



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

 no files selected Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': ...}

```
!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
98% 793M/812M [00:02<00:00, 287MB/s]
100% 812M/812M [00:03<00:00, 281MB/s]

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

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

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.

Importing the numpy as np and creating the dataset with 1000 samples with vector 16.

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

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 binary 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: "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 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_3 (Conv2D)	(None, 18, 18, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_4 (Conv2D)	(None, 7, 7, 256)	590080
flatten (Flatten)	(None, 12544)	0
dropout (Dropout)	(None, 12544)	0
dense (Dense)	(None, 1)	12545
Total params: 991041 (3.78 MB)		
Trainable params: 991041 (3.78 MB)		
Non-trainable params: 0 (0.00 Byte)		

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 1/50
63/63 [=====] - 17s 135ms/step - loss: 0.6955 - accu
Epoch 2/50
63/63 [=====] - 4s 60ms/step - loss: 0.6841 - accur
Epoch 3/50
63/63 [=====] - 4s 59ms/step - loss: 0.6632 - accur
Epoch 4/50
63/63 [=====] - 6s 99ms/step - loss: 0.6212 - accur
Epoch 5/50
63/63 [=====] - 4s 62ms/step - loss: 0.6042 - accur
Epoch 6/50
63/63 [=====] - 5s 83ms/step - loss: 0.5438 - accur
Epoch 7/50
63/63 [=====] - 6s 87ms/step - loss: 0.4911 - accur
Epoch 8/50
63/63 [=====] - 4s 59ms/step - loss: 0.4573 - accur
Epoch 9/50
63/63 [=====] - 4s 58ms/step - loss: 0.4108 - accur
Epoch 10/50
63/63 [=====] - 6s 94ms/step - loss: 0.3771 - accur
Epoch 11/50
63/63 [=====] - 4s 57ms/step - loss: 0.3187 - accur
Epoch 12/50
63/63 [=====] - 4s 65ms/step - loss: 0.2478 - accur
Epoch 13/50
63/63 [=====] - 6s 87ms/step - loss: 0.1871 - accur
Epoch 14/50
63/63 [=====] - 4s 60ms/step - loss: 0.1596 - accur
Epoch 15/50
63/63 [=====] - 6s 100ms/step - loss: 0.1397 - accur
Epoch 16/50
63/63 [=====] - 5s 65ms/step - loss: 0.1199 - accur
Epoch 17/50
63/63 [=====] - 4s 57ms/step - loss: 0.1237 - accur

```

```

Epoch 18/50
63/63 [=====] - 5s 68ms/step - loss: 0.0961 - accuracy: 0.8500
Epoch 19/50
63/63 [=====] - 6s 89ms/step - loss: 0.0614 - accuracy: 0.8700
Epoch 20/50
63/63 [=====] - 4s 57ms/step - loss: 0.0483 - accuracy: 0.8800
Epoch 21/50
63/63 [=====] - 4s 58ms/step - loss: 0.0354 - accuracy: 0.8900
Epoch 22/50
63/63 [=====] - 6s 91ms/step - loss: 0.0903 - accuracy: 0.8600
Epoch 23/50
63/63 [=====] - 4s 58ms/step - loss: 0.0468 - accuracy: 0.8800
Epoch 24/50
63/63 [=====] - 4s 59ms/step - loss: 0.0433 - accuracy: 0.8900
Epoch 25/50
63/63 [=====] - 5s 83ms/step - loss: 0.0574 - accuracy: 0.8700
Epoch 26/50
63/63 [=====] - 5s 79ms/step - loss: 0.0563 - accuracy: 0.8700
Epoch 27/50
63/63 [=====] - 4s 57ms/step - loss: 0.0431 - accuracy: 0.8900
Epoch 28/50
63/63 [=====] - 4s 58ms/step - loss: 0.0359 - accuracy: 0.9000
Epoch 29/50
63/63 [=====] - 6s 96ms/step - loss: 0.0208 - accuracy: 0.9200
Epoch 30/50

```

```

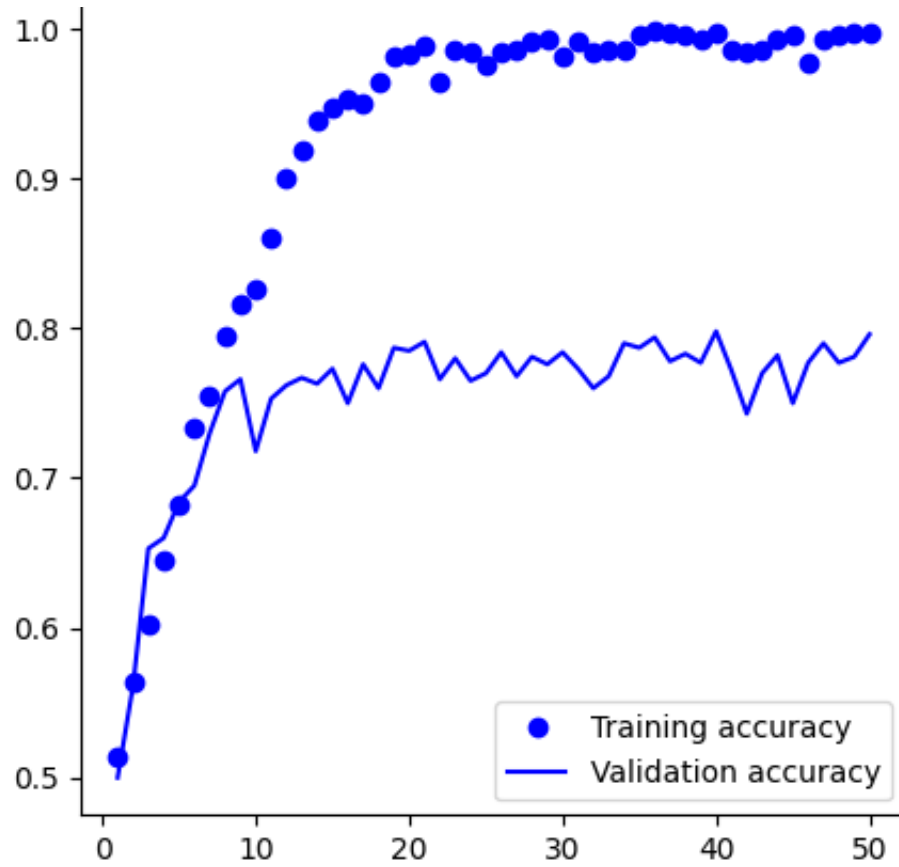
import matplotlib.pyplot as plt

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()

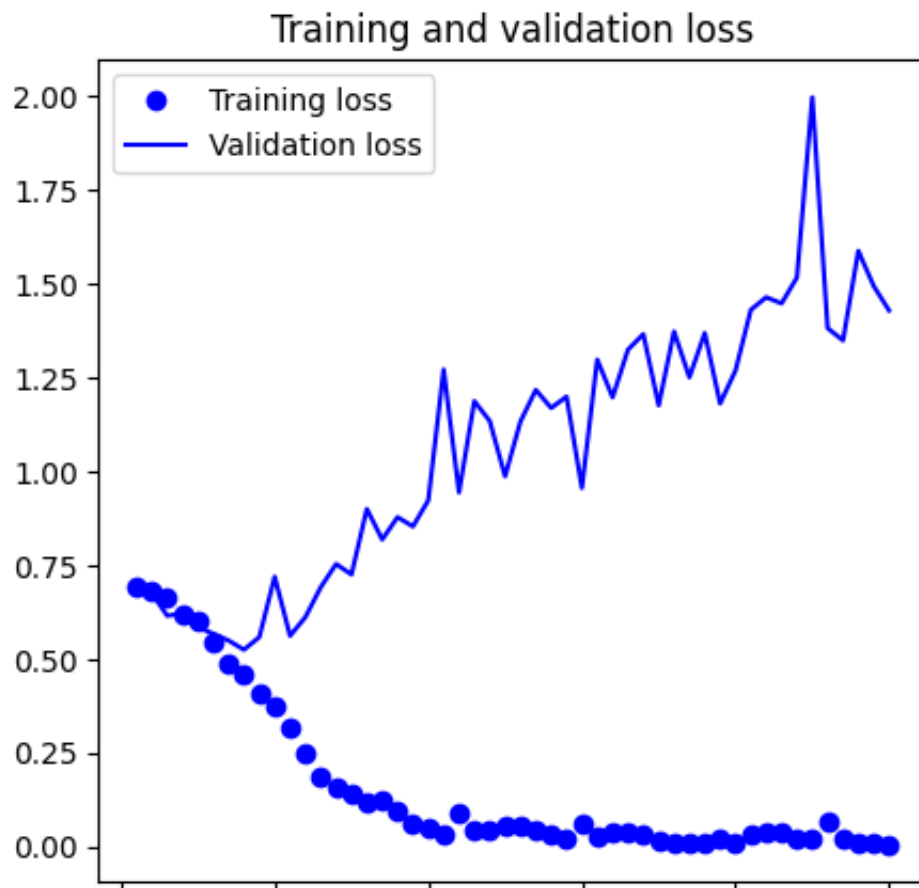
```



Training & validation accuracy



<Figure size 640x480 with 0 Axes>



0 10 20 30 40 50

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

```
32/32 [=====] - 1s 28ms/step - loss: 0.5921 - accuracy: 0.719
Test accuracy: 0.719
```

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 increasing 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)

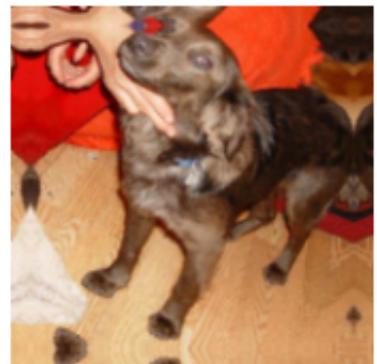
#Creating training, Test and validation sets.
#Training has 1500 samples, test has 500 samples and validation has 500 samples.
make_subset("train", start_index=2000, end_index=3500)
make_subset("validation", start_index=3501, end_index=4001)
make_subset("test", start_index=4002, end_index=4502)
```

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"])

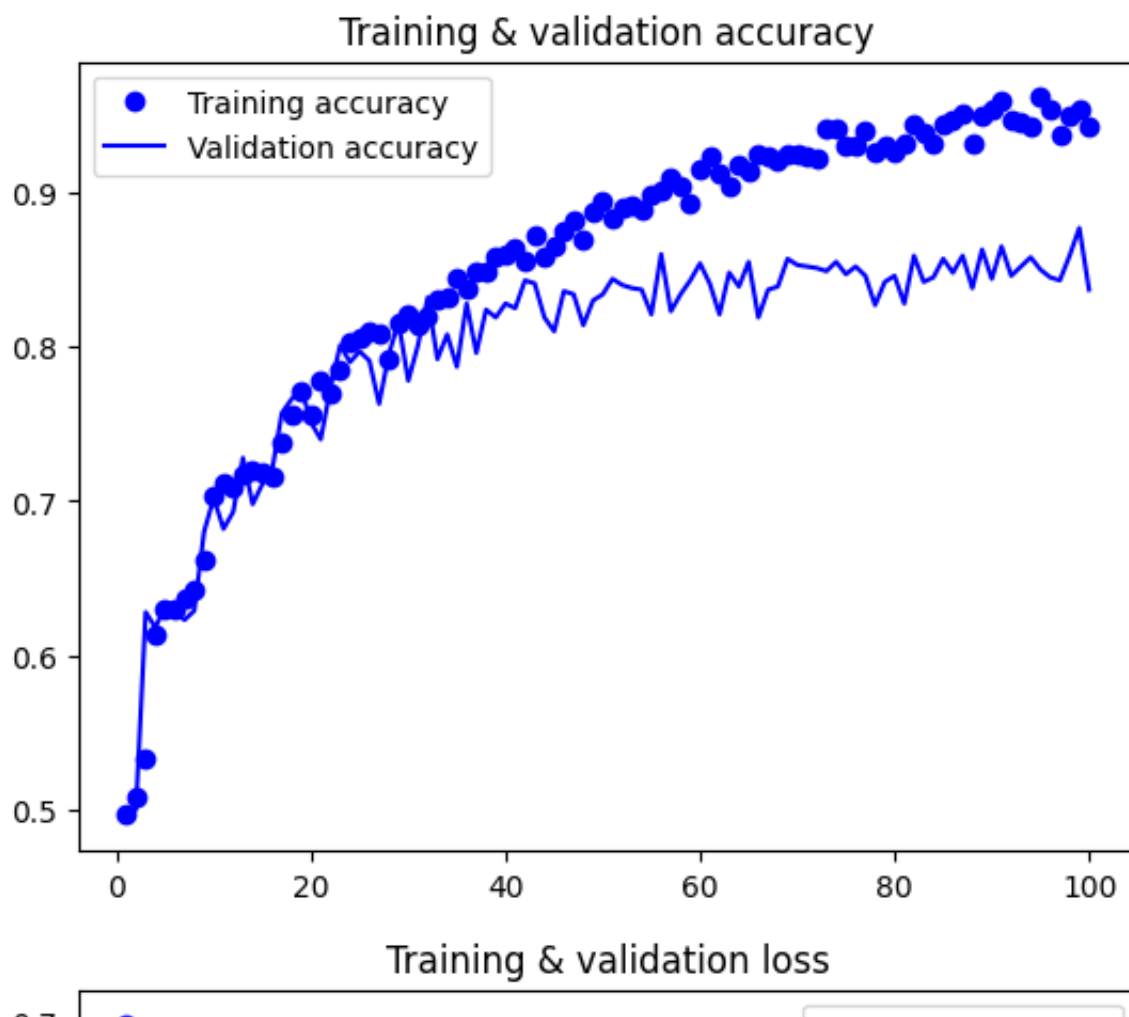
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

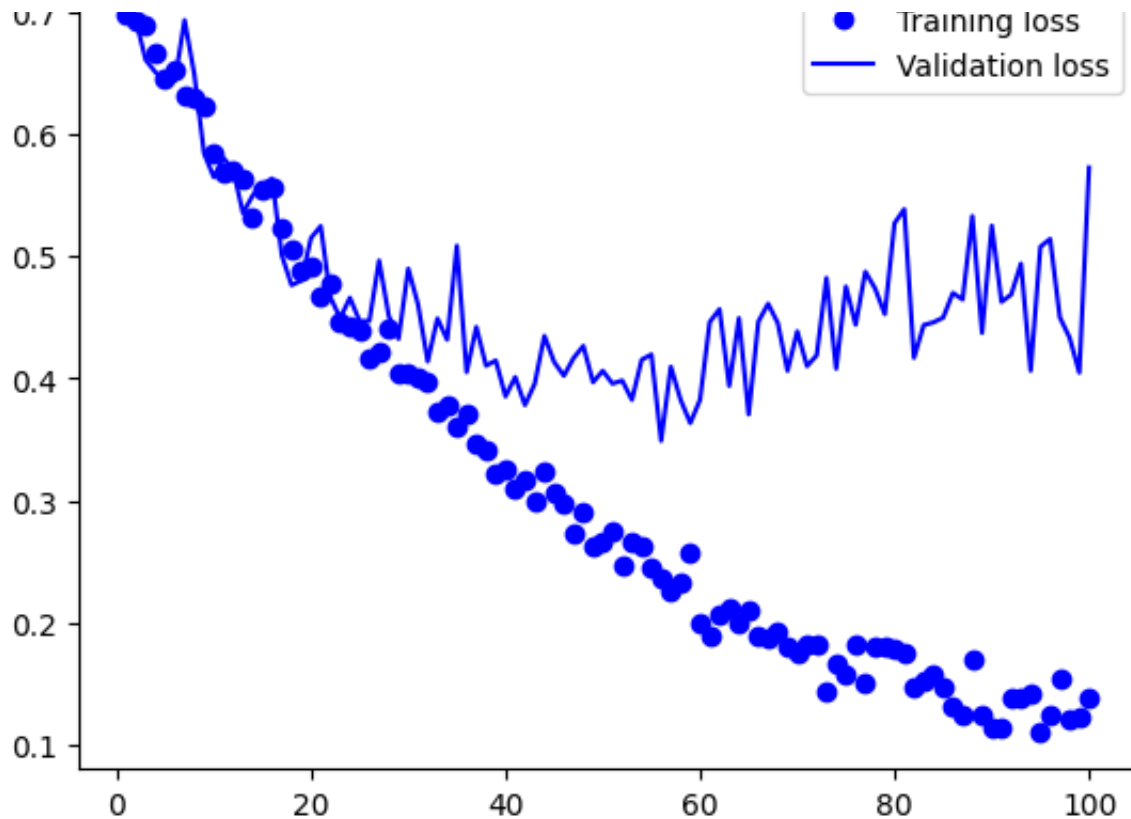
```
➡ Epoch 1/100
63/63 [=====] - 8s 67ms/step - loss: 0.6974 - accuracy: 0.1250
Epoch 2/100
63/63 [=====] - 5s 75ms/step - loss: 0.6936 - accuracy: 0.1250
Epoch 3/100
63/63 [=====] - 6s 90ms/step - loss: 0.6891 - accuracy: 0.1250
Epoch 4/100
63/63 [=====] - 4s 60ms/step - loss: 0.6659 - accuracy: 0.1250
```

```
Epoch 5/100
63/63 [=====] - 4s 60ms/step - loss: 0.6464 - accuracy: 0.7500
Epoch 6/100
63/63 [=====] - 7s 105ms/step - loss: 0.6534 - accuracy: 0.7500
Epoch 7/100
63/63 [=====] - 4s 61ms/step - loss: 0.6325 - accuracy: 0.7500
Epoch 8/100
63/63 [=====] - 4s 58ms/step - loss: 0.6299 - accuracy: 0.7500
Epoch 9/100
63/63 [=====] - 6s 92ms/step - loss: 0.6224 - accuracy: 0.7500
Epoch 10/100
63/63 [=====] - 5s 75ms/step - loss: 0.5843 - accuracy: 0.7500
Epoch 11/100
63/63 [=====] - 4s 59ms/step - loss: 0.5688 - accuracy: 0.7500
Epoch 12/100
63/63 [=====] - 5s 75ms/step - loss: 0.5712 - accuracy: 0.7500
Epoch 13/100
63/63 [=====] - 6s 89ms/step - loss: 0.5632 - accuracy: 0.7500
Epoch 14/100
63/63 [=====] - 4s 58ms/step - loss: 0.5323 - accuracy: 0.7500
Epoch 15/100
63/63 [=====] - 4s 57ms/step - loss: 0.5555 - accuracy: 0.7500
Epoch 16/100
63/63 [=====] - 6s 87ms/step - loss: 0.5565 - accuracy: 0.7500
Epoch 17/100
63/63 [=====] - 5s 78ms/step - loss: 0.5232 - accuracy: 0.7500
Epoch 18/100
63/63 [=====] - 4s 59ms/step - loss: 0.5060 - accuracy: 0.7500
Epoch 19/100
63/63 [=====] - 4s 63ms/step - loss: 0.4880 - accuracy: 0.7500
Epoch 20/100
63/63 [=====] - 7s 109ms/step - loss: 0.4909 - accuracy: 0.7500
Epoch 21/100
63/63 [=====] - 4s 60ms/step - loss: 0.4667 - accuracy: 0.7500
Epoch 22/100
63/63 [=====] - 4s 61ms/step - loss: 0.4775 - accuracy: 0.7500
Epoch 23/100
63/63 [=====] - 6s 100ms/step - loss: 0.4470 - accuracy: 0.7500
Epoch 24/100
63/63 [=====] - 5s 72ms/step - loss: 0.4428 - accuracy: 0.7500
Epoch 25/100
63/63 [=====] - 4s 61ms/step - loss: 0.4384 - accuracy: 0.7500
Epoch 26/100
63/63 [=====] - 5s 83ms/step - loss: 0.4161 - accuracy: 0.7500
Epoch 27/100
63/63 [=====] - 4s 58ms/step - loss: 0.4210 - accuracy: 0.7500
Epoch 28/100
63/63 [=====] - 5s 71ms/step - loss: 0.4415 - accuracy: 0.7500
Epoch 29/100
63/63 [=====] - 6s 94ms/step - loss: 0.4039 - accuracy: 0.7500
```

Epoch 30/100

```
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 27ms/step - loss: 0.4353 - accuracy: 0.8400
Test accuracy: 0.840
```

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)

```

```

↔ Epoch 1/75
63/63 [=====] - 9s 90ms/step - loss: 0.6970 - accuracy: 0.1250
Epoch 2/75
63/63 [=====] - 4s 60ms/step - loss: 0.6932 - accuracy: 0.1250
Epoch 3/75
63/63 [=====] - 4s 60ms/step - loss: 0.6874 - accuracy: 0.1250
Epoch 4/75
63/63 [=====] - 7s 106ms/step - loss: 0.6756 - accuracy: 0.1250
Epoch 5/75
63/63 [=====] - 4s 60ms/step - loss: 0.6899 - accuracy: 0.1250
Epoch 6/75
63/63 [=====] - 4s 62ms/step - loss: 0.6620 - accuracy: 0.1250
Epoch 7/75
63/63 [=====] - 6s 93ms/step - loss: 0.6529 - accuracy: 0.1250
Epoch 8/75
63/63 [=====] - 5s 73ms/step - loss: 0.6442 - accuracy: 0.1250
Epoch 9/75
63/63 [=====] - 34s 545ms/step - loss: 0.6274 - accuracy: 0.1250
Epoch 10/75
63/63 [=====] - 5s 81ms/step - loss: 0.6225 - accuracy: 0.1250
Epoch 11/75
63/63 [=====] - 4s 61ms/step - loss: 0.6122 - accuracy: 0.1250
Epoch 12/75
63/63 [=====] - 4s 64ms/step - loss: 0.5966 - accuracy: 0.1250
Epoch 13/75
63/63 [=====] - 8s 116ms/step - loss: 0.5832 - accuracy: 0.1250
Epoch 14/75
63/63 [=====] - 4s 60ms/step - loss: 0.5763 - accuracy: 0.1250
Epoch 15/75
63/63 [=====] - 4s 60ms/step - loss: 0.5674 - accuracy: 0.1250
Epoch 16/75
63/63 [=====] - 6s 90ms/step - loss: 0.5432 - accuracy: 0.1250
Epoch 17/75
63/63 [=====] - 5s 77ms/step - loss: 0.5274 - accuracy: 0.1250
Epoch 18/75
63/63 [=====] - 4s 60ms/step - loss: 0.5126 - accuracy: 0.1250
Epoch 19/75
63/63 [=====] - 4s 64ms/step - loss: 0.5121 - accuracy: 0.1250
Epoch 20/75

```

```

63/63 [=====] - 6s 92ms/step - loss: 0.4720 - accuracy: 0.85
Epoch 21/75
63/63 [=====] - 4s 60ms/step - loss: 0.4859 - accuracy: 0.85
Epoch 22/75
63/63 [=====] - 8s 113ms/step - loss: 0.4789 - accuracy: 0.85
Epoch 23/75
63/63 [=====] - 4s 58ms/step - loss: 0.4916 - accuracy: 0.85
Epoch 24/75
63/63 [=====] - 5s 81ms/step - loss: 0.4497 - accuracy: 0.85
Epoch 25/75
63/63 [=====] - 4s 58ms/step - loss: 0.4727 - accuracy: 0.85
Epoch 26/75
63/63 [=====] - 4s 65ms/step - loss: 0.4315 - accuracy: 0.85
Epoch 27/75
63/63 [=====] - 6s 92ms/step - loss: 0.4388 - accuracy: 0.85
Epoch 28/75
63/63 [=====] - 4s 59ms/step - loss: 0.4277 - accuracy: 0.85
Epoch 29/75
63/63 [=====] - 4s 61ms/step - loss: 0.4254 - accuracy: 0.85
Epoch 30/75

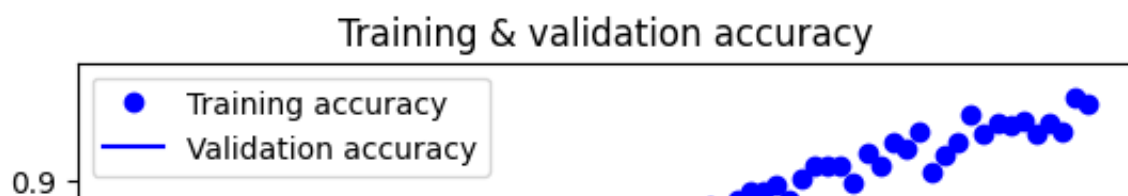
```

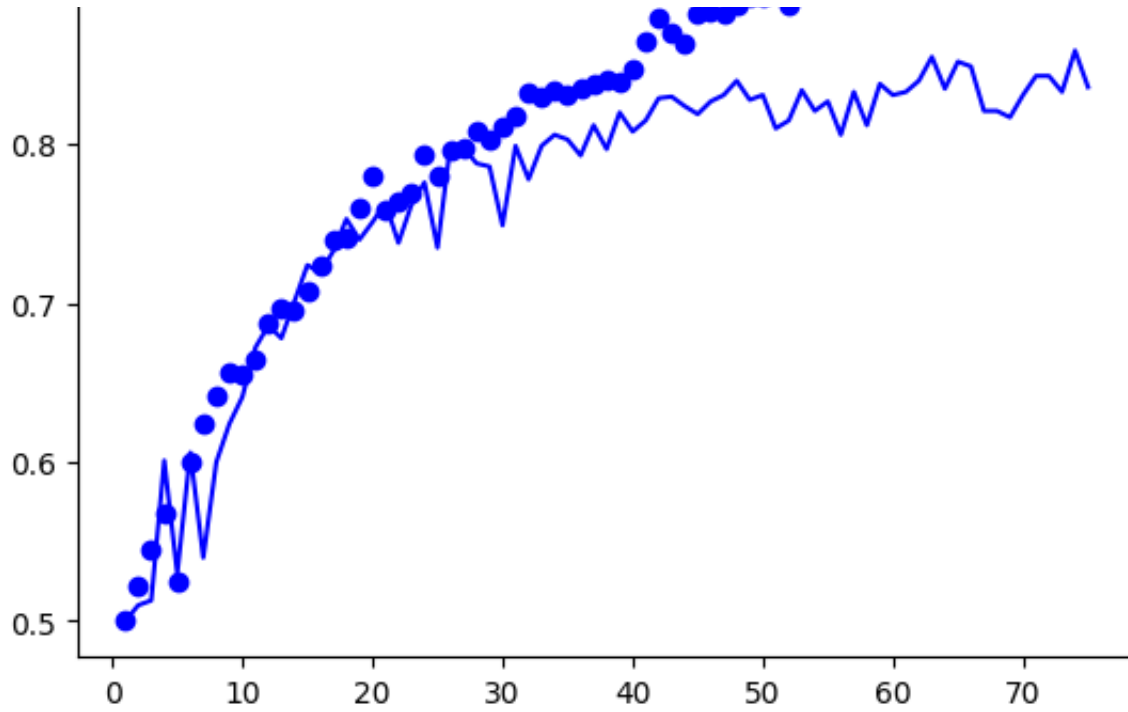
Graph of training and validation accuracy

```

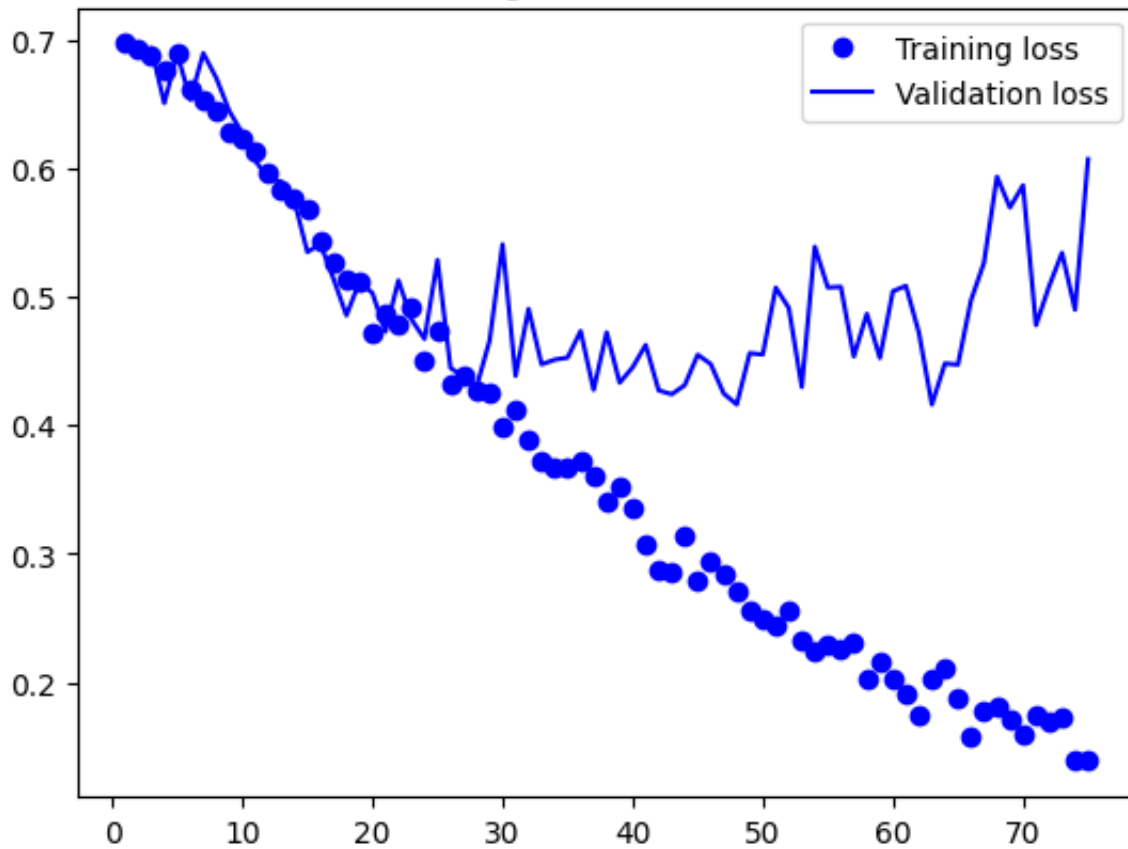
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, 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()

```





Training & validation loss



```
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 28ms/step - loss: 0.4473 - accuracy: 0.833
Test accuracy: 0.833
```

In the beginning as we took only 1000 samples in the first question and we achieved 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/58889256/58889256 [=====] - 0s 0us/step
```

```
convolution_base.summary()
```

⇒ Model: "vgg16"

Layer (type)	Output Shape	Param #
input_7 (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: 14714688 (56.13 MB)		
Trainable params: 14714688 (56.13 MB)		
Non-trainable params: 0 (0.00 Byte)		

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 [=====] - 5s 5s/step
1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 25ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 34ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 24ms/step
```

```

1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 33ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 40ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 33ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 34ms/step
1/1 [=====] - 0s 34ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 41ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 35ms/step
1/1 [=====] - 0s 41ms/step
1/1 [=====] - 0s 34ms/step
1/1 [=====] - 0s 36ms/step
1/1 [=====] - 0s 41ms/step
1/1 [=====] - 0s 33ms/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 1/40
63/63 [=====] - 1s 10ms/step - loss: 13.8645 - accuracy: 0.0000
Epoch 2/40
63/63 [=====] - 1s 8ms/step - loss: 3.3413 - accuracy: 0.0000
Epoch 3/40
63/63 [=====] - 1s 8ms/step - loss: 3.1077 - accuracy: 0.0000
Epoch 4/40
63/63 [=====] - 0s 6ms/step - loss: 0.5525 - accuracy: 0.0000
Epoch 5/40
63/63 [=====] - 1s 8ms/step - loss: 1.6439 - accuracy: 0.0000
Epoch 6/40
63/63 [=====] - 0s 6ms/step - loss: 0.5152 - accuracy: 0.0000
Epoch 7/40
63/63 [=====] - 0s 5ms/step - loss: 0.4014 - accuracy: 0.0000
Epoch 8/40
63/63 [=====] - 0s 5ms/step - loss: 0.9763 - accuracy: 0.0000
Epoch 9/40
63/63 [=====] - 0s 6ms/step - loss: 0.3532 - accuracy: 0.0000
Epoch 10/40
63/63 [=====] - 0s 5ms/step - loss: 0.5668 - accuracy: 0.0000
Epoch 11/40
63/63 [=====] - 0s 5ms/step - loss: 0.5951 - accuracy: 0.0000
Epoch 12/40
63/63 [=====] - 0s 6ms/step - loss: 0.2439 - accuracy: 0.0000
Epoch 13/40
63/63 [=====] - 0s 5ms/step - loss: 0.1363 - accuracy: 0.0000
Epoch 14/40
63/63 [=====] - 0s 5ms/step - loss: 0.1463 - accuracy: 0.0000
Epoch 15/40
63/63 [=====] - 0s 6ms/step - loss: 0.0256 - accuracy: 0.0000
Epoch 16/40
63/63 [=====] - 0s 6ms/step - loss: 0.1540 - accuracy: 0.0000
Epoch 17/40
63/63 [=====] - 1s 8ms/step - loss: 0.0392 - accuracy: 0.0000
Epoch 18/40
63/63 [=====] - 0s 7ms/step - loss: 0.1022 - accuracy: 0.0000
Epoch 19/40
63/63 [=====] - 0s 7ms/step - loss: 0.2228 - accuracy: 0.0000
Epoch 20/40

```



```

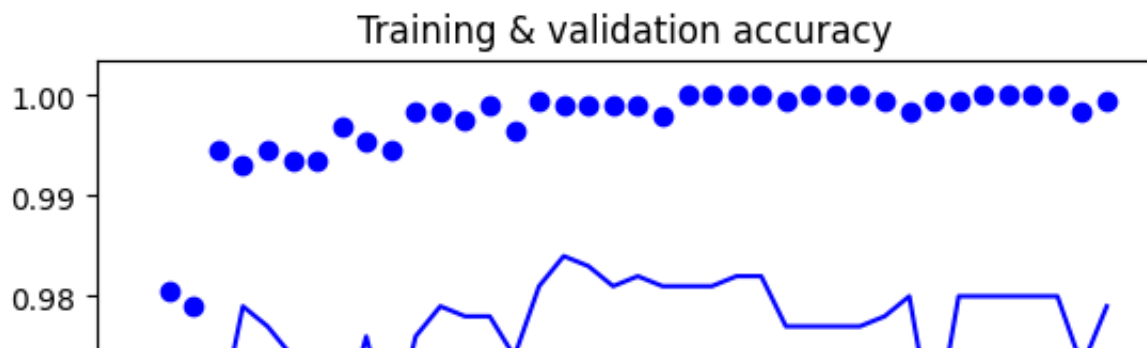
63/63 [=====] - 0s 8ms/step - loss: 0.0472 - accuracy
Epoch 21/40
63/63 [=====] - 0s 7ms/step - loss: 0.0561 - accuracy
Epoch 22/40
63/63 [=====] - 1s 9ms/step - loss: 0.1985 - accuracy
Epoch 23/40
63/63 [=====] - 1s 9ms/step - loss: 0.0000e+00 - accu
Epoch 24/40
63/63 [=====] - 0s 8ms/step - loss: 1.0615e-14 - accu
Epoch 25/40
63/63 [=====] - 0s 8ms/step - loss: 5.1761e-06 - accu
Epoch 26/40
63/63 [=====] - 1s 8ms/step - loss: 7.0063e-19 - accu
Epoch 27/40
63/63 [=====] - 1s 9ms/step - loss: 0.0224 - accuracy
Epoch 28/40
63/63 [=====] - 1s 8ms/step - loss: 1.1320e-06 - accu
Epoch 29/40
63/63 [=====] - 1s 9ms/step - loss: 0.0000e+00 - accu
Epoch 30/40

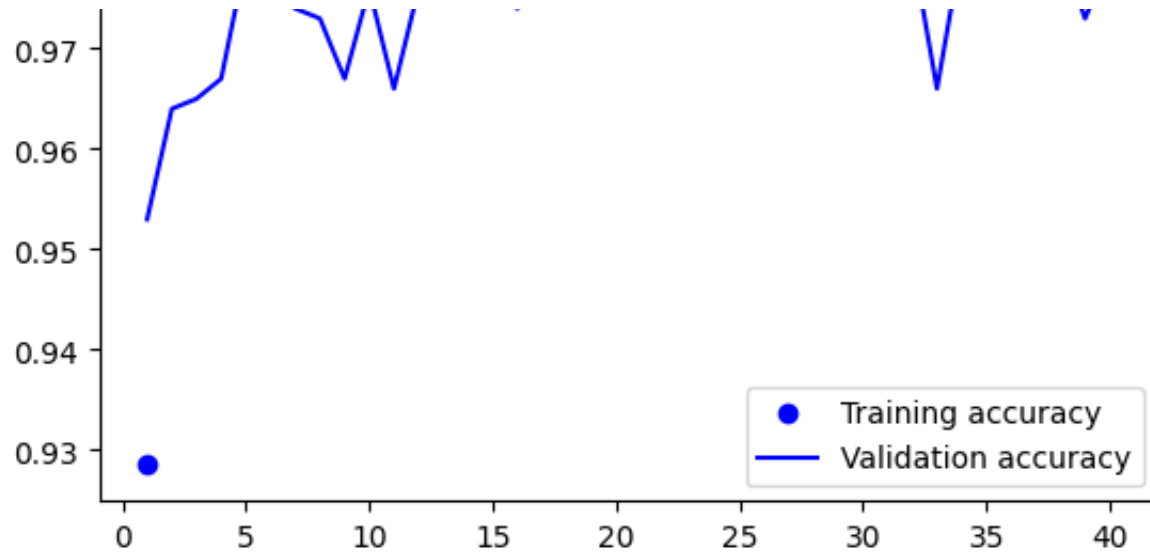
```

```

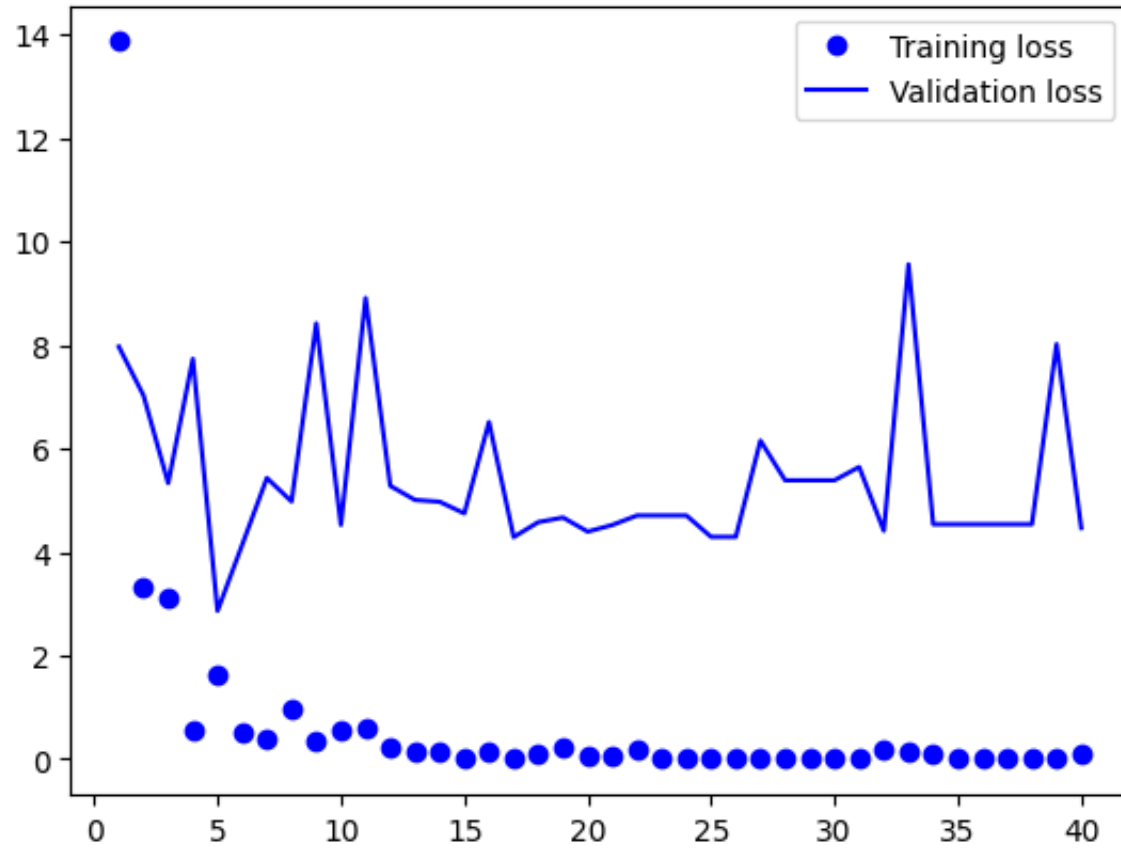
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, 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()

```





Training & validation loss



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
```

⇒ The number of trainable weights required to use the convolution base before it

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

⇒ After the convolution base is frozen, this is the total quantity of trainable

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 1/75
63/63 [=====] - 11s 158ms/step - loss: 21.3565 - accu
Epoch 2/75
63/63 [=====] - 11s 176ms/step - loss: 5.8368 - accu
Epoch 3/75
63/63 [=====] - 11s 177ms/step - loss: 6.2118 - accu
Epoch 4/75
63/63 [=====] - 9s 141ms/step - loss: 5.8927 - accu
Epoch 5/75
63/63 [=====] - 11s 174ms/step - loss: 3.6089 - accu
Epoch 6/75
63/63 [=====] - 10s 148ms/step - loss: 3.3797 - accu
Epoch 7/75
63/63 [=====] - 9s 141ms/step - loss: 3.3175 - accu
Epoch 8/75
63/63 [=====] - 10s 154ms/step - loss: 2.8912 - accu
Epoch 9/75
63/63 [=====] - 9s 144ms/step - loss: 2.3200 - accu
Epoch 10/75
63/63 [=====] - 9s 142ms/step - loss: 3.3377 - accu
Epoch 11/75
63/63 [=====] - 9s 146ms/step - loss: 2.1499 - accu
Epoch 12/75
63/63 [=====] - 9s 138ms/step - loss: 2.7613 - accu
Epoch 13/75
63/63 [=====] - 10s 154ms/step - loss: 1.7239 - accu
Epoch 14/75
63/63 [=====] - 11s 174ms/step - loss: 2.5592 - accu
Epoch 15/75
63/63 [=====] - 9s 141ms/step - loss: 1.9904 - accu
Epoch 16/75
63/63 [=====] - 9s 147ms/step - loss: 1.7074 - accu
Epoch 17/75
63/63 [=====] - 12s 183ms/step - loss: 1.9545 - accu
Epoch 18/75
63/63 [=====] - 9s 145ms/step - loss: 1.8268 - accu
Epoch 19/75
63/63 [=====] - 9s 147ms/step - loss: 2.0651 - accu
Epoch 20/75
63/63 [=====] - 10s 147ms/step - loss: 1.6846 - accu

```

```

Epoch 21/75
63/63 [=====] - 9s 143ms/step - loss: 1.2312 - accuracy: 0.4468
Epoch 22/75
63/63 [=====] - 9s 141ms/step - loss: 1.5208 - accuracy: 0.4468
Epoch 23/75
63/63 [=====] - 9s 144ms/step - loss: 0.4468 - accuracy: 0.4468
Epoch 24/75
63/63 [=====] - 9s 138ms/step - loss: 1.0485 - accuracy: 0.4468
Epoch 25/75
63/63 [=====] - 9s 144ms/step - loss: 1.5633 - accuracy: 0.4468
Epoch 26/75
63/63 [=====] - 11s 174ms/step - loss: 1.1490 - accuracy: 0.4468
Epoch 27/75
63/63 [=====] - 10s 148ms/step - loss: 1.1046 - accuracy: 0.4468
Epoch 28/75
63/63 [=====] - 10s 149ms/step - loss: 1.3581 - accuracy: 0.4468
Epoch 29/75
63/63 [=====] - 9s 140ms/step - loss: 1.0131 - accuracy: 0.4468
Epoch 30/75

```

```

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

```

```

➡ 32/32 [=====] - 4s 88ms/step - loss: 2.1661 - accuracy: 0.979
Test accuracy: 0.979

```

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-tuned 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_9 (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: 14714688 (56.13 MB)		
Trainable params: 0 (0.00 Byte)		
Non-trainable params: 14714688 (56.13 MB)		

```
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 1/50
63/63 [=====] - 13s 178ms/step - loss: 0.3884 - accu
Epoch 2/50
63/63 [=====] - 11s 170ms/step - loss: 0.4428 - accu
Epoch 3/50
63/63 [=====] - 11s 166ms/step - loss: 0.2220 - accu
Epoch 4/50
63/63 [=====] - 12s 192ms/step - loss: 0.0601 - accu
Epoch 5/50
63/63 [=====] - 10s 161ms/step - loss: 0.0859 - accu
Epoch 6/50
63/63 [=====] - 10s 154ms/step - loss: 0.2347 - accu
Epoch 7/50
63/63 [=====] - 10s 163ms/step - loss: 0.2364 - accu
Epoch 8/50
63/63 [=====] - 10s 155ms/step - loss: 0.1495 - accu
Epoch 9/50
63/63 [=====] - 11s 165ms/step - loss: 0.3497 - accu
Epoch 10/50
63/63 [=====] - 10s 157ms/step - loss: 0.5183 - accu
Epoch 11/50
63/63 [=====] - 11s 169ms/step - loss: 0.0668 - accu
Epoch 12/50
63/63 [=====] - 11s 173ms/step - loss: 0.1845 - accu
Epoch 13/50
63/63 [=====] - 11s 165ms/step - loss: 0.0869 - accu
```

```

Epoch 14/50
63/63 [=====] - 11s 162ms/step - loss: 0.1190 - accu
Epoch 15/50
63/63 [=====] - 13s 198ms/step - loss: 0.1532 - accu
Epoch 16/50
63/63 [=====] - 11s 164ms/step - loss: 0.2212 - accu
Epoch 17/50
63/63 [=====] - 10s 157ms/step - loss: 0.0539 - accu
Epoch 18/50
63/63 [=====] - 11s 163ms/step - loss: 0.0486 - accu
Epoch 19/50
63/63 [=====] - 12s 189ms/step - loss: 0.2498 - accu
Epoch 20/50
63/63 [=====] - 10s 158ms/step - loss: 0.1081 - accu
Epoch 21/50
63/63 [=====] - 11s 165ms/step - loss: 0.0792 - accu
Epoch 22/50
63/63 [=====] - 10s 155ms/step - loss: 0.2037 - accu
Epoch 23/50
63/63 [=====] - 10s 160ms/step - loss: 0.0761 - accu
Epoch 24/50
63/63 [=====] - 12s 190ms/step - loss: 0.1135 - accu
Epoch 25/50
63/63 [=====] - 11s 170ms/step - loss: 0.2245 - accu
Epoch 26/50
63/63 [=====] - 10s 157ms/step - loss: 0.0821 - accu
Epoch 27/50
63/63 [=====] - 10s 160ms/step - loss: 0.0804 - accu
Epoch 28/50
63/63 [=====] - 11s 161ms/step - loss: 0.0638 - accu
Epoch 29/50
63/63 [=====] - 11s 167ms/step - loss: 0.1293 - accu
Epoch 30/50

```

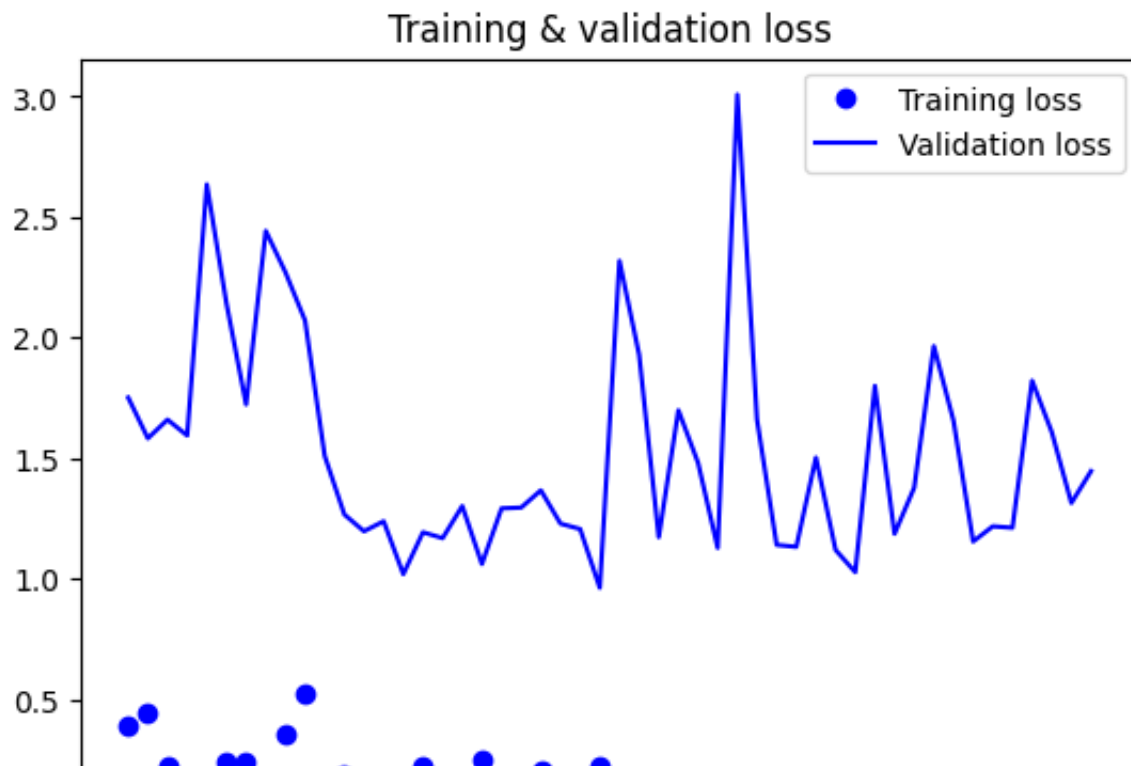
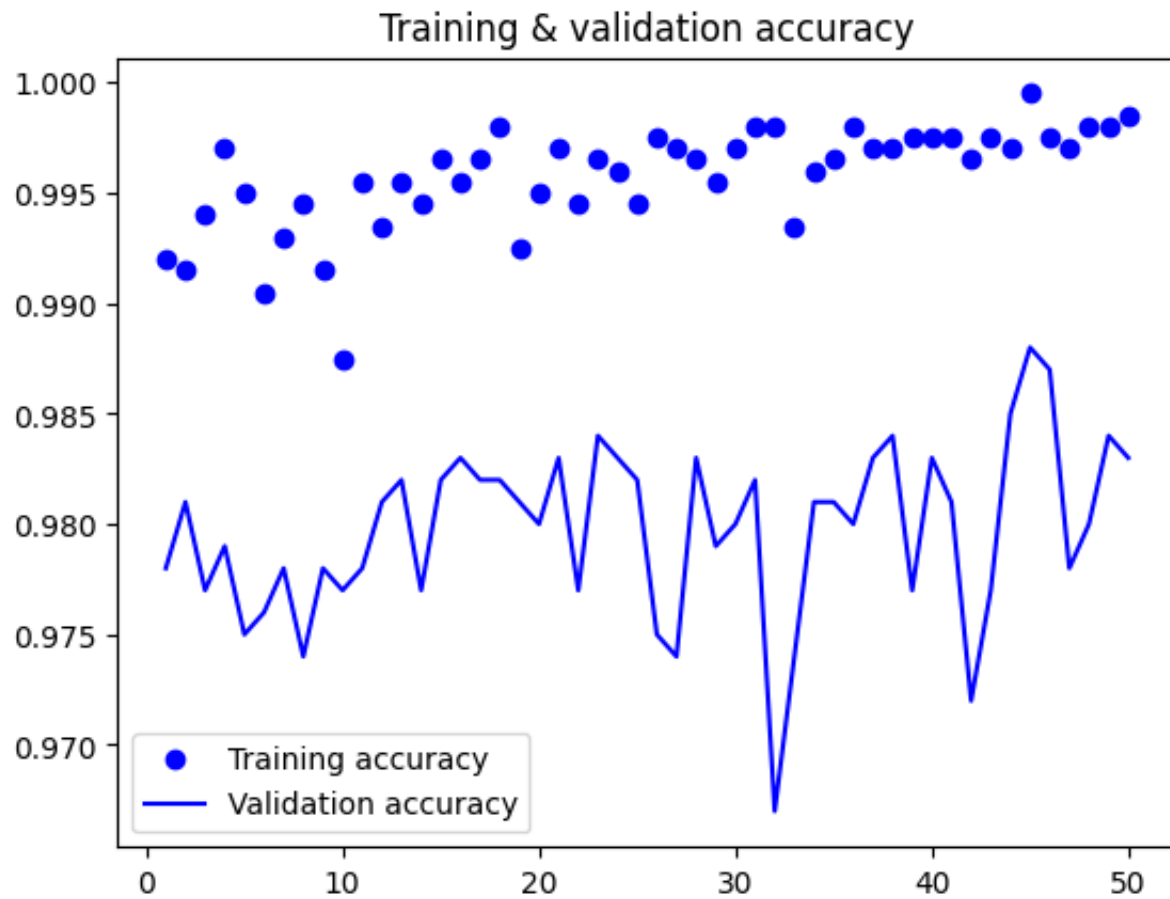
```

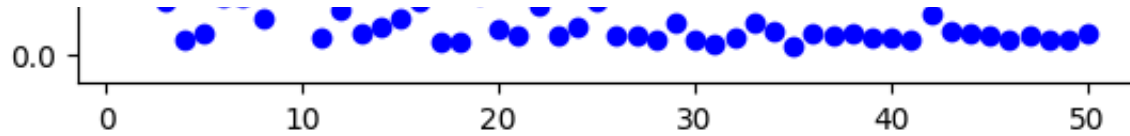
import matplotlib.pyplot as plt
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, 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()
```





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

➡ 32/32 [=====] - 3s 87ms/step - loss: 2.2782 - accuracy: 0.974
Test accuracy: 0.974