=1000, end_index=2000)
```

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

Initializing a compact convolutional neural network for dogs vs. cats image classification.

```
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)

Setting up the model for training

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

Data preparation

```
from tensorflow.keras.utils import image_dataset_from_directory

train_data = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
val_data = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_data = 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.
```

```
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)
```

Showing the dimensions of data and corresponding labels from the dataset

```
for data_batch, labels_batch in train_data:
    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,)
```

Training the model on the dataset

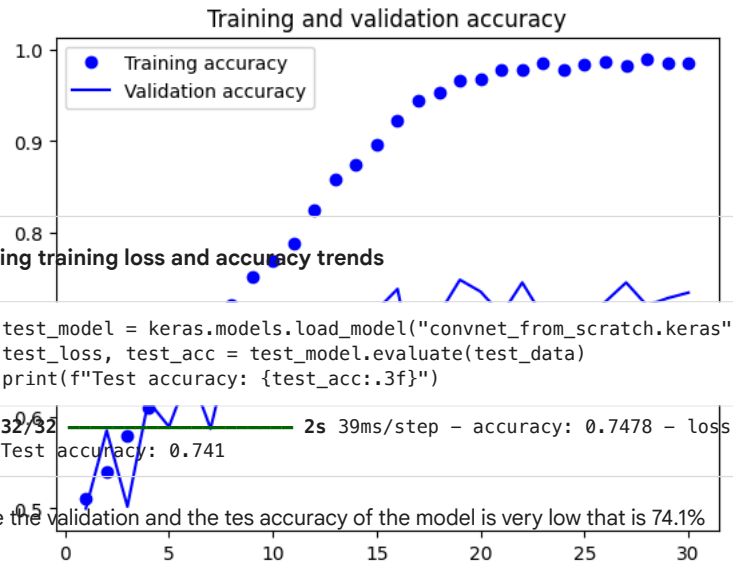
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_data,
    epochs=30,
    validation_data=val_data,
    callbacks=callbacks)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 75ms/step - accuracy: 0.5074 - loss: 0.7013 - val_accuracy: 0.5850 - val_loss: 0.6896
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 51ms/step - accuracy: 0.5689 - loss: 0.6844 - val_accuracy: 0.5020 - val_loss: 0.7994
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.5810 - loss: 0.6839 - val_accuracy: 0.6170 - val_loss: 0.6533
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 66ms/step - accuracy: 0.6604 - loss: 0.6305 - val_accuracy: 0.5890 - val_loss: 0.8273
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.6582 - loss: 0.6385 - val_accuracy: 0.6490 - val_loss: 0.6132
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 57ms/step - accuracy: 0.6975 - loss: 0.5840 - val_accuracy: 0.5870 - val_loss: 0.9872
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.7102 - loss: 0.6203 - val_accuracy: 0.6810 - val_loss: 0.6045
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.7456 - loss: 0.5166 - val_accuracy: 0.6750 - val_loss: 0.6028
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.7719 - loss: 0.4817 - val_accuracy: 0.7050 - val_loss: 0.6418
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 78ms/step - accuracy: 0.7845 - loss: 0.4538 - val_accuracy: 0.6750 - val_loss: 0.7218
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.8228 - loss: 0.4107 - val_accuracy: 0.7110 - val_loss: 0.5852
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.8605 - loss: 0.3322 - val_accuracy: 0.6720 - val_loss: 0.8718
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 71ms/step - accuracy: 0.8651 - loss: 0.3168 - val_accuracy: 0.6260 - val_loss: 1.0703
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.8717 - loss: 0.3088 - val_accuracy: 0.7160 - val_loss: 0.9040
Epoch 16/30
```

```
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 59ms/step - accuracy: 0.9391 - loss: 0.1500 - val_accuracy: 0.7120 - val_loss: 1.0583
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.9704 - loss: 0.0824 - val_accuracy: 0.7490 - val_loss: 1.1701
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.9682 - loss: 0.0808 - val_accuracy: 0.7360 - val_loss: 1.2808
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.9825 - loss: 0.0466 - val_accuracy: 0.7110 - val_loss: 1.5938
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9820 - loss: 0.0564 - val_accuracy: 0.7460 - val_loss: 1.4775
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 72ms/step - accuracy: 0.9839 - loss: 0.0484 - val_accuracy: 0.7090 - val_loss: 1.8275
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.9692 - loss: 0.1385 - val_accuracy: 0.6670 - val_loss: 2.6302
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 0.9747 - loss: 0.0845 - val_accuracy: 0.7030 - val_loss: 2.6017
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.9835 - loss: 0.0545 - val_accuracy: 0.7250 - val_loss: 2.2339
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9851 - loss: 0.0466 - val_accuracy: 0.7460 - val_loss: 1.7138
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9863 - loss: 0.0364 - val_accuracy: 0.7210 - val_loss: 2.2759
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.9829 - loss: 0.0540 - val_accuracy: 0.7290 - val_loss: 2.1253
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 60ms/step - accuracy: 0.9855 - loss: 0.0327 - val_accuracy: 0.7350 - val_loss: 2.2122
```

Plotting training loss and accuracy trends

```
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()
```



Plotting training loss and accuracy trends

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.7478 - loss: 0.5592
Test accuracy: 0.741
```

Since the validation and the tes accuracy of the model is very low that is 74.1%

To improve performance in developing a network that we trained from scratch, we will train our model on following techniques.

Model 1: Applying data augmentation techniques

```
from tensorflow import keras
from tensorflow.keras import layers
augment_layer = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

```
inputs = keras.Input(shape=(180, 180, 3))
x = augment_layer(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)
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_data,
    epochs=30,
    validation_data=val_data,
     callbacks=callbacks)
```

```
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 59ms/step - accuracy: 0.4930 - loss: 0.7038 - val_accuracy: 0.5000 - val_loss: 0.6930
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.5179 - loss: 0.6910 - val_accuracy: 0.5650 - val_loss: 0.6856
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 61ms/step - accuracy: 0.5761 - loss: 0.6867 - val_accuracy: 0.5830 - val_loss: 0.6780
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 59ms/step - accuracy: 0.5953 - loss: 0.6677 - val_accuracy: 0.5800 - val_loss: 0.6674
Epoch 6/30
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.6445 - loss: 0.6292 - val_accuracy: 0.6470 - val_loss: 0.6258
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 60ms/step - accuracy: 0.6708 - loss: 0.6215 - val_accuracy: 0.6400 - val_loss: 0.6529
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.6558 - loss: 0.6136 - val_accuracy: 0.6750 - val_loss: 0.6109
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.6755 - loss: 0.5916 - val_accuracy: 0.6780 - val_loss: 0.6027
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 70ms/step - accuracy: 0.6968 - loss: 0.5873 - val_accuracy: 0.5230 - val_loss: 1.0170
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.7058 - loss: 0.5853 - val_accuracy: 0.7240 - val_loss: 0.5727
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.6861 - loss: 0.5830 - val_accuracy: 0.6840 - val_loss: 0.6097
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.7348 - loss: 0.5500 - val_accuracy: 0.7000 - val_loss: 0.7171
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.6933 - loss: 0.5664 - val_accuracy: 0.7050 - val_loss: 0.5719
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.7203 - loss: 0.5452 - val_accuracy: 0.7150 - val_loss: 0.5720
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 69ms/step - accuracy: 0.7258 - loss: 0.5453 - val_accuracy: 0.7330 - val_loss: 0.5420
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.7546 - loss: 0.5119 - val_accuracy: 0.7380 - val_loss: 0.5619
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 55ms/step - accuracy: 0.7511 - loss: 0.5004 - val_accuracy: 0.7270 - val_loss: 0.5561
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 55ms/step - accuracy: 0.7723 - loss: 0.4825 - val_accuracy: 0.7660 - val_loss: 0.4932
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - accuracy: 0.7837 - loss: 0.4693 - val_accuracy: 0.7470 - val_loss: 0.5171
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 85ms/step - accuracy: 0.7977 - loss: 0.4560 - val_accuracy: 0.7750 - val_loss: 0.5445
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.7930 - loss: 0.4507 - val_accuracy: 0.7350 - val_loss: 0.5447
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 57ms/step - accuracy: 0.7902 - loss: 0.4419 - val_accuracy: 0.6940 - val_loss: 0.6493
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 80ms/step - accuracy: 0.7940 - loss: 0.4575 - val_accuracy: 0.7940 - val_loss: 0.4907
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.8005 - loss: 0.4216 - val_accuracy: 0.7630 - val_loss: 0.5066
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.8129 - loss: 0.4098 - val_accuracy: 0.7520 - val_loss: 0.6202
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 72ms/step - accuracy: 0.8076 - loss: 0.4191 - val_accuracy: 0.7860 - val_loss: 0.4798
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.8209 - loss: 0.4095 - val_accuracy: 0.7830 - val_loss: 0.5120
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.8241 - loss: 0.4116 - val_accuracy: 0.7660 - val_loss: 0.4889
```

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.8225 - loss: 0.4296
Test accuracy: 0.809
```

Creating a data augmentation layer for the image model

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

Visualizing randomly augmented samples from the training set

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

Model 1b: Applying the dropout technique

```
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)
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_dropout.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_data,
    epochs=30,
    validation_data=val_data,
    callbacks=callbacks)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.5287 - loss: 0.6931 - val_accuracy: 0.5000 - val_loss: 0.7037
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 69ms/step - accuracy: 0.5467 - loss: 0.7023 - val_accuracy: 0.6050 - val_loss: 0.6646
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.5907 - loss: 0.6671 - val_accuracy: 0.6030 - val_loss: 0.6515
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 59ms/step - accuracy: 0.6470 - loss: 0.6249 - val_accuracy: 0.6020 - val_loss: 0.7147
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.6602 - loss: 0.6319 - val_accuracy: 0.7030 - val_loss: 0.5876
Epoch 7/30
```

```
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 62ms/step - accuracy: 0.7349 - loss: 0.5338 - val_accuracy: 0.6940 - val_loss: 0.6714
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 60ms/step - accuracy: 0.7497 - loss: 0.5193 - val_accuracy: 0.7170 - val_loss: 0.5612
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - accuracy: 0.7652 - loss: 0.5045 - val_accuracy: 0.7170 - val_loss: 0.5796
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 60ms/step - accuracy: 0.8008 - loss: 0.4408 - val_accuracy: 0.7220 - val_loss: 0.5542
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.8239 - loss: 0.4116 - val_accuracy: 0.7380 - val_loss: 0.5972
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 69ms/step - accuracy: 0.8300 - loss: 0.3742 - val_accuracy: 0.7260 - val_loss: 0.7719
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.8496 - loss: 0.3423 - val_accuracy: 0.7250 - val_loss: 0.6227
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - accuracy: 0.8819 - loss: 0.3014 - val_accuracy: 0.7530 - val_loss: 0.6144
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 66ms/step - accuracy: 0.8969 - loss: 0.2579 - val_accuracy: 0.7120 - val_loss: 0.7932
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9135 - loss: 0.2207 - val_accuracy: 0.7410 - val_loss: 0.6386
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9342 - loss: 0.1873 - val_accuracy: 0.7340 - val_loss: 0.8491
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.9227 - loss: 0.1765 - val_accuracy: 0.7130 - val_loss: 1.0044
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.9399 - loss: 0.1664 - val_accuracy: 0.7460 - val_loss: 0.8566
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 59ms/step - accuracy: 0.9470 - loss: 0.1209 - val_accuracy: 0.7590 - val_loss: 0.9368
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.9683 - loss: 0.0982 - val_accuracy: 0.7030 - val_loss: 1.1350
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.9659 - loss: 0.1020 - val_accuracy: 0.6900 - val_loss: 1.3554
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.9570 - loss: 0.1298 - val_accuracy: 0.7440 - val_loss: 1.2245
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9747 - loss: 0.0820 - val_accuracy: 0.6980 - val_loss: 1.3759
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - accuracy: 0.9734 - loss: 0.0855 - val_accuracy: 0.7500 - val_loss: 1.3971
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9822 - loss: 0.0551 - val_accuracy: 0.7300 - val_loss: 1.6250
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - accuracy: 0.9795 - loss: 0.0779 - val_accuracy: 0.7390 - val_loss: 1.3151
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 69ms/step - accuracy: 0.9716 - loss: 0.0911 - val_accuracy: 0.7630 - val_loss: 1.3118
```

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.7496 - loss: 0.5157
Test accuracy: 0.747
```

Model 1c: Applying a combination of image augmentation and dropout techniques

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

```
inputs = keras.Input(shape=(180, 180, 3))
x = augment_layer(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_dropout.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_data,
    epochs=30,
    validation_data=val_data,
    callbacks=callbacks)
```

```
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.5112 - loss: 0.6965 - val_accuracy: 0.5000 - val_loss: 0.6968
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.5626 - loss: 0.6865 - val_accuracy: 0.5530 - val_loss: 0.6833
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.5369 - loss: 0.6904 - val_accuracy: 0.5250 - val_loss: 0.6863
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.6003 - loss: 0.6661 - val_accuracy: 0.5940 - val_loss: 0.6648
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.6305 - loss: 0.6568 - val_accuracy: 0.6310 - val_loss: 0.6432
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 72ms/step - accuracy: 0.6176 - loss: 0.6597 - val_accuracy: 0.6490 - val_loss: 0.6383
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 59ms/step - accuracy: 0.6252 - loss: 0.6540 - val_accuracy: 0.6550 - val_loss: 0.6235
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 53ms/step - accuracy: 0.6524 - loss: 0.6311 - val_accuracy: 0.6650 - val_loss: 0.6299
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 65ms/step - accuracy: 0.6698 - loss: 0.6144 - val_accuracy: 0.6890 - val_loss: 0.5869
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.6614 - loss: 0.6162 - val_accuracy: 0.5800 - val_loss: 0.7516
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.6789 - loss: 0.5923 - val_accuracy: 0.6050 - val_loss: 0.8373
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - accuracy: 0.6888 - loss: 0.5951 - val_accuracy: 0.6980 - val_loss: 0.5896
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.7193 - loss: 0.5626 - val_accuracy: 0.6280 - val_loss: 0.6982
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.7117 - loss: 0.5687 - val_accuracy: 0.7000 - val_loss: 0.5907
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 57ms/step - accuracy: 0.7329 - loss: 0.5301 - val_accuracy: 0.7000 - val_loss: 0.5730
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.7323 - loss: 0.5247 - val_accuracy: 0.6990 - val_loss: 0.6041
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 68ms/step - accuracy: 0.7296 - loss: 0.5400 - val_accuracy: 0.7350 - val_loss: 0.5613
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - accuracy: 0.7459 - loss: 0.5135 - val_accuracy: 0.6900 - val_loss: 0.5928
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.7589 - loss: 0.5154 - val_accuracy: 0.7570 - val_loss: 0.5156
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.7643 - loss: 0.4918 - val_accuracy: 0.7620 - val_loss: 0.5048
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.7720 - loss: 0.4853 - val_accuracy: 0.7390 - val_loss: 0.5424
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.7872 - loss: 0.4736 - val_accuracy: 0.7350 - val_loss: 0.5363
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 74ms/step - accuracy: 0.7641 - loss: 0.4708 - val_accuracy: 0.6520 - val_loss: 0.7482
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 57ms/step - accuracy: 0.7838 - loss: 0.4685 - val_accuracy: 0.7250 - val_loss: 0.5780
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.7862 - loss: 0.4617 - val_accuracy: 0.7860 - val_loss: 0.4737
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 89ms/step - accuracy: 0.7819 - loss: 0.4789 - val_accuracy: 0.7840 - val_loss: 0.4954
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.8031 - loss: 0.4404 - val_accuracy: 0.7520 - val_loss: 0.5335
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.8031 - loss: 0.4391 - val_accuracy: 0.7740 - val_loss: 0.5238
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 72ms/step - accuracy: 0.7876 - loss: 0.4654 - val_accuracy: 0.7760 - val_loss: 0.5472
```

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.8110 - loss: 0.4337
Test accuracy: 0.822
```

Model 2) Expanding the training dataset to 5,000 samples and incorporating MaxPooling, data augmentation, and a 0.5 dropout rate.

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

```
make_subset("train_1", start_index=0, end_index=5000)
make_subset("validation_1", start_index=5000, end_index=5500)
make_subset("test_1", start_index=5500, end_index=6000)

train_data_1 = image_dataset_from_directory(
    new_base_dir / "train_1",
    image_size=(180, 180),
    batch_size=32)
val_data_1 = image_dataset_from_directory(
    new_base_dir / "validation_1",
    image_size=(180, 180),
    batch_size=32)
test_data_1 = image_dataset_from_directory(
    new_base_dir / "test_1",
    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.
```

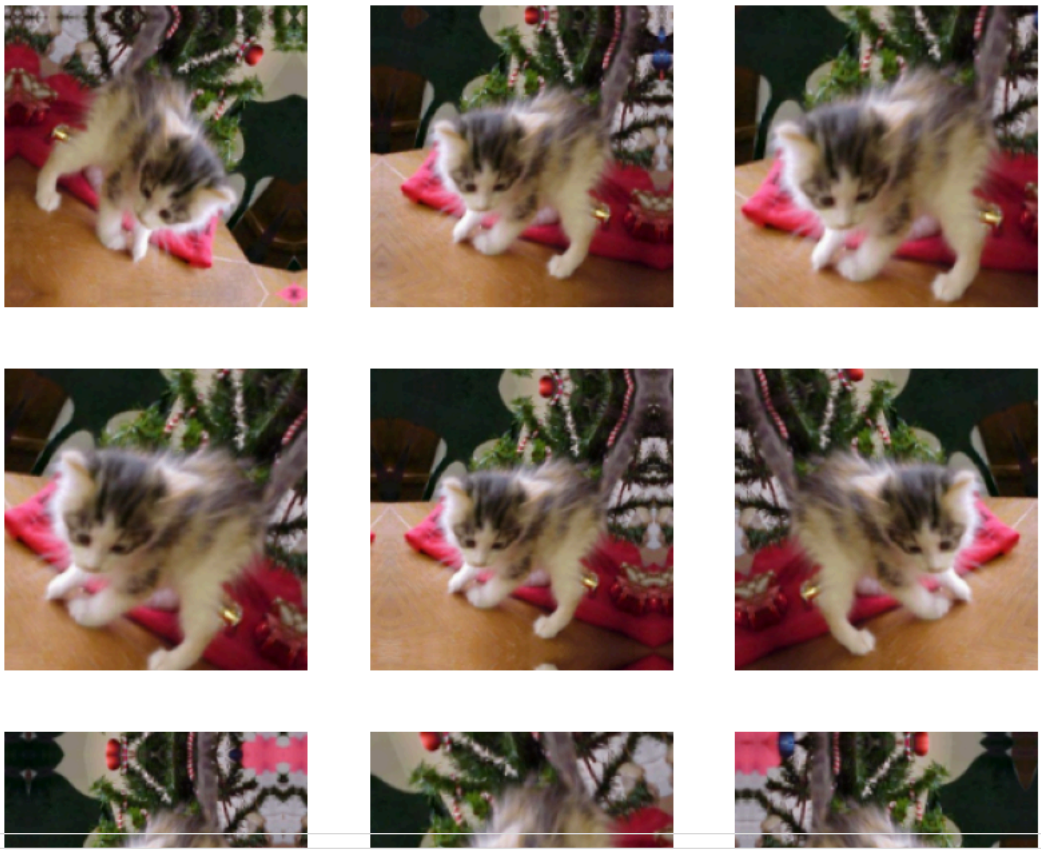
Building a new convolutional network with integrated augmentation and dropout layers

```
inputs = keras.Input(shape=(180, 180, 3))
x = augment_layer(inputs)
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)
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"])
```

```
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from keras.callbacks import EarlyStopping
from keras import regularizers

# used early stopping to stop optimization when it isn't helping any more.
early_stopping_monitor = EarlyStopping(patience=10)
```

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

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

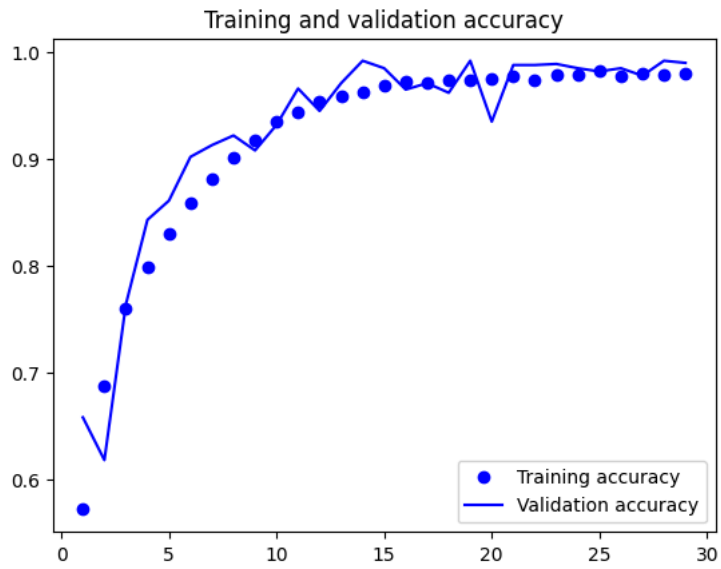


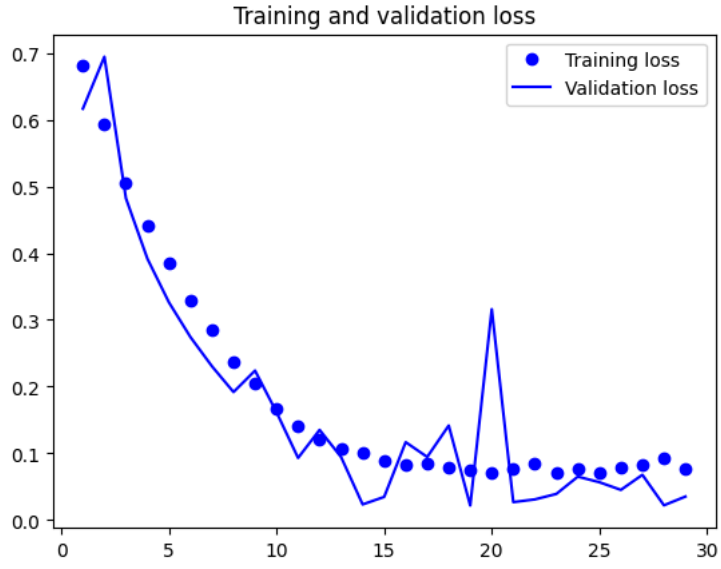
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss"), early_stopping_monitor
]
history = model.fit(
    train_data_1,
    epochs=30,
    validation_data=val_data,
    callbacks=callbacks)
```

```
Epoch 1/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 24s 62ms/step - accuracy: 0.5341 - loss: 0.6931 - val_accuracy: 0.6580 - val_loss: 0.61
Epoch 2/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.6668 - loss: 0.6137 - val_accuracy: 0.6180 - val_loss: 0.69
Epoch 3/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 45ms/step - accuracy: 0.7454 - loss: 0.5271 - val_accuracy: 0.7640 - val_loss: 0.48
Epoch 4/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.7874 - loss: 0.4587 - val_accuracy: 0.8430 - val_loss: 0.39
Epoch 5/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 46ms/step - accuracy: 0.8279 - loss: 0.3936 - val_accuracy: 0.8610 - val_loss: 0.32
Epoch 6/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 46ms/step - accuracy: 0.8505 - loss: 0.3444 - val_accuracy: 0.9020 - val_loss: 0.27
Epoch 7/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.8761 - loss: 0.2914 - val_accuracy: 0.9130 - val_loss: 0.23
Epoch 8/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 45ms/step - accuracy: 0.8974 - loss: 0.2389 - val_accuracy: 0.9220 - val_loss: 0.19
Epoch 9/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9138 - loss: 0.2085 - val_accuracy: 0.9080 - val_loss: 0.22
Epoch 10/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9341 - loss: 0.1667 - val_accuracy: 0.9320 - val_loss: 0.16
Epoch 11/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 46ms/step - accuracy: 0.9446 - loss: 0.1353 - val_accuracy: 0.9660 - val_loss: 0.09
Epoch 12/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 49ms/step - accuracy: 0.9528 - loss: 0.1233 - val_accuracy: 0.9450 - val_loss: 0.13
Epoch 13/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 19s 44ms/step - accuracy: 0.9586 - loss: 0.1077 - val_accuracy: 0.9710 - val_loss: 0.09
Epoch 14/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 45ms/step - accuracy: 0.9628 - loss: 0.0990 - val_accuracy: 0.9920 - val_loss: 0.02
Epoch 15/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 46ms/step - accuracy: 0.9711 - loss: 0.0806 - val_accuracy: 0.9850 - val_loss: 0.03
Epoch 16/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 46ms/step - accuracy: 0.9714 - loss: 0.0822 - val_accuracy: 0.9650 - val_loss: 0.11
Epoch 17/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9718 - loss: 0.0825 - val_accuracy: 0.9710 - val_loss: 0.09
Epoch 18/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 46ms/step - accuracy: 0.9720 - loss: 0.0790 - val_accuracy: 0.9620 - val_loss: 0.14
Epoch 19/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 47ms/step - accuracy: 0.9743 - loss: 0.0752 - val_accuracy: 0.9920 - val_loss: 0.02
Epoch 20/30
```

```
Epoch 21/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9770 - loss: 0.0803 - val_accuracy: 0.9880 - val_loss: 0.02
Epoch 22/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9748 - loss: 0.0888 - val_accuracy: 0.9880 - val_loss: 0.03
Epoch 23/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9775 - loss: 0.0731 - val_accuracy: 0.9890 - val_loss: 0.03
Epoch 24/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 46ms/step - accuracy: 0.9790 - loss: 0.0792 - val_accuracy: 0.9850 - val_loss: 0.06
Epoch 25/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 46ms/step - accuracy: 0.9832 - loss: 0.0717 - val_accuracy: 0.9820 - val_loss: 0.05
Epoch 26/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 20s 46ms/step - accuracy: 0.9753 - loss: 0.0910 - val_accuracy: 0.9850 - val_loss: 0.04
Epoch 27/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 47ms/step - accuracy: 0.9811 - loss: 0.0731 - val_accuracy: 0.9780 - val_loss: 0.06
Epoch 28/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 16s 49ms/step - accuracy: 0.9775 - loss: 0.0858 - val_accuracy: 0.9920 - val_loss: 0.02
Epoch 29/30
313/313 ━━━━━━━━━━━━━━━━━━━━ 14s 45ms/step - accuracy: 0.9815 - loss: 0.0704 - val_accuracy: 0.9900 - val_loss: 0.03
```

```
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_data_1)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.8768 - loss: 0.6293
Test accuracy: 0.876
```

Model 3: Expanding the training dataset to 10,000 samples

```
from tensorflow.keras.utils import image_dataset_from_directory
make_subset("train_4", start_index=0, end_index=10000)
make_subset("validation_4", start_index=10000, end_index=10500)
make_subset("test_4", start_index=10500, end_index=11000)

train_data_4 = image_dataset_from_directory(
    new_base_dir / "train_4",
    image_size=(180, 180),
    batch_size=32)
val_data_4 = image_dataset_from_directory(
    new_base_dir / "validation_4",
    image_size=(180, 180),
    batch_size=32)
test_data_4 = image_dataset_from_directory(
    new_base_dir / "test_4",
    image_size=(180, 180),
    batch_size=32)
```

```
Found 20000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
```

```
inputs = keras.Input(shape=(180, 180, 3))
x = augment_layer(inputs)
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)
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"])
```

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

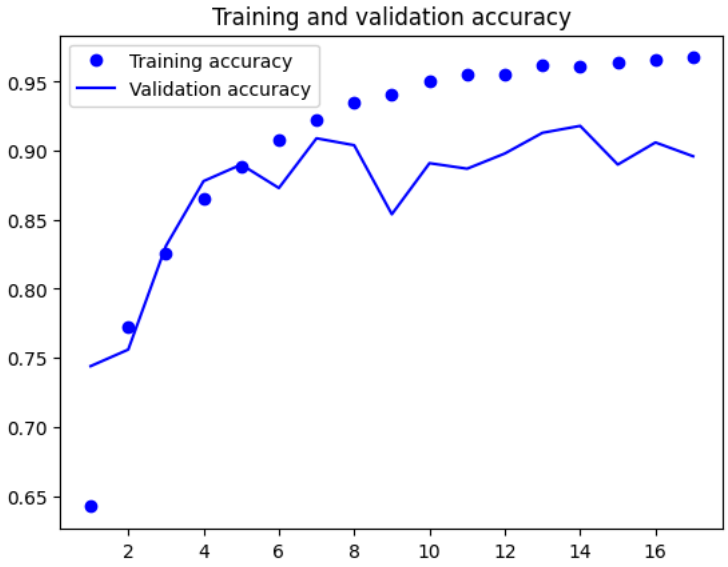
```
plt.figure(figsize=(10, 10))
for images, _ in train_data.take(1):
    for i in range(9):
        augmented_images = augment_layer(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

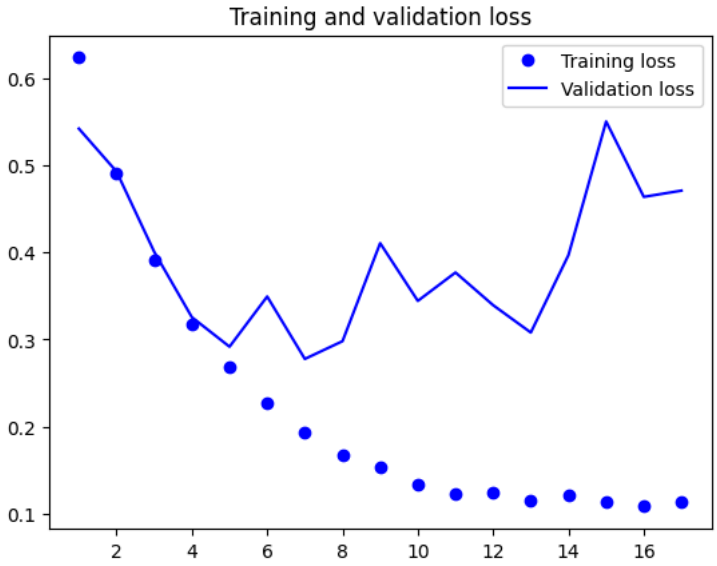
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss"), early_stopping_monitor
]
history = model.fit(
    train_data_4,
    epochs=30,
    validation_data=val_data_4,
    callbacks=callbacks)
```

```
Epoch 1/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 32s 44ms/step - accuracy: 0.5721 - loss: 0.6666 - val_accuracy: 0.7440 - val_loss: 0.542
Epoch 2/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 37s 43ms/step - accuracy: 0.7526 - loss: 0.5233 - val_accuracy: 0.7560 - val_loss: 0.492
Epoch 3/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 28s 44ms/step - accuracy: 0.8134 - loss: 0.4166 - val_accuracy: 0.8310 - val_loss: 0.400
Epoch 4/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 41s 43ms/step - accuracy: 0.8558 - loss: 0.3377 - val_accuracy: 0.8780 - val_loss: 0.325
Epoch 5/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 42s 45ms/step - accuracy: 0.8807 - loss: 0.2877 - val_accuracy: 0.8900 - val_loss: 0.291
Epoch 6/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 26s 42ms/step - accuracy: 0.9025 - loss: 0.2373 - val_accuracy: 0.8730 - val_loss: 0.349
Epoch 7/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9200 - loss: 0.2030 - val_accuracy: 0.9090 - val_loss: 0.277
Epoch 8/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 41s 43ms/step - accuracy: 0.9315 - loss: 0.1761 - val_accuracy: 0.9040 - val_loss: 0.298
Epoch 9/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 44ms/step - accuracy: 0.9378 - loss: 0.1592 - val_accuracy: 0.8540 - val_loss: 0.410
Epoch 10/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 41s 43ms/step - accuracy: 0.9478 - loss: 0.1334 - val_accuracy: 0.8910 - val_loss: 0.344
Epoch 11/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 41s 43ms/step - accuracy: 0.9524 - loss: 0.1288 - val_accuracy: 0.8870 - val_loss: 0.376
Epoch 12/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 41s 44ms/step - accuracy: 0.9529 - loss: 0.1334 - val_accuracy: 0.8980 - val_loss: 0.339
Epoch 13/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9625 - loss: 0.1109 - val_accuracy: 0.9130 - val_loss: 0.307
Epoch 14/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 43ms/step - accuracy: 0.9617 - loss: 0.1168 - val_accuracy: 0.9180 - val_loss: 0.397
Epoch 15/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 28s 45ms/step - accuracy: 0.9614 - loss: 0.1241 - val_accuracy: 0.8900 - val_loss: 0.550
Epoch 16/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 40s 43ms/step - accuracy: 0.9656 - loss: 0.1091 - val_accuracy: 0.9060 - val_loss: 0.463
Epoch 17/30
625/625 ━━━━━━━━━━━━━━━━━━━━ 27s 44ms/step - accuracy: 0.9670 - loss: 0.1138 - val_accuracy: 0.8960 - val_loss: 0.471
```

```
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_data_4)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9066 - loss: 0.2917
Test accuracy: 0.900
```

Combining feature extraction and data augmentation

Creating and locking the VGG16 convolutional backbone

Using a pre-trained model with 1,000 training examples

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_orderin
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
```

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_11 (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)

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

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

Integrating data augmentation and a classifier module with the convolutional backbone

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

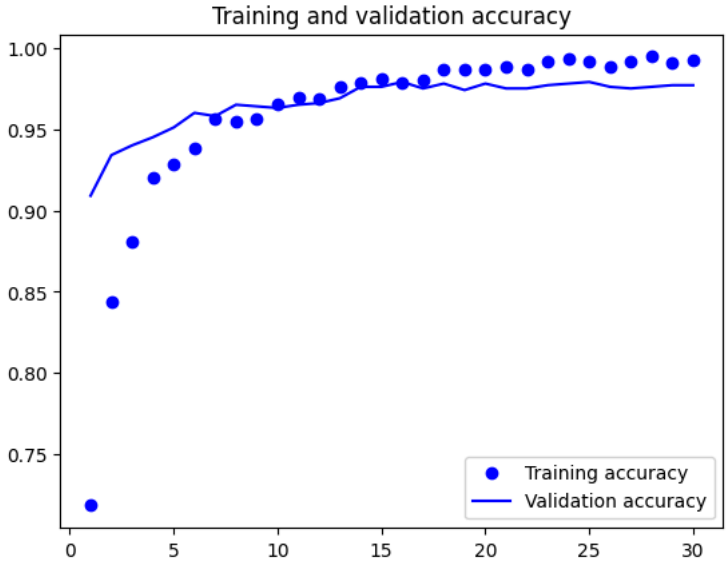
```
inputs = keras.Input(shape=(180, 180, 3))
x = augment_layer(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])
```

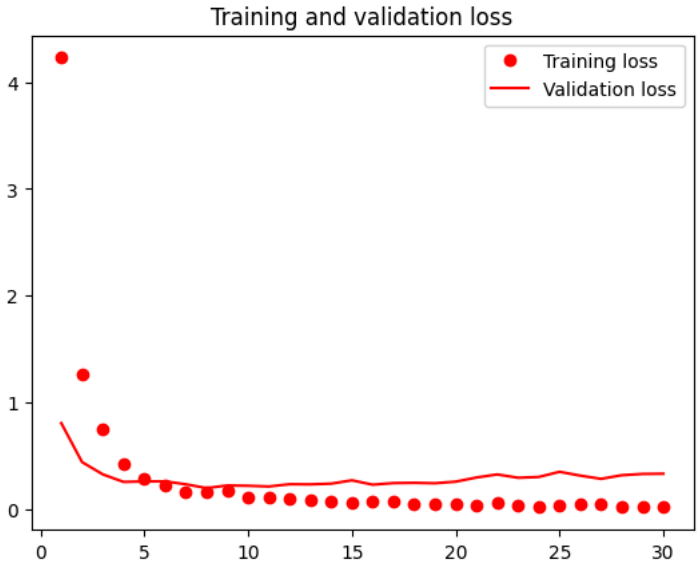
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
```

```
        monitor="val_loss")
]
history = model.fit(
    train_data,
    epochs=30,
    validation_data=val_data,
    callbacks=callbacks)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 172ms/step - accuracy: 0.8394 - loss: 1.3795 - val_accuracy: 0.9340 - val_loss: 0.439
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 169ms/step - accuracy: 0.8624 - loss: 0.8492 - val_accuracy: 0.9400 - val_loss: 0.323
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 166ms/step - accuracy: 0.9226 - loss: 0.4556 - val_accuracy: 0.9450 - val_loss: 0.253
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9258 - loss: 0.2825 - val_accuracy: 0.9510 - val_loss: 0.257
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9435 - loss: 0.1864 - val_accuracy: 0.9600 - val_loss: 0.258
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 161ms/step - accuracy: 0.9633 - loss: 0.1231 - val_accuracy: 0.9580 - val_loss: 0.230
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 161ms/step - accuracy: 0.9555 - loss: 0.1615 - val_accuracy: 0.9650 - val_loss: 0.195
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9576 - loss: 0.1589 - val_accuracy: 0.9640 - val_loss: 0.219
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 192ms/step - accuracy: 0.9648 - loss: 0.1139 - val_accuracy: 0.9630 - val_loss: 0.216
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9744 - loss: 0.0768 - val_accuracy: 0.9650 - val_loss: 0.210
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.9709 - loss: 0.0952 - val_accuracy: 0.9660 - val_loss: 0.231
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.9838 - loss: 0.0614 - val_accuracy: 0.9690 - val_loss: 0.230
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9777 - loss: 0.0573 - val_accuracy: 0.9760 - val_loss: 0.236
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9851 - loss: 0.0430 - val_accuracy: 0.9760 - val_loss: 0.267
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9833 - loss: 0.0595 - val_accuracy: 0.9790 - val_loss: 0.227
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9826 - loss: 0.0690 - val_accuracy: 0.9750 - val_loss: 0.242
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9854 - loss: 0.0418 - val_accuracy: 0.9780 - val_loss: 0.244
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9869 - loss: 0.0528 - val_accuracy: 0.9740 - val_loss: 0.240
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9881 - loss: 0.0399 - val_accuracy: 0.9780 - val_loss: 0.255
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9877 - loss: 0.0310 - val_accuracy: 0.9750 - val_loss: 0.294
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9829 - loss: 0.0511 - val_accuracy: 0.9750 - val_loss: 0.323
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9934 - loss: 0.0202 - val_accuracy: 0.9770 - val_loss: 0.291
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9926 - loss: 0.0217 - val_accuracy: 0.9780 - val_loss: 0.299
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9933 - loss: 0.0182 - val_accuracy: 0.9790 - val_loss: 0.347
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9905 - loss: 0.0263 - val_accuracy: 0.9760 - val_loss: 0.312
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9890 - loss: 0.0377 - val_accuracy: 0.9750 - val_loss: 0.281
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9945 - loss: 0.0141 - val_accuracy: 0.9760 - val_loss: 0.315
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9919 - loss: 0.0185 - val_accuracy: 0.9770 - val_loss: 0.327
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9925 - loss: 0.0196 - val_accuracy: 0.9770 - val_loss: 0.329
```

```
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, "ro", label="Training loss")
plt.plot(epochs, val_loss, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```






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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - accuracy: 0.9688 - loss: 0.1486
Test accuracy: 0.972
```

Using a pre-trained model with 5,000 training examples

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

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

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

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

```
inputs = keras.Input(shape=(180, 180, 3))
x = augment_layer(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=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning2.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_data_1,
    epochs=10,
    validation_data=val_data_1,
    callbacks=callbacks)
```

```
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 40s 121ms/step - accuracy: 0.7894 - loss: 2.9347 - val_accuracy: 0.9690 - val_loss: 0.15
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 43s 127ms/step - accuracy: 0.9290 - loss: 0.2757 - val_accuracy: 0.9670 - val_loss: 0.14
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 39s 120ms/step - accuracy: 0.9456 - loss: 0.1653 - val_accuracy: 0.9730 - val_loss: 0.12
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 118ms/step - accuracy: 0.9602 - loss: 0.1241 - val_accuracy: 0.9750 - val_loss: 0.12
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 118ms/step - accuracy: 0.9626 - loss: 0.1174 - val_accuracy: 0.9770 - val_loss: 0.13
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 38s 120ms/step - accuracy: 0.9658 - loss: 0.1010 - val_accuracy: 0.9760 - val_loss: 0.11
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 38s 120ms/step - accuracy: 0.9733 - loss: 0.0788 - val_accuracy: 0.9750 - val_loss: 0.09
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 118ms/step - accuracy: 0.9731 - loss: 0.0807 - val_accuracy: 0.9770 - val_loss: 0.15
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 119ms/step - accuracy: 0.9778 - loss: 0.0624 - val_accuracy: 0.9780 - val_loss: 0.15
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 40s 126ms/step - accuracy: 0.9785 - loss: 0.0763 - val_accuracy: 0.9800 - val_loss: 0.13
```

```
model = keras.models.load_model("fine_tuning2.keras")
test_loss, test_acc = model.evaluate(test_data_1)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 89ms/step - accuracy: 0.9755 - loss: 0.1240
Test accuracy: 0.978
```

Using a pre-trained model with 10,000 training examples

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

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

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

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

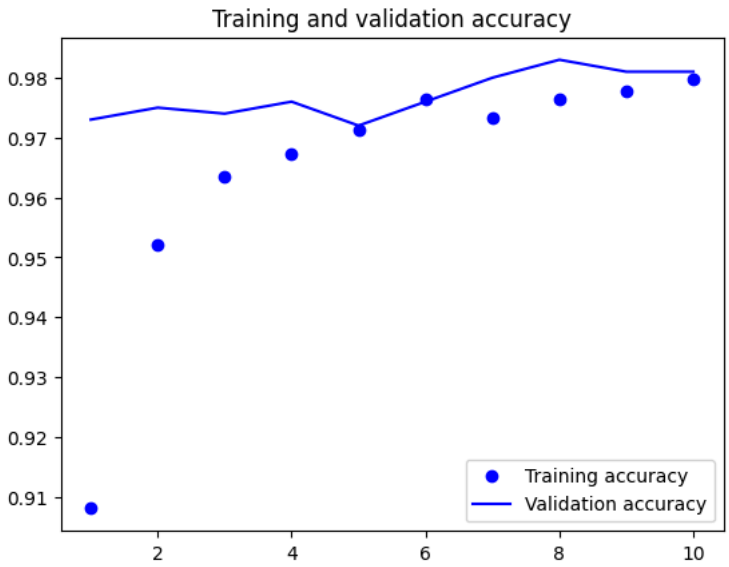
```
inputs = keras.Input(shape=(180, 180, 3))
x = augment_layer(inputs)
```

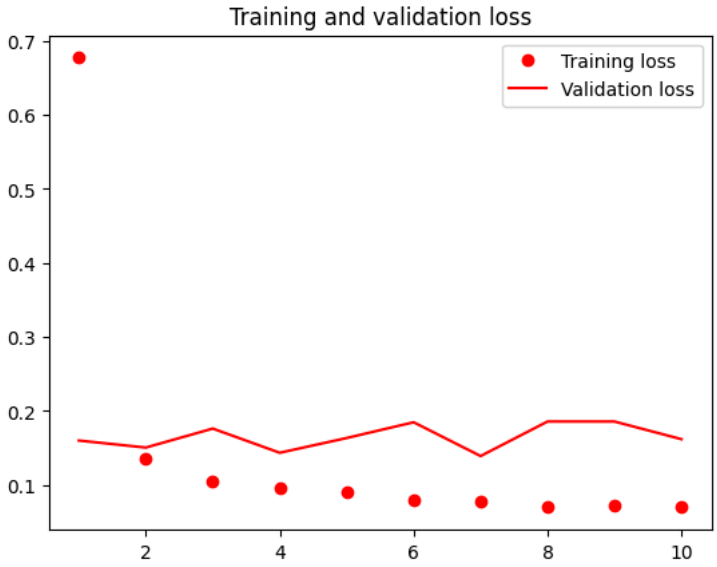
```
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning3.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_data_4,
    epochs=10,
    validation_data=val_data_4,
    callbacks=callbacks)
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 75s 116ms/step - accuracy: 0.8607 - loss: 1.6403 - val_accuracy: 0.9730 - val_loss: 0.16
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 72s 116ms/step - accuracy: 0.9474 - loss: 0.1505 - val_accuracy: 0.9750 - val_loss: 0.15
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 74s 118ms/step - accuracy: 0.9619 - loss: 0.1113 - val_accuracy: 0.9740 - val_loss: 0.17
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 72s 115ms/step - accuracy: 0.9690 - loss: 0.0901 - val_accuracy: 0.9760 - val_loss: 0.14
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 114ms/step - accuracy: 0.9691 - loss: 0.0965 - val_accuracy: 0.9720 - val_loss: 0.16
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 114ms/step - accuracy: 0.9767 - loss: 0.0790 - val_accuracy: 0.9760 - val_loss: 0.18
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 72s 115ms/step - accuracy: 0.9715 - loss: 0.0826 - val_accuracy: 0.9800 - val_loss: 0.13
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 84s 118ms/step - accuracy: 0.9765 - loss: 0.0737 - val_accuracy: 0.9830 - val_loss: 0.18
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 114ms/step - accuracy: 0.9779 - loss: 0.0721 - val_accuracy: 0.9810 - val_loss: 0.18
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 71s 114ms/step - accuracy: 0.9811 - loss: 0.0689 - val_accuracy: 0.9810 - val_loss: 0.16
```

```
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, "ro", label="Training loss")
plt.plot(epochs, val_loss, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```






```
model = keras.models.load_model("fine_tuning3.keras")
test_loss, test_acc = model.evaluate(test_data)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - accuracy: 0.9899 - loss: 0.0253
Test accuracy: 0.992
```

Summary

1.Model 1, which is the unregularized Cats-and-Dogs model, was trained on 1,000 images, with 500 for validation and 500 for testing. It reached only about 73% accuracy, showing that the model is overfitting because the training set is too small.

2. To improve the model's performance while keeping the training sample size fixed at 1,000, I applied three optimization techniques. These include:

a) Dropout method b) Data augmentation c) A combination of data augmentation and dropout

3.It was observed that the model trained using both data augmentation and dropout achieved higher accuracy.

4.Training with more data generally improves accuracy. When the training set was increased to 5,000 and then 10,000 samples, the model showed a clear improvement in accuracy.

5.Using pre-trained models led to a substantial increase in accuracy, with results approaching 99%.

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

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

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.