Adding a Dataset of Your Own to TFDS

In [1]:

```
import os
import textwrap
import scipy.io
import pandas as pd

from os import getcwd
```

IMDB Faces Dataset

This is the largest publicly available dataset of face images with gender and age labels for training.

Source: https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/

The IMDb Faces dataset provides a separate .mat file which can be loaded with Matlab containing all the meta information. The format is as follows:
dob: date of birth (Matlab serial date number)
photo_taken: year when the photo was taken
full_path: path to file
gender: 0 for female and 1 for male, NaN if unknown
name: name of the celebrity
face_location: location of the face (bounding box)
face_score: detector score (the higher the better). Inf implies that no face was found in the image and the face_location then just returns the entire image
second_face_score: detector score of the face with the second highest score. This is useful to ignore images with more than one face. second_face_score is NaN if no second face was detected.
celeb_names: list of all celebrity names
celeb_id: index of celebrity name

Next, let's inspect the dataset

Exploring the Data

In [2]:

```
# Inspect the directory structure
imdb_crop_file_path = f"{getcwd()}/../tmp2/imdb_crop"
files = os.listdir(imdb_crop_file_path)
print(textwrap.fill(' '.join(sorted(files)), 80))
```

```
imdb.mat
```

In [3]:

```
# Inspect the meta data
imdb_mat_file_path = f"{getcwd()}/../tmp2/imdb_crop/imdb.mat"
meta = scipy.io.loadmat(imdb_mat_file_path)
```

In [4]:

```
meta
```

Out[4]:

```
{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Sun Jan 17 11:30:27 2016',
 '__version__': '1.0',
 '__globals__': [],
 'imdb': array([[(array([[693726, 693726, 693726, ..., 726831, 726831, 726831]], dtype=int32),
array([[1968, 1970, 1968, ..., 2011, 2011, 2011]], dtype=uint16),
array([[array(['01/nm0000001_rm124825600_1899-5-10_1968.jpg'], dtype='<U43'),
         array(['01/nm0000001_rm3343756032_1899-5-10_1970.jpg'], dtype='<U44'),
         array(['01/nm0000001_rm577153792_1899-5-10_1968.jpg'], dtype='<U43'),
         ...,
         array(['08/nm3994408_rm926592512_1989-12-29_2011.jpg'], dtype='<U44'),
         array(['08/nm3994408_rm943369728_1989-12-29_2011.jpg'], dtype='<U44'),
         array(['08/nm3994408_rm976924160_1989-12-29_2011.jpg'], dtype='<U44')]],
       dtype=object), array([[1., 1., 1., ..., 0., 0., 0.]]), array([[array(['Fred Astaire'],
dtype='<U12'),
         array(['Fred Astaire'], dtype='<U12'),
         array(['Fred Astaire'], dtype='<U12'), ...,
         array(['Jane Levy'], dtype='<U9'),
         array(['Jane Levy'], dtype='<U9'),
         array(['Jane Levy'], dtype='<U9')]], dtype=object), array([[array([[1072.926,  161.838, 12
14.784,  303.696]]),
         array([[477.184, 100.352, 622.592, 245.76 ]]),
         array([[114.96964309, 114.96964309, 451.68657236, 451.68657236]]),
         ..., array([[  1,   1, 453, 640]], dtype=uint16),
         array([[144.75225472, 126.76472288, 305.78804127, 287.80050943]]),
         array([[457.524,  41.748, 518.016, 102.24 ]])]], dtype=object), array([[1.45969291,
2.5431976 , 3.45557949, ...,        -inf, 4.45072452,
         2.13350269]]), array([[1.11897336, 1.85200773, 2.98566022, ...,         nan,         nan,
                nan]]), array([[array(["'Lee' George Quinones"], dtype='<U21'),
         array(["'Weird Al' Yankovic"], dtype='<U19'),
         array(['2 Chainz'], dtype='<U8'), ...,
         array(['Éric Caravaca'], dtype='<U13'),
         array(['Ólafur Darri Ólafsson'], dtype='<U21'),
         array(['Óscar Jaenada'], dtype='<U13')]], dtype=object), array([[6488, 6488, 6488, ..., 84
10, 8410, 8410]], dtype=uint16))]],
       dtype=[('dob', 'O'), ('photo_taken', 'O'), ('full_path', 'O'), ('gender', 'O'), ('name', 'O'
), ('face_location', 'O'), ('face_score', 'O'), ('second_face_score', 'O'), ('celeb_names', 'O'),
('celeb_id', 'O')])}
```

Extraction

Let's clear up the clutter by going to the metadata's most useful key (imdb) and start exploring all the other keys inside it

In [5]:

```
root = meta['imdb'][0, 0]
```

In [6]:

```
desc = root.dtype.descr
desc
```

Out[6]:

```
[('dob', '|O'),
 ('photo_taken', '|O'),
 ('full_path', '|O'),
 ('gender', '|O'),
 ('name', '|O'),
 ('face_location', '|O'),
 ('face_score', '|O'),
 ('second_face_score', '|O'),
 ('celeb_names', '|O'),
 ('celeb_id', '|O')]
```

In [7]:

```
# EXERCISE: Fill in the missing code below.

full_path = root["full_path"][0]

# Do the same for other attributes
names = root["name"][0]
dob = root["dob"][0]
gender = root["gender"][0]
photo_taken = root["photo_taken"][0]
face_score = root["face_score"][0]
face_locations = root["face_location"][0]
face_locations = root["face_location"][0]
second_face_score = root["second_face_score"][0]
celeb_names = root["celeb_names"][0]
celeb_ids = root["celeb_id"][0]

print('Filepaths: {}\n\n'
      'Names: {}\n\n'
      'Dates of birth: {}\n\n'
      'Genders: {}\n\n'
      'Years when the photos were taken: {}\n\n'
      'Face scores: {}\n\n'
      'Face locations: {}\n\n'
      'Second face scores: {}\n\n'
      'Celeb IDs: {}\n\n'
      .format(full_path, names, dob, gender, photo_taken, face_score, face_locations,
second_face_score, celeb_ids))
```

```
Filepaths: [array(['01/nm0000001_rm124825600_1899-5-10_1968.jpg'], dtype='<U43')
 array(['01/nm0000001_rm3343756032_1899-5-10_1970.jpg'], dtype='<U44')
 array(['01/nm0000001_rm577153792_1899-5-10_1968.jpg'], dtype='<U43') ...
 array(['08/nm3994408_rm926592512_1989-12-29_2011.jpg'], dtype='<U44')
 array(['08/nm3994408_rm943369728_1989-12-29_2011.jpg'], dtype='<U44')
 array(['08/nm3994408_rm976924160_1989-12-29_2011.jpg'], dtype='<U44')]

Names: [array(['Fred Astaire'], dtype='<U12')
 array(['Fred Astaire'], dtype='<U12')
 array(['Fred Astaire'], dtype='<U12') ...
 array(['Jane Levy'], dtype='<U9') array(['Jane Levy'], dtype='<U9')
 array(['Jane Levy'], dtype='<U9')]

Dates of birth: [693726 693726 693726 ... 726831 726831 726831]

Genders: [1. 1. 1. ... 0. 0. 0.]

Years when the photos were taken: [1968 1970 1968 ... 2011 2011 2011]

Face scores: [1.45969291 2.5431976  3.45557949 ...       -inf 4.45072452 2.13350269]

Face locations: [array([[1072.926,  161.838, 1214.784,  303.696]])
 array([[477.184, 100.352, 622.592, 245.76 ]])
 array([[114.96964309, 114.96964309, 451.68657236, 451.68657236]]) ...
 array([[  1,   1, 453, 640]], dtype=uint16)
 array([[144.75225472, 126.76472288, 305.78804127, 287.80050943]])
 array([[457.524,  41.748, 518.016, 102.24 ]])]

Second face scores: [1.11897336 1.85200773 2.98566022 ...        nan        nan        nan]

Celeb IDs: [6488 6488 6488 ... 8410 8410 8410]
```

In [8]:

```
print('Celeb names: {}\n\n'.format(celeb_names))
```

```
Celeb names: [array(["'Lee' George Quinones"], dtype='<U21')
 array(["'Weird Al' Yankovic"], dtype='<U19')
 array(['2 Chainz'], dtype='<U8') ...
 array(['Éric Caravaca'], dtype='<U13')
 array(['Ólafur Darri Ólafsson'], dtype='<U21')
 array(['Óscar Jaenada'], dtype='<U13')]
```

Display all the distinct keys and their corresponding values

In [9]:

```
names = [x[0] for x in desc]
names
```

Out[9]:

```
['dob',
 'photo_taken',
 'full_path',
 'gender',
 'name',
 'face_location',
 'face_score',
 'second_face_score',
 'celeb_names',
 'celeb_id']
```

In [10]:

```
values = {key: root[key][0] for key in names}
values
```

Out[10]:

```
{'dob': array([693726, 693726, 693726, ..., 726831, 726831, 726831], dtype=int32),
 'photo_taken': array([1968, 1970, 1968, ..., 2011, 2011, 2011], dtype=uint16),
 'full_path': array([array(['01/nm0000001_rm124825600_1899-5-10_1968.jpg'], dtype='<U43'),
        array(['01/nm0000001_rm3343756032_1899-5-10_1970.jpg'], dtype='<U44'),
        array(['01/nm0000001_rm577153792_1899-5-10_1968.jpg'], dtype='<U43'),
        ...,
        array(['08/nm3994408_rm926592512_1989-12-29_2011.jpg'], dtype='<U44'),
        array(['08/nm3994408_rm943369728_1989-12-29_2011.jpg'], dtype='<U44'),
        array(['08/nm3994408_rm976924160_1989-12-29_2011.jpg'], dtype='<U44')],
       dtype=object),
 'gender': array([1., 1., 1., ..., 0., 0., 0.]),
 'name': array([array(['Fred Astaire'], dtype='<U12'),
        array(['Fred Astaire'], dtype='<U12'),
        array(['Fred Astaire'], dtype='<U12'), ...,
        array(['Jane Levy'], dtype='<U9'),
        array(['Jane Levy'], dtype='<U9'),
        array(['Jane Levy'], dtype='<U9')], dtype=object),
 'face_location': array([array([[1072.926,  161.838, 1214.784,  303.696]]),
        array([[477.184, 100.352, 622.592, 245.76 ]]),
        array([[114.96964309, 114.96964309, 451.68657236, 451.68657236]]),
        ..., array([[  1,   1, 453, 640]], dtype=uint16),
        array([[144.75225472, 126.76472288, 305.78804127, 287.80050943]]),
        array([[457.524,  41.748, 518.016, 102.24 ]])], dtype=object),
 'face_score': array([1.45969291, 2.5431976 , 3.45557949, ...,       -inf, 4.45072452,
        2.13350269]),
 'second_face_score': array([1.11897336, 1.85200773, 2.98566022, ...,        nan,        nan,
               nan]),
 'celeb_names': array([array(["'Lee' George Quinones"], dtype='<U21'),
        array(["'Weird Al' Yankovic"], dtype='<U19'),
        array(['2 Chainz'], dtype='<U8'), ...,
        array(['Éric Caravaca'], dtype='<U13'),
        array(['Ólafur Darri Ólafsson'], dtype='<U21'),
        array(['Óscar Jaenada'], dtype='<U13')], dtype=object),
 'celeb_id': array([6488, 6488, 6488, ..., 8410, 8410, 8410], dtype=uint16)}
```

Cleanup

Pop out the celeb names as they are not relevant for creating the records.

In [11]:

```
del values['celeb_names']
names.pop(names.index('celeb_names'))
```

Out[11]:

```
'celeb_names'
```

Let's see how many values are present in each key

In [12]:

```
for key, value in values.items():
    print(key, len(value))
```

```
dob 460723
photo_taken 460723
full_path 460723
gender 460723
name 460723
face_location 460723
face_score 460723
second_face_score 460723
celeb_id 460723
```

Dataframe

Now, let's try examining one example from the dataset. To do this, let's load all the attributes that we've extracted just now into a Pandas dataframe

In [13]:

```
df = pd.DataFrame(values, columns=names)
df.head()
```

Out[13]:

	dob	photo_taken	full_path	gender	name	face_location	face_score	second_face_score	ce
0	693726	1968	[01/nm0000001_rm124825600_1899-5-10_1968.jpg]	1.0	[Fred Astaire]	[[1072.926, 161.838, 1214.7839999999999, 303.6...	1.459693	1.118973	
1	693726	1970	[01/nm0000001_rm3343756032_1899-5-10_1970.jpg]	1.0	[Fred Astaire]	[[477.184, 100.352, 622.592, 245.76]]	2.543198	1.852008	
2	693726	1968	[01/nm0000001_rm577153792_1899-5-10_1968.jpg]	1.0	[Fred Astaire]	[[114.96964308962852, 114.96964308962852, 451....	3.455579	2.985660	
3	693726	1968	[01/nm0000001_rm946909184_1899-5-10_1968.jpg]	1.0	[Fred Astaire]	[[622.8855056426588, 424.21750383700805, 844.3...	1.872117	NaN	
4	693726	1968	[01/nm0000001_rm980463616_1899-5-10_1968.jpg]	1.0	[Fred Astaire]	[[1013.8590023603723, 233.8820422075853, 1201....	1.158766	NaN	

The Pandas dataframe may contain some Null values or nan. We will have to filter them later on.

In [14]:

```
df.isna().sum()
```

Out[14]:

```
dob                       0
photo_taken               0
full_path                 0
gender                 8462
name                      0
face_location             0
face_score                0
second_face_score    246926
celeb_id                  0
dtype: int64
```

TensorFlow Datasets

TFDS provides a way to transform all those datasets into a standard format, do the preprocessing necessary to make them ready for a machine learning pipeline, and provides a standard input pipeline using `tf.data`.

To enable this, each dataset implements a subclass of `DatasetBuilder`, which specifies:

- Where the data is coming from (i.e. its URL).
- What the dataset looks like (i.e. its features).
- How the data should be split (e.g. TRAIN and TEST).
- The individual records in the dataset.

The first time a dataset is used, the dataset is downloaded, prepared, and written to disk in a standard format. Subsequent access will read from those pre-processed files directly.

Clone the TFDS Repository

The next step will be to clone the GitHub TFDS Repository. For this particular notebook, we will clone a particular version of the repository. You can clone the repository by running the following command:

```
!git clone https://github.com/tensorflow/datasets.git -b v1.2.0
```

However, for simplicity, we have already cloned this repository for you and placed the files locally. Therefore, there is no need to run the above command if you are running this notebook in Coursera environment.

Next, we set the current working directory to `/datasets/` .

In [15]:

```
cd datasets
```

```
/tf/week5/datasets
```

If you want to contribute to TFDS' repo and add a new dataset, you can use the the following script to help you generate a template of the required python file. To use it, you must first clone the tfds repository and then run the following command:

In [16]:

```
%%bash

python tensorflow_datasets/scripts/create_new_dataset.py \
  --dataset my_dataset \
  --type image
```

```
Dataset generated in /usr/local/lib/python3.6/dist-packages/tensorflow_datasets
You can start with searching TODO(my_dataset).
Please check this `https://github.com/tensorflow/datasets/blob/master/docs/add_dataset.md`for
details.
```

If you wish to see the template generated by the `create_new_dataset.py` file, navigate to the folder indicated in the above cell output. Then go to the `/image/` folder and look for a file called `my_dataset.py` . Feel free to open the file and inspect it. You will see a template with place holders, indicated with the word `TODO` , where you have to fill in the information.

Now we will use IPython's `%%writefile` in-built magic command to write whatever is in the current cell into a file. To create or overwrite a file you can use:

```
%%writefile filename
```

Let's see an example:

In [17]:

```
%%writefile something.py
x = 10
```

```
Overwriting something.py
```

Now that the file has been written, let's inspect its contents.

In [18]:

```
!cat something.py
```

```
x = 10
```

Define the Dataset with `GeneratorBasedBuilder`

Most datasets subclass `tfds.core.GeneratorBasedBuilder`, which is a subclass of `tfds.core.DatasetBuilder` that simplifies defining a dataset. It works well for datasets that can be generated on a single machine. Its subclasses implement:

- `_info`: builds the DatasetInfo object describing the dataset
- `_split_generators`: downloads the source data and defines the dataset splits
- `_generate_examples`: yields (key, example) tuples in the dataset from the source data

In this exercise, you will use the `GeneratorBasedBuilder`.

EXERCISE: Fill in the missing code below.

In [24]:

```
%%writefile tensorflow_datasets/image/imdb_faces.py

# coding=utf-8
# Copyright 2019 The TensorFlow Datasets Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""IMDB Faces dataset."""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import collections
import os
import re

import tensorflow as tf
import tensorflow_datasets.public_api as tfds

_DESCRIPTION = """\
Since the publicly available face image datasets are often of small to medium size, rarely exceeding tens of thousands of images, and often without age information we decided to collect a large dataset of celebrities. For this purpose, we took the list of the most popular 100,000 actors as listed on the IMDb website and (automatically) crawled from their profiles date of birth, name, gender and all images related to that person. Additionally we crawled all profile images from pages of people from Wikipedia with the same meta information. We removed the images without timestamp (the date when the photo was taken). Assuming that the images with single faces are likely to show the actor and that the timestamp and date of birth are correct, we were able to assign to each such image the biological (real) age. Of course, we can not vouch for the accuracy of the assigned age information. Besides wrong timestamps, many images are stills from movies - movies that can have extended production times. There are 62,328 images from Wikipedia.

"""

_URL = ("https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/")
_DATASET_ROOT_DIR = 'imdb_crop'
_ANNOTATION_FILE = 'imdb.mat'


_CITATION = """\
@article{Rothe-IJCV-2016,
  author = {Rasmus Rothe and Radu Timofte and Luc Van Gool},
  title = {Deep expectation of real and apparent age from a single image without facial landmarks}
,
  journal = {International Journal of Computer Vision (IJCV)},
  year = {2016},
  month = {July},
}
@InProceedings{Rothe-ICCVW-2015,
  author = {Rasmus Rothe and Radu Timofte and Luc Van Gool},
  title = {DEX: Deep EXpectation of apparent age from a single image},
  booktitle = {IEEE International Conference on Computer Vision Workshops (ICCVW)},
  year = {2015},
  month = {December},
}
"""

# Source URL of the IMDB faces dataset
_TARBALL_URL = "https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/static/imdb_crop.tar"

class ImdbFaces(tfds.core.GeneratorBasedBuilder):
    """IMDB Faces dataset."""
    VERSION = tfds.core.Version("0.1.0")

    def _info(self):
        return tfds.core.DatasetInfo(
            builder=self,
            description=_DESCRIPTION,
            # Describe the features of the dataset by following this url
            # https://www.tensorflow.org/datasets/api_docs/python/tfds/features
            features=tfds.features.FeaturesDict({
                "image": tfds.features.Image(),
                "gender": tfds.features.ClassLabel(num_classes=2),
                "dob": tf.int32,
                "photo_taken": tf.int32,
                "face_location": tfds.features.BBoxFeature(),
                "face_score": tf.float32,
                "second_face_score": tf.float32,
                "celeb_id": tf.int32
            }),
            supervised_keys=("image", "gender"),
            urls=[_URL],
            citation=_CITATION)

    def _split_generators(self, dl_manager):
        # Download the dataset and then extract it.
        download_path = dl_manager.download([_TARBALL_URL])
        extracted_path = dl_manager.download_and_extract([_TARBALL_URL])

        # Parsing the mat file which contains the list of train images
        def parse_mat_file(file_name):
            with tf.io.gfile.GFile(file_name, "rb") as f:
                # Add a lazy import for scipy.io and import the loadmat method to
                # load the annotation file
                dataset = tfds.core.lazy_imports.scipy.io.loadmat(file_name)['imdb']
            return dataset

        # Parsing the mat file by using scipy's loadmat method
        # Pass the path to the annotation file using the downloaded/extracted paths above
        meta = parse_mat_file(os.path.join(extracted_path[0], _DATASET_ROOT_DIR, _ANNOTATION_FILE))

        # Get the names of celebrities from the metadata
        celeb_names = meta[0, 0]["celeb_names"][0]

        # Create tuples out of the distinct set of genders and celeb names
        self.info.features['gender'].names = ('Female', 'Male')
        self.info.features['celeb_id'].names = tuple([x[0] for x in celeb_names])

        return [
            tfds.core.SplitGenerator(
                name=tfds.Split.TRAIN,
                gen_kwargs={
                    "image_dir": extracted_path[0],
                    "metadata": meta,
                })
        ]

    def _get_bounding_box_values(self, bbox_annotations, img_width, img_height):
        """Function to get normalized bounding box values.
        Args:
          bbox_annotations: list of bbox values in kitti format
          img_width: image width
          img_height: image height

        Returns:
          Normalized bounding box xmin, ymin, xmax, ymax values
        """

        ymin = bbox_annotations[0] / img_height
        xmin = bbox_annotations[1] / img_width
        ymax = bbox_annotations[2] / img_height
        xmax = bbox_annotations[3] / img_width
        return ymin, xmin, ymax, xmax

    def _get_image_shape(self, image_path):
        image = tf.io.read_file(image_path)
        image = tf.image.decode_image(image, channels=3)
        shape = image.shape[:2]
        return shape

    def _generate_examples(self, image_dir, metadata):
        # Add a lazy import for pandas here (pd)
        pd = tfds.core.lazy_imports.pandas

        # Extract the root dictionary from the metadata so that you can query all the keys inside
it
        root = metadata[0, 0]

        """Extract image names, dobs, genders,
                   face locations,
                   year when the photos were taken,
                   face scores (second face score too),
                   celeb ids
        """
        image_names = root["full_path"][0]
        dobs = root["dob"][0]
        genders = root["gender"][0]
        face_locations = root["face_location"][0]
        photo_taken_years = root["photo_taken"][0]
        face_scores = root["face_score"][0]
        second_face_scores = root["second_face_score"][0]
        celeb_id = root["celeb_id"][0]

        df = pd.DataFrame(
            list(zip(image_names,
                    dobs, genders,
                    face_locations,
                    photo_taken_years,
                    face_scores,
                    second_face_scores,
                    celeb_id)),
            columns=['image_names', 'dobs', 'genders', 'face_locations',
                     'photo_taken_years', 'face_scores', 'second_face_scores', 'celeb_ids']
        )

        # Filter dataframe by only having the rows with face_scores > 1.0
        df = df[df['face_scores'] > 1.0]


        # Remove any records that contain Nulls/NaNs by checking for NaN with .isna()
        df = df[~df['genders'].isna()]
        df = df[~df['second_face_scores'].isna()]

        # Cast genders to integers so that mapping can take place
        df.genders = df.genders = df.genders.astype(int)

        # Iterate over all the rows in the dataframe and map each feature
        for _, row in df.iterrows():
            # Extract filename, gender, dob, photo_taken,
            # face_score, second_face_score and celeb_id
            filename = os.path.join(image_dir, _DATASET_ROOT_DIR, row['image_names'][0])
            gender = row['genders']
            dob = row['dobs']
            photo_taken = row['photo_taken_years']
            face_score = row['face_scores']
            second_face_score = row['second_face_scores']
            celeb_id = row['celeb_ids']

        # Get the image shape
        image_width, image_height = self._get_image_shape(filename)
        # Normalize the bounding boxes by using the face coordinates and the image shape
        bbox = self._get_bounding_box_values(row['face_locations'][0],
                                           image_width, image_height)

        # Yield a feature dictionary
        yield filename, {
          "image": filename,
          "gender": gender,
          "dob": dob,
          "photo_taken": photo_taken,
          "face_location": tfds.features.BBox(
                                ymin=min(bbox[0], 1.0),
                                xmin=min(bbox[1], 1.0),
                                ymax=min(bbox[2], 1.0),
                                xmax=min(bbox[3], 1.0)),
          "face_score": face_score,
          "second_face_score": second_face_score,
          "celeb_id": celeb_id
        }
```

```
Overwriting tensorflow_datasets/image/imdb_faces.py
```

Add an Import for Registration

All subclasses of `tfds.core.DatasetBuilder` are automatically registered when their module is imported such that they can be accessed through `tfds.builder` and `tfds.load`.

If you're contributing the dataset to `tensorflow/datasets`, you must add the module import to its subdirectory's `__init__.py` (e.g. `image/__init__.py`), as shown below:

In [20]:

```
%%writefile tensorflow_datasets/image/__init__.py
# coding=utf-8
# Copyright 2019 The TensorFlow Datasets Authors.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Image datasets."""

from tensorflow_datasets.image.abstract_reasoning import AbstractReasoning
from tensorflow_datasets.image.aflw2k3d import Aflw2k3d
from tensorflow_datasets.image.bigearthnet import Bigearthnet
from tensorflow_datasets.image.binarized_mnist import BinarizedMNIST
from tensorflow_datasets.image.binary_alpha_digits import BinaryAlphaDigits
from tensorflow_datasets.image.caltech import Caltech101
from tensorflow_datasets.image.caltech_birds import CaltechBirds2010
from tensorflow_datasets.image.cats_vs_dogs import CatsVsDogs
from tensorflow_datasets.image.cbis_ddsm import CuratedBreastImagingDDSM
from tensorflow_datasets.image.celeba import CelebA
from tensorflow_datasets.image.celebahq import CelebAHq
from tensorflow_datasets.image.chexpert import Chexpert
from tensorflow_datasets.image.cifar import Cifar10
from tensorflow_datasets.image.cifar import Cifar100
from tensorflow_datasets.image.cifar10_corrupted import Cifar10Corrupted
from tensorflow_datasets.image.clevr import CLEVR
from tensorflow_datasets.image.coco import Coco
from tensorflow_datasets.image.coco2014_legacy import Coco2014
from tensorflow_datasets.image.coil100 import Coil100
from tensorflow_datasets.image.colorectal_histology import ColorectalHistology
from tensorflow_datasets.image.colorectal_histology import ColorectalHistologyLarge
from tensorflow_datasets.image.cycle_gan import CycleGAN
from tensorflow_datasets.image.deep_weeds import DeepWeeds
from tensorflow_datasets.image.diabetic_retinopathy_detection import DiabeticRetinopathyDetection
from tensorflow_datasets.image.downsampled_imagenet import DownsampledImagenet
from tensorflow_datasets.image.dsprites import Dsprites
from tensorflow_datasets.image.dtd import Dtd
from tensorflow_datasets.image.eurosat import Eurosat
from tensorflow_datasets.image.flowers import TFFlowers
from tensorflow_datasets.image.food101 import Food101
from tensorflow_datasets.image.horses_or_humans import HorsesOrHumans
from tensorflow_datasets.image.image_folder import ImageLabelFolder
from tensorflow_datasets.image.imagenet import Imagenet2012
from tensorflow_datasets.image.imagenet2012_corrupted import Imagenet2012Corrupted
from tensorflow_datasets.image.kitti import Kitti
from tensorflow_datasets.image.lfw import LFW
from tensorflow_datasets.image.lsun import Lsun
from tensorflow_datasets.image.mnist import EMNIST
from tensorflow_datasets.image.mnist import FashionMNIST
from tensorflow_datasets.image.mnist import KMNIST
from tensorflow_datasets.image.mnist import MNIST
from tensorflow_datasets.image.mnist_corrupted import MNISTCorrupted
from tensorflow_datasets.image.omniglot import Omniglot
from tensorflow_datasets.image.open_images import OpenImagesV4
from tensorflow_datasets.image.oxford_flowers102 import OxfordFlowers102
from tensorflow_datasets.image.oxford_iiit_pet import OxfordIIITPet
from tensorflow_datasets.image.patch_camelyon import PatchCamelyon
from tensorflow_datasets.image.pet_finder import PetFinder
from tensorflow_datasets.image.quickdraw import QuickdrawBitmap
from tensorflow_datasets.image.resisc45 import Resisc45
from tensorflow_datasets.image.rock_paper_scissors import RockPaperScissors
from tensorflow_datasets.image.scene_parse_150 import SceneParse150
from tensorflow_datasets.image.shapes3d import Shapes3d
from tensorflow_datasets.image.smallnorb import Smallnorb
from tensorflow_datasets.image.so2sat import So2sat
from tensorflow_datasets.image.stanford_dogs import StanfordDogs
from tensorflow_datasets.image.stanford_online_products import StanfordOnlineProducts
from tensorflow_datasets.image.sun import Sun397
from tensorflow_datasets.image.svhn import SvhnCropped
from tensorflow_datasets.image.uc_merced import UcMerced
from tensorflow_datasets.image.visual_domain_decathlon import VisualDomainDecathlon

# EXERCISE: Import your dataset module here

from tensorflow_datasets.image.imdb_faces import ImdbFaces
```

```
Overwriting tensorflow_datasets/image/__init__.py
```

URL Checksums

If you're contributing the dataset to `tensorflow/datasets`, add a checksums file for the dataset. On first download, the DownloadManager will automatically add the sizes and checksums for all downloaded URLs to that file. This ensures that on subsequent data generation, the downloaded files are as expected.

In [21]:

```
!touch tensorflow_datasets/url_checksums/imdb_faces.txt
```

Build the Dataset

In [22]:

```
# EXERCISE: Fill in the name of your dataset.
# The name must be a string.
DATASET_NAME = "imdb_faces"
```

We then run the `download_and_prepare` script locally to build it, using the following command:

```
%%bash -s $DATASET_NAME
python -m tensorflow_datasets.scripts.download_and_prepare \
  --register_checksums \
  --datasets=$1
```

NOTE: It may take more than 30 minutes to download the dataset and then write all the preprocessed files as TFRecords. Due to the enormous size of the data involved, we are unable to run the above script in the Coursera environment.

Load the Dataset

Once the dataset is built you can load it in the usual way, by using `tfds.load`, as shown below:

```
import tensorflow_datasets as tfds
dataset, info = tfds.load('imdb_faces', with_info=True)
```

Note: Since we couldn't build the `imdb_faces` dataset due to its size, we are unable to run the above code in the Coursera environment.

Explore the Dataset

Once the dataset is loaded, you can explore it by using the following loop:

```
for feature in tfds.as_numpy(dataset['train']):
  for key, value in feature.items():
    if key == 'image':
      value = value.shape
    print(key, value)
  break
```

Note: Since we couldn't build the `imdb_faces` dataset due to its size, we are unable to run the above code in the Coursera environment.

The expected output from the code block shown above should be:

```
>>>
celeb_id 12387
dob 722957
face_location [1.         0.56327355 1.         1.        ]
face_score 4.0612864
gender 0
image (96, 97, 3)
photo_taken 2007
second_face_score 3.6680346
```

Next steps for publishing

Double-check the citation

It's important that DatasetInfo.citation includes a good citation for the dataset. It's hard and important work contributing a dataset to the community and we want to make it easy for dataset users to cite the work.

If the dataset's website has a specifically requested citation, use that (in BibTex format).

If the paper is on arXiv, find it there and click the bibtex link on the right-hand side.

If the paper is not on arXiv, find the paper on Google Scholar and click the double-quotation mark underneath the title and on the popup, click BibTeX.

If there is no associated paper (for example, there's just a website), you can use the BibTeX Online Editor to create a custom BibTeX entry (the drop-down menu has an Online entry type).

Add a test

Most datasets in TFDS should have a unit test and your reviewer may ask you to add one if you haven't already. See the testing section below.
Check your code style

Follow the PEP 8 Python style guide, except TensorFlow uses 2 spaces instead of 4. Please conform to the Google Python Style Guide,

Most importantly, use tensorflow_datasets/oss_scripts/lint.sh to ensure your code is properly formatted. For example, to lint the image directory See TensorFlow code style guide for more information.

Add release notes Add the dataset to the release notes. The release note will be published for the next release.

Send for review! Send the pull request for review.

For more information, visit https://www.tensorflow.org/datasets/add_dataset

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 frees up resources for your fellow learners.

In [ ]:

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

In [ ]:

```
%%javascript
<!-- Shutdown and close the notebook -->
window.onbeforeunload = null
window.close();
IPython.notebook.session.delete();
```