Cats/Dogs classification using a reduced training dataset

Nizar ZEROUALE

Application 1:

Consider the Dogs vs. Cats dataset, complete the CatsAndDogsClassification.py script (using the dedicated image machine learning platforms (i.e., Tensorsflow with Keras API)) able to recognize the category for an unknown image applied as input. There are 2 classes to predict. The system performance needs to be evaluated using the classification accuracy.

CODE:

```
import tensorflow
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import
VGG16,ResNet50,InceptionV3,Xception,MobileNet
from tensorflow.keras.optimizers import RMSprop
from matplotlib import pyplot
from keras.applications.vgg16 import VGG16
import os
import shutil

#visualization function + saving plot
def visualizeTheTrainingPerformances(history, model_name):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)
    pyplot.title('Training and validation accuracy')
    pyplot.plot(epochs, acc, 'bo', label = 'Training accuracy')
    pyplot.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
    pyplot.legend()
    pyplot.savefig('Accuracy '+model_name+'.png')
    pyplot.figure()
    pyplot.title('Training and validation loss')
    pyplot.plot(epochs, loss, 'bo', label = 'Training loss')
    pyplot.plot(epochs, val_loss, 'b', label = 'Validation loss')
    pyplot.legend
    pyplot.savefig('Loss '+model_name+'.png')

    pyplot.show()

    return



def prepareDatabase(original_directory, base_directory):
```

```
    #If the folder already exist remove everything
    if os.path.exists(base_directory):
        shutil.rmtree(base_directory)

    #Recreate the basefolder
    os.mkdir(base_directory)

    #TODO - Application 1 - Step 1a - Create the training folder in
the base directory
    train_directory = os.path.join(base_directory, 'train')
    os.mkdir(train_directory)

    #TODO - Application 1 - Step 1b - Create the validation folder
in the base directory
    validation_directory = os.path.join(base_directory,
'validation')
    os.mkdir(validation_directory)

    #TODO - Application 1 - Step 1c - Create the test folder in the
base directory
    test_directory = os.path.join(base_directory, 'test')
    os.mkdir(test_directory)

    #TODO - Application 1 - Step 1d - Create the cat/dog
training/validation/testing directories
    # create the train_cats_directory
    train_cats_directory = os.path.join(train_directory, 'cats')
    os.mkdir(train_cats_directory)

    # create the train_dogs_directory
    train_dogs_directory = os.path.join(train_directory, 'dogs')
    os.mkdir(train_dogs_directory)

    # create the validation_cats_directory
    validation_cats_directory = os.path.join(validation_directory,
'cats')
    os.mkdir(validation_cats_directory)

    # create the validation_dogs_directory
    validation_dogs_directory = os.path.join(validation_directory,
'dogs')
    os.mkdir(validation_dogs_directory)

    # create the test_cats_directory
    test_cats_directory = os.path.join(test_directory, 'cats')
    os.mkdir(test_cats_directory)

    # create the test_dogs_directory
    test_dogs_directory = os.path.join(test_directory, 'dogs')
    os.mkdir(test_dogs_directory)

    #TODO - Application 1 - Step 1e - Copy the first 1000 cat images
into the training directory (train_cats_directory)
    original_directory_cats = str(original_directory + '/cats/')
    fnames = ['{}.jpg'.format(i) for i in range(1000)]
    for fname in fnames:
        src = os.path.join(original_directory_cats, fname)
        dst = os.path.join(train_cats_directory, fname)
        shutil.copyfile(src, dst)
```

```
    #TODO - Application 1 - Step 1f - Copy the next 500 cat images
into the validation directory (validation_cats_directory)
    original_directory_cats = str(original_directory + '/cats/')
    fnames = ['{}.jpg'.format(i) for i in range(1000, 1500)]
    for fname in fnames:
        src = os.path.join(original_directory_cats, fname)
        dst = os.path.join(validation_cats_directory, fname)
        shutil.copyfile(src, dst)

    #TODO - Application 1 - Step 1g  - Copy the next 500 cat images
in to the test directory (test_cats_directory)
    original_directory_cats = str(original_directory + '/cats/')
    fnames = ['{}.jpg'.format(i) for i in range(1500, 2000)]
    for fname in fnames:
        src = os.path.join(original_directory_cats, fname)
        dst = os.path.join(test_cats_directory, fname)
        shutil.copyfile(src, dst)

    # TODO - Application 1 - Step 1h - Copy the first 1000 dogs
images into the training directory (train_dogs_directory)
    original_directory_dogs = str(original_directory + '/dogs/')
    fnames = ['{}.jpg'.format(i) for i in range(1000)]
    for fname in fnames:
        src = os.path.join(original_directory_dogs, fname)
        dst = os.path.join(train_dogs_directory, fname)
        shutil.copyfile(src, dst)

    # TODO - Application 1 - Step 1i - Copy the next 500 dogs images
into the validation directory (validation_dogs_directory)
    original_directory_dogs = str(original_directory + '/dogs/')
    fnames = ['{}.jpg'.format(i) for i in range(1000, 1500)]
    for fname in fnames:
        src = os.path.join(original_directory_dogs, fname)
        dst = os.path.join(validation_dogs_directory, fname)
        shutil.copyfile(src, dst)

    # TODO - Application 1 - Step 1j  - Copy the next 500 dogs
images in to the test directory (test_dogs_directory)
    original_directory_dogs = str(original_directory + '/dogs/')
    fnames = ['{}.jpg'.format(i) for i in range(1500, 2000)]
    for fname in fnames:
        src = os.path.join(original_directory_dogs, fname)
        dst = os.path.join(test_dogs_directory, fname)
        shutil.copyfile(src, dst)

    #TODO - Application 1 - Step 1k - As a sanitary check verify how
many pictures are in each directory
    print('Total number of CATS used for training =
{}'.format(len(os.listdir(train_cats_directory))))
    print('Total number of CATS used for validation =
{}'.format(len(os.listdir(validation_cats_directory))))
    print('Total number of CATS used for testing =
{}'.format(len(os.listdir(test_cats_directory))))
    print('Total number of DOGS used for training =
{}'.format(len(os.listdir(train_dogs_directory))))
    print('Total number of DOGS used for validation =
{}'.format(len(os.listdir(validation_dogs_directory))))
    print('Total number of DOGS used for testing =
{}'.format(len(os.listdir(test_dogs_directory))))
```

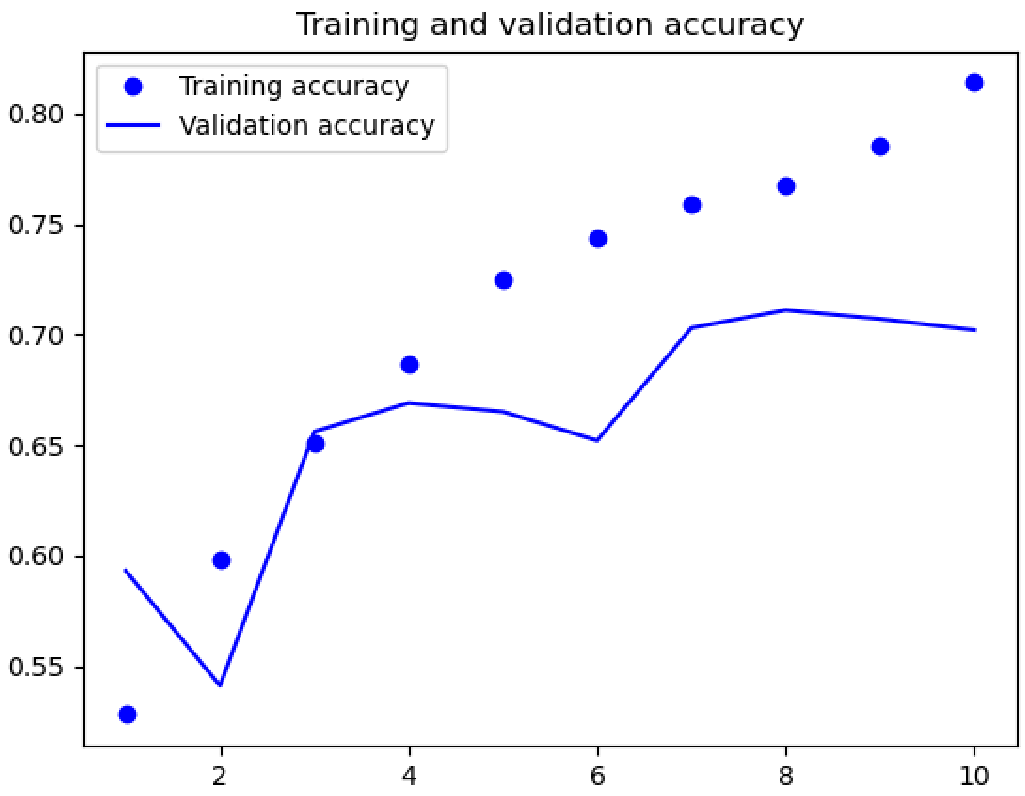
```
    return




def defineCNNModelFromScratch():

    #Application 1 - Step 3a - Initialize the sequential model
    model = models.Sequential()

    #TODO - Application 1 - Step 3b - Create the first hidden layer
as a convolutional layer
    model.add(layers.Conv2D(filters=32, kernel_size=(3, 3),
input_shape=(150, 150, 3), activation='relu'))

    #TODO - Application 1 - Step 3c - Define a maxpooling layer
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    #TODO - Application 1 - Step 3d - Create the third hidden layer
as a convolutional layer
    model.add(layers.Conv2D(filters=64, kernel_size=(3, 3),
activation='relu'))

    #TODO - Application 1 - Step 3e - Define a pooling layer
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    #TODO - Application 1 - Step 3f - Create another convolutional
layer
    model.add(layers.Conv2D(filters=128, kernel_size=(3, 3),
activation='relu'))

    #TODO - Application 1 - Step 3g - Define a pooling layer
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    #TODO - Application 1 - Step 3h - Create another convolutional
layer
    model.add(layers.Conv2D(filters=128, kernel_size=(3, 3),
activation='relu'))

    #TODO - Application 1 - Step 3i - Define a pooling layer
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    #TODO - Application 1 - Step 3j - Define the flatten layer
    model.add(layers.Flatten())
    #model.add(layers.Dropout(rate=0.5))

    #TODO - Application 1 - Step 3k - Define a dense layer of size
512
    model.add(layers.Dense(512, activation='relu'))


    #TODO - Application 1 - Step 3l - Define the output layer
    model.add(layers.Dense(1, activation='sigmoid'))

    #TODO - Application 1 - Step 3m - Visualize the network
arhitecture (list of layers)
    model.summary()
```

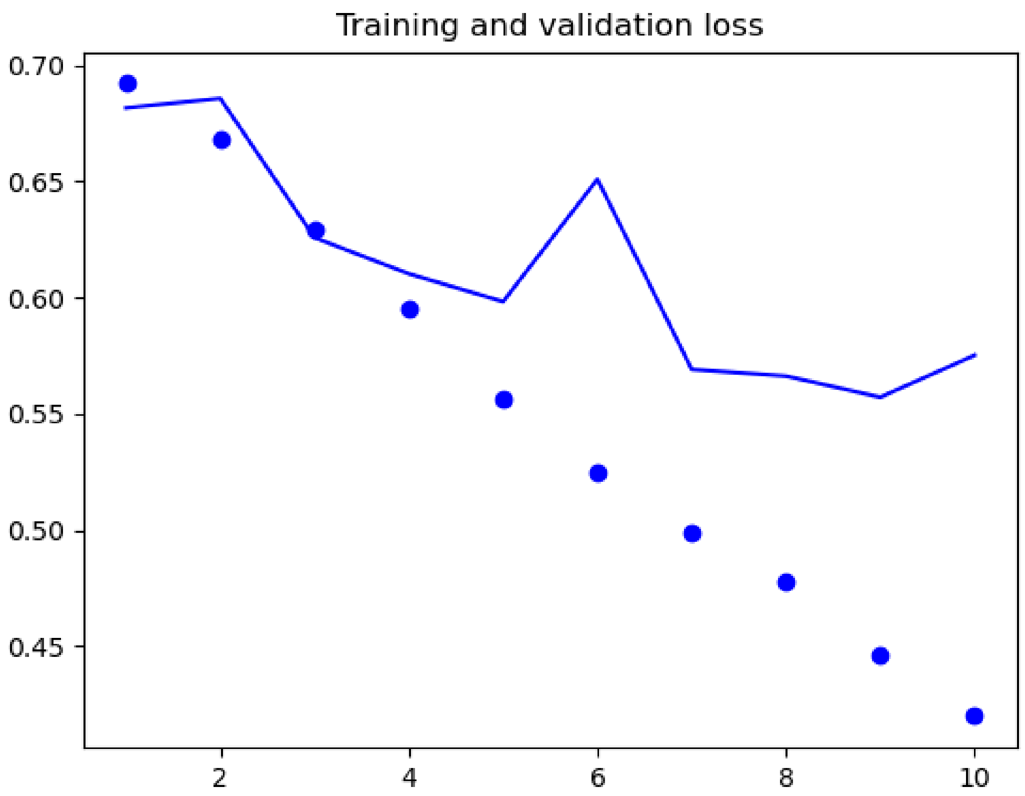
```
    #TODO - Application 1 - Step 3n - Compile the model
    
model.compile(optimizer=tensorflow.keras.optimizers.legacy.RMSprop(l
earning_rate=0.0001), loss='binary_crossentropy',
metrics=['accuracy'])

    return model

def imagePreprocessing(base_directory):
    train_directory = base_directory + '/train'
    validation_directory = base_directory + '/validation'
    #TODO - Application 1 - Step 2 - Create the image data
generators for train and validation
    train_datagen = ImageDataGenerator(rescale=1./255)
    validation_datagen = ImageDataGenerator(rescale=1./255)

    train_generator =
train_datagen.flow_from_directory(train_directory, target_size =
(150, 150), batch_size = 20, class_mode='binary')
    validation_generator =
validation_datagen.flow_from_directory(validation_directory,target_s
ize = (150, 150), batch_size = 20, class_mode='binary')

    #TODO - Application 1 - Step 2 - Analyze the output of the train
and validation generators
    for data_batch, labels_batch in train_generator:
        print('Data batch shape in train: ', data_batch.shape)
        print('Labels batch shape in train: ', labels_batch.shape)
        break
    for data_batch, labels_batch in validation_generator:
        print('Data batch shape in validation: ', data_batch.shape)
        print('Labels batch shape in validation: ',
labels_batch.shape)
        break
    return train_generator, validation_generator



def main():
    original_directory = "./Kaggle_Cats_And_Dogs_Dataset"
    base_directory = "./Kaggle_Cats_And_Dogs_Dataset_Small"

    #TODO - Application 1 - Step 1 - Prepare the dataset
    #prepareDatabase(original_directory, base_directory)

    #TODO - Application 1 - Step 2 - Call the imagePreprocessing
method
    train_generator,
validation_generator=imagePreprocessing(base_directory)

    #TODO - Application 1 - Step 3 - Call the method that creates
the CNN model
    model=defineCNNModelFromScratch()
    #model = defineCNNModelVGGPretained()

    #TODO - Application 1 - Step 4 - Train the model
```

```
    history = model.fit_generator(train_generator,
steps_per_epoch=100, epochs=10,
validation_data=validation_generator, validation_steps=50)

    #TODO - Application 1 - Step 5 - Visualize the system
performance using the diagnostic curves
    visualizeTheTrainingPerformances(history, 'modelfromscratch')
        return

if __name__ == '__main__': main()
```

OUTPUT :

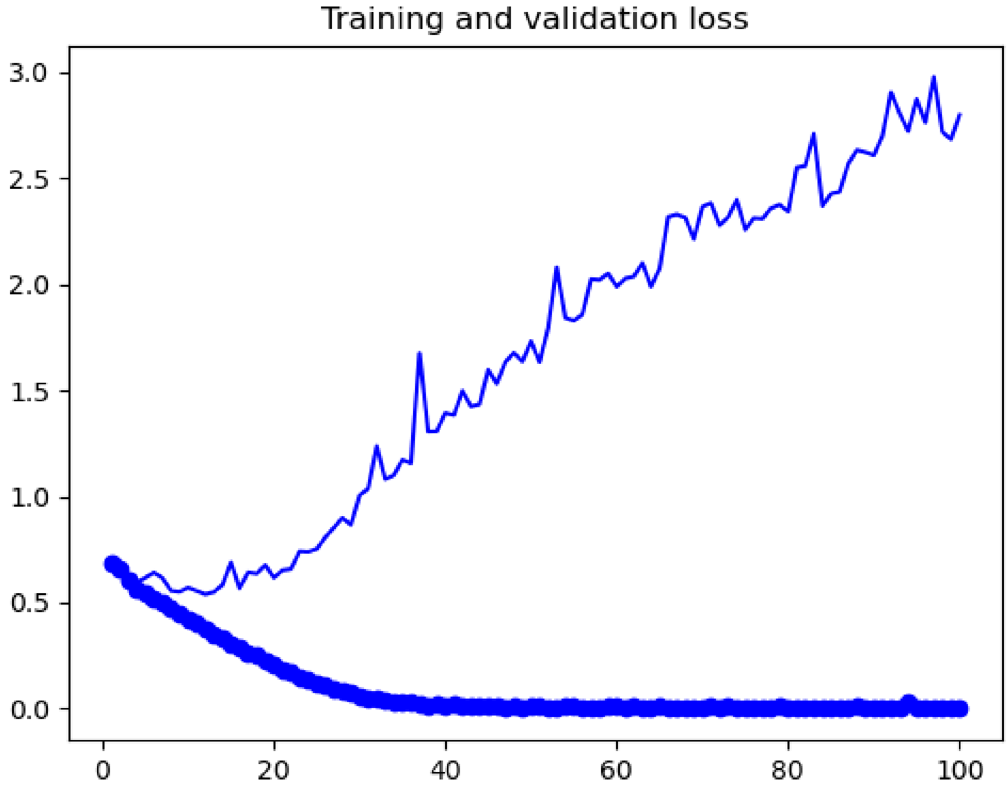
```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Data batch shape in train:  (20, 150, 150, 3)
Labels batch shape in train:  (20,)
Data batch shape in validation:  (20, 150, 150, 3)
Labels batch shape in validation:  (20,)
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_25 (Conv2D)          (None, 148, 148, 32)      896

 max_pooling2d_24 (MaxPooli  (None, 74, 74, 32)        0
 ng2D)

 conv2d_26 (Conv2D)          (None, 72, 72, 64)        18496

 max_pooling2d_25 (MaxPooli  (None, 36, 36, 64)        0
 ng2D)

 conv2d_27 (Conv2D)          (None, 34, 34, 128)       73856

 max_pooling2d_26 (MaxPooli  (None, 17, 17, 128)       0
 ng2D)

 conv2d_28 (Conv2D)          (None, 15, 15, 128)       147584

 max_pooling2d_27 (MaxPooli  (None, 7, 7, 128)         0
 ng2D)

 flatten_6 (Flatten)         (None, 6272)              0

 dense_12 (Dense)            (None, 512)               3211776

 dense_13 (Dense)            (None, 1)                 513

=================================================================
Total params: 3453121 (13.17 MB)
Trainable params: 3453121 (13.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

Epoch 1/10
100/100 [==============================] - 50s 494ms/step - loss: 0.6921 - accuracy: 0.5280
- val_loss: 0.6816 - val_accuracy: 0.5930
.
.
.
Epoch 10/10
100/100 [==============================] - 46s 464ms/step - loss: 0.4201 - accuracy: 0.8140
- val_loss: 0.5752 - val_accuracy: 0.7020
```

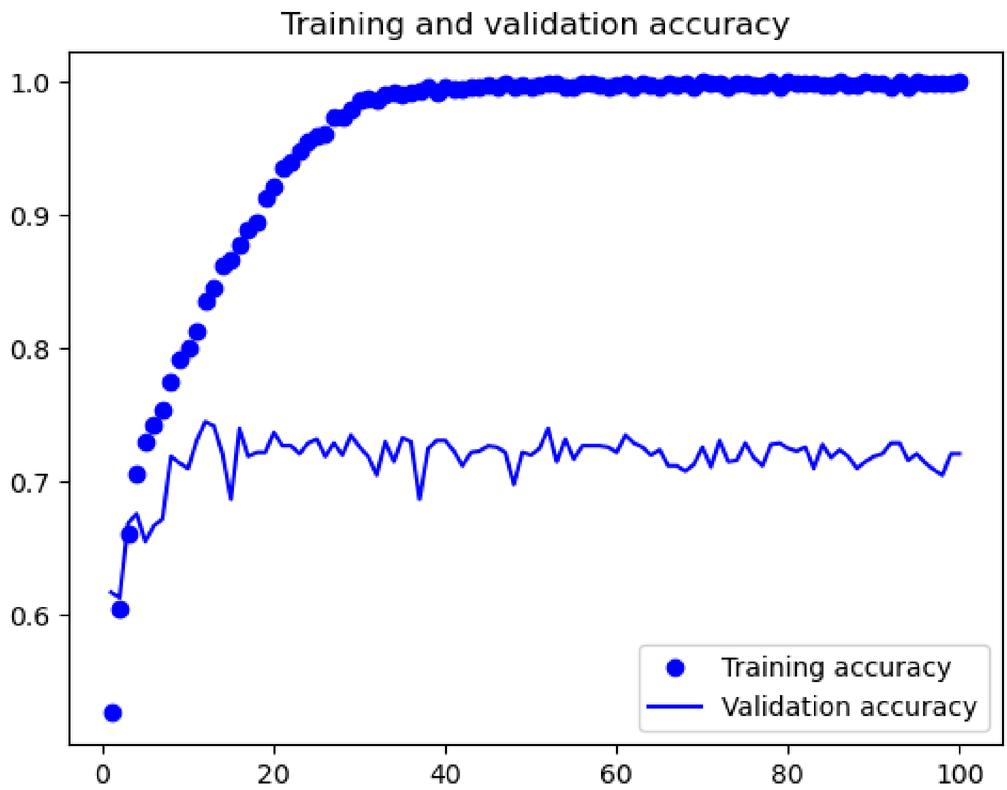

Training and validation accuracy
Training accuracy
Validation accuracy
0.80
0.75
0.70
0.65
0.60
0.55
2
4
6
8
10


Training and validation loss
0.70
0.65
0.60
0.55
0.50
0.45
2
4
6
8
10

Exercise 1:

Save the figures generated by the function visualizeTheTrainingPerformance() when performing the training for 100 epochs.





Exercise 2:

Evaluate the model accuracy on the testing dataset.

We add these lines to the main :

```
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Accuracy}: {test_accuracy:.4f}\n")
```

OUTPUT:

Test Accuracy: 0.7320

Exercise 3:

Using the pre-trained model (saved above), write a Python script able to make an automatic prediction regarding the category for the following images "test1.jpg" and "test2.jpg". The prediction will be performed simultaneously for the two images using a batch of images.

CODE:

```
import cv2
import numpy as np
from tensorflow.keras.models import load_model

def loadImages(image_paths):
    images = []
    for image_path in image_paths:
        # Load the image
        img = cv2.imread(image_path)
        # Convert it to RGB
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        # Resize it to 150x150
        img = cv2.resize(img, (150, 150))
        # Normalize pixel values to [0, 1]
        img = img / 255.0
        images.append(img)
    # Convert the list to a numpy array and add an extra dimension
    images = np.array(images)
    return images

# Path to the saved model
model_path = 'Models_cats_dogs_small_dataset_pretrained.h5'
# Load the pre-trained model
model = load_model(model_path)

# Image paths
image_paths = ['test1.jpg', 'test2.jpg']
# Load and preprocess the images
images = loadImages(image_paths)

# Predict the categories of the images
predictions = model.predict(images)
```

```
# The model outputs a probability close to 1 for dogs and close to 0
for cats
for i, prediction in enumerate(predictions):
    if prediction < 0.5:
        print(f"{image_paths[i]} is a cat with probability {1 -
prediction[0]}")
    else:
        print(f"{image_paths[i]} is a dog with probability
{prediction[0]}")
```

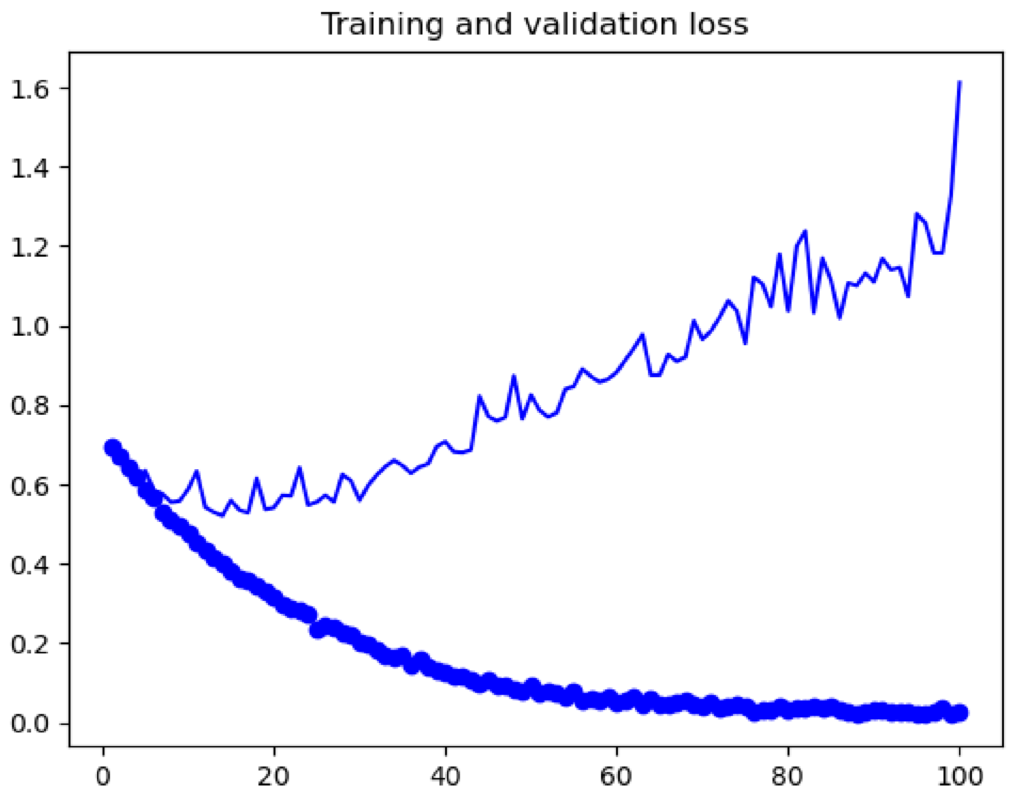
OUTPUT :

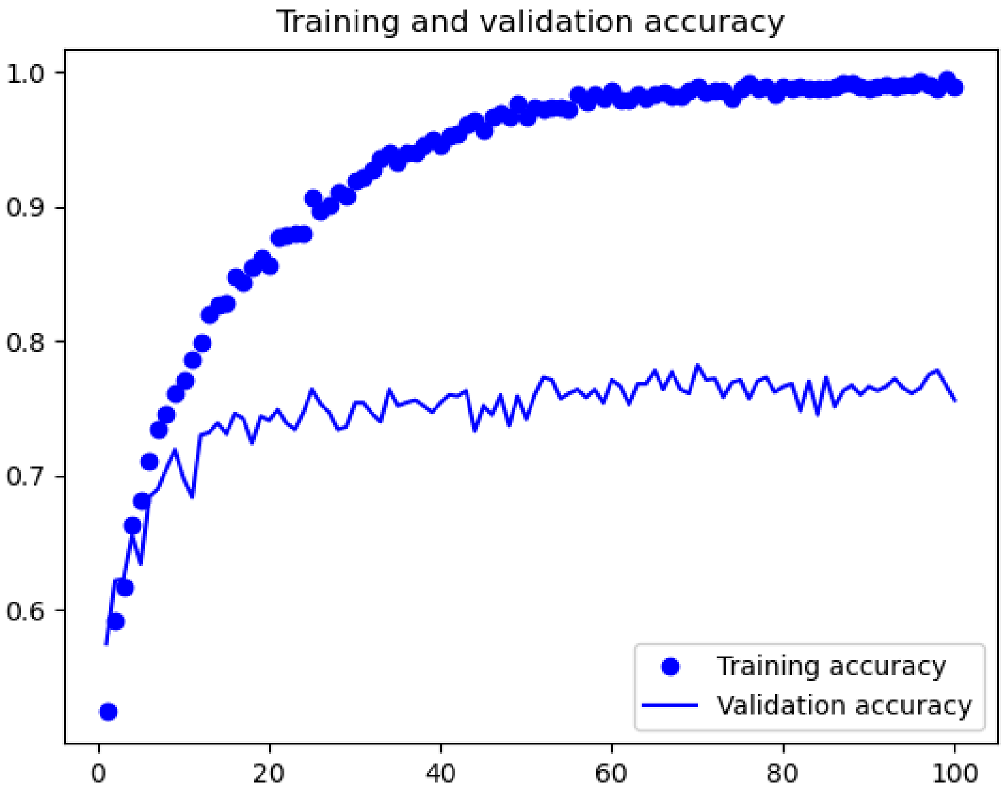
1/1 [==============================] - 0s 106ms/step
test1.jpg is a cat with probability 0.9999955593743834
test2.jpg is a dog with probability 1.0

The tests were indeed successful.

Exercise 4:

Try to further increase the system performances (by reducing the overfitting) by adding a dropout layer to your model that randomly excludes 50% of neurons. The layer should be added right before the densely connected classifier.
In order to further improve the system performance increase the dataset used for training by performing some data augmentation techniques. Let's train the network for 100 epochs using data augmentation techniques as presented bellow. How is the system accuracy influenced by the data augmentation techniques? Compare the graphs with the figure saved at Exercise 1





Test Accuracy : 0.7520

Answer :
We see that there is an increase in the Test Accuracy. However, adding dropout slowed down the convergence time.

Application 2:

In this example, you will need to classify images of cats and dogs by using transfer learning from a pre-trained network. In the ***defineCNNModelVGGPretrained*** *method, load the VGG16 network, previously trained on ImageNet, to extract interesting features from cat and dog images, and then (using the pretrained filters) train a dogs-versus-cats classifier on top of these features.*

CODE :

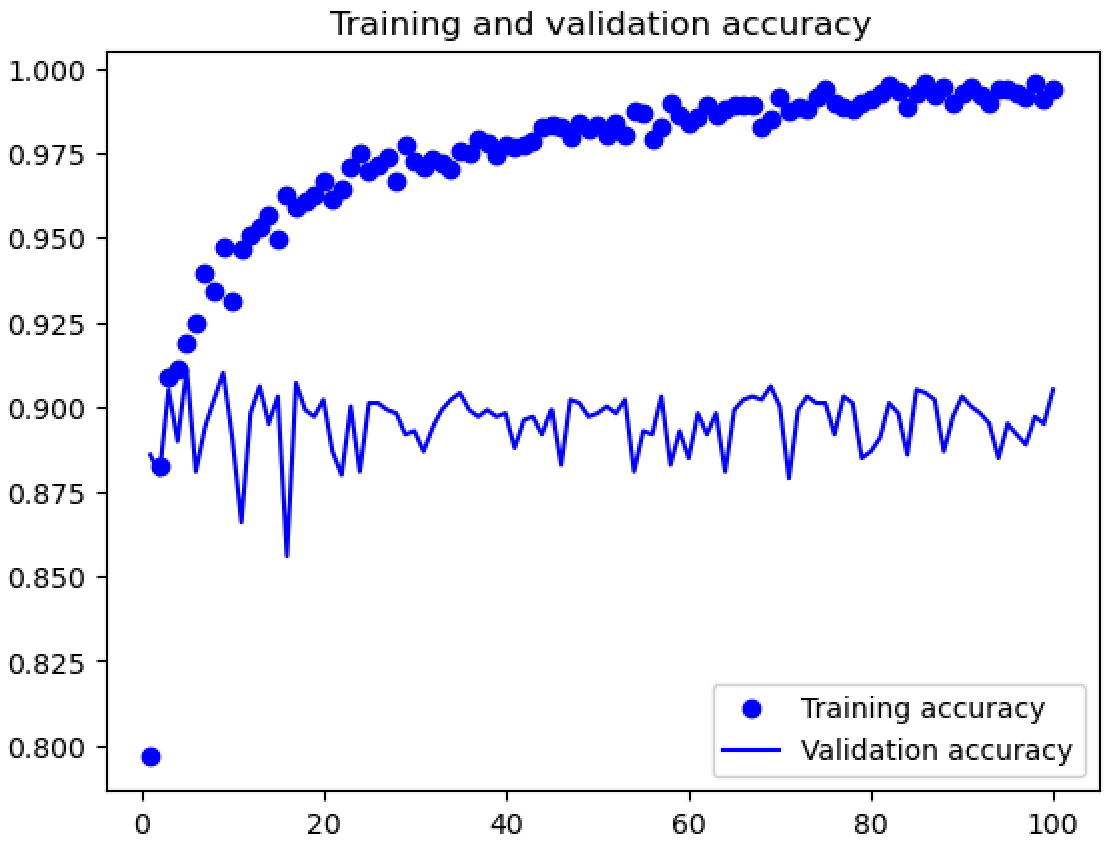
```
def defineCNNModelVGGPretrained():

    #TODO - Application 2 - Step 1 - Load the pretrained VGG16
network in a variable called baseModel
    #The top layers will be omitted; The input_shape will be kept to
(150, 150, 3)
    baseModel = VGG16(weights='imagenet', include_top=False,
input_shape=(150, 150, 3))

    #TODO - Application 2 - Step 2 -  Visualize the network
arhitecture (list of layers)
    print("Base model architecture:")
    baseModel.summary()
```

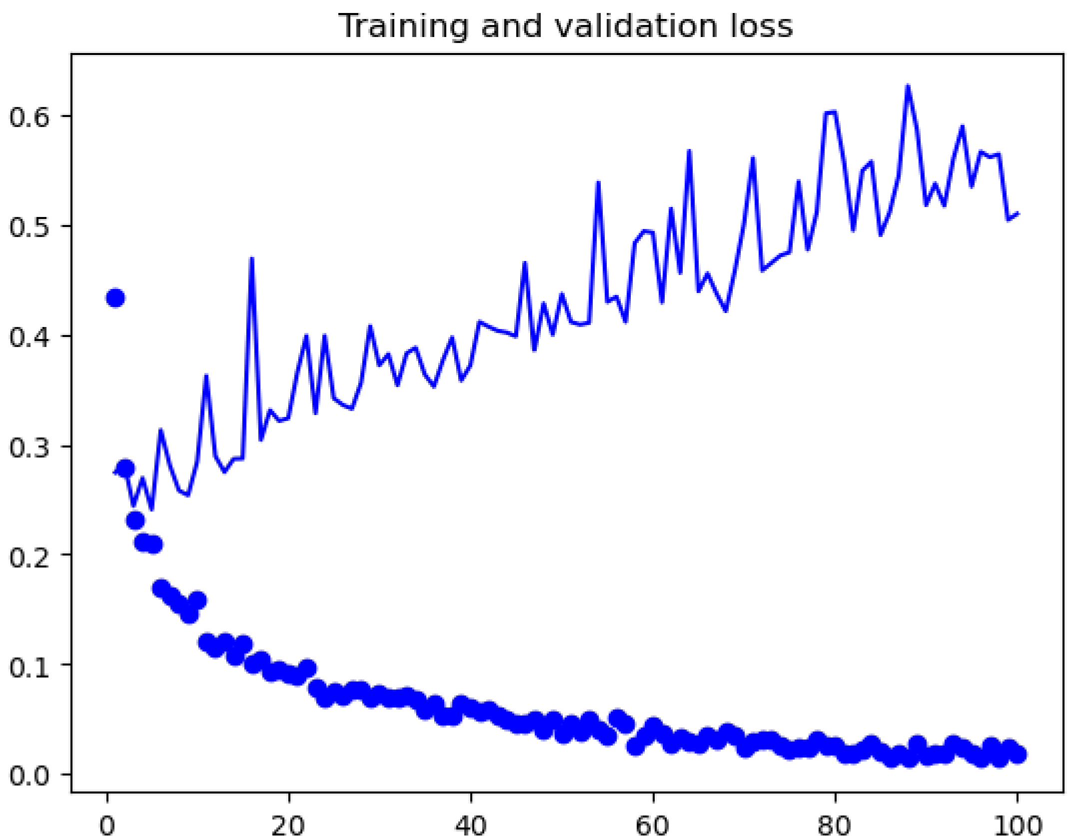
```
    #TODO - Application 2 - Step 3 -  Freeze the baseModel
convolutional layers in order not to allow training
    for layer in baseModel.layers:
        layer.trainable = False

    #TODO - Application 2 - Step 4 - Create the final model and add
the layers from the baseModel
    VGG_model = models.Sequential()
    VGG_model.add(baseModel)

    # TODO - Application 2 - Step 4a - Add the flatten layer
    VGG_model.add(layers.Flatten())

    # TODO - Application 2 - Step 4b - Add the dropout layer
    VGG_model.add(layers.Dropout(0.5))

    # TODO Application 2 - Step 4c - Add a dense layer of size 512
    VGG_model.add(layers.Dense(512, activation='relu'))


    # TODO - Application 2 - Step 4d - Add the output layer
    VGG_model.add(layers.Dense(1, activation='sigmoid'))

    # TODO - Application 2 - Step 4e - Compile the model
    VGG_model.compile(optimizer=RMSprop(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])

    return VGG_model
```

OUTPUT :

```
Total number of CATS used for training = 1000
Total number of CATS used for validation = 500
Total number of CATS used for testing = 500
Total number of DOGS used for training = 1000
Total number of DOGS used for validation = 500
Total number of DOGS used for testing = 500
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Data batch shape in train:  (20, 150, 150, 3)
Labels batch shape in train:  (20,)
Data batch shape in validation:  (20, 150, 150, 3)
Labels batch shape in validation:  (20,)
Data batch shape in test:  (20, 150, 150, 3)
Labels batch shape in test:  (20,)
Base model architecture:
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792
```

```
 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
100/100 [==============================] - 381s 4s/step - loss: 0.4334 - accuracy: 0.7965 - val_loss: 0.2747 - val_accuracy: 0.8860
.
.
.
Epoch 100/100
100/100 [==============================] - 147s 1s/step - loss: 0.0191 - accuracy: 0.9940 - val_loss: 0.5096 - val_accuracy: 0.9050
```


Training and validation accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0
20
40
60
80
100
Training accuracy
Validation accuracy


Training and validation loss
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100

Exercise 5:

In this exercise, you will need to classify images of cats and dogs by using transfer learning and fine-tunning. Write a Python script that uses the VGG16 network, previously trained on ImageNet, to extract interesting features from cat and dog images, unfreeze some of the top layers of the frozen model and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.

You'll fine-tune the last three convolutional layers, which mean that all layers up to block4_pool should be frozen, and the other layers block5_conv1, block5_conv2, and block5_conv3 should be trainable.

CODE :

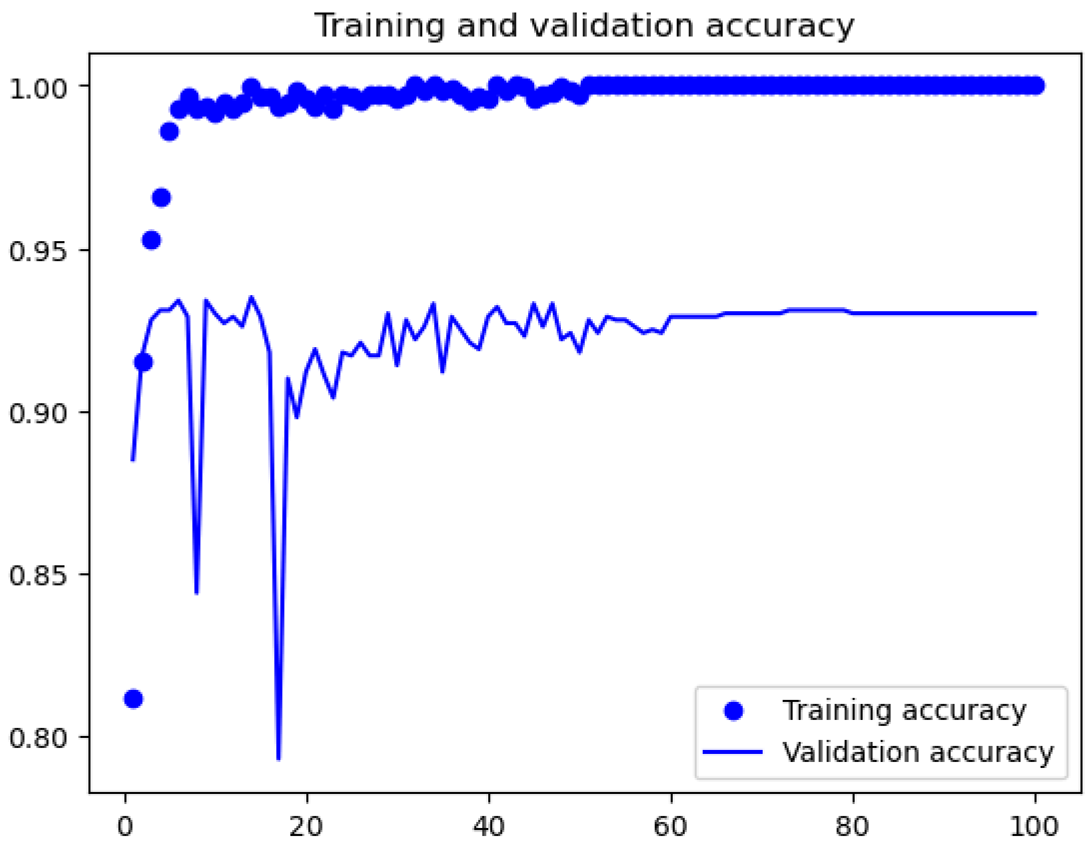
```
def defineCNNModelVGGPretrained():

    #TODO - Application 2 - Step 1 - Load the pretrained VGG16
network in a variable called baseModel
    #The top layers will be omitted; The input_shape will be kept to
(150, 150, 3)
    baseModel = VGG16(weights='imagenet', include_top=False,
input_shape=(150, 150, 3))

    #TODO - Application 2 - Step 2 -  Visualize the network
arhitecture (list of layers)
    print("Base model architecture:")

    #TODO - Application 2 - Step 3 -  Freeze the baseModel
convolutional layers in order not to allow training
    for layer in baseModel.layers:
        layer.trainable = False

    #TODO - Application 2 - Step 4 - Create the final model and add
the layers from the baseModel
    VGG_model = models.Sequential()
    VGG_model.add(baseModel)

    # TODO - Application 2 - Step 4a - Add the flatten layer
    VGG_model.add(layers.Flatten())

    # TODO - Application 2 - Step 4b - Add the dropout layer
    VGG_model.add(layers.Dropout(0.5))

    # TODO Application 2 - Step 4c - Add a dense layer of size 512
    VGG_model.add(layers.Dense(512, activation='relu'))


    # TODO - Application 2 - Step 4d - Add the output layer
    VGG_model.add(layers.Dense(1, activation='sigmoid'))

    # TODO - Application 2 - Step 4e - Compile the model
    VGG_model.compile(optimizer=RMSprop(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
    # Unfreeze the last three convolutional layers in the base model
```

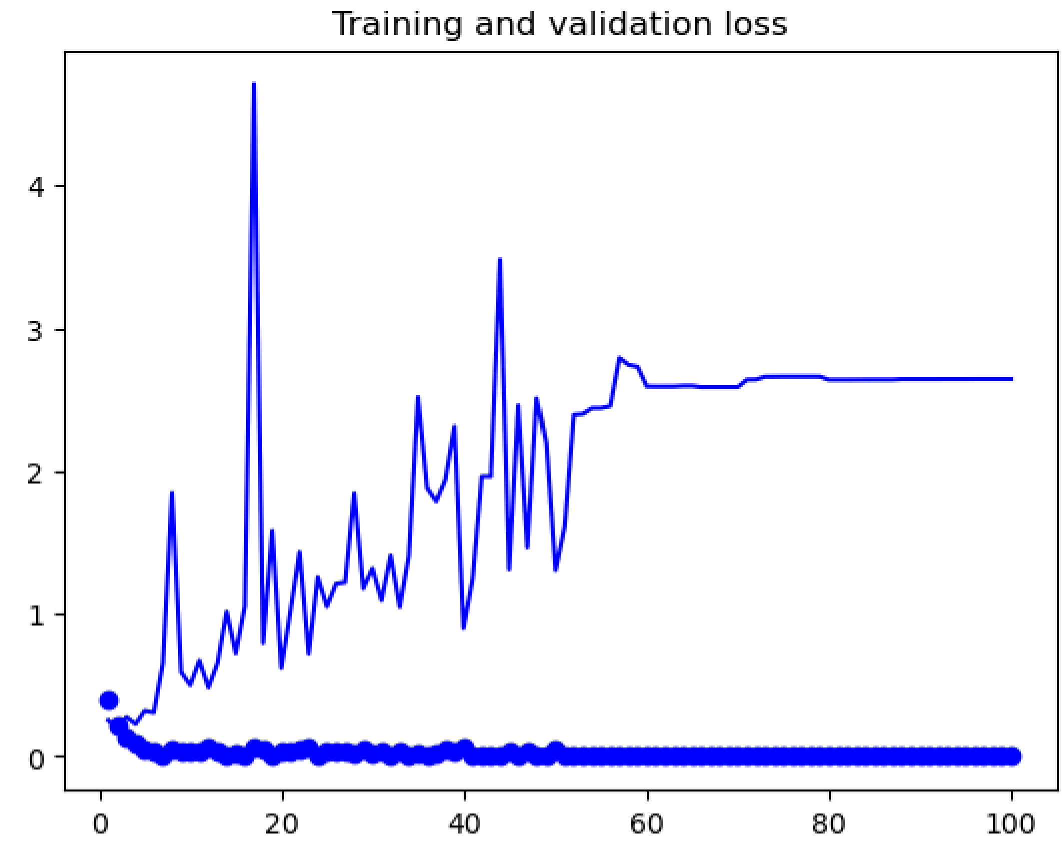
```
    for layer in baseModel.layers:
        if layer.name in ['block5_conv1', 'block5_conv2',
'block5_conv3']:
            layer.trainable = True
        else:
            layer.trainable = False

    baseModel.summary()

    # Recompile the model
    VGG_model.compile(loss='binary_crossentropy',
optimizer=RMSprop(learning_rate=0.0001), metrics=['accuracy'])


    return VGG_model
```

OUTPUT :

```
Total number of CATS used for training = 1000
Total number of CATS used for validation = 500
Total number of CATS used for testing = 500
Total number of DOGS used for training = 1000
Total number of DOGS used for validation = 500
Total number of DOGS used for testing = 500
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Data batch shape in train:  (20, 150, 150, 3)
Labels batch shape in train:  (20,)
Data batch shape in validation:  (20, 150, 150, 3)
Labels batch shape in validation:  (20,)
Data batch shape in test:  (20, 150, 150, 3)
Labels batch shape in test:  (20,)
Base model architecture:
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808
```

```
 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 7,079,424
Non-trainable params: 7,635,264
_________________________________________________________________
Epoch 1/100
100/100 [==============================] - 379s 4s/step - loss: 0.4200 -
accuracy: 0.8165 - val_loss: 0.2021 - val_accuracy: 0.9170
.
.
.
Epoch 100/100
100/100 [==============================] - 187s 2s/step - loss: 3.1774e-13
- accuracy: 1.0000 - val_loss: 2.6425 - val_accuracy: 0.9300
```





We see that we obtained better accuracy results in a faster convergence time

Exercise 6:

Write a Python script that uses the network architectures presented in Table 1, previously trained on ImageNet, to extract interesting features from cat and dog images, and then (using the pretrained filters) train a dogs-versus-cats classifier on top of these features. How is the system accuracy influenced by this network topology type?

CODE :

```
# We create two functions : create_and_train_model and
get_input_size

def create_and_train_model(model_function, input_size,
train_generator, validation_generator, epochs=50,
steps_per_epoch=100, validation_steps=50):

    print("Input size:", input_size)
    print("Model function:", model_function)

    input_shape=input_size + (3,)
    # Load the pre-trained base model without the top layers
    base_model = model_function(weights='imagenet',
include_top=False, input_shape=input_shape)

    # Freeze the layers of the base model
    for layer in base_model.layers:
        layer.trainable = False
```

```
    # Define the custom top layers for classification
    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    # Compile the model
    model.compile(optimizer=RMSprop(learning_rate=0.0001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    # Train the model
    history = model.fit(
        train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=validation_steps
    )

    return model, history


def get_input_size(model_name):
    if model_name in ['Xception', 'InceptionV3']:
        return (299, 299)
    else:
        return (150, 150)
```

Here is the updated main function :

```
def main():
    original_directory = "./Kaggle_Cats_And_Dogs_Dataset"
    base_directory = "./Kaggle_Cats_And_Dogs_Dataset_Small"

    #TODO - Application 1 - Step 1 - Prepare the dataset
    prepareDatabase(original_directory, base_directory)

    #TODO - Application 1 - Step 2 - Call the imagePreprocessing
method
    #train_generator, validation_generator,
test_generator=imagePreprocessing(base_directory)

    #TODO - Application 1 - Step 3 - Call the method that creates
the CNN model
    #model=defineCNNModelFromScratch()
    #model = defineCNNModelVGGPretrained()

    #TODO - Application 1 - Step 4 - Train the model
    #history = model.fit(train_generator, steps_per_epoch=100,
epochs=100, validation_data=validation_generator,
validation_steps=50)

    #TODO - Application 1 - Step 5 - Visualize the system
performance using the diagnostic curves
```

```
    #visualizeTheTrainingPerformances(history, 'VGGModel')
    #model.save('Models_VGGpretrained.h5')

    # Define a dictionary of pre-trained models to evaluate
    pretrained_models = {
        'VGG16': VGG16,
        'Xception': Xception,  # Requires input images to be 299x299
        'InceptionV3': InceptionV3,  # Requires input images to be
299x299
        'ResNet50': ResNet50,
        'MobileNet': MobileNet
    }

    for model_name, model_function in pretrained_models.items():
        print(f"Training with {model_name}...")

        # Adjust the input size for each model
        input_size = get_input_size(model_name)

        # Prepare the data generators with the correct input size
        train_generator, validation_generator, test_generator =
imagePreprocessing(base_directory, input_size)

        # Define and train the model
        model, history =
create_and_train_model(model_function=model_function,
input_size=input_size, train_generator=train_generator,
validation_generator=validation_generator, epochs=50,
steps_per_epoch=100, validation_steps=50)

        # Evaluate the model
        print(f"Evaluating {model_name}...")
        test_loss, test_accuracy = model.evaluate(test_generator)
        print(f"Test Accuracy for {model_name}:
{test_accuracy:.4f}\n")

        # Optionally save the model
        model.save(f'Models_{model_name}_pretrained.h5')


    return
```

OUTPUT :

CNN architecture	VGG16	Xception	ResNet50	Inception	MobileNet
System accuracy	0.8630	0.9970	0.6190	0.9930	0.9720

The comparison of CNN architectures for image classification showcases varying performances: VGG16 achieves a respectable accuracy of 0.8630, illustrating its reliable but computationally intensive nature. Xception stands out with a remarkable accuracy of 0.9970, emphasizing the efficiency of depthwise separable convolutions. Inception also performs impressively at 0.9930, benefiting from its multi-scale processing. ResNet50, however, underperforms at 0.6190, which might reflect a mismatch with dataset specifics or training setup. MobileNet achieves a high accuracy of 0.9720, proving its effectiveness as a lightweight model optimized for speed.