assignment-aml-cats

April 1, 2024

Group - 03 Assignment by Gaurav Kudeshia & Anurodh Singh

Assignment-2: Exploring Convolutional Neural Networks (CNNs)

The primary aim of this assignment is to explore the capabilities of Convolutional Neural Networks in identifying and classifying objects within images. Students will engage in practical application of CNNs on image datasets, leveraging pre-trained models to evaluate their effectiveness and performance.

[1]:
```
!unzip -/fs/ess/PGS0341/BA_64061_KSU_SEC1/data/dogs-vs-cats.zip
```

```
UnZip 6.00 of 20 April 2009, by Info-ZIP.  Maintained by C. Spieler.  Send
bug reports using http://www.info-zip.org/zip-bug.html; see README for details.

Usage: unzip [-Z] [-opts[modifiers]] file[.zip] [list] [-x xlist] [-d exdir]
  Default action is to extract files in list, except those in xlist, to exdir;
  file[.zip] may be a wildcard.  -Z => ZipInfo mode ("unzip -Z" for usage).

  -p  extract files to pipe, no messages     -l  list files (short format)
  -f  freshen existing files, create none    -t  test compressed archive data
  -u  update files, create if necessary      -z  display archive comment only
  -v  list verbosely/show version info       -T  timestamp archive to latest
  -x  exclude files that follow (in xlist)   -d  extract files into exdir
modifiers:
  -n  never overwrite existing files         -q  quiet mode (-qq => quieter)
  -o  overwrite files WITHOUT prompting      -a  auto-convert any text files
  -j  junk paths (do not make directories)   -aa treat ALL files as text
  -U  use escapes for all non-ASCII Unicode  -UU ignore any Unicode fields
  -C  match filenames case-insensitively     -L  make (some) names lowercase
  -X  restore UID/GID info                   -V  retain VMS version numbers
  -K  keep setuid/setgid/tacky permissions   -M  pipe through "more" pager
  -O CHARSET  specify a character encoding for DOS, Windows and OS/2 archives
  -I CHARSET  specify a character encoding for UNIX and other archives

See "unzip -hh" or unzip.txt for more help.  Examples:
  unzip data1 -x joe   => extract all files except joe from zipfile data1.zip
  unzip -p foo | more  => send contents of foo.zip via pipe into program more
  unzip -fo foo ReadMe => quietly replace existing ReadMe if archive file newer
```

[2]:
```
!unzip -o -qq /fs/ess/PGS0341/BA_64061_KSU_SEC1/data/dogs-vs-cats.zip
```

[3]:
```
!unzip -qq/fs/ess/PGS0341/BA_64061_KSU_SEC1/data/dogs-vs-cats train.zip
```

```
UnZip 6.00 of 20 April 2009, by Info-ZIP.  Maintained by C. Spieler.  Send
bug reports using http://www.info-zip.org/zip-bug.html; see README for details.

Usage: unzip [-Z] [-opts[modifiers]] file[.zip] [list] [-x xlist] [-d exdir]
  Default action is to extract files in list, except those in xlist, to exdir;
  file[.zip] may be a wildcard.  -Z => ZipInfo mode ("unzip -Z" for usage).

  -p  extract files to pipe, no messages     -l  list files (short format)
  -f  freshen existing files, create none    -t  test compressed archive data
  -u  update files, create if necessary      -z  display archive comment only
  -v  list verbosely/show version info       -T  timestamp archive to latest
  -x  exclude files that follow (in xlist)   -d  extract files into exdir
modifiers:
  -n  never overwrite existing files         -q  quiet mode (-qq => quieter)
  -o  overwrite files WITHOUT prompting      -a  auto-convert any text files
  -j  junk paths (do not make directories)   -aa treat ALL files as text
  -U  use escapes for all non-ASCII Unicode  -UU ignore any Unicode fields
  -C  match filenames case-insensitively     -L  make (some) names lowercase
  -X  restore UID/GID info                   -V  retain VMS version numbers
  -K  keep setuid/setgid/tacky permissions   -M  pipe through "more" pager
  -O CHARSET  specify a character encoding for DOS, Windows and OS/2 archives
  -I CHARSET  specify a character encoding for UNIX and other archives

See "unzip -hh" or unzip.txt for more help.  Examples:
  unzip data1 -x joe   => extract all files except joe from zipfile data1.zip
  unzip -p foo | more  => send contents of foo.zip via pipe into program more
  unzip -fo foo ReadMe => quietly replace existing ReadMe if archive file newer
```

Q1, Consider the Cats & Dogs example. Start initially with a training sample of 1000, a validation sample of 500, and a test sample of 500 (like in the text). Use any technique to reduce overfitting and improve performance in developing a network that you train from scratch. What performance did you achieve?

We're going to build a convolutional neural network from the ground up. Having loaded our dataset, it's time to split it into distinct subsets for training, validation, and testing. For this project, we'll allocate 1,000 images for training purposes, 500 images for validation, and another 500 images will be used for the test set.

Organizing the Dataset into Training, Validation, and Testing Groups

[4]:
```python
import os, shutil, pathlib

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

```python
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)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src = original_dir / fname
            dst = dir / fname
            shutil.copyfile(src, dst)

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)
```

0.0.1 Constructing the Model

For this network, we begin by processing images, represented as 3D tensors, which are initially reshaped. The process involves applying convolution operations using a 3x3 window (referred to as kernel_size), followed by max pooling operations with a 2x2 window (known as pool_size).

The objective of this task is to categorize the images into two classes: "cat" or "dog". To achieve this, the architecture incorporates a dense layer towards the end, which plays a crucial role in determining the classification of the output as either "cat" or "dog". This classification is facilitated by a single output node corresponding to these categories. Before reaching the dense layer, it's necessary to transform the 3D tensor structure into a 1D format, a step accomplished by introducing a flattening layer.

Creating a Compact Convolutional Network for Cat vs. Dog Classification

[5]:
```python
from tensorflow import keras
from tensorflow.keras import layers

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

[6]:
```python
model.summary()
```

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

Configuring the model for training

[7]:
```python
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

0.0.2 Data preprocessing

Using image_dataset_from_directory to read images

[8]:
```python
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.
```

[9]:
```
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)
```

[10]:
```
for i, element in enumerate(dataset):
    print(element.shape)
    if i >= 2:
        break
```

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

[11]:
```
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)
```

[12]:
```
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 Labels Provided by the Dataset

[13]:
```
for data_batch, labels_batch in train_dataset:
    print("data batch shape:", data_batch.shape)
    print("labels batch shape:", labels_batch.shape)
    break
```

```
data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
```

Training the Model with a Dataset

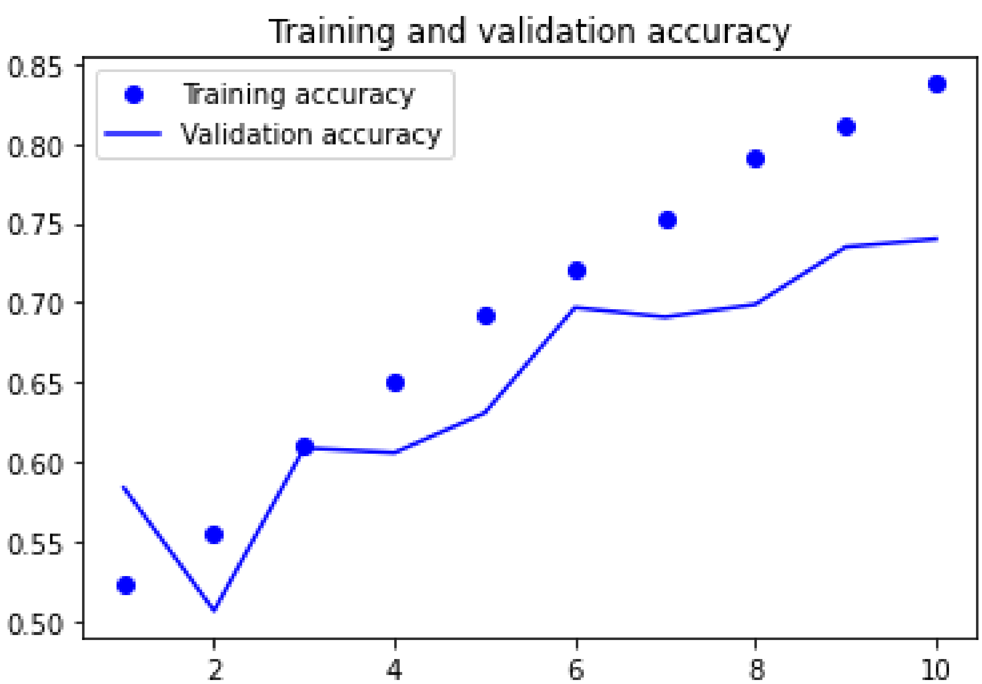
[14]:
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

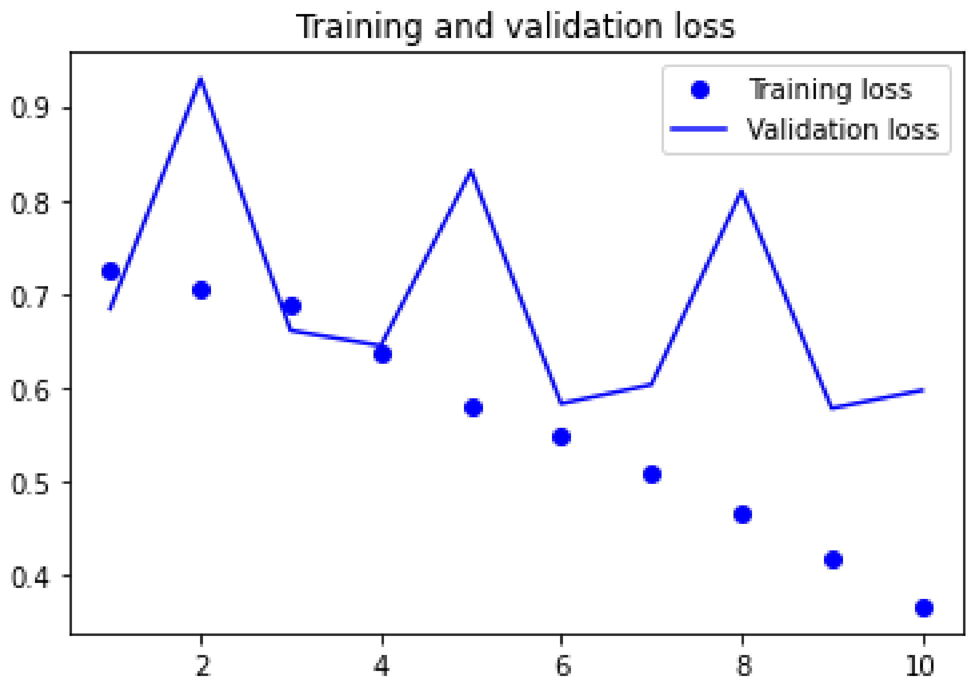
```
Epoch 1/10
63/63 [==============================] - 46s 726ms/step - loss: 0.7247 -
accuracy: 0.5230 - val_loss: 0.6839 - val_accuracy: 0.5840
Epoch 2/10
63/63 [==============================] - 45s 711ms/step - loss: 0.7062 -
accuracy: 0.5555 - val_loss: 0.9294 - val_accuracy: 0.5070
Epoch 3/10
63/63 [==============================] - 45s 713ms/step - loss: 0.6871 -
accuracy: 0.6100 - val_loss: 0.6607 - val_accuracy: 0.6090
Epoch 4/10
63/63 [==============================] - 45s 713ms/step - loss: 0.6377 -
accuracy: 0.6505 - val_loss: 0.6445 - val_accuracy: 0.6060
Epoch 5/10
63/63 [==============================] - 45s 713ms/step - loss: 0.5790 -
accuracy: 0.6920 - val_loss: 0.8312 - val_accuracy: 0.6310
Epoch 6/10
63/63 [==============================] - 45s 714ms/step - loss: 0.5475 -
accuracy: 0.7215 - val_loss: 0.5828 - val_accuracy: 0.6970
Epoch 7/10
63/63 [==============================] - 45s 708ms/step - loss: 0.5099 -
accuracy: 0.7525 - val_loss: 0.6031 - val_accuracy: 0.6910
Epoch 8/10
63/63 [==============================] - 45s 709ms/step - loss: 0.4653 -
```

```
accuracy: 0.7915 - val_loss: 0.8097 - val_accuracy: 0.6990
Epoch 9/10
63/63 [==============================] - 45s 706ms/step - loss: 0.4182 -
accuracy: 0.8110 - val_loss: 0.5778 - val_accuracy: 0.7350
Epoch 10/10
63/63 [==============================] - 45s 714ms/step - loss: 0.3663 -
accuracy: 0.8375 - val_loss: 0.5970 - val_accuracy: 0.7400
```

Visualizing Training Loss and Accuracy Curves

[15]:
```python
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





Assessing the model's performance using the test dataset.

[16]:
```
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 [==============================] - 10s 160ms/step - loss: 0.6221 -
accuracy: 0.7300
Test accuracy: 0.730
```

The model's validation and test accuracy currently stands at a relatively low 73%. - To enhance the model's performance, we plan to implement several strategies, including:

a) Data Augmentation

b) Dropout Technique

c) A combination of Data Augmentation and the Dropout Technique

0.0.3 a, By employing data augmentation

Implement a Data Augmentation Stage for an Image Processing Model

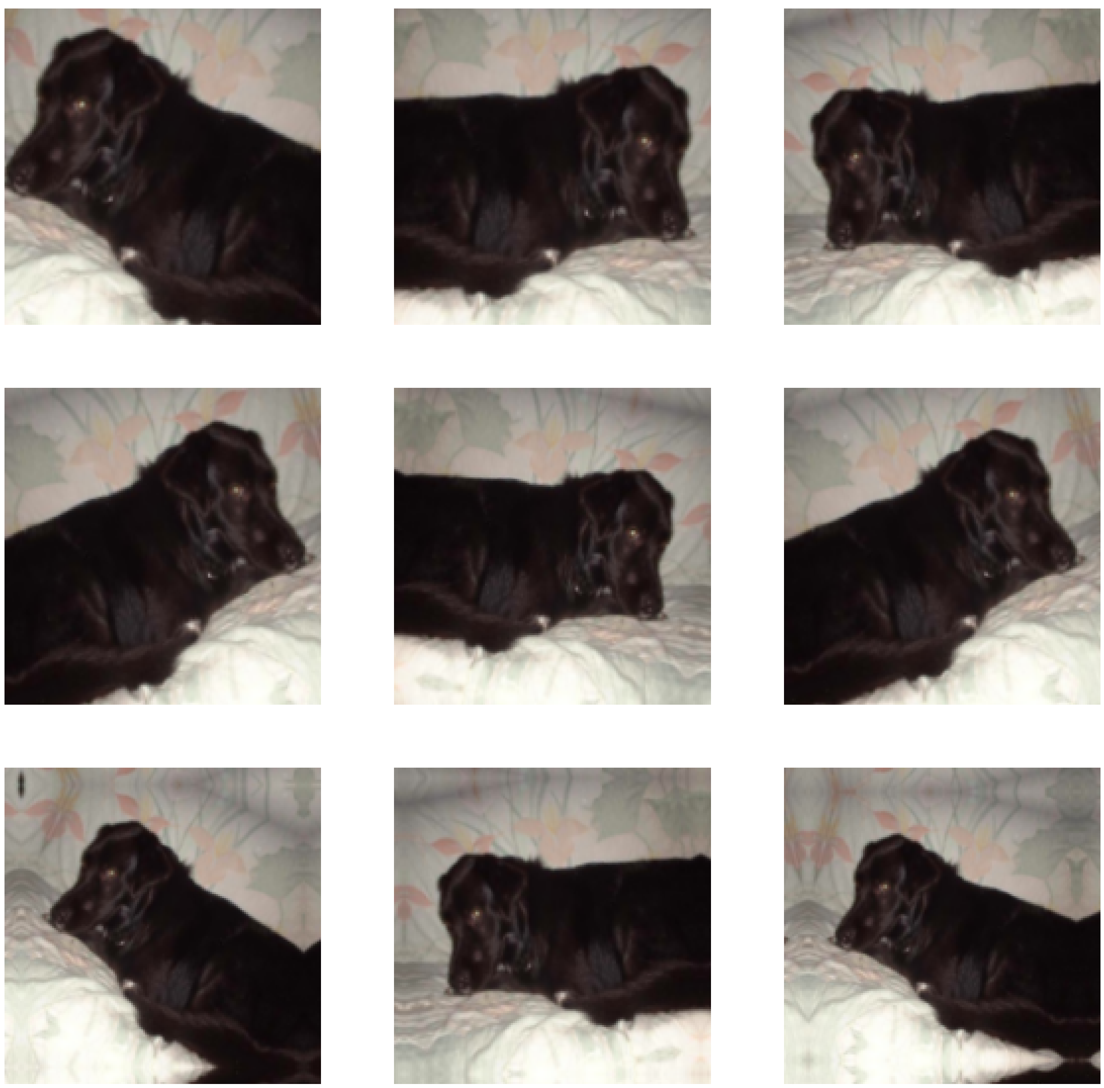
[17]:
```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
```

```
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

Displaying some randomly augmented training images

[18]:

```
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")
```



Q2. Increasing Training sample size to 2870 samples

[19]:
```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_01")

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)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src = original_dir / fname
            dst = dir / fname
            shutil.copyfile(src, dst)

make_subset("train", start_index=0, end_index=1870)
make_subset("validation", start_index=1870, end_index=2370)
make_subset("test", start_index=2370, end_index=2870)
```

Defining a new convnet that includes image augmentation and dropout

[20]:
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

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

Training the regularized convnet

[22]:
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
```

```
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=45,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/45
63/63 [==============================] - 47s 752ms/step - loss: 0.6935 -
accuracy: 0.5275 - val_loss: 0.6748 - val_accuracy: 0.5590
Epoch 2/45
63/63 [==============================] - 47s 742ms/step - loss: 0.6701 -
accuracy: 0.6140 - val_loss: 0.6567 - val_accuracy: 0.6130
Epoch 3/45
63/63 [==============================] - 47s 745ms/step - loss: 0.6581 -
accuracy: 0.6535 - val_loss: 0.6519 - val_accuracy: 0.5830
Epoch 4/45
63/63 [==============================] - 47s 747ms/step - loss: 0.6348 -
accuracy: 0.6590 - val_loss: 0.6216 - val_accuracy: 0.6320
Epoch 5/45
63/63 [==============================] - 47s 747ms/step - loss: 0.6437 -
accuracy: 0.6595 - val_loss: 0.6396 - val_accuracy: 0.6200
Epoch 6/45
63/63 [==============================] - 47s 747ms/step - loss: 0.6065 -
accuracy: 0.6735 - val_loss: 0.6303 - val_accuracy: 0.6260
Epoch 7/45
63/63 [==============================] - 47s 747ms/step - loss: 0.5992 -
accuracy: 0.6690 - val_loss: 0.5956 - val_accuracy: 0.6630
Epoch 8/45
63/63 [==============================] - 47s 749ms/step - loss: 0.5961 -
accuracy: 0.6875 - val_loss: 0.5844 - val_accuracy: 0.6910
Epoch 9/45
63/63 [==============================] - 47s 748ms/step - loss: 0.5702 -
accuracy: 0.7085 - val_loss: 0.5684 - val_accuracy: 0.6940
Epoch 10/45
63/63 [==============================] - 47s 748ms/step - loss: 0.5564 -
accuracy: 0.7190 - val_loss: 0.6759 - val_accuracy: 0.6940
Epoch 11/45
63/63 [==============================] - 47s 748ms/step - loss: 0.5551 -
accuracy: 0.7145 - val_loss: 0.5375 - val_accuracy: 0.7350
Epoch 12/45
63/63 [==============================] - 47s 747ms/step - loss: 0.5614 -
accuracy: 0.7240 - val_loss: 0.5541 - val_accuracy: 0.7200
Epoch 13/45
```

```
63/63 [==============================] - 47s 748ms/step - loss: 0.5202 -
accuracy: 0.7625 - val_loss: 0.5239 - val_accuracy: 0.7460
Epoch 14/45
63/63 [==============================] - 47s 740ms/step - loss: 0.5167 -
accuracy: 0.7505 - val_loss: 0.6480 - val_accuracy: 0.6910
Epoch 15/45
63/63 [==============================] - 47s 747ms/step - loss: 0.5192 -
accuracy: 0.7450 - val_loss: 0.5331 - val_accuracy: 0.7320
Epoch 16/45
63/63 [==============================] - 47s 743ms/step - loss: 0.4957 -
accuracy: 0.7645 - val_loss: 0.5934 - val_accuracy: 0.7250
Epoch 17/45
63/63 [==============================] - 47s 745ms/step - loss: 0.4875 -
accuracy: 0.7605 - val_loss: 0.6310 - val_accuracy: 0.7470
Epoch 18/45
63/63 [==============================] - 47s 750ms/step - loss: 0.4869 -
accuracy: 0.7750 - val_loss: 0.5260 - val_accuracy: 0.7760
Epoch 19/45
63/63 [==============================] - 47s 745ms/step - loss: 0.4682 -
accuracy: 0.7845 - val_loss: 0.4882 - val_accuracy: 0.7840
Epoch 20/45
63/63 [==============================] - 47s 743ms/step - loss: 0.4687 -
accuracy: 0.7785 - val_loss: 0.4578 - val_accuracy: 0.7940
Epoch 21/45
63/63 [==============================] - 47s 747ms/step - loss: 0.4672 -
accuracy: 0.7850 - val_loss: 0.4628 - val_accuracy: 0.7920
Epoch 22/45
63/63 [==============================] - 47s 747ms/step - loss: 0.4405 -
accuracy: 0.8045 - val_loss: 0.4707 - val_accuracy: 0.7860
Epoch 23/45
63/63 [==============================] - 47s 750ms/step - loss: 0.4471 -
accuracy: 0.7800 - val_loss: 0.4806 - val_accuracy: 0.7860
Epoch 24/45
63/63 [==============================] - 47s 742ms/step - loss: 0.4304 -
accuracy: 0.8065 - val_loss: 0.7241 - val_accuracy: 0.6730
Epoch 25/45
63/63 [==============================] - 47s 747ms/step - loss: 0.4344 -
accuracy: 0.7920 - val_loss: 0.4272 - val_accuracy: 0.8180
Epoch 26/45
63/63 [==============================] - 47s 746ms/step - loss: 0.4119 -
accuracy: 0.8170 - val_loss: 0.4968 - val_accuracy: 0.7980
Epoch 27/45
63/63 [==============================] - 47s 748ms/step - loss: 0.4060 -
accuracy: 0.8140 - val_loss: 0.4344 - val_accuracy: 0.8150
Epoch 28/45
63/63 [==============================] - 47s 746ms/step - loss: 0.4196 -
accuracy: 0.8070 - val_loss: 0.4894 - val_accuracy: 0.7980
Epoch 29/45
```

```
63/63 [==============================] - 47s 743ms/step - loss: 0.4063 -
accuracy: 0.8240 - val_loss: 0.5169 - val_accuracy: 0.7650
Epoch 30/45
63/63 [==============================] - 47s 748ms/step - loss: 0.3670 -
accuracy: 0.8360 - val_loss: 0.4758 - val_accuracy: 0.8190
Epoch 31/45
63/63 [==============================] - 47s 747ms/step - loss: 0.3940 -
accuracy: 0.8235 - val_loss: 0.5819 - val_accuracy: 0.7850
Epoch 32/45
63/63 [==============================] - 47s 743ms/step - loss: 0.3843 -
accuracy: 0.8310 - val_loss: 0.4293 - val_accuracy: 0.8140
Epoch 33/45
63/63 [==============================] - 47s 746ms/step - loss: 0.3713 -
accuracy: 0.8325 - val_loss: 0.4447 - val_accuracy: 0.8090
Epoch 34/45
63/63 [==============================] - 47s 745ms/step - loss: 0.3788 -
accuracy: 0.8385 - val_loss: 0.5805 - val_accuracy: 0.7680
Epoch 35/45
63/63 [==============================] - 47s 750ms/step - loss: 0.3473 -
accuracy: 0.8465 - val_loss: 0.4287 - val_accuracy: 0.8280
Epoch 36/45
63/63 [==============================] - 47s 748ms/step - loss: 0.3319 -
accuracy: 0.8480 - val_loss: 0.4798 - val_accuracy: 0.8050
Epoch 37/45
63/63 [==============================] - 47s 747ms/step - loss: 0.3308 -
accuracy: 0.8600 - val_loss: 0.3999 - val_accuracy: 0.8290
Epoch 38/45
63/63 [==============================] - 47s 745ms/step - loss: 0.3331 -
accuracy: 0.8590 - val_loss: 0.5823 - val_accuracy: 0.7700
Epoch 39/45
63/63 [==============================] - 47s 750ms/step - loss: 0.3450 -
accuracy: 0.8580 - val_loss: 0.4573 - val_accuracy: 0.8190
Epoch 40/45
63/63 [==============================] - 47s 745ms/step - loss: 0.3291 -
accuracy: 0.8655 - val_loss: 0.4326 - val_accuracy: 0.8270
Epoch 41/45
63/63 [==============================] - 47s 748ms/step - loss: 0.3089 -
accuracy: 0.8770 - val_loss: 0.4746 - val_accuracy: 0.8000
Epoch 42/45
63/63 [==============================] - 47s 745ms/step - loss: 0.2976 -
accuracy: 0.8730 - val_loss: 0.5727 - val_accuracy: 0.8080
Epoch 43/45
63/63 [==============================] - 47s 746ms/step - loss: 0.3086 -
accuracy: 0.8775 - val_loss: 0.4007 - val_accuracy: 0.8400
Epoch 44/45
63/63 [==============================] - 47s 742ms/step - loss: 0.3114 -
accuracy: 0.8755 - val_loss: 0.4103 - val_accuracy: 0.8260
Epoch 45/45
```

```
63/63 [==============================] - 47s 745ms/step - loss: 0.2869 -
accuracy: 0.8855 - val_loss: 0.4800 - val_accuracy: 0.8250
```

Assessment of the Model Using the Test Data

[23]:
```
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 [==============================] - 10s 159ms/step - loss: 0.4255 -
accuracy: 0.8135
Test accuracy: 0.813
```

Q3. Increasing Training sample size to 3100 samples

The number of training samples has been expanded to 3100, and the impact on model performance is discussed in the provided summary.

[24]:
```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_02")

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)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            src = original_dir / fname
            dst = dir / fname
            shutil.copyfile(src, dst)

make_subset("train", start_index=0, end_index=2100)
make_subset("validation", start_index=2100, end_index=2600)
make_subset("test", start_index=2600, end_index=3100)
```

Defining a new convnet that includes image augmentation and dropout

[25]:
```
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"])
```

Now, we proceed to train the convnet that has been regularized.

[26]:
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=20,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/20
63/63 [==============================] - 49s 755ms/step - loss: 0.7548 -
accuracy: 0.4920 - val_loss: 0.6919 - val_accuracy: 0.5030
Epoch 2/20
63/63 [==============================] - 47s 751ms/step - loss: 0.6993 -
accuracy: 0.5115 - val_loss: 0.6876 - val_accuracy: 0.5360
Epoch 3/20
63/63 [==============================] - 47s 748ms/step - loss: 0.6978 -
accuracy: 0.5535 - val_loss: 0.6811 - val_accuracy: 0.5560
Epoch 4/20
63/63 [==============================] - 47s 750ms/step - loss: 0.6664 -
accuracy: 0.6055 - val_loss: 0.6350 - val_accuracy: 0.6570
Epoch 5/20
63/63 [==============================] - 47s 742ms/step - loss: 0.6583 -
accuracy: 0.6355 - val_loss: 0.6771 - val_accuracy: 0.6010
Epoch 6/20
63/63 [==============================] - 47s 744ms/step - loss: 0.6433 -
accuracy: 0.6460 - val_loss: 0.6111 - val_accuracy: 0.6590
Epoch 7/20
63/63 [==============================] - 47s 744ms/step - loss: 0.6218 -
accuracy: 0.6550 - val_loss: 0.5932 - val_accuracy: 0.6990
Epoch 8/20
63/63 [==============================] - 47s 748ms/step - loss: 0.5977 -
```

```
accuracy: 0.6840 - val_loss: 0.5715 - val_accuracy: 0.6990
Epoch 9/20
63/63 [==============================] - 47s 745ms/step - loss: 0.5801 -
accuracy: 0.7030 - val_loss: 0.5786 - val_accuracy: 0.6920
Epoch 10/20
63/63 [==============================] - 47s 744ms/step - loss: 0.5806 -
accuracy: 0.6990 - val_loss: 0.7779 - val_accuracy: 0.6100
Epoch 11/20
63/63 [==============================] - 47s 745ms/step - loss: 0.5682 -
accuracy: 0.7140 - val_loss: 0.5766 - val_accuracy: 0.7150
Epoch 12/20
63/63 [==============================] - 47s 743ms/step - loss: 0.5529 -
accuracy: 0.7375 - val_loss: 0.7651 - val_accuracy: 0.6420
Epoch 13/20
63/63 [==============================] - 47s 747ms/step - loss: 0.5450 -
accuracy: 0.7335 - val_loss: 0.6488 - val_accuracy: 0.6840
Epoch 14/20
63/63 [==============================] - 47s 750ms/step - loss: 0.5450 -
accuracy: 0.7350 - val_loss: 0.5149 - val_accuracy: 0.7420
Epoch 15/20
63/63 [==============================] - 47s 747ms/step - loss: 0.5254 -
accuracy: 0.7395 - val_loss: 0.4938 - val_accuracy: 0.7630
Epoch 16/20
63/63 [==============================] - 47s 744ms/step - loss: 0.5226 -
accuracy: 0.7390 - val_loss: 0.5729 - val_accuracy: 0.6760
Epoch 17/20
63/63 [==============================] - 47s 752ms/step - loss: 0.5022 -
accuracy: 0.7630 - val_loss: 0.4732 - val_accuracy: 0.7790
Epoch 18/20
63/63 [==============================] - 47s 739ms/step - loss: 0.4907 -
accuracy: 0.7725 - val_loss: 0.5743 - val_accuracy: 0.7230
Epoch 19/20
63/63 [==============================] - 47s 746ms/step - loss: 0.4887 -
accuracy: 0.7730 - val_loss: 0.5555 - val_accuracy: 0.7540
Epoch 20/20
63/63 [==============================] - 47s 747ms/step - loss: 0.4712 -
accuracy: 0.7770 - val_loss: 0.4672 - val_accuracy: 0.7980
```

Evaluating the test dataset

[27]:
```
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 [==============================] - 10s 160ms/step - loss: 0.5118 -
accuracy: 0.7660
Test accuracy: 0.766
```

Q4.Fine Tuning of the pretrained models

We will adjust the pretrained model by experimenting with various sizes of training samples and then assess its effectiveness based on the performance of the models we previously constructed.

Pre-Trained Model with 1000 Training Samples

Initializing the VGG16 convolutional base and setting it to a non-trainable state

Instantiating the VGG16 convolutional base

[28]:
```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

[29]:
```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_4 (InputLayer)        [(None, 180, 180, 3)]     0

block1_conv1 (Conv2D)       (None, 180, 180, 64)      1792

block1_conv2 (Conv2D)       (None, 180, 180, 64)      36928

block1_pool (MaxPooling2D)  (None, 90, 90, 64)        0

block2_conv1 (Conv2D)       (None, 90, 90, 128)       73856

block2_conv2 (Conv2D)       (None, 90, 90, 128)       147584

block2_pool (MaxPooling2D)  (None, 45, 45, 128)       0

block3_conv1 (Conv2D)       (None, 45, 45, 256)       295168

block3_conv2 (Conv2D)       (None, 45, 45, 256)       590080

block3_conv3 (Conv2D)       (None, 45, 45, 256)       590080

block3_pool (MaxPooling2D)  (None, 22, 22, 256)       0

block4_conv1 (Conv2D)       (None, 22, 22, 512)       1180160

block4_conv2 (Conv2D)       (None, 22, 22, 512)       2359808

block4_conv3 (Conv2D)       (None, 22, 22, 512)       2359808

```

```
block4_pool (MaxPooling2D)   (None, 11, 11, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 5, 5, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Fast feature extraction without data augmentation **Extracting the VGG16 features and corresponding labels**

[30]:
```
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)
```

[31]:
```
train_features.shape
```

[31]:
```
(2000, 5, 5, 512)
```

Defining and training the densely connected classifier

[33]:
```
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=30,
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

```
Epoch 1/30
63/63 [==============================] - 2s 33ms/step - loss: 15.5443 -
accuracy: 0.9190 - val_loss: 5.5484 - val_accuracy: 0.9650
Epoch 2/30
63/63 [==============================] - 2s 33ms/step - loss: 2.9176 - accuracy:
0.9765 - val_loss: 8.3800 - val_accuracy: 0.9630
Epoch 3/30
63/63 [==============================] - 2s 33ms/step - loss: 1.9215 - accuracy:
0.9870 - val_loss: 4.2566 - val_accuracy: 0.9730
Epoch 4/30
63/63 [==============================] - 2s 32ms/step - loss: 1.3324 - accuracy:
0.9885 - val_loss: 5.3442 - val_accuracy: 0.9680
Epoch 5/30
63/63 [==============================] - 2s 33ms/step - loss: 0.6369 - accuracy:
0.9945 - val_loss: 5.4410 - val_accuracy: 0.9730
Epoch 6/30
63/63 [==============================] - 2s 33ms/step - loss: 0.5982 - accuracy:
0.9950 - val_loss: 4.4904 - val_accuracy: 0.9730
Epoch 7/30
63/63 [==============================] - 2s 33ms/step - loss: 0.7402 - accuracy:
0.9935 - val_loss: 3.9737 - val_accuracy: 0.9790
Epoch 8/30
63/63 [==============================] - 2s 32ms/step - loss: 0.1541 - accuracy:
0.9960 - val_loss: 3.8200 - val_accuracy: 0.9760
Epoch 9/30
63/63 [==============================] - 2s 32ms/step - loss: 0.2506 - accuracy:
0.9965 - val_loss: 3.9778 - val_accuracy: 0.9770
Epoch 10/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0647 - accuracy:
0.9990 - val_loss: 3.3827 - val_accuracy: 0.9770
Epoch 11/30
63/63 [==============================] - 2s 33ms/step - loss: 0.4268 - accuracy:
0.9965 - val_loss: 6.7100 - val_accuracy: 0.9660
```

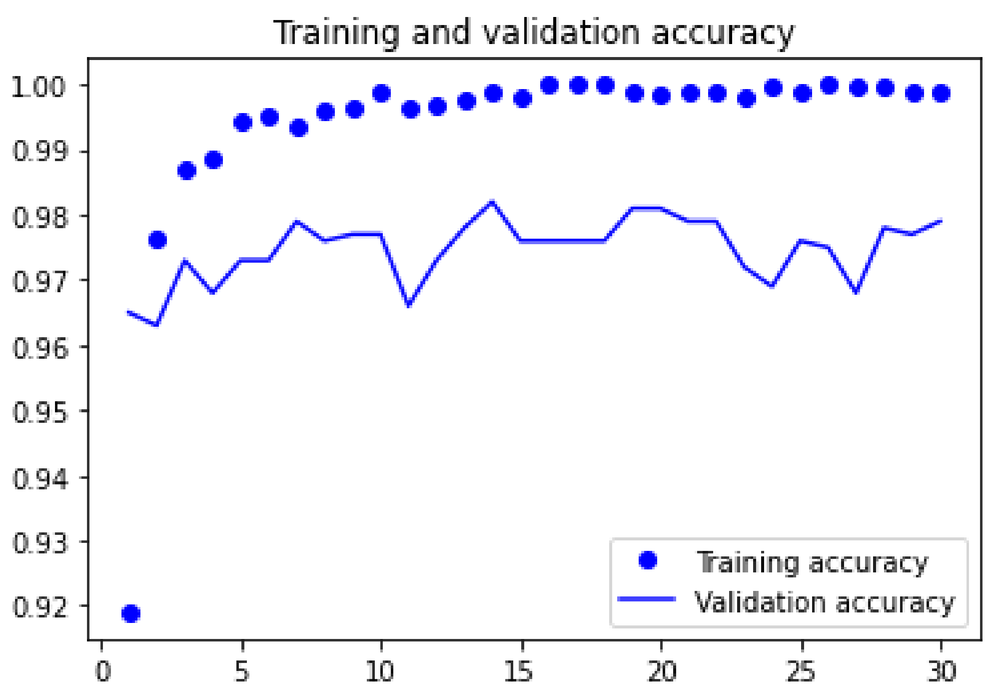
```
Epoch 12/30
63/63 [==============================] - 2s 33ms/step - loss: 0.2706 - accuracy:
0.9970 - val_loss: 4.8512 - val_accuracy: 0.9730
Epoch 13/30
63/63 [==============================] - 2s 33ms/step - loss: 0.1408 - accuracy:
0.9975 - val_loss: 4.6766 - val_accuracy: 0.9780
Epoch 14/30
63/63 [==============================] - 2s 32ms/step - loss: 0.1350 - accuracy:
0.9990 - val_loss: 4.6217 - val_accuracy: 0.9820
Epoch 15/30
63/63 [==============================] - 2s 33ms/step - loss: 0.3154 - accuracy:
0.9980 - val_loss: 4.4168 - val_accuracy: 0.9760
Epoch 16/30
63/63 [==============================] - 2s 33ms/step - loss: 2.0408e-31 -
accuracy: 1.0000 - val_loss: 4.4168 - val_accuracy: 0.9760
Epoch 17/30
63/63 [==============================] - 2s 33ms/step - loss: 1.4306e-15 -
accuracy: 1.0000 - val_loss: 4.4168 - val_accuracy: 0.9760
Epoch 18/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0000e+00 -
accuracy: 1.0000 - val_loss: 4.4168 - val_accuracy: 0.9760
Epoch 19/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0043 - accuracy:
0.9990 - val_loss: 4.1333 - val_accuracy: 0.9810
Epoch 20/30
63/63 [==============================] - 2s 33ms/step - loss: 0.1289 - accuracy:
0.9985 - val_loss: 4.3503 - val_accuracy: 0.9810
Epoch 21/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0066 - accuracy:
0.9990 - val_loss: 4.4840 - val_accuracy: 0.9790
Epoch 22/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0709 - accuracy:
0.9990 - val_loss: 4.3030 - val_accuracy: 0.9790
Epoch 23/30
63/63 [==============================] - 2s 33ms/step - loss: 0.1636 - accuracy:
0.9980 - val_loss: 5.5896 - val_accuracy: 0.9720
Epoch 24/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0022 - accuracy:
0.9995 - val_loss: 5.9060 - val_accuracy: 0.9690
Epoch 25/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0603 - accuracy:
0.9990 - val_loss: 5.0379 - val_accuracy: 0.9760
Epoch 26/30
63/63 [==============================] - 2s 33ms/step - loss: 4.9801e-09 -
accuracy: 1.0000 - val_loss: 5.5193 - val_accuracy: 0.9750
Epoch 27/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0435 - accuracy:
0.9995 - val_loss: 8.3777 - val_accuracy: 0.9680
```

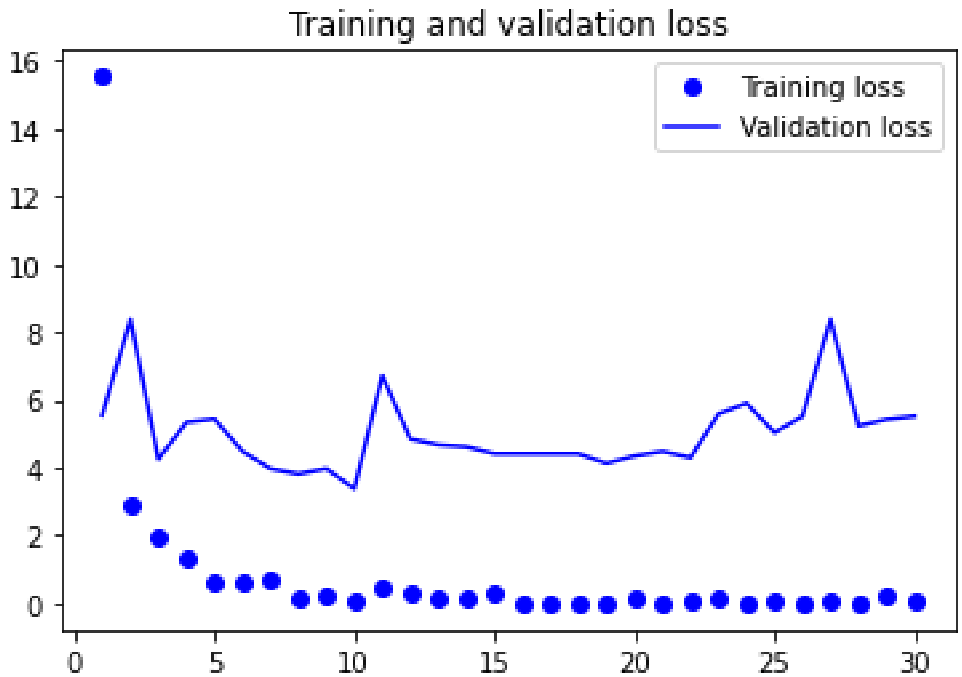
```
Epoch 28/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0068 - accuracy:
0.9995 - val_loss: 5.2382 - val_accuracy: 0.9780
Epoch 29/30
63/63 [==============================] - 2s 33ms/step - loss: 0.2427 - accuracy:
0.9990 - val_loss: 5.4280 - val_accuracy: 0.9770
Epoch 30/30
63/63 [==============================] - 2s 33ms/step - loss: 0.0633 - accuracy:
0.9990 - val_loss: 5.5203 - val_accuracy: 0.9790
```

Plotting the results

[34]:

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





Feature extraction together with data augmentation **Instantiating and freezing the VGG16 convolutional base**

[35]:
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

Printing the list of trainable weights before and after freezing

[36]:
```
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
```

[37]:
```
conv_base.trainable = False
print("This is the number of trainable weights "
      "after freezing the conv base:", len(conv_base.trainable_weights))
```

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

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

[38]:
```
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"])
```

[39]:
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/10
63/63 [==============================] - 168s 3s/step - loss: 18.4987 -
accuracy: 0.8920 - val_loss: 14.4387 - val_accuracy: 0.9230
Epoch 2/10
63/63 [==============================] - 167s 3s/step - loss: 6.1704 - accuracy:
0.9530 - val_loss: 8.7896 - val_accuracy: 0.9510
Epoch 3/10
63/63 [==============================] - 167s 3s/step - loss: 6.2106 - accuracy:
0.9510 - val_loss: 4.6838 - val_accuracy: 0.9730
Epoch 4/10
63/63 [==============================] - 167s 3s/step - loss: 5.0829 - accuracy:
0.9590 - val_loss: 7.3436 - val_accuracy: 0.9640
Epoch 5/10
63/63 [==============================] - 167s 3s/step - loss: 4.5701 - accuracy:
```

```
0.9620 - val_loss: 4.5137 - val_accuracy: 0.9720
Epoch 6/10
63/63 [==============================] - 167s 3s/step - loss: 3.9205 - accuracy:
0.9685 - val_loss: 3.1031 - val_accuracy: 0.9790
Epoch 7/10
63/63 [==============================] - 167s 3s/step - loss: 3.7451 - accuracy:
0.9685 - val_loss: 4.0067 - val_accuracy: 0.9780
Epoch 8/10
63/63 [==============================] - 166s 3s/step - loss: 3.0027 - accuracy:
0.9730 - val_loss: 6.3938 - val_accuracy: 0.9700
Epoch 9/10
63/63 [==============================] - 167s 3s/step - loss: 3.2352 - accuracy:
0.9770 - val_loss: 4.9734 - val_accuracy: 0.9720
Epoch 10/10
63/63 [==============================] - 166s 3s/step - loss: 2.5900 - accuracy:
0.9770 - val_loss: 4.2494 - val_accuracy: 0.9760
```

Evaluating the model on the test set

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

```
 5/63 [=>…] - ETA: 1:41 - loss: 1.6594 - accuracy:
0.9875
```

0.0.4 Fine-tuning a pretrained model

[39]:
```
conv_base.summary()
```

```
Model: "vgg16"

_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_6 (InputLayer)        [(None, None, None, 3)]   0

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

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

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

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

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

_________________________________________________________________
block2_pool (MaxPooling2D)  (None, None, None, 128)   0
```

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

Freezing all layers until the fourth from the last

[40]:
```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

Fine-tuning the model

[41]:
```
model.compile(loss="binary_crossentropy",
              optimizer=keras.optimizers.RMSprop(learning_rate=1e-5),
              metrics=["accuracy"])

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

```
history = model.fit(
    train_dataset,
    epochs=10,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
63/63 [==============================] - 192s 3s/step - loss: 2.5842 - accuracy:
0.9730 - val_loss: 2.7219 - val_accuracy: 0.9780
```

[42]:
```
model = keras.models.load_model("fine_tuning.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
63/63 [==============================] - 107s 2s/step - loss: 2.9272 - accuracy:
0.9735
Test accuracy: 0.974
```

[43]:
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

[44]:
```
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
```

[46]:
```
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=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_dataset,
    epochs=5,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
63/63 [==============================] - 193s 3s/step - loss: 3.6172 - accuracy:
0.7560 - val_loss: 0.5617 - val_accuracy: 0.9330
```

[47]:
```
model = keras.models.load_model("fine_tuning2.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
63/63 [==============================] - 107s 2s/step - loss: 0.6919 - accuracy:
0.9325
Test accuracy: 0.933
```

[48]:
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

[49]:
```
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
```

[51]:
```
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=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_dataset,
    epochs=5,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
63/63 [==============================] - 192s 3s/step - loss: 3.8077 - accuracy:
0.7520 - val_loss: 0.8551 - val_accuracy: 0.9150
```

[53]:
```
model = keras.models.load_model("fine_tuning3.keras")
test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
63/63 [==============================] - 107s 2s/step - loss: 0.8319 - accuracy:
0.9205
Test accuracy: 0.920
```