```
import os

from tensorflow.keras import layers
from tensorflow.keras import Model
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/inception_v3_weights_tf_dim_ordering_tf_kerne
    -O /tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5

from tensorflow.keras.applications.inception_v3 import InceptionV3

local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'

pre_trained_model = InceptionV3(input_shape = (150, 150, 3),
                                include_top = False,
                                weights = None)

pre_trained_model.load_weights(local_weights_file)

for layer in pre_trained_model.layers:
  layer.trainable = False

# pre_trained_model.summary()

last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape: ', last_layer.output_shape)
last_output = last_layer.output


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 = ['accuracy'])



!wget --no-check-certificate \
```

```
                https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip  \
               -O  /tmp/cats_and_dogs_filtered.zip

from  tensorflow.keras.preprocessing.image  import  ImageDataGenerator

import  os
import  zipfile

local_zip  =  '//tmp/cats_and_dogs_filtered.zip'

zip_ref  =  zipfile.ZipFile(local_zip,  'r')

zip_ref.extractall('/tmp')
zip_ref.close()

#  Define  our  example  directories  and  files
base_dir  =  '/tmp/cats_and_dogs_filtered'

train_dir  =  os.path.join(  base_dir,  'train')
validation_dir  =  os.path.join(  base_dir,  'validation')


train_cats_dir  =  os.path.join(train_dir,  'cats')  #  Directory  with  our  training  cat  pictu
train_dogs_dir  =  os.path.join(train_dir,  'dogs')  #  Directory  with  our  training  dog  pictu
validation_cats_dir  =  os.path.join(validation_dir,  'cats')  #  Directory  with  our  validation
validation_dogs_dir  =  os.path.join(validation_dir,  'dogs')#  Directory  with  our  validation  c

train_cat_fnames  =  os.listdir(train_cats_dir)
train_dog_fnames  =  os.listdir(train_dogs_dir)

#  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  =  1.0/255.  )

#  Flow  training  images  in  batches  of  20  using  train_datagen  generator
train_generator  =  train_datagen.flow_from_directory(train_dir,




#  Flow  validation  images  in  batches  of  20  using  test_datagen  generator
validation_generator  =    test_datagen.flow_from_directory(  validation_dir,
```

```
history = model.fit(
                        train_generator,
                        validation_data = validation_generator,
                        steps_per_epoch = 100,
                        epochs = 20,
                        validation_steps = 50,
                        verbose = 2)


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()
```