AML_Assignment-2_Convolution

Sushma Chiluveru

Uploading Kaggle API JSON File by creating a token and Downloading Dogs vs Cats dataset from Kaggle website

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

Choose Files kaggle.json

- **kaggle.json**(application/json) - 71 bytes, last modified: 7/10/2024 - 100% done

```
Saving kaggle.json to kaggle (1).json
{'kaggle (1).json':
b'{"username":"chiluverusushma","key":"6ca0632035ecba498140262fbc039094"}'}
```

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

```
!kaggle competitions download -c dogs-vs-cats
!unzip -qq dogs-vs-cats.zip
!unzip -qq train.zip
```

```
Downloading dogs-vs-cats.zip to /content
100% 810M/812M [00:51<00:00, 17.6MB/s]
100% 812M/812M [00:51<00:00, 16.6MB/s]
```

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?

Organizing and copying data into separate directories for testing, training, and validation

```
import os, shutil, pathlib
o_dir = pathlib.Path("train")
n_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = n_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 = o_dir / fname
            dst = dir / fname
            shutil.copyfile(src, dst)

make_subset("train", start_index=500, end_index=1500)
make_subset("validation", start_index=1500, end_index=2000)
make_subset("test", start_index=2000, end_index=2500)
```

Constructing a simple convolutional neural network model for classifying dogs and cats.

```
from tensorflow.keras.utils import image_dataset_from_directory

train_data = image_dataset_from_directory(n_dir / "train",image_size=(180, 180),batch_size=32)

valid_data = image_dataset_from_directory(n_dir / "validation",image_size=(180, 180),batch_size=32)

test_data= image_dataset_from_directory(n_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.
```

Generate a dataset instance with 1000 random samples, each having a vector size of 16, using a NumPy array.

```
import numpy as np
import tensorflow as tf

ran_num = np.random.normal(size=(1000, 16))
dataset = tf.data.Dataset.from_tensor_slices(ran_num)
for i, element in enumerate(dataset):
    print(element.shape)
    if i >= 2:
       break

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

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
```

```
(16,)
(16,)
(16,)
(32, 16)
(32, 16)
(32, 16)
(4, 4)
(4, 4)
(4, 4)
```

Displaying the dimensions of the data and labels generated by the Dataset.

```
for dataset_batch, label_batch in train_data:
    print("data batch shape:", dataset_batch.shape)
    print("labels batch shape:", label_batch.shape)
    break
```

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

Identifying a small convolution for dogs vs. cats categories

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

input_1000 = keras.Input(shape=(180, 180, 3))
d_1000 = layers.Rescaling(1./255)(input_1000)
d_1000 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(d_1000)
d_1000 = layers.MaxPooling2D(pool_size=2)(d_1000)
d_1000 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(d_1000)
d_1000 = layers.MaxPooling2D(pool_size=2)(d_1000)
d_1000 = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(d_1000)
d_1000 = layers.MaxPooling2D(pool_size=2)(d_1000)
d_1000 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(d_1000)
d_1000 = layers.MaxPooling2D(pool_size=2)(d_1000)
d_1000 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(d_1000)
d_1000 = layers.Flatten()(d_1000)
d_1000 = layers.Dropout(0.5)(d_1000)
output_1000 = layers.Dense(1, activation="sigmoid")(d_1000)
model_1000 = keras.Model(inputs=input_1000, outputs=output_1000)
```

Model Training

```
model_1000.compile(loss="binary_crossentropy",
optimizer="adam",
metrics=["accuracy"])
```

The training dataset is utilized to train the model once it's constructed. The validation dataset is employed to assess the model's performance after each epoch. To minimize the execution time per epoch, I am using a GPU.

```
model_1000.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 (MaxPooling2  (None, 89, 89, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 43, 43, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 20, 20, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPoolin  (None, 9, 9, 256)         0
 g2D)

 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)
_________________________________________________________________
```

Model Fitting

```
callback_1000 = [keras.callbacks.ModelCheckpoint(filepath="convnet_from_scratch.keras",save_best_only=True,monitor="val_loss")]

history_1000 = model_1000.fit(train_data,epochs=100,validation_data=valid_data,callbacks=callback_1000)
```

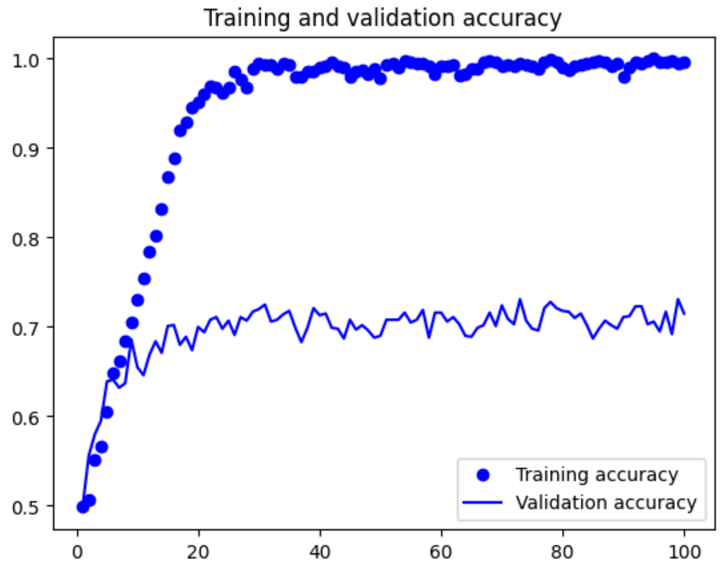
```
Epoch 85/100
63/63 [==============================] - 7s 108ms/step - loss: 0.0137 - accuracy: 0.9955 - val_loss: 2.8027 - val_accuracy: 0.6870
Epoch 86/100
63/63 [==============================] - 6s 86ms/step - loss: 0.0133 - accuracy: 0.9970 - val_loss: 2.4931 - val_accuracy: 0.6980
Epoch 87/100
63/63 [==============================] - 6s 93ms/step - loss: 0.0168 - accuracy: 0.9955 - val_loss: 2.4998 - val_accuracy: 0.7070
Epoch 88/100
63/63 [==============================] - 6s 84ms/step - loss: 0.0193 - accuracy: 0.9925 - val_loss: 2.3262 - val_accuracy: 0.7020
Epoch 89/100
63/63 [==============================] - 4s 60ms/step - loss: 0.0177 - accuracy: 0.9940 - val_loss: 2.5093 - val_accuracy: 0.6980
Epoch 90/100
63/63 [==============================] - 7s 111ms/step - loss: 0.0656 - accuracy: 0.9795 - val_loss: 2.0489 - val_accuracy: 0.7110
Epoch 91/100
63/63 [==============================] - 4s 62ms/step - loss: 0.0247 - accuracy: 0.9905 - val_loss: 2.1659 - val_accuracy: 0.7120
Epoch 92/100
63/63 [==============================] - 6s 85ms/step - loss: 0.0187 - accuracy: 0.9960 - val_loss: 2.1287 - val_accuracy: 0.7230
Epoch 93/100
63/63 [==============================] - 6s 93ms/step - loss: 0.0095 - accuracy: 0.9950 - val_loss: 2.2282 - val_accuracy: 0.7230
Epoch 94/100
63/63 [==============================] - 4s 60ms/step - loss: 0.0057 - accuracy: 0.9975 - val_loss: 2.3406 - val_accuracy: 0.7030
Epoch 95/100
63/63 [==============================] - 6s 93ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 2.6054 - val_accuracy: 0.7060
Epoch 96/100
63/63 [==============================] - 6s 84ms/step - loss: 0.0096 - accuracy: 0.9960 - val_loss: 2.8397 - val_accuracy: 0.6950
Epoch 97/100
63/63 [==============================] - 4s 63ms/step - loss: 0.0103 - accuracy: 0.9955 - val_loss: 2.8918 - val_accuracy: 0.7170
Epoch 98/100
63/63 [==============================] - 7s 104ms/step - loss: 0.0079 - accuracy: 0.9975 - val_loss: 3.1390 - val_accuracy: 0.6920
Epoch 99/100
63/63 [==============================] - 6s 80ms/step - loss: 0.0164 - accuracy: 0.9950 - val_loss: 2.2767 - val_accuracy: 0.7310
Epoch 100/100
63/63 [==============================] - 4s 62ms/step - loss: 0.0180 - accuracy: 0.9955 - val_loss: 2.5289 - val_accuracy: 0.7150
```

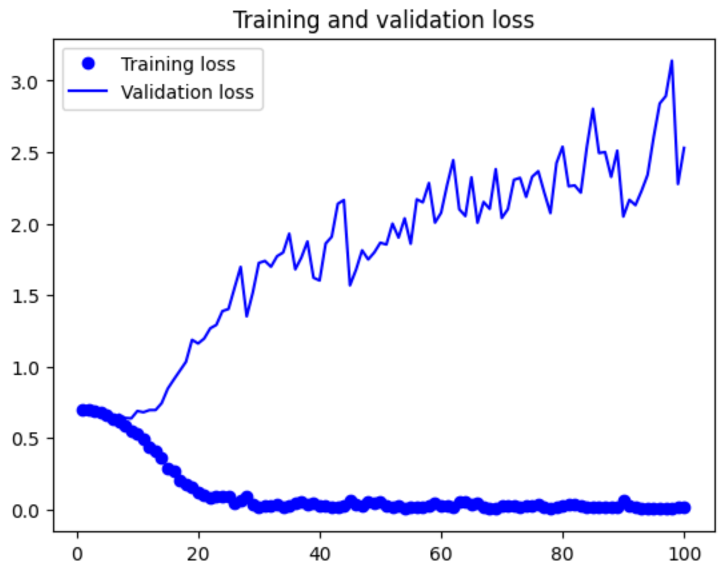
Plot for loss and accuracy during training

```
import matplotlib.pyplot as plt

accuracy = history_1000.history["accuracy"]
val_accuracy = history_1000.history["val_accuracy"]
loss = history_1000.history["loss"]
val_loss = history_1000.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

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





Test Accuracy of the model

```
test_1000 = keras.models.load_model("convnet_from_scratch.keras")
test_loss, test_acc = test_1000.evaluate(test_data)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 2s 33ms/step - loss: 0.6077 - accuracy: 0.6680
Test accuracy: 0.668
```

Q2. 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?

Using data augmentation

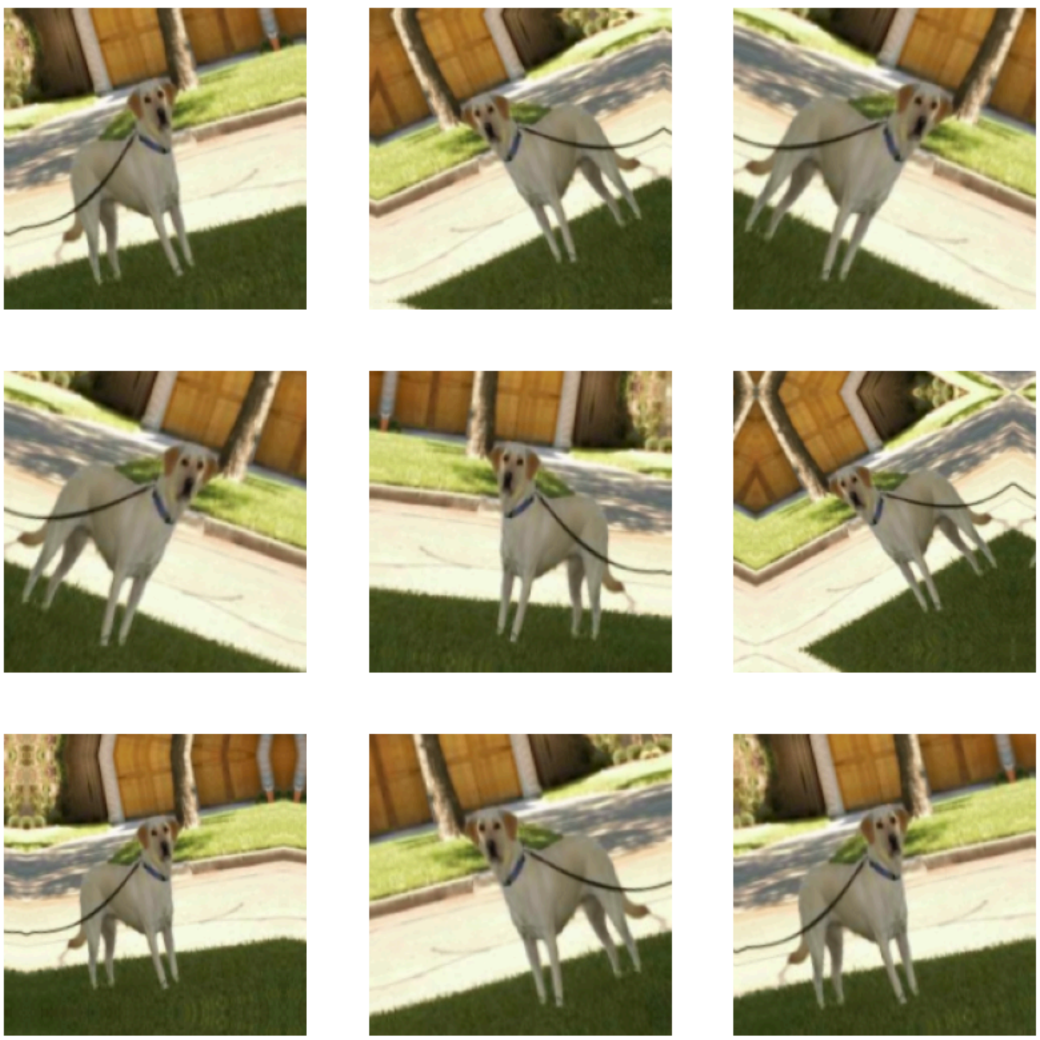
```
shutil.rmtree("./cats_vs_dogs_small_Q2", ignore_errors=True)
o_dir = pathlib.Path("train")
n_dir = pathlib.Path("cats_vs_dogs_small_Q2")
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = n_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg" for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=o_dir / fname,
            dst=dir / fname)

make_subset("train", start_index=500, end_index=2000)
make_subset("validation", start_index=2000, end_index=2500)
make_subset("test", start_index=2500, end_index=3000)

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

plt.figure(figsize=(10, 10))

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



Convolutional neural network with dropout and picture augmentation

```
input_1500 = keras.Input(shape=(180, 180, 3))

d_2000 = augmentation(input_1500)
d_2000 = layers.Rescaling(1./255)(d_2000)
d_2000 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.Flatten()(d_2000)
d_2000 = layers.Dropout(0.5)(d_2000)

output_1500 = layers.Dense(1, activation="sigmoid")(d_2000)
model_1500 = keras.Model(inputs=input_1500, outputs=output_1500)

model_1500.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])

callback_1500 = [keras.callbacks.ModelCheckpoint(filepath="convnet_from_scratch_with_augmentation_info.keras",save_best_only=True,monitor="v

history_1500 = model_1500.fit(train_data,epochs=100,validation_data=valid_data,callbacks=callback_1500)
```

```
Epoch 1/100
63/63 [==============================] - 9s 71ms/step - loss: 0.6953 - accuracy: 0.5010 - val_loss: 0.6929 - val_accuracy: 0.5000
Epoch 2/100
63/63 [==============================] - 5s 78ms/step - loss: 0.6934 - accuracy: 0.4875 - val_loss: 0.6921 - val_accuracy: 0.5970
Epoch 3/100
63/63 [==============================] - 7s 102ms/step - loss: 0.6930 - accuracy: 0.5175 - val_loss: 0.6909 - val_accuracy: 0.5400
Epoch 4/100
63/63 [==============================] - 4s 63ms/step - loss: 0.6836 - accuracy: 0.5690 - val_loss: 0.6815 - val_accuracy: 0.6020
Epoch 5/100
63/63 [==============================] - 8s 117ms/step - loss: 0.6712 - accuracy: 0.5995 - val_loss: 0.6786 - val_accuracy: 0.6050
Epoch 6/100
63/63 [==============================] - 7s 109ms/step - loss: 0.6591 - accuracy: 0.6195 - val_loss: 0.6458 - val_accuracy: 0.6390
Epoch 7/100
63/63 [==============================] - 7s 103ms/step - loss: 0.6526 - accuracy: 0.6225 - val_loss: 0.6640 - val_accuracy: 0.5930
Epoch 8/100
63/63 [==============================] - 4s 63ms/step - loss: 0.6333 - accuracy: 0.6365 - val_loss: 0.6373 - val_accuracy: 0.6490
Epoch 9/100
63/63 [==============================] - 4s 66ms/step - loss: 0.6390 - accuracy: 0.6490 - val_loss: 0.6284 - val_accuracy: 0.6570
Epoch 10/100
63/63 [==============================] - 8s 116ms/step - loss: 0.6339 - accuracy: 0.6350 - val_loss: 0.6163 - val_accuracy: 0.6740
Epoch 11/100
63/63 [==============================] - 4s 63ms/step - loss: 0.6250 - accuracy: 0.6570 - val_loss: 0.5983 - val_accuracy: 0.6840
Epoch 12/100
63/63 [==============================] - 4s 62ms/step - loss: 0.6200 - accuracy: 0.6590 - val_loss: 0.6220 - val_accuracy: 0.6700
Epoch 13/100
63/63 [==============================] - 7s 111ms/step - loss: 0.6136 - accuracy: 0.6545 - val_loss: 0.6237 - val_accuracy: 0.6540
Epoch 14/100
63/63 [==============================] - 5s 70ms/step - loss: 0.6110 - accuracy: 0.6765 - val_loss: 0.6102 - val_accuracy: 0.6560
Epoch 15/100
63/63 [==============================] - 4s 59ms/step - loss: 0.5881 - accuracy: 0.6865 - val_loss: 0.6656 - val_accuracy: 0.6550
Epoch 16/100
63/63 [==============================] - 6s 87ms/step - loss: 0.5900 - accuracy: 0.6975 - val_loss: 0.6098 - val_accuracy: 0.6630
Epoch 17/100
63/63 [==============================] - 6s 89ms/step - loss: 0.5913 - accuracy: 0.6910 - val_loss: 0.6481 - val_accuracy: 0.6560
Epoch 18/100
63/63 [==============================] - 5s 75ms/step - loss: 0.5713 - accuracy: 0.7005 - val_loss: 0.5844 - val_accuracy: 0.7080
Epoch 19/100
63/63 [==============================] - 8s 118ms/step - loss: 0.5389 - accuracy: 0.7230 - val_loss: 0.5701 - val_accuracy: 0.7120
Epoch 20/100
63/63 [==============================] - 5s 67ms/step - loss: 0.5434 - accuracy: 0.7415 - val_loss: 0.5381 - val_accuracy: 0.7300
Epoch 21/100
63/63 [==============================] - 4s 60ms/step - loss: 0.5254 - accuracy: 0.7340 - val_loss: 0.5625 - val_accuracy: 0.7230
Epoch 22/100
63/63 [==============================] - 7s 104ms/step - loss: 0.5170 - accuracy: 0.7470 - val_loss: 0.6288 - val_accuracy: 0.7040
Epoch 23/100
63/63 [==============================] - 5s 75ms/step - loss: 0.4858 - accuracy: 0.7705 - val_loss: 0.5158 - val_accuracy: 0.7430
Epoch 24/100
63/63 [==============================] - 4s 62ms/step - loss: 0.4598 - accuracy: 0.7830 - val_loss: 0.5388 - val_accuracy: 0.7420
Epoch 25/100
63/63 [==============================] - 5s 71ms/step - loss: 0.4684 - accuracy: 0.7830 - val_loss: 0.5244 - val_accuracy: 0.7520
Epoch 26/100
63/63 [==============================] - 6s 96ms/step - loss: 0.4855 - accuracy: 0.7685 - val_loss: 0.5201 - val_accuracy: 0.7570
Epoch 27/100
63/63 [==============================] - 4s 63ms/step - loss: 0.4666 - accuracy: 0.7765 - val_loss: 0.5181 - val_accuracy: 0.7630
Epoch 28/100
63/63 [==============================] - 4s 62ms/step - loss: 0.4408 - accuracy: 0.7865 - val_loss: 0.5043 - val_accuracy: 0.7680
Epoch 29/100
```

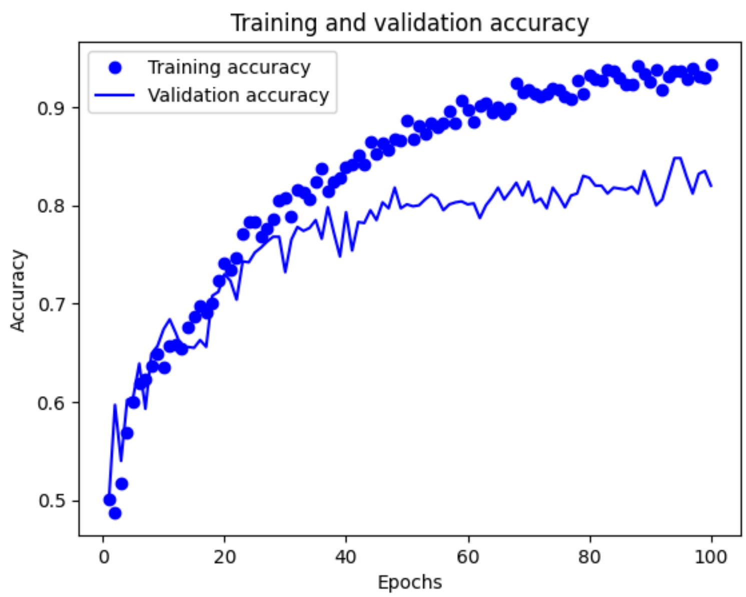
```
63/63 [==============================] - 7s 108ms/step - loss: 0.4205 - accuracy: 0.8055 - val_loss: 0.5002 - val_accuracy: 0.7680
```

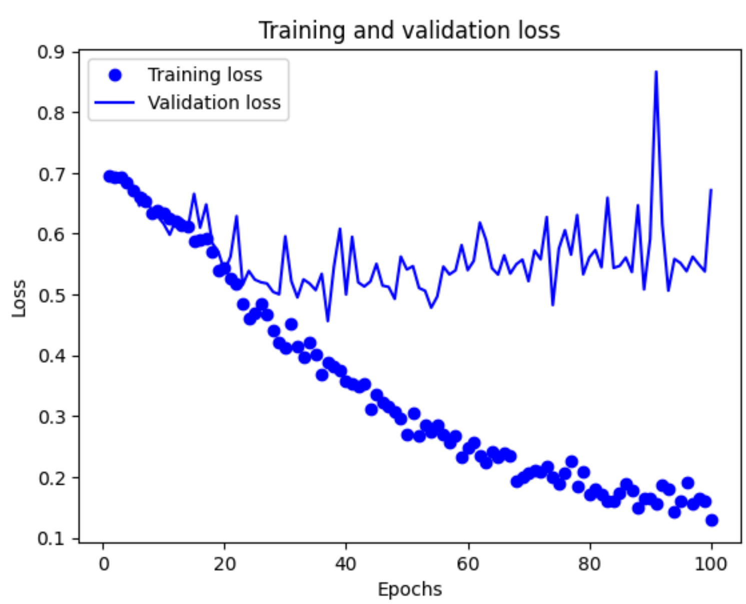
Plots for loss and accuracy during training

```
accuracy_1500 = history_1500.history["accuracy"]
valac_1500 = history_1500.history["val_accuracy"]
loss_1500 = history_1500.history["loss"]
valloss_1500 = history_1500.history["val_loss"]
epochs = range(1, len(accuracy_1500) + 1)

plt.plot(epochs, accuracy_1500, "bo", label="Training accuracy")
plt.plot(epochs, valac_1500, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss_1500, "bo", label="Training loss")
plt.plot(epochs, valloss_1500, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```





Test Accuracy of model

```
test_1500 = keras.models.load_model("convnet_from_scratch_with_augmentation_info.keras")

test_loss, test_acc = test_1500.evaluate(test_data)

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

```
32/32 [==============================] - 1s 30ms/step - loss: 0.4643 - accuracy: 0.7840
Test accuracy: 0.784
```

Q3. 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.

Expanding the training sample to 2000, while maintaining the validation and test sets at their previous size of 500 samples each.

```
n_dir = pathlib.Path("cats_vs_dogs_small_Q3")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = n_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:
            shutil.copyfile(src=o_dir / fname,
            dst=dir / fname)

make_subset("train", start_index=500, end_index=2500)
make_subset("validation", start_index=2500, end_index=3000)
make_subset("test", start_index=3000, end_index=3500)

input_2000 = keras.Input(shape=(180, 180, 3))
d_2000 = augmentation(input_2000)
d_2000 = layers.Rescaling(1./255)(d_2000)
d_2000 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.MaxPooling2D(pool_size=2)(d_2000)
d_2000 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(d_2000)
d_2000 = layers.Flatten()(d_2000)
d_2000 = layers.Dropout(0.5)(d_2000)

output_2000 = layers.Dense(1, activation="sigmoid")(d_2000)
model_2000 = keras.Model(inputs=input_2000, outputs=output_2000)

model_2000.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])

callback_2000 = [keras.callbacks.ModelCheckpoint(filepath="convnet_from_scratch_with_augmentation_info.keras",save_best_only=True,monitor="v

history_2000 = model_2000.fit(train_data,epochs=100,validation_data=valid_data,callbacks=callback_2000)
```

```
Epoch 1/100
63/63 [==============================] - 10s 118ms/step - loss: 0.6953 - accuracy: 0.5090 - val_loss: 0.6931 - val_accuracy: 0.4990
Epoch 2/100
63/63 [==============================] - 4s 63ms/step - loss: 0.6935 - accuracy: 0.4795 - val_loss: 0.6931 - val_accuracy: 0.4980
Epoch 3/100
63/63 [==============================] - 4s 60ms/step - loss: 0.6963 - accuracy: 0.5040 - val_loss: 0.6934 - val_accuracy: 0.5000
Epoch 4/100
63/63 [==============================] - 6s 86ms/step - loss: 0.6936 - accuracy: 0.4935 - val_loss: 0.6930 - val_accuracy: 0.5000
Epoch 5/100
63/63 [==============================] - 8s 122ms/step - loss: 0.6933 - accuracy: 0.4965 - val_loss: 0.6918 - val_accuracy: 0.5430
Epoch 6/100
63/63 [==============================] - 6s 82ms/step - loss: 0.6920 - accuracy: 0.5055 - val_loss: 0.6872 - val_accuracy: 0.5450
Epoch 7/100
63/63 [==============================] - 6s 96ms/step - loss: 0.6920 - accuracy: 0.5110 - val_loss: 0.6862 - val_accuracy: 0.5590
Epoch 8/100
63/63 [==============================] - 6s 83ms/step - loss: 0.6862 - accuracy: 0.5485 - val_loss: 0.6849 - val_accuracy: 0.5680
```

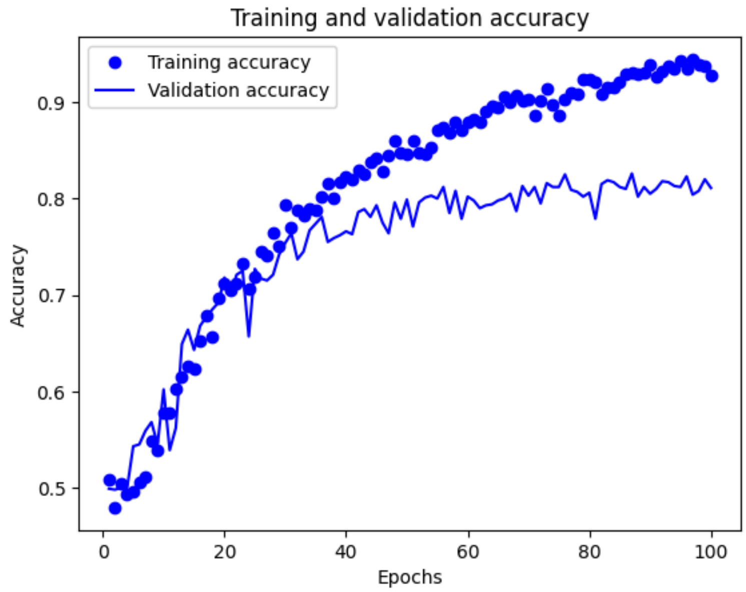
```
Epoch 9/100
63/63 [==============================] - 9s 146ms/step - loss: 0.6848 - accuracy: 0.5395 - val_loss: 0.6870 - val_accuracy: 0.5450
Epoch 10/100
63/63 [==============================] - 6s 76ms/step - loss: 0.6796 - accuracy: 0.5775 - val_loss: 0.6719 - val_accuracy: 0.6020
Epoch 11/100
63/63 [==============================] - 4s 61ms/step - loss: 0.6724 - accuracy: 0.5780 - val_loss: 0.6910 - val_accuracy: 0.5390
Epoch 12/100
63/63 [==============================] - 5s 84ms/step - loss: 0.6707 - accuracy: 0.6030 - val_loss: 0.6906 - val_accuracy: 0.5620
Epoch 13/100
63/63 [==============================] - 4s 63ms/step - loss: 0.6505 - accuracy: 0.6150 - val_loss: 0.6380 - val_accuracy: 0.6490
Epoch 14/100
63/63 [==============================] - 4s 64ms/step - loss: 0.6473 - accuracy: 0.6260 - val_loss: 0.6358 - val_accuracy: 0.6640
Epoch 15/100
63/63 [==============================] - 8s 115ms/step - loss: 0.6416 - accuracy: 0.6240 - val_loss: 0.6403 - val_accuracy: 0.6430
Epoch 16/100
63/63 [==============================] - 8s 122ms/step - loss: 0.6282 - accuracy: 0.6525 - val_loss: 0.6174 - val_accuracy: 0.6680
Epoch 17/100
63/63 [==============================] - 4s 62ms/step - loss: 0.6132 - accuracy: 0.6785 - val_loss: 0.5971 - val_accuracy: 0.6770
Epoch 18/100
63/63 [==============================] - 5s 75ms/step - loss: 0.6264 - accuracy: 0.6570 - val_loss: 0.5981 - val_accuracy: 0.6850
Epoch 19/100
63/63 [==============================] - 7s 98ms/step - loss: 0.5866 - accuracy: 0.6960 - val_loss: 0.5850 - val_accuracy: 0.6920
Epoch 20/100
63/63 [==============================] - 4s 63ms/step - loss: 0.5739 - accuracy: 0.7120 - val_loss: 0.5636 - val_accuracy: 0.7180
Epoch 21/100
63/63 [==============================] - 4s 60ms/step - loss: 0.5733 - accuracy: 0.7050 - val_loss: 0.5736 - val_accuracy: 0.7010
Epoch 22/100
63/63 [==============================] - 6s 96ms/step - loss: 0.5675 - accuracy: 0.7125 - val_loss: 0.5543 - val_accuracy: 0.7210
Epoch 23/100
63/63 [==============================] - 5s 79ms/step - loss: 0.5182 - accuracy: 0.7325 - val_loss: 0.5625 - val_accuracy: 0.7250
Epoch 24/100
63/63 [==============================] - 6s 84ms/step - loss: 0.5495 - accuracy: 0.7065 - val_loss: 0.6411 - val_accuracy: 0.6570
Epoch 25/100
63/63 [==============================] - 6s 92ms/step - loss: 0.5495 - accuracy: 0.7190 - val_loss: 0.5337 - val_accuracy: 0.7270
Epoch 26/100
63/63 [==============================] - 7s 107ms/step - loss: 0.5108 - accuracy: 0.7455 - val_loss: 0.5415 - val_accuracy: 0.7170
Epoch 27/100
63/63 [==============================] - 4s 61ms/step - loss: 0.5235 - accuracy: 0.7415 - val_loss: 0.5686 - val_accuracy: 0.7150
Epoch 28/100
63/63 [==============================] - 4s 61ms/step - loss: 0.4822 - accuracy: 0.7640 - val_loss: 0.5715 - val_accuracy: 0.7210
Epoch 29/100
63/63 [==============================] - 7s 110ms/step - loss: 0.5044 - accuracy: 0.7510 - val_loss: 0.5046 - val_accuracy: 0.7420
```

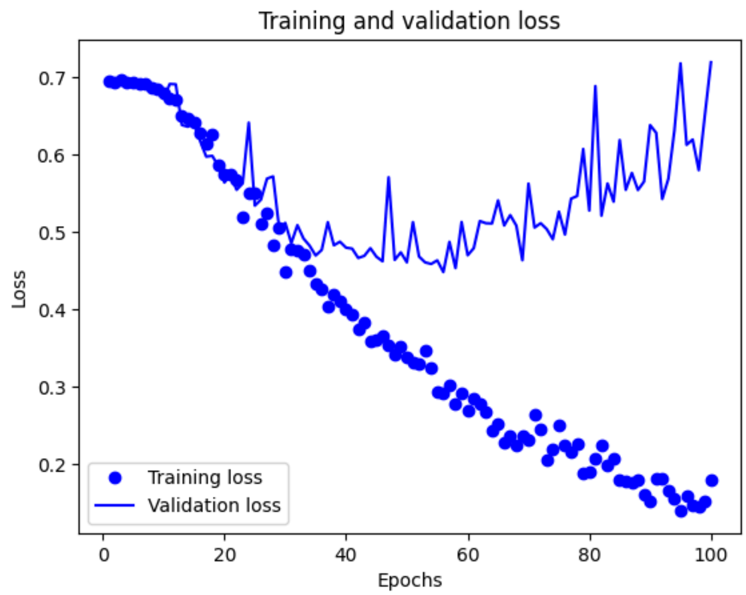
Plots for loss and accuracy during training

```
accuracy_2000 = history_2000.history["accuracy"]
valac_2000 = history_2000.history["val_accuracy"]
loss_2000 = history_2000.history["loss"]
valloss_2000 = history_2000.history["val_loss"]
epochs = range(1, len(accuracy_2000) + 1)

plt.plot(epochs, accuracy_2000, "bo", label="Training accuracy")
plt.plot(epochs, valac_2000, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss_2000, "bo", label="Training loss")
plt.plot(epochs, valloss_2000, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```





Test Accuracy of the model

```
test_2000 = keras.models.load_model("convnet_from_scratch_with_augmentation_info.keras")

test_loss, test_acc = test_2000.evaluate(test_data)

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

```
32/32 [==============================] - 2s 55ms/step - loss: 0.4756 - accuracy: 0.7950
Test accuracy: 0.795
```

Q4. 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.

Instantiating the VGG16 convolutional base

```
convoluted_b = 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.
58889256/58889256 [==============================] - 4s 0us/step
```

```
convoluted_b.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)
_________________________________________________________________
```

Pretrained model for feature extraction without data augmentation

```
def get_features_and_labels(dataset):
    all_feature = []
    all_label = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = convoluted_b.predict(preprocessed_images)
        all_feature.append(features)
        all_label.append(labels)
    return np.concatenate(all_feature), np.concatenate(all_label)

train_features, train_labels = get_features_and_labels(train_data)
val_features, val_labels = get_features_and_labels(valid_data)
test_features, test_labels = get_features_and_labels(test_data)
```

```
1/1 [==============================] - 5s 5s/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 35ms/step
```

```
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 82ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 50ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 28ms/step
```

```
train_features.shape
```

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

Model Fitting

```
input_6000 = keras.Input(shape=(5, 5, 512))

d_6000 = layers.Flatten()(input_6000)
d_6000 = layers.Dense(256)(d_6000)
d_6000 = layers.Dropout(0.5)(d_6000)

output_6000 = layers.Dense(1, activation="sigmoid")(d_6000)

model_6000 = keras.Model(input_6000, output_6000)
model_6000.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accuracy"])

callback_6000 = [keras.callbacks.ModelCheckpoint(filepath="feature_extraction.keras",save_best_only=True,monitor="val_loss")]

history_6000 = model_6000.fit(train_features, train_labels,epochs=100,validation_data=(val_features, val_labels),callbacks=callback_6000)
```

```
Epoch 1/100
63/63 [==============================] - 2s 12ms/step - loss: 16.1286 - accuracy: 0.9180 - val_loss: 5.8465 - val_accuracy: 0.9650
Epoch 2/100
63/63 [==============================] - 0s 5ms/step - loss: 2.7064 - accuracy: 0.9795 - val_loss: 8.9551 - val_accuracy: 0.9540
Epoch 3/100
63/63 [==============================] - 0s 8ms/step - loss: 2.0651 - accuracy: 0.9850 - val_loss: 4.8202 - val_accuracy: 0.9710
Epoch 4/100
63/63 [==============================] - 0s 8ms/step - loss: 0.8275 - accuracy: 0.9930 - val_loss: 4.5501 - val_accuracy: 0.9740
Epoch 5/100
63/63 [==============================] - 0s 5ms/step - loss: 0.6588 - accuracy: 0.9915 - val_loss: 6.7217 - val_accuracy: 0.9710
Epoch 6/100
63/63 [==============================] - 0s 5ms/step - loss: 1.0188 - accuracy: 0.9940 - val_loss: 5.5107 - val_accuracy: 0.9750
```

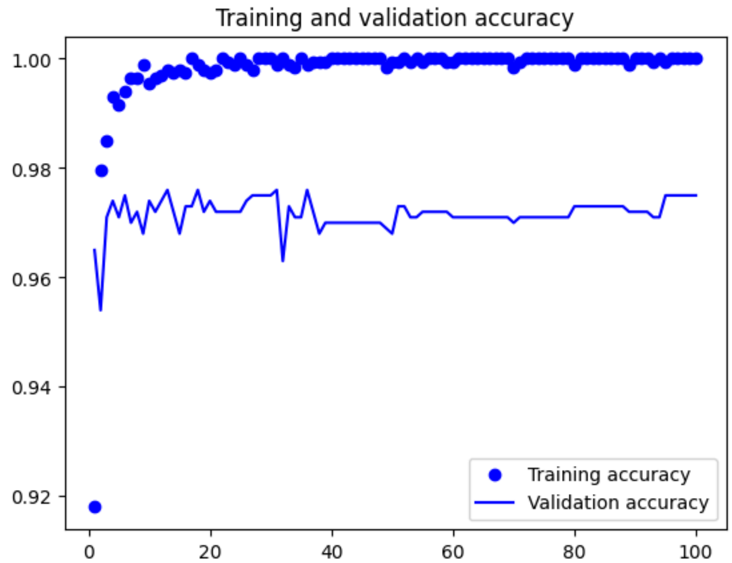
```
Epoch 7/100
63/63 [==============================] - 0s 7ms/step - loss: 0.4168 - accuracy: 0.9965 - val_loss: 6.8885 - val_accuracy: 0.9700
Epoch 8/100
63/63 [==============================] - 0s 8ms/step - loss: 0.4991 - accuracy: 0.9965 - val_loss: 7.2080 - val_accuracy: 0.9720
Epoch 9/100
63/63 [==============================] - 1s 9ms/step - loss: 0.0522 - accuracy: 0.9990 - val_loss: 6.4618 - val_accuracy: 0.9680
Epoch 10/100
63/63 [==============================] - 1s 8ms/step - loss: 0.3578 - accuracy: 0.9955 - val_loss: 8.8637 - val_accuracy: 0.9740
Epoch 11/100
63/63 [==============================] - 1s 9ms/step - loss: 0.3205 - accuracy: 0.9965 - val_loss: 7.2026 - val_accuracy: 0.9720
Epoch 12/100
63/63 [==============================] - 1s 8ms/step - loss: 0.1597 - accuracy: 0.9970 - val_loss: 6.5647 - val_accuracy: 0.9740
Epoch 13/100
63/63 [==============================] - 1s 9ms/step - loss: 0.2190 - accuracy: 0.9980 - val_loss: 7.0175 - val_accuracy: 0.9760
Epoch 14/100
63/63 [==============================] - 1s 8ms/step - loss: 0.4229 - accuracy: 0.9975 - val_loss: 9.0027 - val_accuracy: 0.9720
Epoch 15/100
63/63 [==============================] - 1s 8ms/step - loss: 0.3797 - accuracy: 0.9980 - val_loss: 8.7803 - val_accuracy: 0.9680
Epoch 16/100
63/63 [==============================] - 1s 9ms/step - loss: 0.2501 - accuracy: 0.9975 - val_loss: 6.4431 - val_accuracy: 0.9730
Epoch 17/100
63/63 [==============================] - 0s 8ms/step - loss: 1.1497e-27 - accuracy: 1.0000 - val_loss: 6.4431 - val_accuracy: 0.9730
Epoch 18/100
63/63 [==============================] - 1s 9ms/step - loss: 0.0892 - accuracy: 0.9990 - val_loss: 6.5740 - val_accuracy: 0.9760
Epoch 19/100
63/63 [==============================] - 1s 8ms/step - loss: 0.2666 - accuracy: 0.9980 - val_loss: 7.0606 - val_accuracy: 0.9720
Epoch 20/100
63/63 [==============================] - 1s 9ms/step - loss: 0.3169 - accuracy: 0.9975 - val_loss: 7.3652 - val_accuracy: 0.9740
Epoch 21/100
63/63 [==============================] - 0s 8ms/step - loss: 0.2282 - accuracy: 0.9980 - val_loss: 6.9580 - val_accuracy: 0.9720
Epoch 22/100
63/63 [==============================] - 1s 8ms/step - loss: 7.2121e-10 - accuracy: 1.0000 - val_loss: 6.9577 - val_accuracy: 0.9720
Epoch 23/100
63/63 [==============================] - 0s 5ms/step - loss: 0.0611 - accuracy: 0.9995 - val_loss: 7.6014 - val_accuracy: 0.9720
Epoch 24/100
63/63 [==============================] - 0s 6ms/step - loss: 0.2457 - accuracy: 0.9990 - val_loss: 7.6962 - val_accuracy: 0.9720
Epoch 25/100
63/63 [==============================] - 0s 5ms/step - loss: 1.0433e-37 - accuracy: 1.0000 - val_loss: 7.6962 - val_accuracy: 0.9720
Epoch 26/100
63/63 [==============================] - 0s 5ms/step - loss: 0.0591 - accuracy: 0.9990 - val_loss: 6.6485 - val_accuracy: 0.9740
Epoch 27/100
63/63 [==============================] - 0s 7ms/step - loss: 0.1373 - accuracy: 0.9980 - val_loss: 7.1424 - val_accuracy: 0.9750
Epoch 28/100
63/63 [==============================] - 1s 8ms/step - loss: 2.3124e-36 - accuracy: 1.0000 - val_loss: 7.1424 - val_accuracy: 0.9750
```

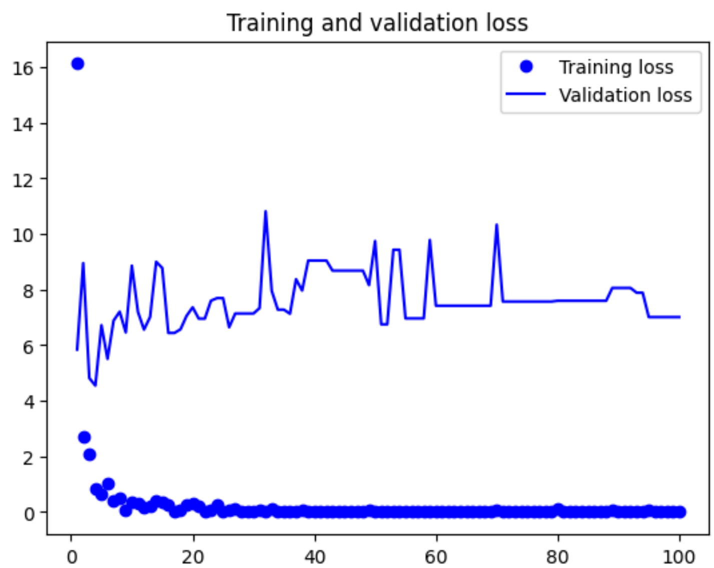
Plot for loss and accuracy during training

```
accuracy_6000 = history_6000.history["accuracy"]
valac_6000 = history_6000.history["val_accuracy"]
loss_6000 = history_6000.history["loss"]
valloss_6000 = history_6000.history["val_loss"]

epochs = range(1, len(accuracy_6000) + 1)
plt.plot(epochs, accuracy_6000, "bo", label="Training accuracy")
plt.plot(epochs, valac_6000, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()

plt.plot(epochs, loss_6000, "bo", label="Training loss")
plt.plot(epochs, valloss_6000, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```





Freezing and Unfreezing the Pre-trained Convolutional Base

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

convoluted_base.trainable = False
convoluted_base.trainable = True

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

convoluted_base.trainable = False

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

```
This is the number of trainable weights before freezing the conv base: 26
This is the number of trainable weights after freezing the conv base: 0
```

Model is now performing with a classifier and agumentation to convulation base

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

input_cb = keras.Input(shape=(180, 180, 3))
d_cb = augmented_cb(input_cb)
d_cb =keras.layers.Lambda(lambda x: keras.applications.vgg16.preprocess_input(x))(d_cb)
d_cb = convoluted_base(d_cb)
d_cb = layers.Flatten()(d_cb)
d_cb = layers.Dense(256)(d_cb)
d_cb = layers.Dropout(0.5)(d_cb)

output_cb = layers.Dense(1, activation="sigmoid")(d_cb)

modelx = keras.Model(input_cb, output_cb)

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

callback_cb = [keras.callbacks.ModelCheckpoint(filepath="features_extraction_with_augmentation2.keras",save_best_only=True,monitor="val_loss

history_cb = modelx.fit(train_data,epochs=100,validation_data=valid_data,callbacks=callback_cb)
```

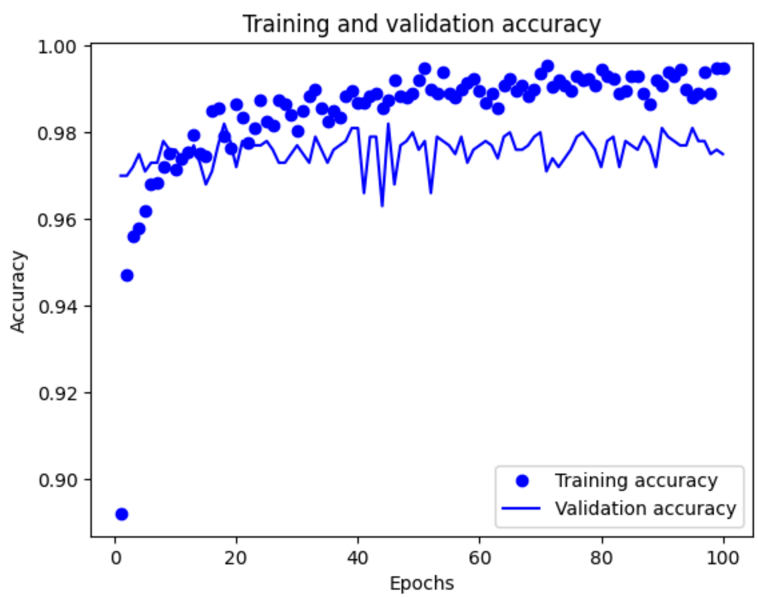
```
Epoch 1/100
63/63 [==============================] - 13s 162ms/step - loss: 20.5766 - accuracy: 0.8920 - val_loss: 5.1385 - val_accuracy: 0.9700
Epoch 2/100
63/63 [==============================] - 11s 168ms/step - loss: 8.1243 - accuracy: 0.9470 - val_loss: 3.9042 - val_accuracy: 0.9700
Epoch 3/100
63/63 [==============================] - 10s 163ms/step - loss: 5.1447 - accuracy: 0.9560 - val_loss: 3.7450 - val_accuracy: 0.9720
Epoch 4/100
63/63 [==============================] - 10s 152ms/step - loss: 5.0218 - accuracy: 0.9580 - val_loss: 3.9683 - val_accuracy: 0.9750
Epoch 5/100
63/63 [==============================] - 10s 152ms/step - loss: 4.7088 - accuracy: 0.9620 - val_loss: 5.9792 - val_accuracy: 0.9710
Epoch 6/100
63/63 [==============================] - 10s 154ms/step - loss: 3.8578 - accuracy: 0.9680 - val_loss: 4.2969 - val_accuracy: 0.9730
Epoch 7/100
63/63 [==============================] - 11s 177ms/step - loss: 3.6115 - accuracy: 0.9685 - val_loss: 5.7969 - val_accuracy: 0.9730
Epoch 8/100
63/63 [==============================] - 12s 186ms/step - loss: 3.1932 - accuracy: 0.9720 - val_loss: 3.7103 - val_accuracy: 0.9780
Epoch 9/100
63/63 [==============================] - 11s 164ms/step - loss: 2.9859 - accuracy: 0.9750 - val_loss: 3.5621 - val_accuracy: 0.9760
Epoch 10/100
63/63 [==============================] - 10s 152ms/step - loss: 2.8674 - accuracy: 0.9715 - val_loss: 3.8491 - val_accuracy: 0.9760
Epoch 11/100
63/63 [==============================] - 10s 158ms/step - loss: 2.2934 - accuracy: 0.9740 - val_loss: 3.8310 - val_accuracy: 0.9740
Epoch 12/100
63/63 [==============================] - 12s 183ms/step - loss: 1.8120 - accuracy: 0.9755 - val_loss: 5.0486 - val_accuracy: 0.9740
Epoch 13/100
63/63 [==============================] - 10s 159ms/step - loss: 2.2276 - accuracy: 0.9795 - val_loss: 3.4299 - val_accuracy: 0.9770
Epoch 14/100
63/63 [==============================] - 10s 147ms/step - loss: 2.5800 - accuracy: 0.9750 - val_loss: 4.1733 - val_accuracy: 0.9730
Epoch 15/100
63/63 [==============================] - 10s 159ms/step - loss: 3.1009 - accuracy: 0.9745 - val_loss: 4.8212 - val_accuracy: 0.9680
Epoch 16/100
63/63 [==============================] - 10s 159ms/step - loss: 1.0595 - accuracy: 0.9850 - val_loss: 2.6045 - val_accuracy: 0.9710
Epoch 17/100
63/63 [==============================] - 12s 182ms/step - loss: 1.1761 - accuracy: 0.9855 - val_loss: 2.7369 - val_accuracy: 0.9770
Epoch 18/100
63/63 [==============================] - 11s 161ms/step - loss: 1.9031 - accuracy: 0.9790 - val_loss: 2.4094 - val_accuracy: 0.9820
Epoch 19/100
63/63 [==============================] - 10s 149ms/step - loss: 1.6831 - accuracy: 0.9765 - val_loss: 2.8524 - val_accuracy: 0.9780
Epoch 20/100
63/63 [==============================] - 10s 155ms/step - loss: 0.8528 - accuracy: 0.9865 - val_loss: 3.5713 - val_accuracy: 0.9720
Epoch 21/100
63/63 [==============================] - 12s 185ms/step - loss: 1.3964 - accuracy: 0.9835 - val_loss: 3.3081 - val_accuracy: 0.9780
Epoch 22/100
63/63 [==============================] - 10s 148ms/step - loss: 1.5905 - accuracy: 0.9775 - val_loss: 3.0353 - val_accuracy: 0.9770
Epoch 23/100
63/63 [==============================] - 10s 157ms/step - loss: 1.5273 - accuracy: 0.9810 - val_loss: 2.7537 - val_accuracy: 0.9770
Epoch 24/100
63/63 [==============================] - 11s 179ms/step - loss: 0.6598 - accuracy: 0.9875 - val_loss: 2.9536 - val_accuracy: 0.9770
Epoch 25/100
63/63 [==============================] - 10s 149ms/step - loss: 1.0744 - accuracy: 0.9825 - val_loss: 2.7996 - val_accuracy: 0.9780
Epoch 26/100
63/63 [==============================] - 12s 187ms/step - loss: 1.2579 - accuracy: 0.9815 - val_loss: 2.3291 - val_accuracy: 0.9760
Epoch 27/100
63/63 [==============================] - 10s 150ms/step - loss: 0.6084 - accuracy: 0.9875 - val_loss: 2.9719 - val_accuracy: 0.9730
Epoch 28/100
63/63 [==============================] - 12s 185ms/step - loss: 0.8149 - accuracy: 0.9865 - val_loss: 2.6522 - val_accuracy: 0.9730
Epoch 29/100
63/63 [==============================] - 10s 154ms/step - loss: 1.0614 - accuracy: 0.9840 - val_loss: 3.5974 - val_accuracy: 0.9750
```

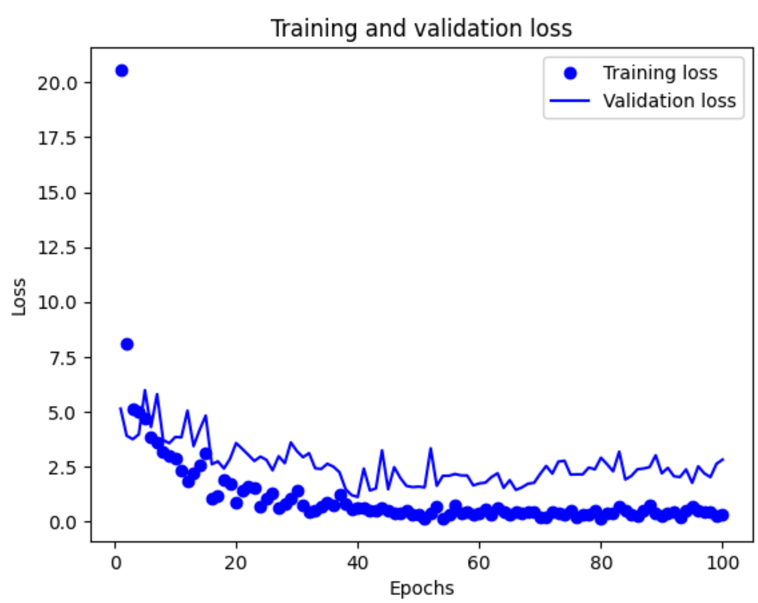
Plot for loss and accuracy during training

```
accuracy_cb = history_cb.history["accuracy"]
valac_cb = history_cb.history["val_accuracy"]
loss_cb = history_cb.history["loss"]
valloss_cb = history_cb.history["val_loss"]
epochs = range(1, len(accuracy_cb) + 1)

plt.plot(epochs, accuracy_cb, "bo", label="Training accuracy")
plt.plot(epochs, valac_cb, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss_cb, "bo", label="Training loss")
plt.plot(epochs, valloss_cb, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```





Test Accuracy of the model

```
test_cb = keras.models.load_model("features_extraction_with_augmentation2.keras",safe_mode=False)

test_loss, test_acc = test_cb.evaluate(test_data)

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

```
32/32 [==============================] - 4s 92ms/step - loss: 2.5513 - accuracy: 0.9720
Test accuracy: 0.972
```

Fine-tuning a pretrained model

```
convoluted_base.trainable = True

for layer in convoluted_base.layers[:-4]:
    layer.trainable = False

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

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

history_tuning = modelx.fit(train_data,epochs=100,validation_data=valid_data,callbacks=callback_tuning)
```

```
Epoch 63/100
63/63 [==============================] - 12s 181ms/step - loss: 0.0622 - accuracy: 0.9980 - val_loss: 1.5330 - val_accuracy: 0.9760
Epoch 64/100
63/63 [==============================] - 11s 176ms/step - loss: 0.0415 - accuracy: 0.9980 - val_loss: 1.5459 - val_accuracy: 0.9780
Epoch 65/100
63/63 [==============================] - 11s 172ms/step - loss: 0.1867 - accuracy: 0.9960 - val_loss: 1.5336 - val_accuracy: 0.9780
Epoch 66/100
63/63 [==============================] - 13s 208ms/step - loss: 0.1026 - accuracy: 0.9985 - val_loss: 1.5110 - val_accuracy: 0.9780
Epoch 67/100
63/63 [==============================] - 12s 180ms/step - loss: 0.0759 - accuracy: 0.9985 - val_loss: 1.4829 - val_accuracy: 0.9760
Epoch 68/100
63/63 [==============================] - 13s 206ms/step - loss: 0.0725 - accuracy: 0.9965 - val_loss: 1.4418 - val_accuracy: 0.9770
Epoch 69/100
63/63 [==============================] - 12s 176ms/step - loss: 0.0367 - accuracy: 0.9995 - val_loss: 2.2424 - val_accuracy: 0.9780
Epoch 70/100
63/63 [==============================] - 11s 175ms/step - loss: 0.1862 - accuracy: 0.9960 - val_loss: 1.5632 - val_accuracy: 0.9840
Epoch 71/100
63/63 [==============================] - 11s 163ms/step - loss: 1.6877e-11 - accuracy: 1.0000 - val_loss: 1.5632 - val_accuracy: 0.98
Epoch 72/100
63/63 [==============================] - 11s 168ms/step - loss: 0.0921 - accuracy: 0.9970 - val_loss: 1.4894 - val_accuracy: 0.9810
Epoch 73/100
63/63 [==============================] - 12s 180ms/step - loss: 0.0159 - accuracy: 0.9990 - val_loss: 1.3530 - val_accuracy: 0.9820
Epoch 74/100
63/63 [==============================] - 13s 199ms/step - loss: 0.0513 - accuracy: 0.9990 - val_loss: 1.6436 - val_accuracy: 0.9820
Epoch 75/100
63/63 [==============================] - 11s 167ms/step - loss: 0.0852 - accuracy: 0.9990 - val_loss: 1.4018 - val_accuracy: 0.9830
Epoch 76/100
63/63 [==============================] - 11s 175ms/step - loss: 0.0584 - accuracy: 0.9990 - val_loss: 1.5203 - val_accuracy: 0.9830
Epoch 77/100
63/63 [==============================] - 11s 173ms/step - loss: 0.0376 - accuracy: 0.9985 - val_loss: 1.5500 - val_accuracy: 0.9810
Epoch 78/100
63/63 [==============================] - 11s 178ms/step - loss: 0.0488 - accuracy: 0.9980 - val_loss: 1.6516 - val_accuracy: 0.9810
Epoch 79/100
63/63 [==============================] - 11s 170ms/step - loss: 0.0176 - accuracy: 0.9990 - val_loss: 1.5627 - val_accuracy: 0.9800
Epoch 80/100
63/63 [==============================] - 13s 198ms/step - loss: 0.0806 - accuracy: 0.9985 - val_loss: 1.5944 - val_accuracy: 0.9850
Epoch 81/100
63/63 [==============================] - 10s 163ms/step - loss: 0.0183 - accuracy: 0.9990 - val_loss: 1.6598 - val_accuracy: 0.9820
Epoch 82/100
63/63 [==============================] - 13s 200ms/step - loss: 0.0126 - accuracy: 0.9995 - val_loss: 1.7734 - val_accuracy: 0.9800
Epoch 83/100
63/63 [==============================] - 12s 175ms/step - loss: 0.0390 - accuracy: 0.9990 - val_loss: 1.9215 - val_accuracy: 0.9820
Epoch 84/100
63/63 [==============================] - 11s 173ms/step - loss: 0.0193 - accuracy: 0.9990 - val_loss: 1.7286 - val_accuracy: 0.9800
Epoch 85/100
63/63 [==============================] - 12s 196ms/step - loss: 0.0790 - accuracy: 0.9970 - val_loss: 1.7727 - val_accuracy: 0.9800
Epoch 86/100
63/63 [==============================] - 11s 171ms/step - loss: 0.1157 - accuracy: 0.9965 - val_loss: 2.8173 - val_accuracy: 0.9820
Epoch 87/100
63/63 [==============================] - 11s 166ms/step - loss: 0.0814 - accuracy: 0.9980 - val_loss: 1.7807 - val_accuracy: 0.9790
Epoch 88/100
63/63 [==============================] - 11s 166ms/step - loss: 0.0985 - accuracy: 0.9985 - val_loss: 1.7663 - val_accuracy: 0.9800
Epoch 89/100
63/63 [==============================] - 11s 173ms/step - loss: 0.0116 - accuracy: 0.9990 - val_loss: 1.8214 - val_accuracy: 0.9810
Epoch 90/100
63/63 [==============================] - 13s 196ms/step - loss: 0.0430 - accuracy: 0.9985 - val_loss: 1.7327 - val_accuracy: 0.9820
Epoch 91/100
```

Plot for loss and accuracy during training

```
accuracy_tuning = history_tuning.history["accuracy"]
val_tuning = history_tuning.history["val_accuracy"]
loss_tuning = history_tuning.history["loss"]
val_loss_tuning = history_tuning.history["val_loss"]
epochs = range(1, len(accuracy_tuning) + 1)

plt.plot(epochs, accuracy_tuning, "bo", label="Training accuracy")
plt.plot(epochs, val_tuning, "b", label="Validation accuracy")
plt.title("Fine-tuning: Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

plt.figure()
plt.plot(epochs, loss_tuning, "bo", label="Training loss")
plt.plot(epochs, val_loss_tuning, "b", label="Validation loss")
plt.title("Fine-tuning: Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Fine-tuning: Training and validation accuracy