gulpio2

Release 0.0.4

Will Price & TwentyBN

Mar 18, 2021

CONTENTS:

Binary storage format for deep learning on videos.

CHAPTER

ONE

USAGE

1.1 'Gulp' a Dataset

The `gulpio` package has been designed to be infinitely hackable and to support arbitrary datasets. To this end, we are providing a so-called *adapter pattern*. Specifically there exists an abstract class in `gulpio.adapters`: the `AbstractDatasetAdapter`. In order to ingest your dataset, you basically need to implement your own custom adapter that inherits from this.

You should be able to get going quickly by looking at the following examples, that we use internally to gulp our video datasets.

- The class: `gulpio.adapters.Custom20BNJsonAdapter` adapters.py
- The script: `gulp2_20bn_json_videos` command line script

And an example invocation would be:

```
gulp2_20bn_json_videos videos.json input_dir output_dir
```

Additionally, if you would like to ingest your dataset from the command line, the `register_adapter` script can be used to generate the command line interface for the new adapter. Write your adapter that inherits from the `AbstractDatasetAdapter` in the `adapter.py` file, then simply call:

```
gulp2_register_adapter gulpio.adapters <NewAdapterClassName>
```

The script that provides the command line interface will be in the main directory of the repository. To use it, execute `./new_adapter_class_name`.

1.2 Sanity check the 'Gulped' Files

A very basic test to check the correctness of the gulped files is provided by the `gulp2_sanity_check` script. For execution run:

```
gulp2_sanity_check <folder containing the gulped files>
```

It tests:

- The presence of any content in the `.gulp` and `.gmeta`-files
- The file size of the `.gulp` file corresponds to the required file size that is given in the `.gmeta` file
- Duplicate appearances of any video-ids

The file names of the files where any test fails will be printed. Currently no script to fix possible errors is provided, 'regulping' is the only solution.

1.3 Read a 'Gulped' Dataset

In order to read from the gulps, you can let yourself be inspired by the following snippet:

```
from gulpio2 import GulpDirectory
# You can either read greyscale (`colorspace="GRAY"`) or RGB (`colorspace="RGB"`)
# images.
gulp_directory = GulpDirectory('/tmp/something_something_gulps')
# iterate over all chunks
for chunk in gulp_directory:
    # for each 'video' get the metadata and all frames
    for frames, meta in chunk:
        # do something with the metadata
        for i, f in enumerate(frames):
            # do something with the frames
            pass
```

Alternatively, a video with a specific `id` can be directly accessed via:

```
from gulpio2 import GulpDirectory
gulp_directory = GulpDirectory('/tmp/something_something_gulps')
frames, meta = gulp_directory[<id>]
```

For down-sampling or loading only a part of a video, a python slice or list of indices can be passed as well:

```
frames, meta = gulp_directory[<id>, slice(1,10,2)]

frames, meta = gulp_directory[<id>, [1, 5, 6, 8]]
```

or:

```
frames, meta = gulp_directory[<id>, 1:10:2]
```

1.4 Loading Data

Below is an example loading an image dataset and defining an augmentation pipeline using torchvision. Transformations are applied to each instance on the fly.

```
from torch.utils.data import DataLoader
from torchvision.transforms import Scale, CenterCrop, Compose, Normalize


class GulpImageDataset:
    def __init__(self, gulp_dir: GulpDirectory, transform=None):
        self.gulp_dir = gulp_dir
        self.transform = transform if transform is not None else lambda x: x
        self.example_ids = list(gulp_dir.merged_meta_dict.keys())

    def __getitem__(self, idx):
        if isinstance(idx, int):
```

(continues on next page)

(continued from previous page)

```
            example_id = self.example_ids[idx]
        else:
            example_id = idx
        imgs, meta = self.gulp_dir[example_id]
        return self.transform(imgs[0]), meta

    def __len__(self):
        return len(self.gulp_dir.merged_meta_dict)

# define data augmentations. Notice that there are different functions for videos and␣
↪images
transform = Compose([
    Resize(120),
    CenterCrop(112),
    Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
])

# define dataset wrapper and pick this up by the data loader interface.
dataset = GulpImageDataset(GulpDirectory('/path/to/train_data'), transform=transforms)
loader = DataLoader(dataset, batch_size=256, shuffle=True, num_workers=0, drop_
↪last=True)

dataset_val = GulpImageDataset(GulpDirectory('/path/to/validation_data/'),␣
↪transform=transforms)
loader_val = DataLoader(dataset_val, batch_size=256, shuffle=False, num_workers=0,␣
↪drop_last=True)
```

Here we iterate through the dataset we loaded. Iterator returns data and label as numpy arrays. You might need to cast these into the format of you deep learning library.

```
for data, label in loader:
    # train your model here
    # ...
```

CHAPTER

TWO

FORMAT DESCRIPTION

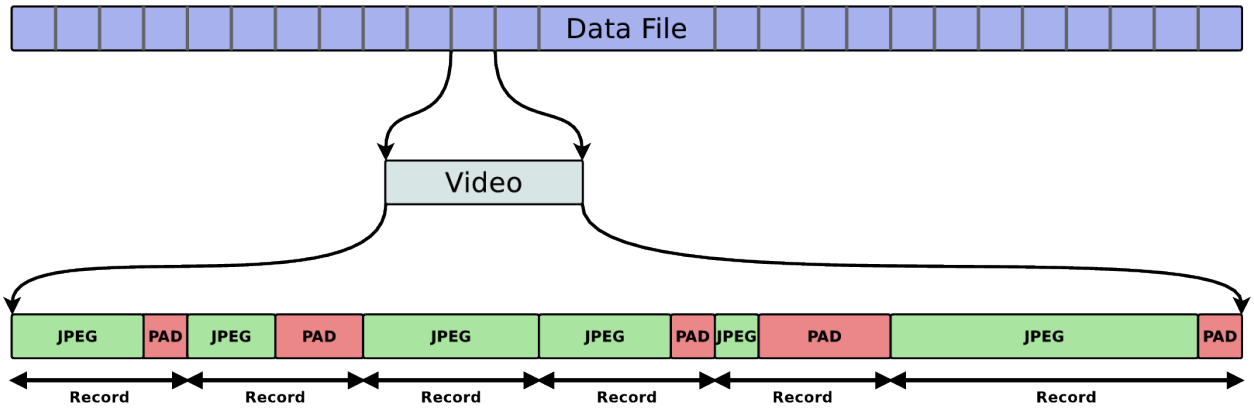
When gulping a dataset, two different files are created for every chunk: a `*.gulp` data file that contains the actual data and a `*.gmeta` meta file that contains the metadata.

The layout of the `*.gulp` file is as follows:

```
|-jpeg-|-pad-|-jpeg-|-pad-|...
```

Essentially, the data file is simply a series of concatenated JPEG images, i.e. the frames of the video. Each frame is padded to be divisible by four bytes, since this makes it easier to read JPEGs from disk.

Here is a more visual example:



As you can see there are 6 *records* in the example. They have the following paddings and lengths:

FRAME	LEN	PAD
0	4	1
1	4	2
2	4	0
3	4	1
4	4	3
5	8	1

The layout of the meta file is a mapping, where each `id` representing a video is mapped to two further mappings, `meta_data`, which contains arbitrary, user-defined meta-data. And a triplet, `frame_info`, which contains the offset (index) into the data file, the number of bytes used for padding and the total length of the frame (including padding). (`[<offset>, <padding>, <total_length>]`.) The *frame_info* is required to recover the frames from the data file.

```
'id'
  |
  |-> meta_data: [{}]
  |
  |-> frame_info: [[], [], ...]
.
.
.
```

By default, the meta file is serialized in JSON format.

For example, here is a meta file snippet:

```
{"702766": {"frame_info": [[0, 3, 7260],
                           [7260, 3, 7252],
                           [14512, 2, 7256],
                           [21768, 2, 7260],
                           [29028, 1, 7308],
                           [36336, 1, 7344],
                           [43680, 0, 7352],
                           [51032, 1, 7364],
                           [58396, 0, 7348],
                           [65744, 1, 7352],
                           [73096, 1, 7352],
                           [80448, 1, 7408],
                           [87856, 1, 7400],
                           [95256, 0, 7376],
                           [102632, 1, 7384],
                           [110016, 2, 7404],
                           [117420, 0, 7396],
                           [124816, 1, 7400],
                           [132216, 2, 7428],
                           [139644, 1, 7420],
                           [147064, 0, 7428],
                           [154492, 2, 7472],
                           [161964, 3, 7456],
                           [169420, 2, 7444],
                           [176864, 2, 7436]],
            "meta_data":  [{"label": "something something",
                            "id":    702766}]},
 "803959": {"frame_info": [[184300, 1, 9256],
                           [193556, 3, 9232],
                           [202788, 2, 9340],
                           [212128, 2, 9184],
                           [221312, 1, 9112],
                           [230424, 3, 9100],
                           [239524, 0, 9144],
                           [248668, 1, 9120],
                           [257788, 0, 9104],
                           [266892, 0, 9220],
                           [276112, 1, 9140],
                           [285252, 1, 9076],
                           [294328, 2, 9100],
                           [303428, 0, 9224],
                           [312652, 3, 9200],
                           [321852, 3, 9136],
                           [330988, 2, 9136],
                           [340124, 1, 9152],
```

(continues on next page)

(continued from previous page)


```
                            [349276, 0, 8984],
                            [358260, 1, 9048],
                            [367308, 0, 9116],
                            [376424, 1, 9136],
                            [385560, 1, 9108],
                            [394668, 2, 9084],
                            [403752, 1, 9112],
                            [412864, 2, 9108]],
            "meta_data":  [{"label": "something something",
                            "id":    803959}]},
"803957": {"frame_info": [[421972, 2, 8592],
                            [430564, 1, 8608],
                            [439172, 2, 8872],
                            [448044, 3, 8852],
                            [456896, 2, 8860],
                            [465756, 0, 8908],
                            [474664, 2, 8912],
                            [483576, 1, 8884],
                            [492460, 1, 8752],
                            [501212, 3, 8692],
                            [509904, 0, 8612],
                            [518516, 0, 8816],
                            [527332, 2, 8784],
                            [536116, 1, 8840],
                            [544956, 1, 8844],
                            [553800, 1, 8988],
                            [562788, 0, 8992],
                            [571780, 0, 8972],
                            [580752, 3, 9044],
                            [589796, 2, 9012],
                            [598808, 3, 9060],
                            [607868, 2, 9032],
                            [616900, 1, 9052],
                            [625952, 2, 9056],
                            [635008, 0, 9084],
                            [644092, 2, 9100]],
            "meta_data":  [{"label": "something something",
                            "id":    803957}]},
"773430": {"frame_info": [[653192, 1, 7964],
                            [661156, 2, 7996],
                            [669152, 1, 7960],
                            [677112, 0, 8024],
                            [685136, 0, 8008],
                            [693144, 1, 7972],
                            [701116, 0, 7980],
                            [709096, 0, 8036],
                            [717132, 0, 8016],
                            [725148, 0, 8016],
                            [733164, 1, 8004],
                            [741168, 1, 8008],
                            [749176, 1, 7996],
                            [757172, 1, 8016],
                            [765188, 1, 8032],
                            [773220, 0, 8040],
                            [781260, 2, 8044],
                            [789304, 2, 8004],
                            [797308, 1, 8008],
```


(continues on next page)

(continued from previous page)


```
                            [805316, 0, 8056],
                            [813372, 3, 8088],
                            [821460, 0, 8044]],
            "meta_data":  [{"label": "something something",
                            "id":    773430}]},
 "803963": {"frame_info": [[829504, 2, 8952],
                            [838456, 1, 8928],
                            [847384, 0, 8972],
                            [856356, 1, 8992],
                            [865348, 1, 8936],
                            [874284, 1, 8992],
                            [883276, 3, 8988],
                            [892264, 1, 9008],
                            [901272, 2, 8996],
                            [910268, 2, 8976],
                            [919244, 0, 9180],
                            [928424, 0, 9128],
                            [937552, 2, 9100],
                            [946652, 2, 9096],
                            [955748, 3, 9044],
                            [964792, 0, 9096],
                            [973888, 2, 9068],
                            [982956, 1, 8996],
                            [991952, 3, 8928],
                            [1000880, 1, 9040],
                            [1009920, 0, 9084],
                            [1019004, 0, 9076],
                            [1028080, 2, 9056],
                            [1037136, 2, 9040],
                            [1046176, 2, 9052],
                            [1055228, 3, 9096]],
            "meta_data":  [{"label": "something something",
                            "id":    803963}]}
}
```

CHAPTER

THREE

API

class gulpio2.**GulpDirectory**(*output_dir*, *jpeg_decoder=<function jpeg_bytes_to_img>*)

Represents a directory containing *.gulp and *.gmeta files.

Parameters

- **output_dir** (*str*) – Path to the directory containing the files.
- **jpeg_decoder** (callable that takes a JPEG stored as bytes and returns) – the desired decoded image format (e.g. np.ndarray)

all_meta_dicts

All meta dicts from all chunks as a list.

Type list of dicts

chunk_lookup

Mapping element id to chunk index.

Type dict: int -> str

chunk_objs_lookup

Mapping element id to chunk index.

Type dict: int -> GulpChunk

merged_meta_dict

all meta dicts merged

Type dict: id -> meta dict

chunks()

Return a generator over existing GulpChunk objects which are ready to be opened and read from.

new_chunks(*total_new_chunks*)

Return a generator over freshly setup GulpChunk objects which are ready to be opened and written to.

Parameters **total_new_chunks** (*int*) – The total number of new chunks to initialize.

class gulpio2.**GulpChunk**(*data_file_path*, *meta_file_path*, *serializer=<gulpio2.fileio.JSONSerializer object>*, *jpeg_decoder=<function jpeg_bytes_to_img>*)

Represents a gulp chunk on disk.

Parameters

- **data_file_path** (*str*) – Path to the *.gulp file.
- **meta_file_path** (*str*) – Path to the *.gmeta file.
- **serializer** (*subclass of AbstractSerializer*) – The type of serializer to use.

- **jpeg_decoder** (callable that takes a JPEG stored as `bytes` and returns) – the desired decoded image format (e.g. np.ndarray)

append(*id_*, *meta_data*, *frames*)
Append an item to the gulp.

Parameters

- **id** (*str*) – The ID of the item
- **meta_data** (*dict*) – The meta-data associated with the item.
- **frames** (*list of numpy arrays*) – The frames of the item as a list of numpy dictionaries consisting of image pixel values.

flush()
Flush all buffers and write the meta file.

iter_all(*accepted_ids=None*, *shuffle=False*)
Iterate over all frames in the gulp.

Parameters

- **accepted_ids** (*list of str*) – A filter for accepted ids.
- **shuffle** (*bool*) – Shuffle the items or not.

Returns An iterator that yield a series of frames,meta tuples. See *read_frames* for details.

Return type iterator

open(*flag='rb'*)
Open the gulp chunk for reading.

Parameters **flag** (*str*) – 'rb': Read binary 'wb': Write binary 'ab': Append to binary

Notes

Works as a context manager but returns None.

read_frames(*id_*, *slice_=None*)
Read frames for a single item.

Parameters

- **id** (*str*) – The ID of the item
- **slice** (*slice or list of ints:*) – A slice or list of indices with which to select frames.

Returns The frames of the item as a list of numpy arrays consisting of image pixel values. And the metadata.

Return type frames (int), meta(dict)

class gulpio2.**GulpIngestor**(*adapter*, *output_folder*, *videos_per_chunk*, *num_workers*)
Ingest items from an adapter into an gulp chunks.

Parameters

- **adapter** (*subclass of AbstractDatasetAdapter*) – The adapter to ingest from.
- **output_folder** (*str*) – The folder/directory to write to.
- **videos_per_chunk** (*int*) – The total number of items per chunk.

- **num_workers** (*int*) – The level of parallelism.

class gulpio2.**ChunkWriter**(*adapter*)

Can write from an adapter to a gulp chunk.

Parameters **adapter** (*subclass of AbstractDatasetAdapter*) – The adapter to get items from.

write_chunk(*output_chunk*, *input_slice*)

Write from an input slice in the adapter to an output chunk.

Parameters

- **output_chunk** (GulpChunk) – The chunk to write to
- **input_slice** (*slice*) – The slice to use from the adapter.

CHAPTER

FOUR

INSTALLATION

```
pip install gulpio2
```

CHAPTER

FIVE

MIGRATING FROM GULPIO

If you've been using gulpio, then you can use `gulpio2` as a drop-in replacement. Simply replace all `import gulpio ...` statements with `import gulpio2 ...`.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

SEVEN

LICENSE

All original gulpio code is Copyright (c) Twenty Billion Neurons and licensed under the MIT License, see the file `LICENSE.txt` for details. Subsequent code is Copyright (c) Will Price, and is also licensed under the MIT License.

PYTHON MODULE INDEX

INDEX