In [2]:
```
# ATTENTION: Please do not alter any of the provided code in the exercise. Only c
# ATTENTION: Please do not add or remove any cells in the exercise. The grader wi
# 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 [3]:
```
path_inception = f"{getcwd()}/../tmp2/inception_v3_weights_tf_dim_ordering_tf_ker

# 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

# 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][(
#__________________________________________________________________________________
#activation_273 (Activation)     (None, 3, 3, 320)    0           batch_normaliz
#__________________________________________________________________________________
#mixed9_1 (Concatenate)          (None, 3, 3, 768)    0           activation_275[
#                                                                 activation_276[
#__________________________________________________________________________________
#concatenate_5 (Concatenate)     (None, 3, 3, 768)    0           activation_279[
#                                                                 activation_280[
#__________________________________________________________________________________
#activation_281 (Activation)     (None, 3, 3, 192)    0           batch_normaliz
#__________________________________________________________________________________
#mixed10 (Concatenate)           (None, 3, 3, 2048)   0           activation_273[
#                                                                 mixed9_1[0][0]
#                                                                 concatenate_5[(
#                                                                 activation_281[
#==================================================================================
#Total params: 21,802,784
#Trainable params: 0
#Non-trainable params: 21,802,784
```

```
[0]
__________________________________________________________________________________
__________________
batch_normalization_43 (BatchNo (None, 7, 7, 192)    576         conv2d_43[0]
[0]
__________________________________________________________________________________
__________________
batch_normalization_48 (BatchNo (None, 7, 7, 192)    576         conv2d_48[0]
[0]
__________________________________________________________________________________
__________________
```

```
batch_normalization_49 (BatchNo (None, 7, 7, 192)    576         conv2d_49[0]
[0]
__________________________________________________________________________
__________________
activation_40 (Activation)      (None, 7, 7, 192)    0           batch_normal
ization_40[0][0]
__________________________________________________________________________
__________________
activation_43 (Activation)      (None, 7, 7, 192)    0           batch_normal
```

In [4]:
```
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output

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

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

In [5]:
```
# 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
```

In [6]:
```
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)

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

model.summary()

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

# mixed7 (Concatenate)            (None, 7, 7, 768)    0           activation_248
#                                                                  activation_251
#                                                                  activation_256
#                                                                  activation_257
# __________________________________________________________________________________
# 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_32 (BatchNo (None, 7, 7, 128)    384         conv2d_32[0]
[0]
__________________________________________________________________________________
__________________
batch_normalization_37 (BatchNo (None, 7, 7, 128)    384         conv2d_37[0]
[0]
__________________________________________________________________________________
__________________
activation_32 (Activation)      (None, 7, 7, 128)    0           batch_normal
ization_32[0][0]
__________________________________________________________________________________
__________________
activation_37 (Activation)      (None, 7, 7, 128)    0           batch_normal
ization_37[0][0]
__________________________________________________________________________________
__________________
average_pooling2d_3 (AveragePoo (None, 7, 7, 768)    0           mixed3[0][0]
```

In [7]:
```
# 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.z
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 [8]:
```
# Define our example directories and files
train_dir = '/tmp/training'
validation_dir = '/tmp/validation'

train_horses_dir = os.path.join(train_dir, "horses")
train_humans_dir = os.path.join(train_dir, "humans")
validation_horses_dir = os.path.join(validation_dir, "horses")
validation_humans_dir = os.path.join(validation_dir, "humans")

train_horses_fnames = os.listdir(train_horses_dir)
train_humans_fnames = os.listdir(train_humans_dir)
validation_horses_fnames = os.listdir(validation_horses_dir)
validation_humans_fnames = os.listdir(validation_humans_dir)

print(len(train_horses_fnames))
print(len(train_humans_fnames))
print(len(validation_horses_fnames))
print(len(validation_humans_fnames))

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

```
500
527
128
128
```

In [9]:
```
# 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,
    fill_mode='nearest'
)

# 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,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

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

# 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 [10]:
```
# 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,
    epochs=3,
    validation_data=validation_generator,
    verbose=1,
    callbacks=[callbacks]
)
```

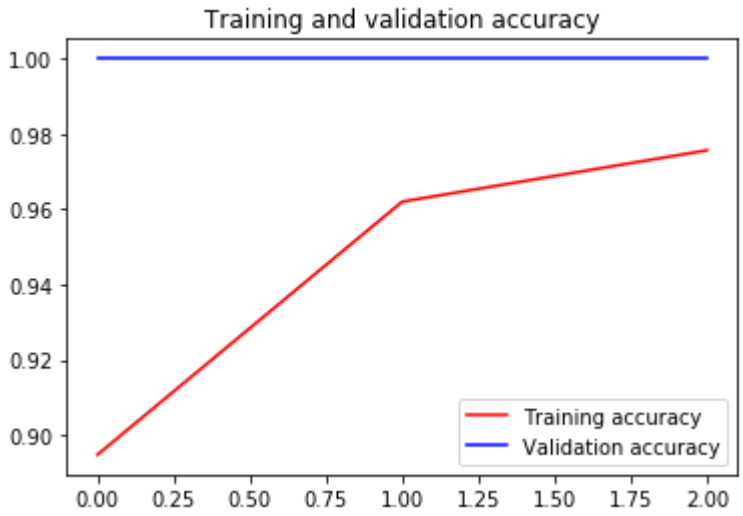
```
Epoch 1/3
52/52 [==============================] - 52s 1s/step - loss: 0.2500 - acc: 0.89
48 - val_loss: 0.0061 - val_acc: 1.0000
Epoch 2/3
52/52 [==============================] - 44s 850ms/step - loss: 0.1073 - acc:
0.9620 - val_loss: 4.7177e-04 - val_acc: 1.0000
Epoch 3/3
51/52 [============================>.] - ETA: 0s - loss: 0.0685 - acc: 0.9752
Reached 97.0% accuracy so cancelling training!
52/52 [==============================] - 46s 879ms/step - loss: 0.0672 - acc:
0.9757 - val_loss: 1.1070e-04 - val_acc: 1.0000
```

In [11]:
```
%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()
```




```
<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);
```