In [1]:

```
# ATTENTION: Please do not alter any of the provided code in the exercise. Only add your own
# ATTENTION: Please do not add or remove any cells in the exercise. The grader will check sp
# 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
```

In [2]:

```
path_inception = f"{getcwd()}/../tmp2/inception_v3_weights_tf_dim_ordering_tf_kernels_notop

# 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)

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()

# 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_normalization_v1_27
#__________________________________________________________________________________________
#mixed9_1 (Concatenate)           (None, 3, 3, 768)    0           activation_275[0][0]
#                                                                  activation_276[0][0]
#__________________________________________________________________________________________
#concatenate_5 (Concatenate)      (None, 3, 3, 768)    0           activation_279[0][0]
#                                                                  activation_280[0][0]
#__________________________________________________________________________________________
#activation_281 (Activation)      (None, 3, 3, 192)    0           batch_normalization_v1_28
#__________________________________________________________________________________________
#mixed10 (Concatenate)            (None, 3, 3, 2048)   0           activation_273[0][0]
#                                                                  mixed9_1[0][0]
#                                                                  concatenate_5[0][0]
#                                                                  activation_281[0][0]
#==========================================================================================
#Total params: 21,802,784
#Trainable params: 0
#Non-trainable params: 21,802,784
```

```
Model: "inception_v3"
__________________________________________________________________________________________
__________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
==========
input_1 (InputLayer)            [(None, 150, 150, 3) 0
__________________________________________________________________________________________
__________
conv2d (Conv2D)                 (None, 74, 74, 32)   864         input_1[0][0]
__________________________________________________________________________________________
__________
batch_normalization (BatchNorma (None, 74, 74, 32)   96          conv2d[0][0]
__________________________________________________________________________________________
__________
activation (Activation)         (None, 74, 74, 32)   0           batch_normalization[0]
[0]
__________________________________________________________________________________________
__________
conv2d_1 (Conv2D)               (None, 72, 72, 32)   9216        activation[0][0]
```

In [3]:

```
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output# Your Code Here

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

```
last layer output shape:  (None, 7, 7, 768)
```

In [4]:

```
# 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('accuracy')>=0.97):
      print("\nReached 97.0% accuracy so cancelling training!")
      self.model.stop_training = True
```

In [5]:

```
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')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a final sigmoid layer for classification
x = layers.Dense  (1,activation='sigmoid')(x)

model = Model(pre_trained_model.input, x) #first model and second model

model.compile(optimizer = RMSprop(lr=0.0001),
              loss = 'binary_crossentropy',
              metrics =['accuracy'])

model.summary()

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

# mixed7 (Concatenate)            (None, 7, 7, 768)    0           activation_248[0][0]
#                                                                  activation_251[0][0]
#                                                                  activation_256[0][0]
#                                                                  activation_257[0][0]
# ______________________________________________________________________________________
# flatten_4 (Flatten)             (None, 37632)        0           mixed7[0][0]
# ______________________________________________________________________________________
# dense_8 (Dense)                 (None, 1024)         38536192    flatten_4[0][0]
# ______________________________________________________________________________________
# dropout_4 (Dropout)             (None, 1024)         0           dense_8[0][0]
# ______________________________________________________________________________________
# dense_9 (Dense)                 (None, 1)            1025        dropout_4[0][0]
# ======================================================================================
# Total params: 47,512,481
# Trainable params: 38,537,217
# Non-trainable params: 8,975,264
```

```
________________________________________________________________________________________
__________
batch_normalization (BatchNorma (None, 74, 74, 32)   96          conv2d[0][0]
________________________________________________________________________________________
__________
activation (Activation)         (None, 74, 74, 32)   0           batch_normalization[0]
[0]
________________________________________________________________________________________
__________
conv2d_1 (Conv2D)               (None, 72, 72, 32)   9216        activation[0][0]
________________________________________________________________________________________
__________
batch_normalization_1 (BatchNor (None, 72, 72, 32)   96          conv2d_1[0][0]
________________________________________________________________________________________
__________
activation_1 (Activation)       (None, 72, 72, 32)   0           batch_normalization_1
[0][0]
________________________________________________________________________________________
__________
conv2d_2 (Conv2D)               (None, 72, 72, 64)   18432       activation_1[0][0]
```

In [6]:

```
# 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()
```

In [7]:

```
# Define our example directories and files
train_dir = '/tmp/training'
validation_dir = '/tmp/validation'

train_horses_dir = "/tmp/training/horses" # Your Code Here
train_humans_dir = "/tmp/training/humans" # Your Code Here
validation_horses_dir = "/tmp/validation/horses" # Your Code Here
validation_humans_dir = "/tmp/validation/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_humans_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

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

```
500
527
128
128
```

In [8]:

```
# 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)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale = 1./255. )

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 20,
                                                    class_mode = 'binary',
                                                    target_size = (150, 150))

# Flow validation images in batches of 20 using test_datagen generator
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size = 20,
                                                         class_mode = 'binary',
                                                         target_size = (150, 150))

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

```
Found 1027 images belonging to 2 classes.
Found 256 images belonging to 2 classes.
```

In [9]:

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

callbacks = myCallback()
history = model.fit_generator(train_generator,
                              validation_data=validation_generator,
                              epochs=3,
                              verbose=1,
                              callbacks=[callbacks])
```

```
Epoch 1/3
52/52 [==============================] - 52s 996ms/step - loss: 0.2467 - accuracy: 0.9036
- val_loss: 0.1487 - val_accuracy: 0.9258
Epoch 2/3
52/52 [==============================] - 44s 840ms/step - loss: 0.0768 - accuracy: 0.9679
- val_loss: 0.0121 - val_accuracy: 0.9961
Epoch 3/3
51/52 [============================>.] - ETA: 0s - loss: 0.0537 - accuracy: 0.9791
Reached 97.0% accuracy so cancelling training!
52/52 [==============================] - 43s 835ms/step - loss: 0.0537 - accuracy: 0.9786
- val_loss: 2.1995e-04 - val_accuracy: 1.0000
```

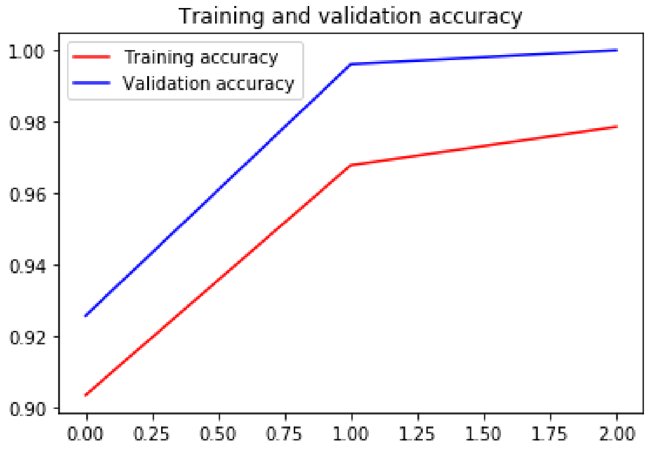
In [15]:

```
%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()
```



```
<Figure size 432x288 with 0 Axes>
```

Submission Instructions

In [ ]:

```
# 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.

In [ ]:

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

In [ ]:

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