# Import all the necessary files!

import os

import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras import Model

from os import getcwd

path\_inception = f"{getcwd()}/../tmp2/inception\_v3\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5"

# Import the inception model

from tensorflow.keras.applications.inception\_v3 import InceptionV3

# Create an instance of the inception model from the local pre-trained weights

local\_weights\_file = path\_inception

pre\_trained\_model = InceptionV3(

input\_shape=(150, 150, 3),

include\_top=False,

weights=None

) # Your Code Here

pre\_trained\_model.load\_weights(local\_weights\_file)

# Make all the layers in the pre-trained model non-trainable

for layer in pre\_trained\_model.layers:

layer.trainable = False

# Your Code Here

# Print the model summary

pre\_trained\_model.summary()

last\_layer = pre\_trained\_model.get\_layer('mixed7' # Your Code Here

)

print('last layer output shape: ', last\_layer.output\_shape)

last\_output = last\_layer.output # Your Code Here

# Define a Callback class that stops training once accuracy reaches 97.0%

class myCallback(tf.keras.callbacks.Callback):

def on\_epoch\_end(self, epoch, logs={}):

try:

if(logs.get('accuracy')>0.97):

print("\nReached 97.0% accuracy so cancelling training!")

self.model.stop\_training = True

except:

if(logs.get('accuracy')>0.97):

print("\nReached 97.0% accuracy so cancelling training!")

self.model.stop\_training = True

from tensorflow.keras.optimizers import RMSprop

# Flatten the output layer to 1 dimension

x = layers.Flatten()(last\_output)

# Add a fully connected layer with 1,024 hidden units and ReLU activation

x = layers.Dense(1024, activation='relu' # Your Code Here

)(x)

# Add a dropout rate of 0.2

x = layers.Dropout(0.2# Your Code Here

)(x)

# Add a final sigmoid layer for classification

x = layers.Dense (1, activation='sigmoid'# Your Code Here

)(x)

model = Model( pre\_trained\_model.input, x # Your Code Here

)

model.compile(optimizer = RMSprop(lr=0.0001),

loss = 'binary\_crossentropy', # Your Code Here

metrics = ['accuracy'] # Your Code Here

)

model.summary()

# Get the Horse or Human dataset

path\_horse\_or\_human = f"{getcwd()}/../tmp2/horse-or-human.zip"

# Get the Horse or Human Validation dataset

path\_validation\_horse\_or\_human = f"{getcwd()}/../tmp2/validation-horse-or-human.zip"

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

import zipfile

import shutil

shutil.rmtree('/tmp')

local\_zip = path\_horse\_or\_human

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

zip\_ref.extractall('/tmp/training')

zip\_ref.close()

local\_zip = path\_validation\_horse\_or\_human

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

zip\_ref.extractall('/tmp/validation')

zip\_ref.close()

# Define our example directories and files

train\_dir = '/tmp/training'

validation\_dir = '/tmp/validation'

train\_horses\_dir = os.path.join(train\_dir, 'horses')# Your Code Here

train\_humans\_dir = os.path.join(train\_dir, 'humans')# Your Code Here

validation\_horses\_dir = os.path.join(validation\_dir, 'horses')# Your Code Here

validation\_humans\_dir = os.path.join(validation\_dir, 'humans')# Your Code Here

train\_horses\_fnames = os.listdir(train\_horses\_dir) # Your Code Here

train\_humans\_fnames = os.listdir(train\_humans\_dir) # Your Code Here

validation\_horses\_fnames = os.listdir(validation\_horses\_dir) # Your Code Here

validation\_humans\_fnames = os.listdir(validation\_horses\_dir) # Your Code Here

print(len(train\_horses\_fnames) # Your Code Here

)

print(len(train\_humans\_fnames) # Your Code Here

)

print(len(validation\_horses\_fnames) # Your Code Here

)

print(len(validation\_humans\_fnames) # Your Code Here

)

# Add our data-augmentation parameters to ImageDataGenerator

train\_datagen = ImageDataGenerator(rescale = 1./255.,

rotation\_range = 40,

width\_shift\_range = 0.2,

height\_shift\_range = 0.2,

shear\_range = 0.2,

zoom\_range = 0.2,

horizontal\_flip = True # Your Code Here

)

# Note that the validation data should not be augmented!

test\_datagen = ImageDataGenerator(rescale = 1./255. # Your Code Here

)

# Flow training images in batches of 20 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(train\_dir,

batch\_size=64,

class\_mode='binary',

target\_size=(150,150) # Your Code Here

)

# Flow validation images in batches of 20 using test\_datagen generator

validation\_generator = test\_datagen.flow\_from\_directory( validation\_dir,

batch\_size=64,

class\_mode='binary',

target\_size=(150,150) # Your Code Here

)

# Run this and see how many epochs it should take before the callback

# fires, and stops training at 97% accuracy

callbacks = myCallback() # Your Code Here

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

epochs=3,

validation\_data=validation\_generator,

callbacks=[callbacks] # Your Code Here (set epochs = 3)

)

%matplotlib inline

import matplotlib.pyplot as plt

acc = history.history['accuracy']

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

loss = history.history['loss']

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

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')

plt.plot(epochs, val\_acc, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

plt.legend(loc=0)

plt.figure()

plt.show()