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

```
Mounted at /content/drive
```

```
!unzip /content/drive/MyDrive/dogs-vs-cats.zip
```

```
Archive:  /content/drive/MyDrive/dogs-vs-cats.zip
  inflating: sampleSubmission.csv
  inflating: test1.zip
  inflating: train.zip
```

```
!unzip /content/train.zip
```

```
  inflating: train/dog.9981.jpg
  inflating: train/dog.9982.jpg
  inflating: train/dog.9983.jpg
  inflating: train/dog.9984.jpg
  inflating: train/dog.9985.jpg
  inflating: train/dog.9986.jpg
  inflating: train/dog.9987.jpg
  inflating: train/dog.9988.jpg
  inflating: train/dog.9989.jpg
  inflating: train/dog.999.jpg
  inflating: train/dog.9990.jpg
  inflating: train/dog.9991.jpg
  inflating: train/dog.9992.jpg
  inflating: train/dog.9993.jpg
  inflating: train/dog.9994.jpg
  inflating: train/dog.9995.jpg
  inflating: train/dog.9996.jpg
  inflating: train/dog.9997.jpg
  inflating: train/dog.9998.jpg
  inflating: train/dog.9999.jpg
```

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

We are initially taking the train sample of 1000 by taking first 1000 values in the dataset. Taking 500 validation samples starting from 1000 to 1500 values in the dataset, taking 500 test samples starting from 1500 to 2000 values in the dataset.

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_1")

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

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

Here we are preprocessing the data

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

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

```
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,)
```

Here we are taking 32 as batch size for the data

```
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)
```

Reshaping the dataset using dataset.map

```
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)
```

```
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,)
```

Using Keras with convolutions and Maxpooling: Creates convolutions kernel that is convolved with the layer

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

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

Configuring the model for training using biary crossentropy as loss function, adam optimizer and accuracy to measure the performance of the model.

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

```
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
dropout (Dropout)	(None, 12544)	0
dense (Dense)	(None, 1)	12,545

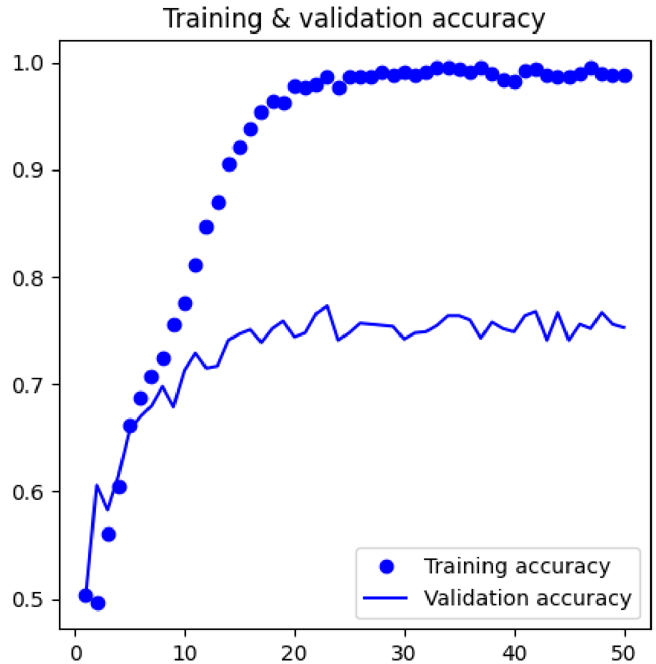
A record of the training measurements and loss values at different epochs, along with validation metrics and loss values, is called a history attribute.

```
from keras.callbacks import ModelCheckpoint, EarlyStopping

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

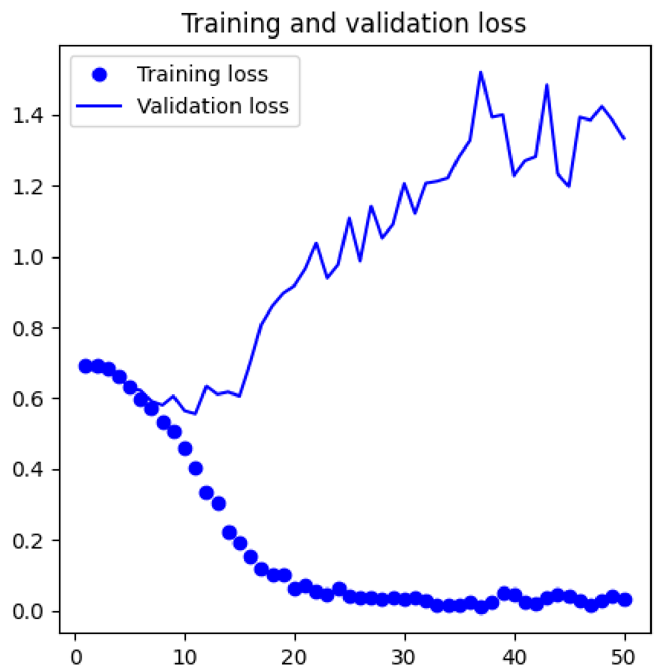
```
Epoch 22/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 91ms/step - accuracy: 0.9783 - loss: 0.0623 - val_accuracy: 0.7660 - val_loss: 1.0373
Epoch 23/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.9829 - loss: 0.0430 - val_accuracy: 0.7730 - val_loss: 0.9390
Epoch 24/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 58ms/step - accuracy: 0.9879 - loss: 0.0351 - val_accuracy: 0.7410 - val_loss: 0.9775
Epoch 25/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 82ms/step - accuracy: 0.9846 - loss: 0.0411 - val_accuracy: 0.7480 - val_loss: 1.1086
Epoch 26/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.9866 - loss: 0.0348 - val_accuracy: 0.7570 - val_loss: 0.9876
Epoch 27/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.9877 - loss: 0.0422 - val_accuracy: 0.7560 - val_loss: 1.1421
Epoch 28/50
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 88ms/step - accuracy: 0.9909 - loss: 0.0321 - val_accuracy: 0.7560 - val_loss: 1.3836
Epoch 50/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 56ms/step - accuracy: 0.9889 - loss: 0.0375 - val_accuracy: 0.7530 - val_loss: 1.3339
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(5, 5))
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 & validation accuracy")
plt.legend()
plt.figure()
plt.figure(figsize=(5, 5))
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()
```




```
<Figure size 640x480 with 0 Axes>
```



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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.7092 - loss: 0.5987
Test accuracy: 0.701
```

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

Here we are incresing the train sample size to 1500 by taking the values from 2000 to 3500 and keeping the validation and test values constant i.e., 500.

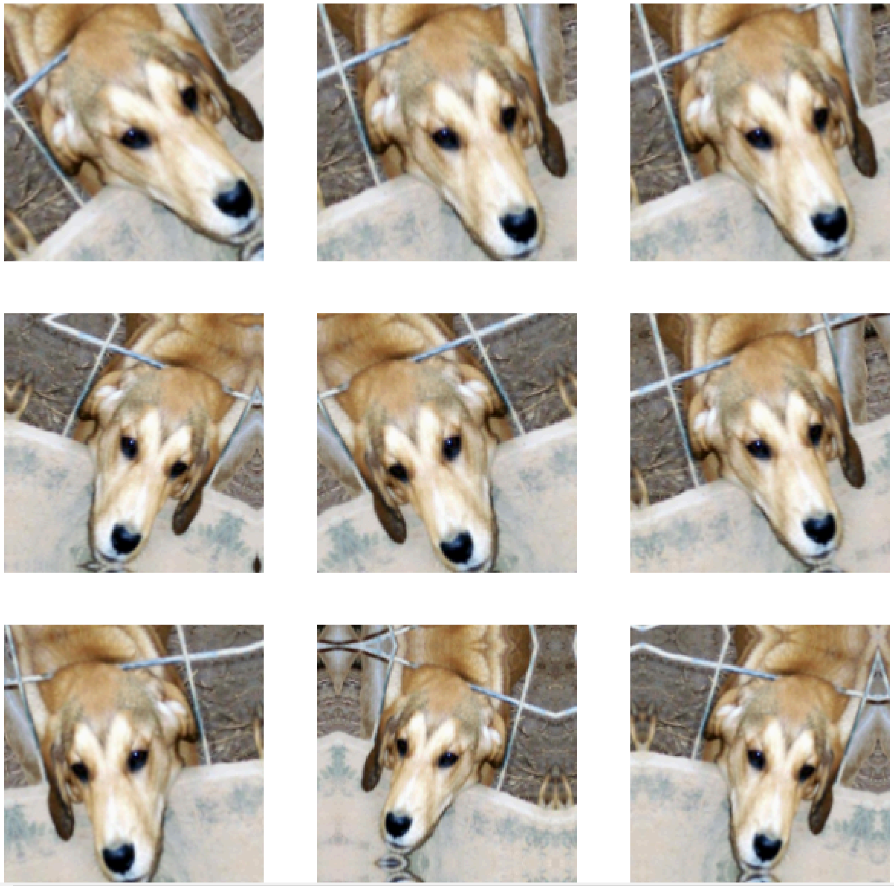
```
import os, shutil, pathlib

shutil.rmtree("./cats_vs_dogs_small_Q2", ignore_errors=True)

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q2")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)
#Creating training, Test and validation sets.
#Training has 1500 samples, test has 500 samples and validation has 500 samples.
make_subset("train", start_index=2000, end_index=3500)
make_subset("validation", start_index=3501, end_index=4001)
make_subset("test", start_index=4002, end_index=4502)
```

Here we are using the data augmentation technique to optimize the model performance as we are dealing with large datasets (increased the train sample size to 1500)

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

Display of few sample images in the dataset.

```
plt.figure(figsize=(7.5,7.5 ))
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")
```



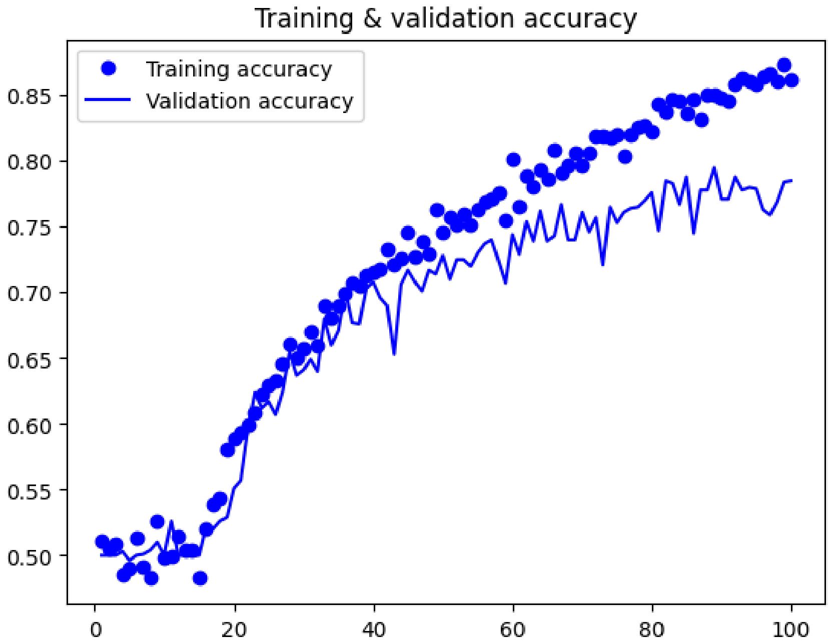
Using Data Augmentation and Dropout to optimize the model. Dropout layer only applies when training is set to True such that no values are dropped during inference

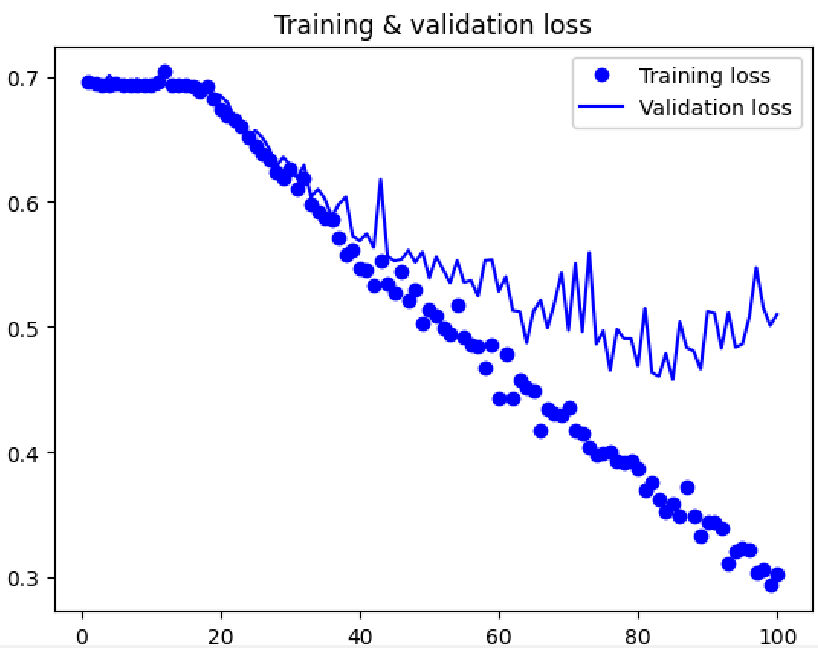
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
```

```
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="adam",
              metrics=["accuracy"])

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

```
Epoch 83/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 96ms/step - accuracy: 0.8579 - loss: 0.3461 - val_accuracy: 0.7830 - val_loss: 0.4605
Epoch 84/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 57ms/step - accuracy: 0.8433 - loss: 0.3430 - val_accuracy: 0.7670 - val_loss: 0.4788
Epoch 85/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 73ms/step - accuracy: 0.8295 - loss: 0.3624 - val_accuracy: 0.7880 - val_loss: 0.4582
Epoch 86/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 88ms/step - accuracy: 0.8546 - loss: 0.3402 - val_accuracy: 0.7450 - val_loss: 0.5041
Epoch 87/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8353 - loss: 0.3646 - val_accuracy: 0.7780 - val_loss: 0.4834
Epoch 88/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 62ms/step - accuracy: 0.8704 - loss: 0.3156 - val_accuracy: 0.7780 - val_loss: 0.4808
Epoch 89/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 100ms/step - accuracy: 0.8515 - loss: 0.3283 - val_accuracy: 0.7950 - val_loss: 0.4665
Epoch 90/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 62ms/step - accuracy: 0.8649 - loss: 0.3171 - val_accuracy: 0.7710 - val_loss: 0.5127
Epoch 91/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.8547 - loss: 0.3281 - val_accuracy: 0.7710 - val_loss: 0.5110
Epoch 92/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 89ms/step - accuracy: 0.8693 - loss: 0.3142 - val_accuracy: 0.7880 - val_loss: 0.4830
Epoch 93/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.8799 - loss: 0.2873 - val_accuracy: 0.7780 - val_loss: 0.5115
Epoch 94/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 62ms/step - accuracy: 0.8641 - loss: 0.3233 - val_accuracy: 0.7800 - val_loss: 0.4839
Epoch 95/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 91ms/step - accuracy: 0.8676 - loss: 0.3115 - val_accuracy: 0.7790 - val_loss: 0.4863
Epoch 96/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 57ms/step - accuracy: 0.8673 - loss: 0.3171 - val_accuracy: 0.7630 - val_loss: 0.5077
Epoch 97/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.8692 - loss: 0.2899 - val_accuracy: 0.7590 - val_loss: 0.5473
Epoch 98/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 91ms/step - accuracy: 0.8653 - loss: 0.2938 - val_accuracy: 0.7690 - val_loss: 0.5158
Epoch 99/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 56ms/step - accuracy: 0.8854 - loss: 0.2773 - val_accuracy: 0.7840 - val_loss: 0.5011
Epoch 100/100
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 101ms/step - accuracy: 0.8689 - loss: 0.3007 - val_accuracy: 0.7850 - val_loss: 0.5102
```

```
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 & 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 & validation loss")
plt.legend()
plt.show()
```






```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
```

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.7922 - loss: 0.4741
Test accuracy: 0.786
```

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

Increasing the training sample size to 2000 taking the values from 4000 to 6000 from dataset and keeping the validation and test and validation sample sizes to 500 only.

```
original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small_Q3")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)
#Creating training, Test and validation sets.
#Training has 2000 samples, test has 500 samples and validation has 500 samples.
make_subset("train", start_index=4000, end_index=6000)
make_subset("validation", start_index=6001, end_index=6501)
make_subset("test", start_index=6502, end_index=7002)
```

A new convent:

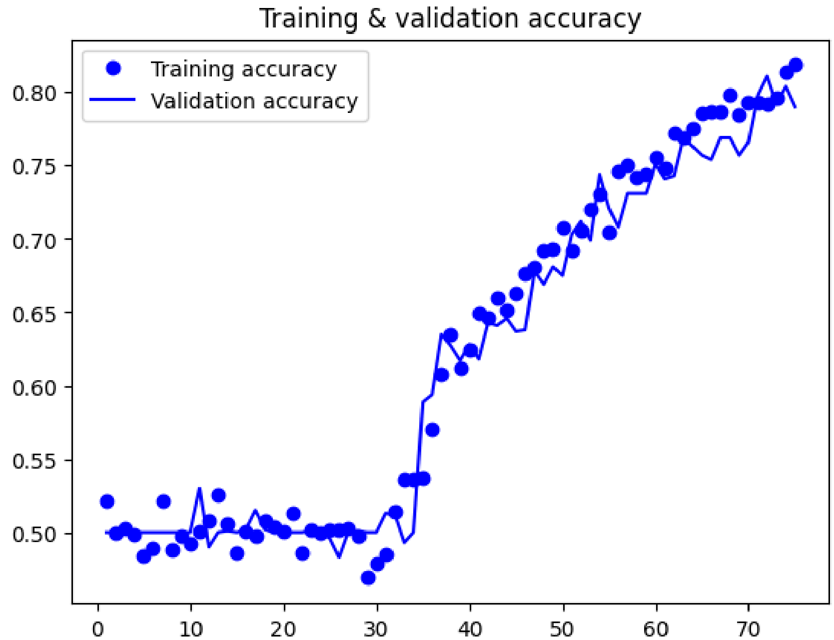
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
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)
```

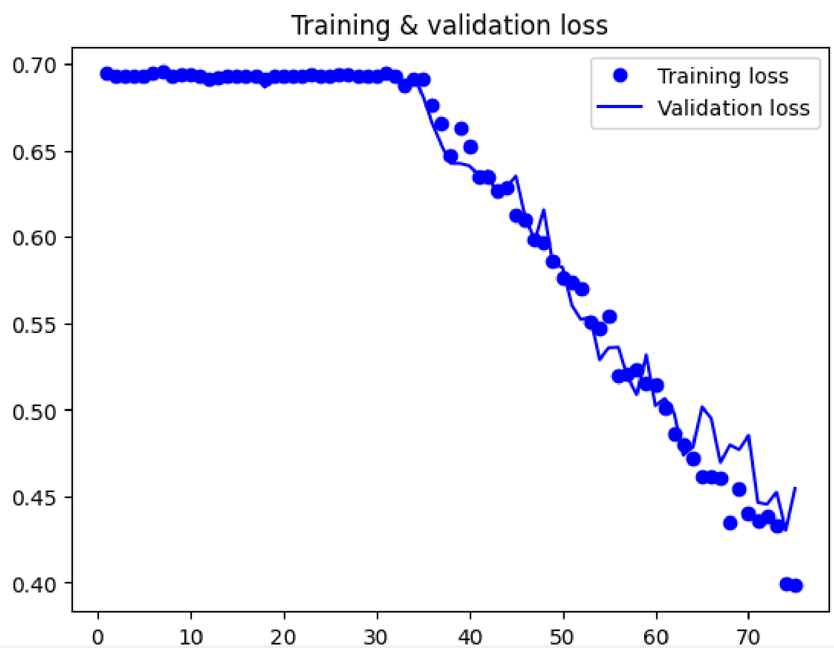
```
model = keras.Model(inputs=inputs, outputs=outputs)

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


callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation1.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=75,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 92ms/step - accuracy: 0.7888 - loss: 0.4573 - val_accuracy: 0.7630 - val_loss: 0.4783
Epoch 65/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 68ms/step - accuracy: 0.7963 - loss: 0.4440 - val_accuracy: 0.7570 - val_loss: 0.5016
Epoch 66/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.7888 - loss: 0.4675 - val_accuracy: 0.7540 - val_loss: 0.4952
Epoch 67/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 75ms/step - accuracy: 0.7906 - loss: 0.4566 - val_accuracy: 0.7690 - val_loss: 0.4696
Epoch 68/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 112ms/step - accuracy: 0.7981 - loss: 0.4386 - val_accuracy: 0.7690 - val_loss: 0.4798
Epoch 69/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 7s 57ms/step - accuracy: 0.7817 - loss: 0.4501 - val_accuracy: 0.7570 - val_loss: 0.4770
Epoch 70/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 101ms/step - accuracy: 0.7945 - loss: 0.4303 - val_accuracy: 0.7660 - val_loss: 0.4850
Epoch 71/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 64ms/step - accuracy: 0.7894 - loss: 0.4389 - val_accuracy: 0.7980 - val_loss: 0.4464
Epoch 72/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 5s 84ms/step - accuracy: 0.8057 - loss: 0.4164 - val_accuracy: 0.8110 - val_loss: 0.4454
Epoch 73/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 62ms/step - accuracy: 0.7899 - loss: 0.4272 - val_accuracy: 0.7920 - val_loss: 0.4523
Epoch 74/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.8189 - loss: 0.3931 - val_accuracy: 0.8040 - val_loss: 0.4303
Epoch 75/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 6s 94ms/step - accuracy: 0.8216 - loss: 0.3939 - val_accuracy: 0.7900 - val_loss: 0.4544
```

Graph of training and validation accuracy

```
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 & 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 & validation loss")
plt.legend()
plt.show()
```






```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation1.keras")
```

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.7936 - loss: 0.4714
Test accuracy: 0.788
```

In the beginning as we took only 1000 samples in the first question and we acheived an accuracy of 74% but the same when we saw above with increasing the sample size to double we received 83% accuracy, the problem was overfitting and hence we generalized the model. As there was overfitting we used techniques like data augmentation and dropout to generalize the model.

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

Using pretrained model with Feature extraction technique

Using the VGG16 convolutional base which describes the first several layers of the the architecture, which are in charge of taking hierarchical features out of input images.

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

```
convolution_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_4 (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

Feature extraction without data augmentation using a pretrained model

```
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 = convolution_base.predict(preprocessed_images)
```

```
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
```

```
train_features.shape
```

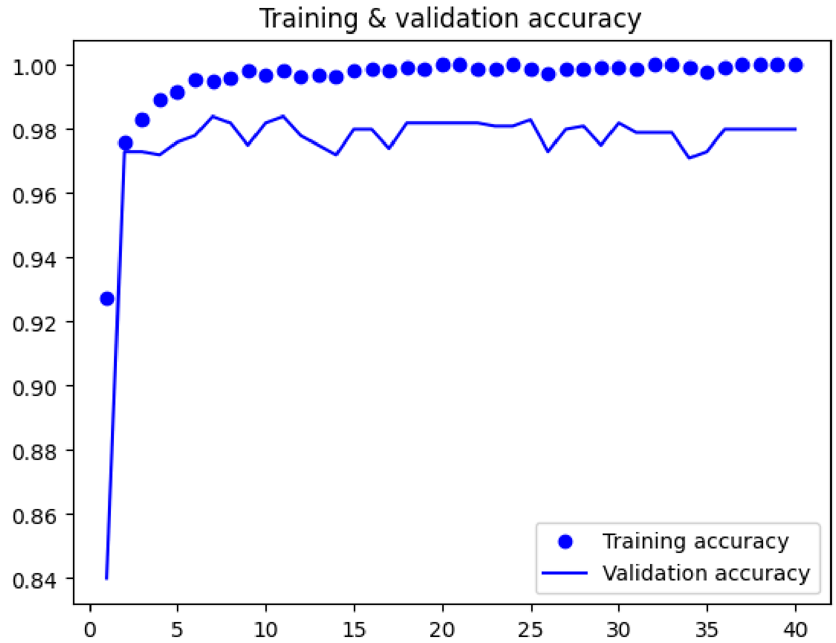
```
(2000, 5, 5, 512)
```

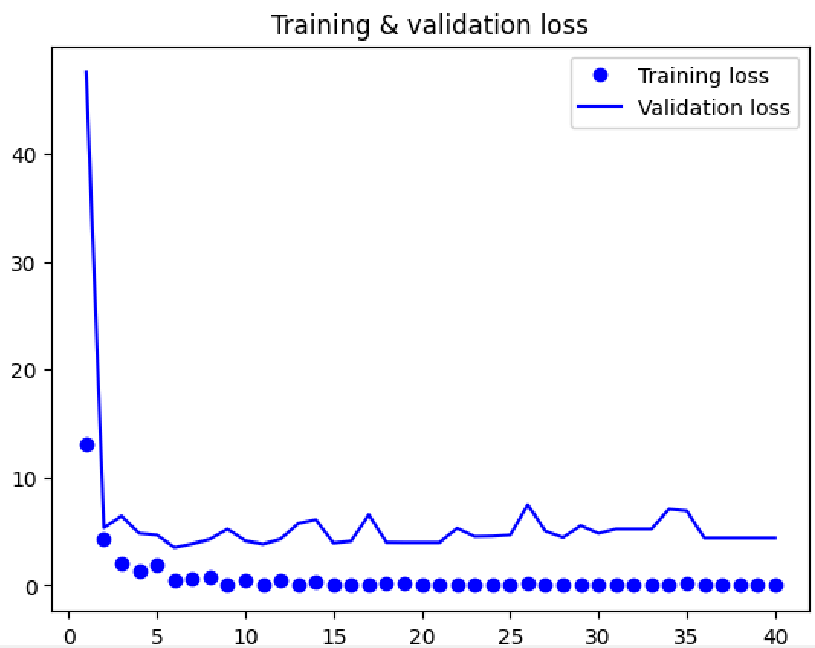
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
          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

callbacks = [
    keras.callbacks.ModelCheckpoint(
      filepath="feature_extraction.keras",
      save_best_only=True,
      monitor="val_loss")
]
history = model.fit(
    train_features, train_labels,
    epochs=40,
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

```
Epoch 19/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9970 - loss: 0.4450 - val_accuracy: 0.9820 - val_loss: 3.9779
Epoch 20/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 9.1143e-14 - val_accuracy: 0.9820 - val_loss: 3.9779
Epoch 21/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 8.0309e-13 - val_accuracy: 0.9820 - val_loss: 3.9779
Epoch 22/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9997 - loss: 0.0310 - val_accuracy: 0.9820 - val_loss: 5.3180
Epoch 23/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9996 - loss: 0.0400 - val_accuracy: 0.9810 - val_loss: 4.5393
Epoch 24/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 7.7287e-07 - val_accuracy: 0.9810 - val_loss: 4.5716
Epoch 25/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9991 - loss: 0.0401 - val_accuracy: 0.9830 - val_loss: 4.6812
Epoch 26/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9991 - loss: 0.0515 - val_accuracy: 0.9730 - val_loss: 7.4718
Epoch 27/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9983 - loss: 0.1135 - val_accuracy: 0.9800 - val_loss: 5.0380
Epoch 28/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9994 - loss: 0.0294 - val_accuracy: 0.9810 - val_loss: 4.4556
Epoch 29/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9997 - loss: 0.0088 - val_accuracy: 0.9750 - val_loss: 5.5643
Epoch 30/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9994 - loss: 0.0349 - val_accuracy: 0.9820 - val_loss: 4.8367
Epoch 31/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9989 - loss: 0.0392 - val_accuracy: 0.9790 - val_loss: 5.2452
Epoch 32/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 3.0856e-34 - val_accuracy: 0.9790 - val_loss: 5.2452
Epoch 33/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9790 - val_loss: 5.2452
Epoch 34/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 0.9710 - val_loss: 7.0786
Epoch 35/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9987 - loss: 0.3955 - val_accuracy: 0.9730 - val_loss: 6.9198
Epoch 36/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9989 - loss: 0.0312 - val_accuracy: 0.9800 - val_loss: 4.3969
Epoch 37/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9800 - val_loss: 4.3969
Epoch 38/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9800 - val_loss: 4.3969
Epoch 39/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 1.1448e-34 - val_accuracy: 0.9800 - val_loss: 4.3969
Epoch 40/40
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 0.9800 - val_loss: 4.3969
```

```
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 & 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 & validation loss")
plt.legend()
plt.show()
```





Freezing the VGG16 convolutional base as in feature extraction we freeze the initial trained base

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

```
convolution_base.trainable = True
print("The number of trainable weights required to use the convolution base before it freezes is as follows:", len(convolution_base.trainable_weights))
```

```
The number of trainable weights required to use the convolution base before it freezes is as follows: 26
```

```
convolution_base.trainable = False
print("After the convolution base is frozen, this is the total quantity of trainable weights:", len(convolution_base.trainable_weights))
```

```
After the convolution base is frozen, this is the total quantity of trainable weights: 0
```

Adding data augmentation:

```
augmentation2 = keras.Sequential(
[
layers.RandomFlip("horizontal"),
layers.RandomRotation(0.1),
layers.RandomZoom(0.2),
]
)
input22 = keras.Input(shape=(180, 180, 3))
x1 = augmentation2(input22)
x1 =keras.layers.Lambda(
lambda x: keras.applications.vgg16.preprocess_input(x))(x1)
x1 = convolution_base(x1)
x1 = layers.Flatten()(x1)
x1 = layers.Dense(256)(x1)
x1 = layers.Dropout(0.5)(x1)
outputs = layers.Dense(1, activation="sigmoid")(x1)
model = keras.Model(input22, outputs)
model.compile(loss="binary_crossentropy",
optimizer="rmsprop",
metrics=["accuracy"])
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="feature_extraction_with_data_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=75,
```

```
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 73/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 192ms/step - accuracy: 0.9890 - loss: 0.4092 - val_accuracy: 0.9800 - val_loss: 2.4140
Epoch 74/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 161ms/step - accuracy: 0.9907 - loss: 0.6050 - val_accuracy: 0.9820 - val_loss: 2.2798
Epoch 75/75
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 163ms/step - accuracy: 0.9934 - loss: 0.3279 - val_accuracy: 0.9820 - val_loss: 2.0438
```

A pretrained VGG16 model with Fine-tuning

Fine-tuning the pretrained model which already discovered some useful characteristics from a large set of data, speed-to-convergence is accelerated as compared to training from scratch. The model may overfit the dataset that the model is tuned on if it is not fine-tunned on a new dataset. This may result in it meshing its learned features better to the features of this new dataset leading to improved generalization performance.

```
convolution_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_6 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1,792
block1_conv2 (Conv2D)	(None, None, None, 64)	36,928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73,856
block2_conv2 (Conv2D)	(None, None, None, 128)	147,584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295,168
block3_conv2 (Conv2D)	(None, None, None, 256)	590,080
block3_conv3 (Conv2D)	(None, None, None, 256)	590,080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1,180,160
block4_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block4_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

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

```
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=50,
        validation_data=validation_dataset,
        callbacks=callbacks)
```

```
Epoch 22/50
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 178ms/step - accuracy: 0.9992 - loss: 0.0111 - val_accuracy: 0.9700 - val_loss: 3.5891
Epoch 44/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 207ms/step - accuracy: 0.9967 - loss: 0.0438 - val_accuracy: 0.9780 - val_loss: 2.8906
Epoch 45/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 203ms/step - accuracy: 0.9937 - loss: 0.3912 - val_accuracy: 0.9870 - val_loss: 1.6918
Epoch 46/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 211ms/step - accuracy: 0.9975 - loss: 0.0443 - val_accuracy: 0.9820 - val_loss: 1.9703
Epoch 47/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 207ms/step - accuracy: 0.9996 - loss: 0.0151 - val_accuracy: 0.9870 - val_loss: 1.7109
Epoch 48/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 215ms/step - accuracy: 0.9975 - loss: 0.0445 - val_accuracy: 0.9880 - val_loss: 1.4771
Epoch 49/50
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 178ms/step - accuracy: 0.9959 - loss: 0.2099 - val_accuracy: 0.9870 - val_loss: 1.6063
Epoch 50/50
```