Transfer learning - Cats vs. Dogs

Nuova sezione

Unzip data

```
# !unzip train.zip -d train/
# !unzip test.zip -d test/
```

Code to initiliaze Tensorflow 2.0 in Colab

```
from __future__ import absolute_import, division, print_function, unicode_literals
%tensorflow_version 2.x
import tensorflow as tf
%load_ext tensorboard
import datetime
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

[TODO] Create a data loader function that returns a tuple with a tf.float32 tensor for the image and a label. Images must be resized to 128x128. N.B.: filenames are formatted as class.number.jpg

```
def load_and_preprocess_image(filename):  # load images
    image = tf.io.read_file(filename)   # read the raw data from the file as a string
    image = tf.image.decode_jpeg(image, channels=3) # decode the jpeg image to a tensor

    image = tf.image.resize(image, [128, 128])  # resize the image to 128x128
    image = tf.cast(image, tf.float32) / 255.0  # transform the image to a tf.float32 type and normalize it to [0, 1]
    return image

def parse_filename(filename): # load labels
    label = tf.strings.split(filename, sep='/') # split the filename by '/' (label[0]: 'train', label[1]: 'class.number.jpg')
    label = tf.strings.split(label[-1], sep='.')    # split the last element of the filename by '.' (label[0]: 'class', label[1]: 'number', label[2]: 'jpg')
    label = tf.strings.to_number(label[0], out_type=tf.int32)   # convert the label[0] to a tf.int32 type
    return label
```

[TODO] Create a tf.Dataset, map the loader function and prepare a batch object for training

```
trainDataSet = tf.data.Dataset.list_files('train/*')    # create a dataset from the filenames (filename: train/class.number.jpg)
testDataSet = tf.data.Dataset.list_files('test/*')  # create a dataset from the filenames (filename: test/class.number.jpg)
trainDataSet = trainDataSet.map(lambda x: (load_and_preprocess_image(x), parse_filename(x)))    # for each filename, load the image and the label
testDataSet = testDataSet.map(lambda x: (load_and_preprocess_image(x), parse_filename(x)))  # for each filename, load the image and the label
```

Prepare Keras callback for Tensorboard

```
# logdir = "logs/scalars/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
# %tensorboard --logdir logs
# tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir, update_freq='batch')
```

[TODO] Import the MobileNetV2 model trained on ImageNet without the final layer

```
# because build a deep learning model can be too complex, we can consider to use a pre-trained model for performing feature extraction, and then add the final layer to perfo
# classification as we want

# import the MobileNetV2 model, input_shape is the shape of the images. They have 3 channels cause they are RGB images
# include_top=False means that we exclude the last fully connected layer of the model
# weights='imagenet' means that we initialize the model with pre-trained weights on ImageNet
base_model = tf.keras.applications.MobileNetV2(input_shape=(128, 128, 3), include_top=False, weights='imagenet')    # load the MobileNetV2 model7
base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_128_no_top.h5
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

[TODO] Add a final classification layer for 2 classes and create the final Keras model

```
# We know that the final layer, the one that performs the classification through the softmax function, need that the input is a vector. This is why, as seen in the previous 
# have performed a flattening operation on the input. The problem of flattening is that than the classification layer depends on the size of the input (HxW), so if the model
# with different shapes then the classification will not work. This is why in this case we use the GlobalAveragePooling2D layer, cause we want that our model could work with
# different shapes. What GlobalAvaragePooling2D does is to evaluate the mean value of each image feature map and build a F dimensional vector where each value rappresent the
# the image on that channel.
x = base_model.output   # get the output of the model, on it we will add the final layers
x = tf.keras.layers.GlobalAveragePooling2D()(x) # evaluate the mean value of each image feature map. So we receive an input of shape (HxWxF) and we return an output of shape
y = tf.keras.layers.Dense(2, activation='softmax')(x)  # build a pdf with the two possible classes, dog and cat
model = tf.keras.Model(inputs=base_model.input, outputs=y)    # create the final model
```

[TODO] Compile the Keras model: specify the optimization algorithm, the loss function and the test metric

```
lr = 0.01   # learning rate
model.compile(optimizer = tf.keras.optimizers.Adam(lr), loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
```

[TODO] Train the Keras model

```
model.fit(trainDataSet.batch(32), epochs=5)    # train the model
```

```
Epoch 1/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 15s 78ms/step - accuracy: 0.8386 - loss: 0.6377
Epoch 2/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.9694 - loss: 0.0894
Epoch 3/5
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 49ms/step - accuracy: 0.9721 - loss: 0.0838
Epoch 4/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.9756 - loss: 0.0627
Epoch 5/5
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.9947 - loss: 0.0183
<keras.src.callbacks.history.History at 0x7f87b49d1bd0>
```

[TODO] Print model summary

```
# model.summary()
```

[TODO] Test the Keras model by computing the accuracy the whole test set

```
model.evaluate(trainDataSet.batch(32))    # evaluate the model on the training set
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 83ms/step - accuracy: 0.9837 - loss: 0.0314
[0.03328186646103859, 0.984499990940094]
```

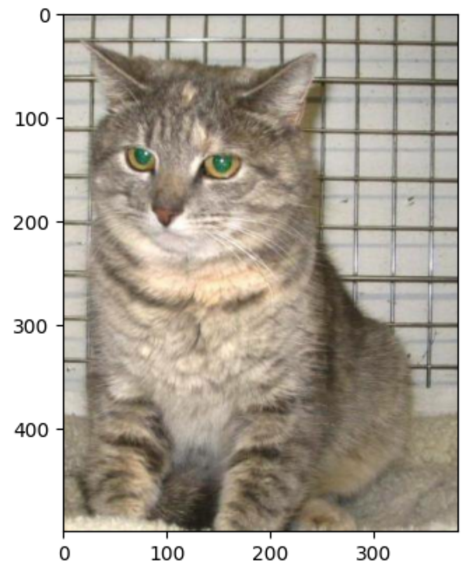
[TODO] Load Test image 'test/0.1047.jpg', visualize it and check the network prediction

```
# plt.imshow(x_test[47].reshape(28, 28), cmap='gray')
# plt.title(f'Label: {y_test[47]}')
# plt.show()

# y_pred = model.predict(x_test[47][np.newaxis, :, :, :])
# print(f'Predicted label: {np.argmax(y_pred)}')

showImage = tf.io.read_file('test/0.1047.jpg')   # read the raw data from the file as a string
showImage = tf.image.decode_jpeg(showImage)  # decode the jpeg image to a tensor
plt.imshow(showImage)   # show the image
plt.show()

# prepare the image for the model
showImage = tf.image.resize(showImage, [128, 128])  # resize the image to 128x128
showImage = tf.cast(showImage, tf.float32) / 255.0  # transform the image to a tf.float32 type and normalize it to [0, 1]
showImage = showImage[tf.newaxis, :, :, :]  # add a batch dimension
predict = model.predict(showImage)  # predict the label of the image
print(f'Predicted label: {np.argmax(predict)}')    # print the predicted label
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 7s 7s/step
Predicted label: 0
```