# ATTENTION: Please do not alter any of the provided code in the exercise. Only add your own code where indicated

# ATTENTION: Please do not add or remove any cells in the exercise. The grader will check specific cells based on the cell position.

# ATTENTION: Please use the provided epoch values when training.

# In this exercise you will train a CNN on the FULL Cats-v-dogs dataset

# This will require you doing a lot of data preprocessing because

# the dataset isn't split into training and validation for you

# This code block has all the required inputs

import os

import zipfile

import random

import tensorflow as tf

import shutil

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from shutil import copyfile

from os import getcwd

path\_cats\_and\_dogs = f"{getcwd()}/../tmp2/cats-and-dogs.zip"

shutil.rmtree('/tmp')

local\_zip = path\_cats\_and\_dogs

zip\_ref = zipfile.ZipFile(local\_zip, 'r')

zip\_ref.extractall('/tmp')

zip\_ref.close()

print(len(os.listdir('/tmp/PetImages/Cat/')))

print(len(os.listdir('/tmp/PetImages/Dog/')))

# Expected Output:

# 1500

# 1500

# Use os.mkdir to create your directories

# You will need a directory for cats-v-dogs, and subdirectories for training

# and testing. These in turn will need subdirectories for 'cats' and 'dogs'

to\_create = [

'/tmp/cats-v-dogs',

'/tmp/cats-v-dogs/training',

'/tmp/cats-v-dogs/testing',

'/tmp/cats-v-dogs/training/cats',

'/tmp/cats-v-dogs/training/dogs',

'/tmp/cats-v-dogs/testing/cats',

'/tmp/cats-v-dogs/testing/dogs'

]

for directory in to\_create:

try:

os.mkdir(directory)

print(directory, 'created')

#YOUR CODE GOES HERE

except OSError:

print(directory, 'failed')

# Write a python function called split\_data which takes

# a SOURCE directory containing the files

# a TRAINING directory that a portion of the files will be copied to

# a TESTING directory that a portion of the files will be copie to

# a SPLIT SIZE to determine the portion

# The files should also be randomized, so that the training set is a random

# X% of the files, and the test set is the remaining files

# SO, for example, if SOURCE is PetImages/Cat, and SPLIT SIZE is .9

# Then 90% of the images in PetImages/Cat will be copied to the TRAINING dir

# and 10% of the images will be copied to the TESTING dir

# Also -- All images should be checked, and if they have a zero file length,

# they will not be copied over

#

# os.listdir(DIRECTORY) gives you a listing of the contents of that directory

# os.path.getsize(PATH) gives you the size of the file

# copyfile(source, destination) copies a file from source to destination

# random.sample(list, len(list)) shuffles a list

def split\_data(SOURCE, TRAINING, TESTING, SPLIT\_SIZE):

# YOUR CODE STARTS HERE

all\_files=[]

for file\_name in os.listdir(SOURCE):

file\_path = SOURCE + file\_name

if os.path.getsize(file\_path):

all\_files.append(file\_name)

else:

print('{} is zero length, so ignoring'.format(file\_name))

n\_files = len(all\_files)

split\_point = int(n\_files \* SPLIT\_SIZE)

shuffled = random.sample(all\_files, n\_files)

train\_set = shuffled[:split\_point]

test\_set = shuffled[split\_point:]

for file\_name in train\_set:

copyfile(SOURCE + file\_name, TRAINING + file\_name)

for file\_name in test\_set:

copyfile(SOURCE + file\_name, TESTING + file\_name)

# YOUR CODE ENDS HERE

CAT\_SOURCE\_DIR = "/tmp/PetImages/Cat/"

TRAINING\_CATS\_DIR = "/tmp/cats-v-dogs/training/cats/"

TESTING\_CATS\_DIR = "/tmp/cats-v-dogs/testing/cats/"

DOG\_SOURCE\_DIR = "/tmp/PetImages/Dog/"

TRAINING\_DOGS\_DIR = "/tmp/cats-v-dogs/training/dogs/"

TESTING\_DOGS\_DIR = "/tmp/cats-v-dogs/testing/dogs/"

split\_size = .9

split\_data(CAT\_SOURCE\_DIR, TRAINING\_CATS\_DIR, TESTING\_CATS\_DIR, split\_size)

split\_data(DOG\_SOURCE\_DIR, TRAINING\_DOGS\_DIR, TESTING\_DOGS\_DIR, split\_size)

print(len(os.listdir('/tmp/cats-v-dogs/training/cats/')))

print(len(os.listdir('/tmp/cats-v-dogs/training/dogs/')))

print(len(os.listdir('/tmp/cats-v-dogs/testing/cats/')))

print(len(os.listdir('/tmp/cats-v-dogs/testing/dogs/')))

# Expected output:

# 1350

# 1350

# 150

# 150

# DEFINE A KERAS MODEL TO CLASSIFY CATS V DOGS

# USE AT LEAST 3 CONVOLUTION LAYERS

model = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(32, (3,3), activation='relu', input\_shape=(150, 150, 3)),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(64, (2,2), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dense(1, activation='sigmoid')

# YOUR CODE HERE

])

model.compile(optimizer=RMSprop(lr=0.001), loss='binary\_crossentropy', metrics=['acc'])

TRAINING\_DIR = '/tmp/cats-v-dogs/training' #YOUR CODE HERE

train\_datagen = ImageDataGenerator(rescale=1 / 255) #YOUR CODE HERE

# NOTE: YOU MUST USE A BATCH SIZE OF 10 (batch\_size=10) FOR THE

# TRAIN GENERATOR.

train\_generator = train\_datagen.flow\_from\_directory(TRAINING\_DIR,

batch\_size=10,

class\_mode='binary',

target\_size=(150, 150)) #YOUR CODE HERE

VALIDATION\_DIR = '/tmp/cats-v-dogs/testing' #YOUR CODE HERE

validation\_datagen = ImageDataGenerator(rescale= 1/255) #YOUR CODE HERE

# NOTE: YOU MUST USE A BACTH SIZE OF 10 (batch\_size=10) FOR THE

# VALIDATION GENERATOR.

validation\_generator =validation\_datagen.flow\_from\_directory(VALIDATION\_DIR,

batch\_size=10,

class\_mode='binary',

target\_size=(150, 150)) #YOUR CODE HERE

# Expected Output:

# Found 2700 images belonging to 2 classes.

# Found 300 images belonging to 2 classes.

history = model.fit\_generator(train\_generator,

epochs=2,

verbose=1,

validation\_data=validation\_generator)

# PLOT LOSS AND ACCURACY

%matplotlib inline

import matplotlib.image as mpimg

import matplotlib.pyplot as plt

#-----------------------------------------------------------

# Retrieve a list of list results on training and test data

# sets for each training epoch

#-----------------------------------------------------------

acc=history.history['acc']

val\_acc=history.history['val\_acc']

loss=history.history['loss']

val\_loss=history.history['val\_loss']

epochs=range(len(acc)) # Get number of epochs

#------------------------------------------------

# Plot training and validation accuracy per epoch

#------------------------------------------------

plt.plot(epochs, acc, 'r', "Training Accuracy")

plt.plot(epochs, val\_acc, 'b', "Validation Accuracy")

plt.title('Training and validation accuracy')

plt.figure()

#------------------------------------------------

# Plot training and validation loss per epoch

#------------------------------------------------

plt.plot(epochs, loss, 'r', "Training Loss")

plt.plot(epochs, val\_loss, 'b', "Validation Loss")

plt.title('Training and validation loss')

# Desired output. Charts with training and validation metrics. No crash :)