hy-23 Created using Colaboratory History

1 contributor

1.01 MB

Open in Colab

In [1]:
```
import tensorflow as tf
from tensorflow.data import AUTOTUNE
```

In [2]:
```
import pathlib
data_root_orig = tf.keras.utils.get_file(origin='https://storage.googleapis.com/download.tensorflow.org/
                                         fname='flower_photos', untar=True)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photo
s.tgz
228818944/228813984 [==============================] - 2s 0us/step
228827136/228813984 [==============================] - 2s 0us/step
```

In [3]:
```
# data_root_orig: string: /root/.keras/datasets/flower_photos

# create a pathlib.PosixPath object from the string.
data_root = pathlib.Path(data_root_orig)

# root directory: /root/.keras/datasets/flower_photos.
```

In [4]:
```
# check all the available directories in your root directory.
for folder in data_root.iterdir():
  print(folder)
```

```
/root/.keras/datasets/flower_photos/tulips
/root/.keras/datasets/flower_photos/daisy
/root/.keras/datasets/flower_photos/dandelion
/root/.keras/datasets/flower_photos/LICENSE.txt
/root/.keras/datasets/flower_photos/sunflowers
/root/.keras/datasets/flower_photos/roses
```

In [5]:
```
import random

# iterate over the subtree "data_root" and yield all files that matches the pattern.
# i.e., [*/][*]
# [*/]:= select_anything_and_everything_in_root
# [*] := select_anything_and_everything_in_this
all_image_paths = list(data_root.glob('*/*'))
all_image_paths = [str(path) for path in all_image_paths]
random.shuffle(all_image_paths)

image_count = len(all_image_paths)
print("Number of files: {}".format(image_count))
```

```
Number of files: 3670
```

In [6]:
```
all_image_paths[:10]
```

Out[6]:
```
['/root/.keras/datasets/flower_photos/sunflowers/18237156988_9ceb46a8de_n.jpg',
 '/root/.keras/datasets/flower_photos/dandelion/4632757134_40156d7d5b.jpg',
 '/root/.keras/datasets/flower_photos/dandelion/7280221020_98b473b20d_n.jpg',
 '/root/.keras/datasets/flower_photos/tulips/14090534565_5857ce4b7c_n.jpg',
 '/root/.keras/datasets/flower_photos/roses/14221192676_eb8c89a7d6_n.jpg',
 '/root/.keras/datasets/flower_photos/sunflowers/15026703621_e15e9d55f0_n.jpg',
 '/root/.keras/datasets/flower_photos/roses/8523394349_61b31fdd8f_m.jpg',
 '/root/.keras/datasets/flower_photos/dandelion/14313509432_6f2343d6c8_m.jpg',
 '/root/.keras/datasets/flower_photos/tulips/3626132563_d955973447_n.jpg',
 '/root/.keras/datasets/flower_photos/daisy/515112668_a49c69455a.jpg']
```

Create Caption

In [7]:
```
# read the License.txt file in the root directory.

attributions = (data_root/"LICENSE.txt").open(encoding='utf-8').readlines()[4:]
# show first 3 lines.
attributions[:3]
```

Out[7]:
```
['daisy/7568630428_8cf0fc16ff_n.jpg CC-BY by A Guy Taking Pictures - https://www.flickr.com/photos/809013
81@N04/7568630428/\n',
 'daisy/7410356270_9dff4d0e2e_n.jpg CC-BY by martinak15 - https://www.flickr.com/photos/martinaphotograph
y/7410356270/\n',
 'daisy/4286053334_a75541f20b_m.jpg CC-BY by jenny downing - https://www.flickr.com/photos/jenny-pics/428
6053334/\n']
```

In [8]:
```
attributions = [line.split(' CC-BY') for line in attributions]
attributions[:3]
```

Out[8]:
```
[['daisy/7568630428_8cf0fc16ff_n.jpg',
  ' by A Guy Taking Pictures - https://www.flickr.com/photos/80901381@N04/7568630428/\n'],
 ['daisy/7410356270_9dff4d0e2e_n.jpg',
  ' by martinak15 - https://www.flickr.com/photos/martinaphotography/7410356270/\n'],
 ['daisy/4286053334_a75541f20b_m.jpg',
  ' by jenny downing - https://www.flickr.com/photos/jenny-pics/4286053334/\n']]
```

In [9]:
```
attributions = dict(attributions)
for key in attributions:
  print("key  : " , key)
  print("value:", attributions[key])
  break
```

```
key  :  daisy/7568630428_8cf0fc16ff_n.jpg
value:  by A Guy Taking Pictures - https://www.flickr.com/photos/80901381@N04/7568630428/
```

In [10]:
```
import IPython.display as display

def caption_image(image_path):
  # image_path := string
  # := /root/.keras/datasets/flower_photos/daisy/5512287917_9f5d3f0f98_n.jpg
  # data_root  := PosixPath
  # := /root/.keras/datasets/flower_photos
  image_rel = pathlib.Path(image_path).relative_to(data_root)
  # image_rel  := PosixPath
  # := daisy/5512287917_9f5d3f0f98_n.jpg

  # return string
  # := Image (CC BY 2.0)  by Vicente Villamón

  # attribution key  :  daisy/7568630428_8cf0fc16ff_n.jpg
  # attribution value:  by A Guy Taking Pictures - https://www.flickr.com/photos/80901381@N04/7568630428/
  return "Image (CC BY 2.0) " + ' - '.join(attributions[str(image_rel)].split(' - ')[:-1])
```

In [11]:
```
for n in range(3):
  image_path = random.choice(all_image_paths)
  display.display(display.Image(image_path))
  print(caption_image(image_path))
  print()
```

```
Image (CC BY 2.0)  by Susanne Nilsson
```

Image (CC BY 2.0) by liz west

Image (CC BY 2.0) by jnyemb

Prepare label for each image.

In [12]:
```
# select anything and everything in the root directory.
all = list(data_root.glob('*/'))
all
```

Out[12]:
```
[PosixPath('/root/.keras/datasets/flower_photos/tulips'),
 PosixPath('/root/.keras/datasets/flower_photos/daisy'),
 PosixPath('/root/.keras/datasets/flower_photos/dandelion'),
 PosixPath('/root/.keras/datasets/flower_photos/LICENSE.txt'),
 PosixPath('/root/.keras/datasets/flower_photos/sunflowers'),
 PosixPath('/root/.keras/datasets/flower_photos/roses')]
```

In [13]:
```
for item in data_root.glob('*/'):
  if item.is_dir():
    print(item.name)
```

```
tulips
daisy
dandelion
sunflowers
roses
```

In [14]:
```
label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())
print(label_names)
dict_label2idx = dict((name, index) for index, name in enumerate(label_names))
dict_label2idx
```

```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

Out[14]:
```
{'daisy': 0, 'dandelion': 1, 'roses': 2, 'sunflowers': 3, 'tulips': 4}
```

In [15]:
```
# path := /root/.keras/datasets/flower_photos/tulips/2271507463_15c48d41c4_n.jpg
# parent.name := tulips
all_image_labels = [dict_label2idx[pathlib.Path(path).parent.name]
                    for path in all_image_paths]
```

In [16]:
```
print("First 10 labels indices: ", all_image_labels[:10])
print("First 10 image paths: ", all_image_paths[:10])
```

```
First 10 labels indices:  [3, 1, 1, 4, 2, 3, 2, 1, 4, 0]
First 10 image paths:  ['/root/.keras/datasets/flower_photos/sunflowers/18237156988_9ceb46a8de_n.jpg', '/root/.keras/datasets/flower_photos/dandelion/4632757134_40156d7d5b.jpg', '/root/.keras/datasets/flower_photos/dandelion/7280221020_98b473b20d_n.jpg', '/root/.keras/datasets/flower_photos/tulips/14090534565_5857ce4b7c_n.jpg', '/root/.keras/datasets/flower_photos/roses/14221192676_eb8c89a7d6_n.jpg', '/root/.keras/datasets/flower_photos/sunflowers/15026703621_e15e9d55f0_n.jpg', '/root/.keras/datasets/flower_photos/roses/8523394349_61b31fdd8f_m.jpg', '/root/.keras/datasets/flower_photos/dandelion/14313509432_6f2343d6c8_m.jpg', '/root/.keras/datasets/flower_photos/tulips/3626132563_d955973447_n.jpg', '/root/.keras/datasets/flower_photos/daisy/515112668_a49c69455a.jpg']
```

Prepare tf.data now

Prepare tf.data flow

Before that see what the process is.

In [17]:
```
img_path = all_image_paths[0]
print("image path:", img_path)

# read the file
img_raw = tf.io.read_file(img_path)
# img_raw is tensorflow.python.framework.ops.EagerTensor

# Decode it.
image = tf.image.decode_jpeg(img_raw, channels=3)

# Reason for decoding.
# dtype of img_raw is string, whereas dtype of decoded image is uint8.
print("\nType of img_raw : ", type(img_raw))
print("Type of image   : ", type(image))
print("\nBut.,\n")
print("dtype of img_raw: ", img_raw.dtype)
print("dtype of image  : ", image.dtype)
```

```
image path: /root/.keras/datasets/flower_photos/sunflowers/18237156988_9ceb46a8de_n.jpg

Type of img_raw :  <class 'tensorflow.python.framework.ops.EagerTensor'>
Type of image   :  <class 'tensorflow.python.framework.ops.EagerTensor'>

But.,

dtype of img_raw:  <dtype: 'string'>
dtype of image  :  <dtype: 'uint8'>
```

Do the processing.

In [18]:
```
def preprocess_image(image):
  # image: is output of tf.io.read('image_path')
  # := 'tensorflow.python.framework.ops.EagerTensor'
  image = tf.image.decode_jpeg(image, channels=3)
  image = tf.image.resize(image, [192, 192])
  image /= 255.0  # normalize to [0,1] range
  return image

def load_and_preprocess_image(path):
  image = tf.io.read_file(path)
  return preprocess_image(image)
```

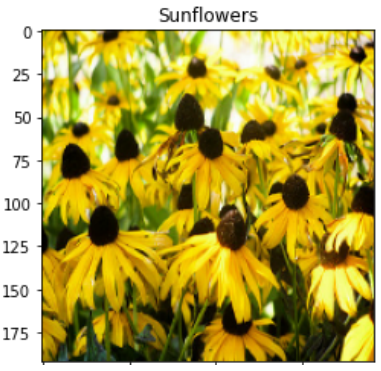
In [19]:
```
label_names
```

Out[19]:
```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

In [20]:
```
import matplotlib.pyplot as plt

# get an image path from the list of paths.
image_path = all_image_paths[0]
# get the corresponding label.
label = all_image_labels[0]

plt.imshow(load_and_preprocess_image(img_path))
plt.grid(False)
plt.xlabel(caption_image(img_path))
plt.title(label_names[label].title())
# label_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
print()
```



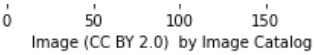


Image (CC BY 2.0) by Image Catalog

Build tf.data.Dataset

Prepare dataset for images.

The easiest way to build a tf.data.Dataset is using `from_tensor_slices` method.

1. Slicing the array of strings `all_image_paths`, results in dataset of strings.

In [21]:
```
all_image_paths_ds = tf.data.Dataset.from_tensor_slices(all_image_paths)
print(all_image_paths_ds)
```

```
<TensorSliceDataset element_spec=TensorSpec(shape=(), dtype=tf.string, name=None)>
```

Now create a new dataset that loads and formats images on the fly by mapping preprocess_image over the dataset of paths.

In [22]:
```
images_ds = all_image_paths_ds.map(load_and_preprocess_image, num_parallel_calls=AUTOTUNE)
# from here on all_image_paths_ds is not needed anymore.
```

In [23]:
```
for image in images_ds.take(1):
  print(type(images_ds))
  print(type(image))
```

```
<class 'tensorflow.python.data.ops.dataset_ops.ParallelMapDataset'>
<class 'tensorflow.python.framework.ops.EagerTensor'>
```

In [24]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,8))
# take 4 first 4 EagerTensors from the dataset.
for n, image in enumerate(images_ds.take(4)):
  plt.subplot(2,2,n+1)
  plt.imshow(image)
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  plt.xlabel(caption_image(all_image_paths[n]))
  plt.show()
```


Image (CC BY 2.0) by Image Catalog


Image (CC BY 2.0) by cloud2013


Image (CC BY 2.0) by Paul Hudson

Image (CC BY 2.0) by Blondinrikard Fröberg

Prepare dataset for labels

In [25]:
```
all_image_labels_t64 = tf.cast(all_image_labels, tf.int64)
labels_ds = tf.data.Dataset.from_tensor_slices(all_image_labels_t64)

# seems like no need to decode this dataset.
# or, .numpy() itself is decoding.
for label in labels_ds.take(10):
  print(label.numpy())
  print(label_names[label.numpy()])
```

```
3
sunflowers
1
dandelion
1
dandelion
4
tulips
2
roses
3
sunflowers
2
roses
1
dandelion
4
tulips
0
daisy
```

In [26]:
```
# Since the datasets are in the same order you can just zip them together to get
# a dataset of (image, label) pairs:

image_label_ds = tf.data.Dataset.zip((images_ds, labels_ds))
```

In [27]:
```
print(image_label_ds)
```

```
<ZipDataset element_spec=(TensorSpec(shape=(192, 192, 3), dtype=tf.float32, name=None), TensorSpec(shape=
(), dtype=tf.int64, name=None))>
```

Alternatively, you can club both.

In [28]:
```
def load_and_preprocess_from_path_label(path, label):
  return load_and_preprocess_image(path), label
```

In [29]:
```
# tuples -> dataset of tuples.
ds = tf.data.Dataset.from_tensor_slices((all_image_paths, all_image_labels))
```

In [30]:
```
image_label_ds = ds.map(load_and_preprocess_from_path_label)
image_label_ds
```

Out[30]:
```
<MapDataset element_spec=(TensorSpec(shape=(192, 192, 3), dtype=tf.float32, name=None), TensorSpec(shape=
(), dtype=tf.int32, name=None))>
```

In [31]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,8))
# take 4 first 4 EagerTensors from the dataset.
for n, (image, label) in enumerate(image_label_ds.take(4)):
  plt.subplot(2,2,n+1)
  plt.imshow(image)
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  plt.xlabel(caption_image(all_image_paths[n]))
  plt.title(label_names[label.numpy()])
  plt.show()
```

sunflowers

Image (CC BY 2.0) by Image Catalog

dandelion

Image (CC BY 2.0) by cloud2013

dandelion

Image (CC BY 2.0) by Paul Hudson

tulips

Image (CC BY 2.0) by Blondinrikard Fröberg

Train the model

To train a model with this dataset you will want the data:

1. To be well shuffled.
2. To be batched.
3. To repeat forever.
4. Batches to be available as soon as possible.

In [32]:

```
ds = image_label_ds.shuffle(buffer_size=image_count)
ds = ds.repeat()
ds = ds.batch(batch_size=64)
ds = ds.cache()
ds = ds.prefetch(buffer_size=AUTOTUNE)
ds
```

Out[32]:

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 192, 192, 3), dtype=tf.float32, name=None), Tensor
Spec(shape=(None,), dtype=tf.int32, name=None))>
```

There are a few things to note here:

1. The order is important.
   - .shuffle after a .repeat would shuffle items across epoch boundaries (some items will be seen twice before others are seen at all).
   - .shuffle after a .batch would shuffle the order of the batches, but not shuffle the items across batches.
2. You use a buffer_size the same size as the dataset for a full shuffle. Up to the dataset size, large values provide better randomization, but use more memory.
3. The shuffle buffer is filled before any elements are pulled from it. So a large buffer_size may cause a delay when your Dataset is starting.
4. The shuffeled dataset doesn't report the end of a dataset until the shuffle-buffer is completely empty. The Dataset

is restarted by .repeat, causing another wait for the shuffle-buffer to be filled.

Pipe the dataset to a model

In [33]:
```
mobile_net = tf.keras.applications.MobileNetV2(input_shape=(192, 192, 3), include_top=False)
mobile_net.trainable=False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet
_v2_weights_tf_dim_ordering_tf_kernels_1.0_192_no_top.h5
9412608/9406464 [==============================] - 0s 0us/step
9420800/9406464 [==============================] - 0s 0us/step
```

In [34]:
```
help(tf.keras.applications.mobilenet_v2.preprocess_input)
```

```
Help on function preprocess_input in module keras.applications.mobilenet_v2:

preprocess_input(x, data_format=None)
    Preprocesses a tensor or Numpy array encoding a batch of images.

    Usage example with `applications.MobileNet`:

    ```python
    i = tf.keras.layers.Input([None, None, 3], dtype = tf.uint8)
    x = tf.cast(i, tf.float32)
    x = tf.keras.applications.mobilenet.preprocess_input(x)
    core = tf.keras.applications.MobileNet()
    x = core(x)
    model = tf.keras.Model(inputs=[i], outputs=[x])

    image = tf.image.decode_png(tf.io.read_file('file.png'))
    result = model(image)
    ```

    Args:
      x: A floating point `numpy.array` or a `tf.Tensor`, 3D or 4D with 3 color
        channels, with values in the range [0, 255].
        The preprocessed data are written over the input data
        if the data types are compatible. To avoid this
        behaviour, `numpy.copy(x)` can be used.
      data_format: Optional data format of the image tensor/array. Defaults to
        None, in which case the global setting
        `tf.keras.backend.image_data_format()` is used (unless you changed it,
        it defaults to "channels_last").

    Returns:
        Preprocessed `numpy.array` or a `tf.Tensor` with type `float32`.

        The inputs pixel values are scaled between -1 and 1, sample-wise.

    Raises:

      ValueError: In case of unknown `data_format` argument.
```

In [35]:
```
# range in our images.
im_path = all_image_paths[0]
im_raw  = tf.io.read_file(im_path)
im      = tf.image.decode_jpeg(im_raw, channels=3)
print(im.numpy().min())
print(im.numpy().max())
```

```
0
255
```

In [36]:
```
# range is [0, 255] -> and we already changed it to [0, 1].
# Before you pass the input to the MobilNet model, you need to convert it from
# a range of [0,1] to [-1,1]:

def change_range(image, label):
  return 2*image-1, label

mobilenet_ds = ds.map(change_range)
```

In [37]:
```
# The dataset may take a few seconds to start, as it fills its shuffle buffer.
image_batch, label_batch = next(iter(mobilenet_ds))
```

In [38]:

In [38]:
```
image_batch.shape
```

Out[38]:
```
TensorShape([64, 192, 192, 3])
```

In [39]:
```
feature_map_batch = mobile_net(image_batch)
print(feature_map_batch.shape)
```

```
(64, 6, 6, 1280)
```

Build a model wrapped around MobileNet and use tf.keras.layers.GlobalAveragePooling2D to average over those space dimensions before the output tf.keras.layers.Dense layer:

In [40]:
```
model = tf.keras.models.Sequential([mobile_net,
                                    tf.keras.layers.GlobalAveragePooling2D(),
                                    tf.keras.layers.Dense(len(label_names),
                                                          activation = 'softmax')])
```

In [41]:
```
logit_batch = model(image_batch).numpy()

print("min logit:", logit_batch.min())
print("max logit:", logit_batch.max())
print()

print("Shape:", logit_batch.shape)
```

```
min logit: 0.0059083644
max logit: 0.7955431

Shape: (64, 5)
```

In [42]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_192 (Funct  (None, 6, 6, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 5)                 6405

=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [43]:
```
# There are 2 trainable variables - the Dense weights and bias.
len(model.trainable_variables)
```

Out[43]:
```
2
```

In [44]:
```
# steps_per_epoch: Integer. Total number of steps (batches of samples) to yield
# from generator before declaring one epoch finished and starting the next epoch.
# It should typically be equal to ceil(num_samples / batch_size).
# Optional for Sequence: if unspecified, will use the len(generator) as a number
# of steps.
BATCH_SIZE = 64
steps_per_epoch=tf.math.ceil(len(all_image_paths)/BATCH_SIZE).numpy()
steps_per_epoch
```

Out[44]:
```
58.0
```

In [45]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(),
```