```
# ATTENTION: Please do not alter any of the provided code in the exercise. Only add
# ATTENTION: Please do not add or remove any cells in the exercise. The grader will
# ATTENTION: Please use the provided epoch values when training.

# 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 = # 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:
    # Your Code Here

# Print the model summary
pre_trained_model.summary()

# Expected Output is extremely large, but should end with:

#batch_normalization_v1_281 (Bat (None, 3, 3, 192)     576         conv2d_281[0][0
#__________________________________________________________________________________________________
#activation_273 (Activation)     (None, 3, 3, 320)     0           batch_nor
#__________________________________________________________________________________________________
#mixed9_1 (Concatenate)          (None, 3, 3, 768)     0           acti
#
#__________________________________________________________________________________________________
#concatenate_5 (Concatenate)     (None, 3, 3, 768)     0           activatio
#
#__________________________________________________________________________________________________
#activation_281 (Activation)     (None, 3, 3, 192)     0           batch_nor
#__________________________________________________________________________________________________
#mixed10 (Concatenate)           (None, 3, 3, 2048)    0           acti
#
#
#
#==================================================================================================
#Total params: 21,802,784
#Trainable params: 0
#Non-trainable params: 21,802,784
```

```
last_layer = pre_trained_model.get_layer(# Your Code Here)
print('last layer output shape: ', last_layer.output_shape)
last_output = # Your Code Here

# Expected Output:
# ('last layer output shape: ', (None, 7, 7, 768))


# 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={}):
        if(logs.get('acc')>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(# Your Code Here)(x)
# Add a dropout rate of 0.2
x = layers.Dropout(# Your Code Here)(x)
# Add a final sigmoid layer for classification
x = layers.Dense   (# Your Code Here)(x)

model = Model( # Your Code Here, x)

model.compile(optimizer = RMSprop(lr=0.0001),
              loss = # Your Code Here,
              metrics = # Your Code Here)

model.summary()

# Expected output will be large. Last few lines should be:

# mixed7 (Concatenate)                (None, 7, 7, 768)    0
#
#
#
# ______________________________________________________________________________________________
# flatten_4 (Flatten)                 (None, 37632)        0
# ______________________________________________________________________________________________
# dense_8 (Dense)                     (None, 1024)         38536192
# ______________________________________________________________________________________________
# dropout_4 (Dropout)                 (None, 1024)         0
# ______________________________________________________________________________________________
# dense_9 (Dense)                     (None, 1)            1025
# ==============================================================================================
# Total params: 47,512,481
# Trainable params: 38,537,217
# Non-trainable params: 8,975,264
```

```
# 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 = # Your Code Here
train_humans_dir = # Your Code Here
validation_horses_dir = # Your Code Here
validation_humans_dir = # Your Code Here

train_horses_fnames = # Your Code Here
train_humans_fnames = # Your Code Here
validation_horses_fnames = # Your Code Here
validation_humans_fnames = # Your Code Here

print(# Your Code Here)
print(# Your Code Here)
print(# Your Code Here)
print(# Your Code Here)

# Expected Output:
# 500
# 527
# 128
# 128


# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(# Your Code Here)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(# Your Code Here )
```

```
# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(# Your Code Here)

# Flow validation images in batches of 20 using test_datagen generator
validation_generator =  test_datagen.flow_from_directory( # Your Code Here)

# Expected Output:
# Found 1027 images belonging to 2 classes.
# Found 256 images belonging to 2 classes.
```

```
# Run this and see how many epochs it should take before the callback
# fires, and stops training at 97% accuracy

callbacks = # Your Code Here
history = model.fit_generator(# Your Code Here (set epochs = 3))
```

```
%matplotlib inline
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
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()
```

Submission Instructions

```
# Now click the 'Submit Assignment' button above.
```

When you're done or would like to take a break, please run the two cells below to save your work and close the Notebook. This will free up resources for your fellow learners.

```
%%javascript
<!-- Save the notebook -->
IPython.notebook.save_checkpoint();
```

```
%%javascript
IPython.notebook.session.delete();
window.onbeforeunload = null
setTimeout(function() { window.close(); }, 1000);
```