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

```
!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 [00:07<00:00, 211MB/s]
100% 812M/812M [00:08<00:00, 106MB/s]
```

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

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

The code arranges a cat and dog pictures dataset into subdirectories for training, validation, and testing, copying a specified number of images in each subdirectory from the main folder.

```
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)
```

This code is a Keras model with a TensorFlow backend which creates the datasets based on the directories having images for training, validation, and testing. Each dataset has images resized to 180x180 pixels and batched into groups of 32.

```
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.
```

These piece of the code creates a dataset with 1000 samples consisting of 16 random normal distributions. Then, it iterates through the dataset to print the shape of each element. It stops printing the shape after three elements.

```
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,)
```

This code manages to batch a dataset that contains 32 samples a batch. Output. Then, it starts with printing the shape of the first three batches, after that stopping.

```
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)
```

This code includes a mapping to transform every element in the dataset into a 4x4 matrix and then move over the transformed dataset, which allows printing of the shape of the first three elements.

```
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)
```

The code here will loop through the batches of the training data along with its own labels for printing the shape of every batch of data and labels. Subsequently, it will break out of the loop.

```
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,)
```

This code initializes a CNN model with multiple convolution and pooling layers interconnected by dropout layer for the output layer followed by the sigmoid activation function.

```
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)
```

Here the code that establishes model for learning defines binary crossentropy as the loss function, Adam as the optimizer, and accuracy as the metric of model evaluation.

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

Double-click (or enter) to edit

Double-click (or enter) to edit

```
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 (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)
_________________________________________________________________
```

This code uses 50 epochs for training the neural network with the training dataset. It further validates the model on the validation dataset and keeps the best performing model which has the lowest validation loss using the ModelCheckpoint.

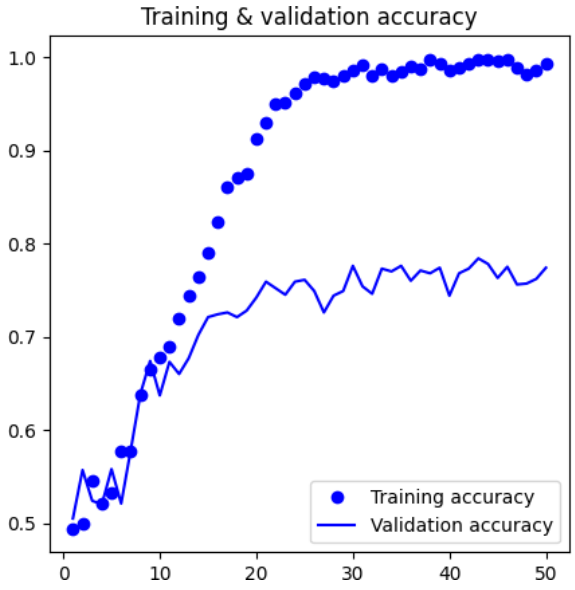
```
from keras.callbacks import ModelCheckpoint, EarlyStopping

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

```
Epoch 22/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.1269 - accuracy: 0.9490 - val_loss: 0.8488 - val_accuracy: 0
Epoch 23/50
63/63 [==============================]  - 6s 99ms/step - loss: 0.1141 - accuracy: 0.9505 - val_loss: 0.8340 - val_accuracy: 0
Epoch 24/50
63/63 [==============================]  - 4s 64ms/step - loss: 0.0968 - accuracy: 0.9615 - val_loss: 0.8928 - val_accuracy: 0
Epoch 25/50
63/63 [==============================]  - 5s 81ms/step - loss: 0.0946 - accuracy: 0.9710 - val_loss: 1.0549 - val_accuracy: 0
Epoch 26/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.0639 - accuracy: 0.9785 - val_loss: 1.1624 - val_accuracy: 0
Epoch 27/50
63/63 [==============================]  - 4s 55ms/step - loss: 0.0616 - accuracy: 0.9765 - val_loss: 1.2904 - val_accuracy: 0
Epoch 28/50
63/63 [==============================]  - 6s 85ms/step - loss: 0.0637 - accuracy: 0.9745 - val_loss: 1.1177 - val_accuracy: 0
Epoch 29/50
63/63 [==============================]  - 4s 56ms/step - loss: 0.0504 - accuracy: 0.9800 - val_loss: 1.2804 - val_accuracy: 0
Epoch 30/50
63/63 [==============================]  - 4s 58ms/step - loss: 0.0409 - accuracy: 0.9860 - val_loss: 1.2107 - val_accuracy: 0
Epoch 31/50
63/63 [==============================]  - 7s 109ms/step - loss: 0.0268 - accuracy: 0.9915 - val_loss: 1.2319 - val_accuracy:
Epoch 32/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.0633 - accuracy: 0.9795 - val_loss: 1.1915 - val_accuracy: 0
Epoch 33/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.0408 - accuracy: 0.9875 - val_loss: 1.2045 - val_accuracy: 0
Epoch 34/50
63/63 [==============================]  - 5s 84ms/step - loss: 0.0483 - accuracy: 0.9805 - val_loss: 1.3111 - val_accuracy: 0
Epoch 35/50
63/63 [==============================]  - 6s 81ms/step - loss: 0.0449 - accuracy: 0.9845 - val_loss: 1.1370 - val_accuracy: 0
Epoch 36/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.0322 - accuracy: 0.9905 - val_loss: 1.1224 - val_accuracy: 0
Epoch 37/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.0292 - accuracy: 0.9875 - val_loss: 1.2164 - val_accuracy: 0
Epoch 38/50
63/63 [==============================]  - 7s 106ms/step - loss: 0.0140 - accuracy: 0.9965 - val_loss: 1.4347 - val_accuracy:
Epoch 39/50
63/63 [==============================]  - 5s 79ms/step - loss: 0.0173 - accuracy: 0.9935 - val_loss: 1.3945 - val_accuracy: 0
Epoch 40/50
63/63 [==============================]  - 4s 63ms/step - loss: 0.0372 - accuracy: 0.9855 - val_loss: 1.4103 - val_accuracy: 0
Epoch 41/50
63/63 [==============================]  - 4s 58ms/step - loss: 0.0319 - accuracy: 0.9880 - val_loss: 1.3349 - val_accuracy: 0
Epoch 42/50
63/63 [==============================]  - 6s 92ms/step - loss: 0.0217 - accuracy: 0.9925 - val_loss: 1.2596 - val_accuracy: 0
Epoch 43/50
63/63 [==============================]  - 5s 74ms/step - loss: 0.0096 - accuracy: 0.9965 - val_loss: 1.4444 - val_accuracy: 0
Epoch 44/50
63/63 [==============================]  - 4s 56ms/step - loss: 0.0080 - accuracy: 0.9975 - val_loss: 1.5229 - val_accuracy: 0
Epoch 45/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.0170 - accuracy: 0.9955 - val_loss: 1.4761 - val_accuracy: 0
Epoch 46/50
63/63 [==============================]  - 7s 100ms/step - loss: 0.0159 - accuracy: 0.9965 - val_loss: 1.3899 - val_accuracy:
Epoch 47/50
63/63 [==============================]  - 4s 63ms/step - loss: 0.0324 - accuracy: 0.9880 - val_loss: 1.3977 - val_accuracy: 0
Epoch 48/50
63/63 [==============================]  - 4s 58ms/step - loss: 0.0584 - accuracy: 0.9810 - val_loss: 1.3903 - val_accuracy: 0
Epoch 49/50
63/63 [==============================]  - 7s 99ms/step - loss: 0.0372 - accuracy: 0.9860 - val_loss: 1.4081 - val_accuracy: 0
Epoch 50/50
63/63 [==============================]  - 4s 57ms/step - loss: 0.0162 - accuracy: 0.9925 - val_loss: 1.4816 - val_accuracy: 0
```

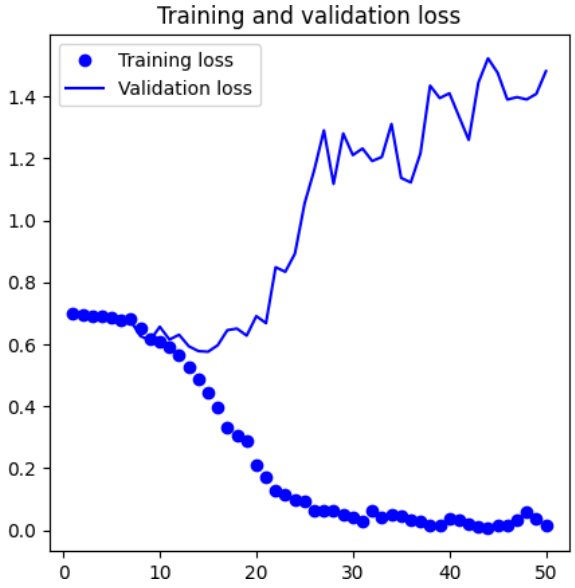
Double-click (or enter) to edit

Using the below code, the graph has been formed as shown in the figures below.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(5, 5))
accuracy = history.history["accuracy"]
val_accuracy = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training & validation accuracy")
plt.legend()
plt.figure()
plt.figure(figsize=(5, 5))
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
```




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



Double-click (or enter) to edit

In this code description, we load the saved model from the "convnet_from_scratch.keras" file, evaluate its performance on the test dataset, and display the test accuracy rounded to three decimal places.

```
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: 0.6068 - accuracy: 0.7200
     Test accuracy: 0.720
```

In this code, the loaded model file name is "convnet_from_scratch.keras" and the test accuracy is printed in the three decimals.

```
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)
```

This objective provides a sequential data augmentation pipeline for image data starting with random horizontal flipping, random rotation less than 0.1 radians, and random zooming up to 0.2 times the original image.

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

Those codes allow visualization of augmented images by using data augmentation techniques such as flipping, rotation, and zooming to the batch of training images and then displaying nine augmented images on a grid that lack axis labels.

```
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")
```

This code generates augmented images and displaying them on a grid without axis labels which is achieved through applying the data augmentation techniques like random flipping, rotation and zoom on a set of images from the training dataset.

```
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=75,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
63/63 [==============================]  - 5s 79ms/step - loss: 0.3901 - accuracy: 0.8190 - val_loss: 0.4790 - val_accuracy:
Epoch 51/75
63/63 [==============================]  - 5s 84ms/step - loss: 0.3706 - accuracy: 0.8425 - val_loss: 0.5361 - val_accuracy: 0
Epoch 52/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.3904 - accuracy: 0.8250 - val_loss: 0.4989 - val_accuracy: 0
Epoch 53/75
63/63 [==============================]  - 4s 65ms/step - loss: 0.3546 - accuracy: 0.8475 - val_loss: 0.4504 - val_accuracy: 0
Epoch 54/75
63/63 [==============================]  - 6s 93ms/step - loss: 0.3670 - accuracy: 0.8370 - val_loss: 0.4771 - val_accuracy: 0
Epoch 55/75
63/63 [==============================]  - 4s 61ms/step - loss: 0.3460 - accuracy: 0.8495 - val_loss: 0.4860 - val_accuracy: 0
Epoch 56/75
63/63 [==============================]  - 4s 62ms/step - loss: 0.3492 - accuracy: 0.8480 - val_loss: 0.4461 - val_accuracy: 0
Epoch 57/75
63/63 [==============================]  - 7s 111ms/step - loss: 0.3405 - accuracy: 0.8565 - val_loss: 0.4912 - val_accuracy:
Epoch 58/75
63/63 [==============================]  - 4s 61ms/step - loss: 0.3366 - accuracy: 0.8645 - val_loss: 0.4228 - val_accuracy: 0
Epoch 59/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.3282 - accuracy: 0.8580 - val_loss: 0.4956 - val_accuracy: 0
Epoch 60/75
63/63 [==============================]  - 6s 96ms/step - loss: 0.3134 - accuracy: 0.8675 - val_loss: 0.4750 - val_accuracy: 0
Epoch 61/75
63/63 [==============================]  - 5s 73ms/step - loss: 0.3171 - accuracy: 0.8670 - val_loss: 0.5087 - val_accuracy: 0
Epoch 62/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.3140 - accuracy: 0.8700 - val_loss: 0.4556 - val_accuracy: 0
Epoch 63/75
63/63 [==============================]  - 4s 61ms/step - loss: 0.3157 - accuracy: 0.8630 - val_loss: 0.5335 - val_accuracy: 0
Epoch 64/75
63/63 [==============================]  - 6s 90ms/step - loss: 0.3193 - accuracy: 0.8635 - val_loss: 0.4802 - val_accuracy: 0
Epoch 65/75
63/63 [==============================]  - 4s 58ms/step - loss: 0.2785 - accuracy: 0.8845 - val_loss: 0.5349 - val_accuracy: 0
Epoch 66/75
63/63 [==============================]  - 4s 61ms/step - loss: 0.3122 - accuracy: 0.8590 - val_loss: 0.5007 - val_accuracy: 0
Epoch 67/75
63/63 [==============================]  - 6s 92ms/step - loss: 0.2828 - accuracy: 0.8805 - val_loss: 0.5511 - val_accuracy: 0
Epoch 68/75
63/63 [==============================]  - 5s 70ms/step - loss: 0.2629 - accuracy: 0.8905 - val_loss: 0.6774 - val_accuracy: 0
Epoch 69/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.3010 - accuracy: 0.8700 - val_loss: 0.5184 - val_accuracy: 0
Epoch 70/75
63/63 [==============================]  - 5s 83ms/step - loss: 0.2711 - accuracy: 0.8900 - val_loss: 0.5954 - val_accuracy: 0
Epoch 71/75
63/63 [==============================]  - 4s 57ms/step - loss: 0.2838 - accuracy: 0.8735 - val_loss: 0.6481 - val_accuracy: 0
Epoch 72/75
63/63 [==============================]  - 5s 83ms/step - loss: 0.2710 - accuracy: 0.8820 - val_loss: 0.4916 - val_accuracy: 0
Epoch 73/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.2557 - accuracy: 0.8910 - val_loss: 0.4920 - val_accuracy: 0
Epoch 74/75
63/63 [==============================]  - 4s 62ms/step - loss: 0.2416 - accuracy: 0.9005 - val_loss: 0.5181 - val_accuracy: 0
Epoch 75/75
63/63 [==============================]  - 7s 100ms/step - loss: 0.2619 - accuracy: 0.8860 - val_loss: 0.4687 - val_accuracy:
```

This code plotted the training & validation accuracy within one graph and training & validation loss within another graph, thereby enabling visualization of the model's performance during training.

Such a code fetches the trained model, that got the highest validation performance, evaluates its accuracy on test set and then prints it rounded to three decimal places.

```
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 28ms/step - loss: 0.4959 - accuracy: 0.7920
Test accuracy: 0.792
```

The block of code, here, partitions the dataset in nutshell with the training set made up of 2000 samples and validation and test sets having 500 samples each.

```
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)
```

With data augmentation, multiple convolutional and max-pooling layers, dropout regularization, and a sigmoid output layer, this code describes a convolutional neural network (CNN) model architecture for image classification. The model is then compiled for training using binary crossentropy loss, the Adam optimizer, and an accuracy metric.

```
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"])
```

With validation, this code trains the specified convolutional neural network model over 75 epochs on the training dataset. It then uses the ModelCheckpoint callback to save the best-performing model based on validation loss.

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

```
63/63 [==============================]  - 4s 61ms/step - loss: 0.2634 - accuracy: 0.8880 - val_loss: 0.4910 - val_accuracy:
Epoch 56/75
63/63 [==============================]  - 6s 97ms/step - loss: 0.2651 - accuracy: 0.8850 - val_loss: 0.5270 - val_accuracy: 0
Epoch 57/75
63/63 [==============================]  - 5s 69ms/step - loss: 0.2410 - accuracy: 0.8945 - val_loss: 0.4651 - val_accuracy: 0
Epoch 58/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.2099 - accuracy: 0.9135 - val_loss: 0.5512 - val_accuracy: 0
Epoch 59/75
63/63 [==============================]  - 6s 85ms/step - loss: 0.2566 - accuracy: 0.8935 - val_loss: 0.5811 - val_accuracy: 0
Epoch 60/75
63/63 [==============================]  - 6s 87ms/step - loss: 0.2336 - accuracy: 0.9010 - val_loss: 0.6210 - val_accuracy: 0
Epoch 61/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.2661 - accuracy: 0.8835 - val_loss: 0.5825 - val_accuracy: 0
Epoch 62/75
63/63 [==============================]  - 5s 84ms/step - loss: 0.2379 - accuracy: 0.9035 - val_loss: 0.5537 - val_accuracy: 0
Epoch 63/75
63/63 [==============================]  - 6s 88ms/step - loss: 0.2387 - accuracy: 0.9065 - val_loss: 0.4978 - val_accuracy: 0
Epoch 64/75
63/63 [==============================]  - 4s 58ms/step - loss: 0.2109 - accuracy: 0.9075 - val_loss: 0.6036 - val_accuracy: 0
Epoch 65/75
63/63 [==============================]  - 4s 64ms/step - loss: 0.2224 - accuracy: 0.9065 - val_loss: 0.5249 - val_accuracy: 0
Epoch 66/75
63/63 [==============================]  - 6s 90ms/step - loss: 0.1963 - accuracy: 0.9160 - val_loss: 0.5233 - val_accuracy: 0
Epoch 67/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.2048 - accuracy: 0.9200 - val_loss: 0.5011 - val_accuracy: 0
Epoch 68/75
63/63 [==============================]  - 4s 60ms/step - loss: 0.2138 - accuracy: 0.9130 - val_loss: 0.6505 - val_accuracy: 0
Epoch 69/75
63/63 [==============================]  - 6s 94ms/step - loss: 0.2057 - accuracy: 0.9150 - val_loss: 0.5584 - val_accuracy: 0
Epoch 70/75
63/63 [==============================]  - 5s 71ms/step - loss: 0.1889 - accuracy: 0.9240 - val_loss: 0.5633 - val_accuracy: 0
Epoch 71/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.1887 - accuracy: 0.9255 - val_loss: 0.5963 - val_accuracy: 0
Epoch 72/75
63/63 [==============================]  - 5s 69ms/step - loss: 0.1723 - accuracy: 0.9320 - val_loss: 0.5771 - val_accuracy: 0
Epoch 73/75
63/63 [==============================]  - 6s 90ms/step - loss: 0.1868 - accuracy: 0.9275 - val_loss: 0.5246 - val_accuracy: 0
Epoch 74/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.1860 - accuracy: 0.9195 - val_loss: 0.6375 - val_accuracy: 0
Epoch 75/75
63/63 [==============================]  - 4s 59ms/step - loss: 0.1988 - accuracy: 0.9140 - val loss: 0.5920 - val accuracy: 0
```

To help evaluate the model's performance during training, this code displays the training and validation loss over epochs in one graph and the training and validation accuracy over epochs in another graph.

```
test_model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation1.keras")
test_loss, test_acc = test_model.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 [==============================] - 1s 30ms/step - loss: 0.4677 - accuracy: 0.8120
Test accuracy: 0.812
```

In the first phase, 1,000 examples were used to train a basic convolutional neural network (convnet), which resulted in a classification accuracy of roughly 70%. It was discovered that overfitting was the primary reason for worry. Afterwards, optimisation methods including dropout, early stopping, regularisation, and data augmentation were applied to enhance the model. The model's accuracy increased to 80% with the use of data augmentation, indicating a significant improvement in performance. The goal was to find the best training samples so that, following this improvement, classification accuracy could be further improved. Various tactics were explored to counteract overfitting, such as altering the
training set and incorporating additional optimization tools. These included decreasing the network's capacity, incorporating dropout, adding weight regularization, and increasing the size of the training sample.

Q4. Proceed with Steps 1-3, utilizing a pretrained network this time. The pretrained network sample sizes you use in Steps 2 and 3 may or may not match those of the network you trained from scratch. Once more, utilize every optimization strategy available to achieve peak performance.

Feature extraction with a pretrained model

Instantiating the VGG16 convolutional base

With the exception of its fully connected layers, this code loads the pre-trained VGG16 convolutional base on ImageNet and sets it up to accept 180x180 input images with three color channels.

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_ke
58889256/58889256 [==============================] - 0s 0us/step
```

```
convolution_base.summary()
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

 block4_pool (MaxPooling2D)  (None, 11, 11, 512)       0

 block5_conv1 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_conv2 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_conv3 (Conv2D)       (None, 11, 11, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 5, 5, 512)         0

=================================================================
Total params: 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 Extracting the

VGG16 features and corresponding labels

Using a pre-trained VGG16 model, this code preprocesses the images, runs them through the model to extract features, and gathers the features and associated labels for training, validation, and test sets.

```
import numpy as np

def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = convolution_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

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

```
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 28ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 28ms/step
 1/1  [==============================]  - 0s 22ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 36ms/step
 1/1  [==============================]  - 0s 45ms/step
 1/1  [==============================]  - 0s 38ms/step
 1/1  [==============================]  - 0s 31ms/step
 1/1  [==============================]  - 0s 31ms/step
 1/1  [==============================]  - 0s 43ms/step
 1/1  [==============================]  - 0s 35ms/step
 1/1  [==============================]  - 0s 33ms/step
 1/1  [==============================]  - 0s 36ms/step
 1/1  [==============================]  - 0s 38ms/step
 1/1  [==============================]  - 0s 34ms/step
 1/1  [==============================]  - 0s 32ms/step
 1/1  [==============================]  - 0s 34ms/step
 1/1  [==============================]  - 0s 37ms/step
 1/1  [==============================]  - 0s 38ms/step
 1/1  [==============================]  - 0s 37ms/step
 1/1  [==============================]  - 0s 39ms/step
 1/1  [==============================]  - 0s 38ms/step
 1/1  [==============================]  - 2s 2s/step
 1/1  [==============================]  - 0s 40ms/step
 1/1  [==============================]  - 0s 25ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 28ms/step
 1/1  [==============================]  - 0s 22ms/step
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 27ms/step
 1/1  [==============================]  - 0s 29ms/step
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 27ms/step
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 30ms/step
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 26ms/step
 1/1  [==============================]  - 0s 30ms/step
 1/1  [==============================]  - 0s 26ms/step
 1/1  [==============================]  - 0s 22ms/step
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 29ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 22ms/step
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 26ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 22ms/step
 1/1  [==============================]  - 0s 25ms/step
 1/1  [==============================]  - 0s 22ms/step
 1/1  [==============================]  - 0s 25ms/step
 1/1  [==============================]  - 0s 24ms/step
 1/1  [==============================]  - 0s 23ms/step
 1/1  [==============================]  - 0s 18ms/step
```

```
train_features.shape
```

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

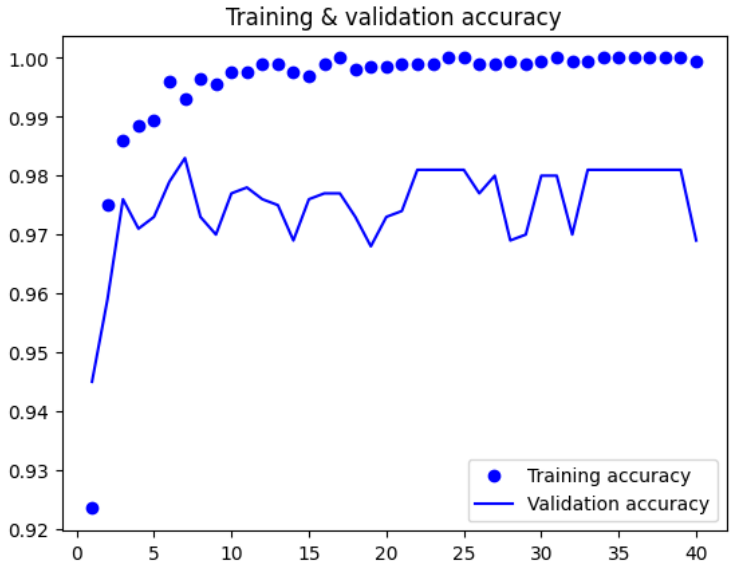
This code creates a neural network model to perform binary classification on extracted features. It then trains the model for 40 epochs using the RMSprop optimizer and binary crossentropy loss, and uses ModelCheckpoint to save the best model based on validation loss.

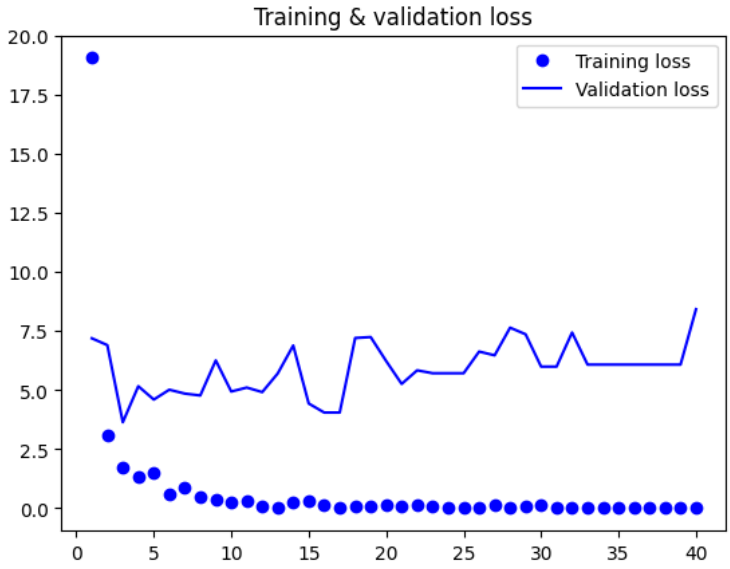
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              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 12ms/step - loss: 19.0687 - accuracy: 0.9235 - val_loss: 7.1989 - val_accuracy:
Epoch 2/40
63/63 [==============================]  - 0s 7ms/step - loss: 3.1118 - accuracy: 0.9750 - val_loss: 6.9091 - val_accuracy: 0.
Epoch 3/40
63/63 [==============================]  - 0s 8ms/step - loss: 1.7449 - accuracy: 0.9860 - val_loss: 3.6428 - val_accuracy: 0.
Epoch 4/40
63/63 [==============================]  - 0s 6ms/step - loss: 1.3205 - accuracy: 0.9885 - val_loss: 5.1684 - val_accuracy: 0.
Epoch 5/40
63/63 [==============================]  - 0s 6ms/step - loss: 1.5003 - accuracy: 0.9895 - val_loss: 4.6042 - val_accuracy: 0.
Epoch 6/40
63/63 [==============================]  - 0s 5ms/step - loss: 0.6215 - accuracy: 0.9960 - val_loss: 5.0204 - val_accuracy: 0.
Epoch 7/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.8572 - accuracy: 0.9930 - val_loss: 4.8568 - val_accuracy: 0.
Epoch 8/40
63/63 [==============================]  - 0s 5ms/step - loss: 0.4568 - accuracy: 0.9965 - val_loss: 4.7765 - val_accuracy: 0.
Epoch 9/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.3636 - accuracy: 0.9955 - val_loss: 6.2617 - val_accuracy: 0.
Epoch 10/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.2640 - accuracy: 0.9975 - val_loss: 4.9441 - val_accuracy: 0.
Epoch 11/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.3348 - accuracy: 0.9975 - val_loss: 5.1121 - val_accuracy: 0.
Epoch 12/40
63/63 [==============================]  - 0s 5ms/step - loss: 0.0816 - accuracy: 0.9990 - val_loss: 4.9193 - val_accuracy: 0.
Epoch 13/40
63/63 [==============================]  - 0s 5ms/step - loss: 0.0161 - accuracy: 0.9990 - val_loss: 5.7095 - val_accuracy: 0.
Epoch 14/40
63/63 [==============================]  - 0s 5ms/step - loss: 0.2320 - accuracy: 0.9975 - val_loss: 6.8929 - val_accuracy: 0.
Epoch 15/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.3165 - accuracy: 0.9970 - val_loss: 4.4371 - val_accuracy: 0.
Epoch 16/40
63/63 [==============================]  - 0s 5ms/step - loss: 0.1208 - accuracy: 0.9990 - val_loss: 4.0539 - val_accuracy: 0.
Epoch 17/40
63/63 [==============================]  - 0s 6ms/step - loss: 1.8794e-31 - accuracy: 1.0000 - val_loss: 4.0539 - val_accuracy
Epoch 18/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.0880 - accuracy: 0.9980 - val_loss: 7.2110 - val_accuracy: 0.
Epoch 19/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.0757 - accuracy: 0.9985 - val_loss: 7.2506 - val_accuracy: 0.
Epoch 20/40
63/63 [==============================]  - 0s 5ms/step - loss: 0.1674 - accuracy: 0.9985 - val_loss: 6.2344 - val_accuracy: 0.
Epoch 21/40
63/63 [==============================]  - 0s 6ms/step - loss: 0.0889 - accuracy: 0.9990 - val_loss: 5.2662 - val_accuracy: 0.
Epoch 22/40
63/63 [==============================]  - 0s 8ms/step - loss: 0.1204 - accuracy: 0.9990 - val_loss: 5.8422 - val_accuracy: 0.
Epoch 23/40
63/63 [==============================]  - 0s 7ms/step - loss: 0.1061 - accuracy: 0.9990 - val_loss: 5.7193 - val_accuracy: 0.
Epoch 24/40
63/63 [==============================]  - 0s 7ms/step - loss: 3.0967e-34 - accuracy: 1.0000 - val_loss: 5.7193 - val_accuracy
Epoch 25/40
63/63 [==============================]  - 0s 8ms/step - loss: 1.3559e-24 - accuracy: 1.0000 - val_loss: 5.7193 - val_accuracy
Epoch 26/40
63/63 [==============================]  - 1s 8ms/step - loss: 0.0298 - accuracy: 0.9990 - val_loss: 6.6370 - val_accuracy: 0.
Epoch 27/40
63/63 [==============================]  - 0s 8ms/step - loss: 0.1504 - accuracy: 0.9990 - val_loss: 6.4768 - val_accuracy: 0.
Epoch 28/40
63/63 [==============================]  - 1s 8ms/step - loss: 0.0196 - accuracy: 0.9995 - val_loss: 7.6487 - val_accuracy: 0.
Epoch 29/40
63/63 [==============================]  - 1s 8ms/step - loss: 0.0647 - accuracy: 0.9990 - val_loss: 7.3683 - val_accuracy: 0.
```

This code provides a visualization of the model's performance during training by plotting the training and validation loss over epochs in one graph and the training and validation accuracy over epochs in another.

```
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()
```






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

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

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

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

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

This code provides a visualization of the model's performance during training by plotting the training and validation loss over epochs in one graph and the training and validation accuracy over epochs in another.

```
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"])
```

Using ModelCheckpoint to save the top-performing model based on validation loss, this code trains the specified neural network model with data augmentation over 75 epochs on the training dataset.

```
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 [==============================]  - 13s 184ms/step  - loss: 24.1926 - accuracy: 0.8950 - val_loss: 10.3057 - val_accurac
Epoch 2/75
63/63 [==============================]  - 12s 189ms/step  - loss: 8.0849 - accuracy: 0.9455 - val_loss: 4.3758 - val_accuracy:
Epoch 3/75
63/63 [==============================]  - 10s 148ms/step  - loss: 6.8959 - accuracy: 0.9505 - val_loss: 4.0495 - val_accuracy:
Epoch 4/75
63/63 [==============================]  - 10s 153ms/step - loss: 6.6901 - accuracy: 0.9560 - val_loss: 3.1329 - val_accuracy:
Epoch 5/75
63/63 [==============================]  - 10s 150ms/step - loss: 4.7576 - accuracy: 0.9645 - val_loss: 3.0542 - val_accuracy:
Epoch 6/75
63/63 [==============================]  - 10s 145ms/step  - loss: 4.5847 - accuracy: 0.9685 - val_loss: 10.0774 - val_accuracy
Epoch 7/75
63/63 [==============================]  - 11s 174ms/step - loss: 2.9833 - accuracy: 0.9725 - val_loss: 4.0811 - val_accuracy:
Epoch 8/75
63/63 [==============================]  - 10s 147ms/step - loss: 3.2815 - accuracy: 0.9760 - val_loss: 3.2017 - val_accuracy:
Epoch 9/75
63/63 [==============================]  - 9s 140ms/step - loss: 3.6867 - accuracy: 0.9665 - val_loss: 3.2883 - val_accuracy:
Epoch 10/75
63/63 [==============================]  - 10s 147ms/step - loss: 2.6142 - accuracy: 0.9730 - val_loss: 8.6223 - val_accuracy:
Epoch 11/75
63/63 [==============================]  - 9s 143ms/step - loss: 2.8282 - accuracy: 0.9745 - val_loss: 3.7969 - val_accuracy:
Epoch 12/75
63/63 [==============================]  - 10s 145ms/step - loss: 2.6029 - accuracy: 0.9780 - val_loss: 4.5574 - val_accuracy:
```

```
Epoch 13/75
63/63 [==============================] - 11s 176ms/step - loss: 1.6483 - accuracy: 0.9815 - val_loss: 3.2872 - val_accuracy:
Epoch 14/75
63/63 [==============================] - 10s 146ms/step - loss: 1.9050 - accuracy: 0.9805 - val_loss: 3.1579 - val_accuracy:
Epoch 15/75
63/63 [==============================]  - 9s 143ms/step - loss: 2.1316 - accuracy: 0.9770 - val_loss: 3.9085 - val_accuracy:
Epoch 16/75
63/63 [==============================]  - 10s 146ms/step - loss: 2.2381 - accuracy: 0.9730 - val_loss: 3.2030 - val_accuracy:
Epoch 17/75
63/63 [==============================]  - 12s 184ms/step - loss: 1.1010 - accuracy: 0.9845 - val_loss: 2.8999 - val_accuracy:
Epoch 18/75
63/63 [==============================]  - 11s 176ms/step  - loss: 1.5024 - accuracy: 0.9825 - val_loss: 3.8265 - val_accuracy:
Epoch 19/75
63/63 [==============================]  - 10s 149ms/step - loss: 1.9317 - accuracy: 0.9775 - val_loss: 3.3848 - val_accuracy:
Epoch 20/75
63/63 [==============================]  - 10s 150ms/step - loss: 1.3992 - accuracy: 0.9830 - val_loss: 2.0959 - val_accuracy:
Epoch 21/75
63/63 [==============================]  - 10s 151ms/step  - loss: 1.4552 - accuracy: 0.9815 - val_loss: 3.6308 - val_accuracy:
Epoch 22/75
63/63 [==============================]  - 9s 144ms/step - loss: 1.0946 - accuracy: 0.9825 - val_loss: 2.4078 - val_accuracy:
Epoch 23/75
63/63 [==============================]  - 10s 149ms/step - loss: 1.1851 - accuracy: 0.9840 - val_loss: 2.9068 - val_accuracy:
Epoch 24/75
63/63 [==============================]  - 10s 149ms/step - loss: 1.4632 - accuracy: 0.9805 - val_loss: 2.2917 - val_accuracy:
Epoch 25/75
63/63 [==============================]  - 10s 146ms/step  - loss: 0.9851 - accuracy: 0.9855 - val_loss: 2.5992 - val_accuracy:
Epoch 26/75
63/63 [==============================]  - 11s 178ms/step - loss: 1.1330 - accuracy: 0.9850 - val_loss: 2.9360 - val_accuracy:
Epoch 27/75
63/63 [==============================]  - 9s 144ms/step - loss: 1.1688 - accuracy: 0.9860 - val_loss: 2.7156 - val_accuracy:
Epoch 28/75
63/63 [==============================]  - 10s 151ms/step  - loss: 0.6923 - accuracy: 0.9875 - val_loss: 2.2375 - val_accuracy:
Epoch 29/75
63/63 [==============================]  - 9s 145ms/step - loss: 1.1098 - accuracy: 0.9855 - val_loss: 2.8026 - val_accuracy:
```

```
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 94ms/step - loss: 1.7070 - accuracy: 0.9810
Test accuracy: 0.981
```

A pretrained VGG16 model with Fine-tuning

Fine-tuning involves training the newly added element of the model (the fully connected classifier in this case) and these top layers simultaneously, as well as unfreezing a few top layers of a frozen model foundation used for feature extraction. Because it adjusts the more abstract representations of the model that are being reused to make them more pertinent to the current task, this process is known as fine- tuning.

```
convolution_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, None, None, 3)]   0

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

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

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

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

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

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

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

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

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

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

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

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

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

 block4_pool (MaxPooling2D)   (None, None, None, 512)   0

 block5_conv1 (Conv2D)        (None, None, None, 512)   2359808

 block5_conv2 (Conv2D)        (None, None, None, 512)   2359808

 block5_conv3 (Conv2D)        (None, None, None, 512)   2359808

 block5_pool (MaxPooling2D)   (None, None, None, 512)   0

=================================================================
Total params: 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
```

Fine-tuning the model

This code trains the model for 50 epochs on the training dataset with validation, storing the best model based on validation loss. It then compiles the model using binary crossentropy loss and RMSprop optimizer with a specified learning rate.

```
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)
```

```
63/63 [==============================]  - 13s 201ms/step - loss:  0.0856 - accuracy: 0.9955 - val_loss: 1.2975 - val_accuracy
Epoch 39/50
63/63 [==============================]  - 12s 191ms/step - loss:  0.0743 - accuracy: 0.9980 - val_loss: 2.1396 - val_accuracy:
Epoch 40/50
63/63 [==============================]  - 11s 166ms/step - loss: 0.0792 - accuracy: 0.9975 - val_loss: 1.8178 - val_accuracy:
Epoch 41/50
63/63 [==============================]  - 11s 161ms/step - loss: 0.0299 - accuracy: 0.9985 - val_loss: 1.6836 - val_accuracy:
Epoch 42/50
63/63 [==============================]  - 11s 164ms/step - loss:  0.1061 - accuracy: 0.9960 - val_loss: 1.9956 - val_accuracy:
Epoch 43/50
63/63 [==============================]  - 12s 190ms/step - loss: 1.0127e-04 - accuracy: 1.0000 - val_loss: 1.8244 - val_accur
Epoch 44/50
63/63 [==============================]  - 11s 162ms/step - loss: 0.0706 - accuracy: 0.9965 - val_loss: 1.5397 - val_accuracy:
Epoch 45/50
63/63 [==============================]  - 11s 168ms/step - loss:  0.0177 - accuracy: 0.9985 - val_loss: 1.6017 - val_accuracy:
Epoch 46/50
63/63 [==============================]  - 12s 191ms/step - loss:  0.0404 - accuracy: 0.9980 - val_loss: 1.9093 - val_accuracy:
Epoch 47/50
63/63 [==============================]  - 11s 168ms/step - loss: 0.0260 - accuracy: 0.9985 - val_loss: 1.6076 - val_accuracy:
Epoch 48/50
63/63 [==============================]  - 12s 190ms/step - loss: 0.1287 - accuracy: 0.9960 - val_loss: 1.5361 - val_accuracy:
Epoch 49/50
63/63 [==============================]  - 11s 166ms/step - loss:  0.0085 - accuracy: 0.9990 - val_loss: 1.8375 - val_accuracy:
Epoch 50/50
63/63 [==============================]  - 12s 190ms/step - loss:  0.0304 - accuracy: 0.9975 - val_loss: 1.5489 - val_accuracy:
```

This code provides a visual representation of the model's performance during training by plotting the training and validation loss over epochs in one graph and the training and validation accuracy over epochs in another graph.

```
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()
```