datadings

Joachim Folz

Apr 28, 2023

CONTENTS

1 Mission statement **3**
- 1.1 Fast, you say? . . . 3
- 1.2 TL;DR . . . 4

2 Contents **5**
- 2.1 Usage . . . 5
- 2.2 Data conventions . . . 6
- 2.3 PyTorch integration . . . 6
- 2.4 File format . . . 9
- 2.5 Creating a custom dataset . . . 11
- 2.6 Contributing a dataset . . . 12
- 2.7 API reference . . . 18

3 Indices and tables **83**

Python Module Index **85**

Index **87**

datadings is a collection of tools to prepare datasets for machine learning, based on two simple principles:

Datasets are collections of individual data samples.

Each sample is a dictionary with descriptive keys.

For supervised training with images samples are dictionaries like this:

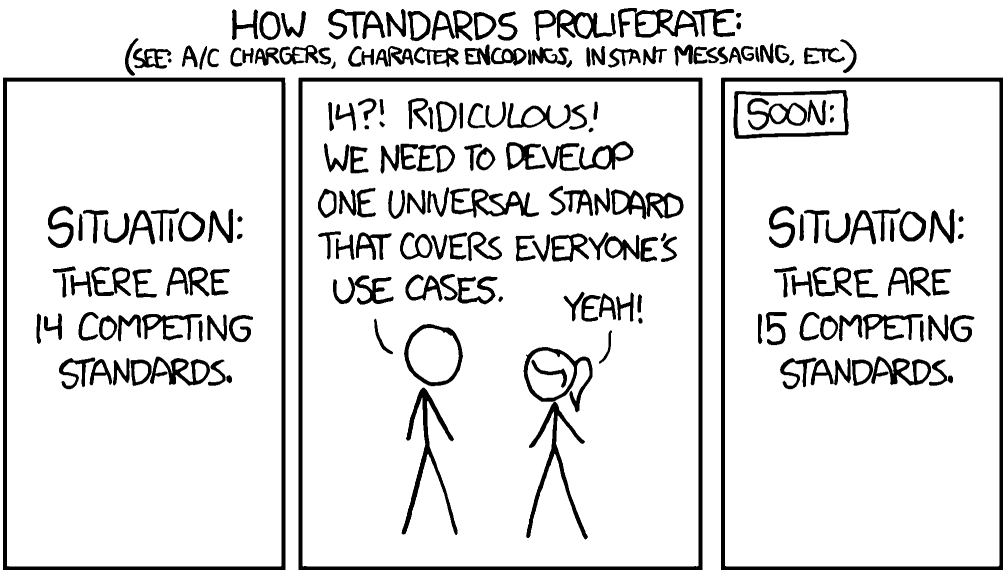
```
{"key": unique_key, "image": imagedata, "label": label}
```

CHAPTER

ONE

MISSION STATEMENT

Dealing with different datasets can be tedious for machine learning practitioners. Two datasets almost never share the same directory structure and often custom file formats are used. How datadings fits into the picture is best explained by XKCD #927:



Slightly less cynically, datadings aims to make dealing with datasets fast and easy. datadings currently supports over 20 different datasets for image classification, segmentation, saliency prediction, and remote sensing. `pip install datadings` and use the *`datadings-write`* command (`datadings-write -h` for more info) to download the source files for any of the included datasets and convert them to the datadings format. And since it's based on the excellent msgpack, a JSON-like format that supports binary data, it's space-efficient, blazingly fast, does not use schema, and has support for over 50 programming languages and environments. You are also not limited to any specific learning framework, only Python if you want to use additional tools provided by datadings.

1.1 Fast, you say?

The ImageNet dataset (the ILSVRC2012 challenge dataset, to be precise) is prominently featured in many scientific publications. Tutorials on how to train models with it usually recommended unpacking the large training and validation set tar files into separate folders. There are now roughly 1.3 million tiny files you need to load per epoch of training. This is bad for HDDs and doubly bad if you access them over the network. While datadings supports reading from datasets like these with the *`DirectoryReader`*, it will only read with a leisurely pace of about 500 samples/s. Reading the whole training set takes about 40 minutes. This is not fast enough for modern GPUs.

Once converted into the datadings format, you can easily saturate 10G ethernet reading well over 20000 samples/s using the *MsgpackReader*.

It also takes several seconds to start reading from the directory tree, whereas reading from msgpack files is almost instant. This makes debugging a breeze. Check out the *file format description* description if you want to know how this is achieved.

1.2 TL;DR

`pip install datadings` and use the *datadings-write* command to create the dataset files (`datadings-write -h` for more info). It creates a `dataset.msgpack` file. In your code, open this file with the *MsgpackReader*. You can now iterate over it:

```
from datadings.reader import MsgpackReader
with MsgpackReader('dataset.msgpack') as reader:
    for sample in reader:
        [do dataset things]
```

CHAPTER

TWO

CONTENTS

2.1 Usage

Each dataset defines modules to read and write in the `datadings.sets` package. For most datasets the reading module only contains additional metadata like class labels and distributions.

Let's consider the *MIT1003* dataset as an example.

`MIT1003_write` is an executable that creates dataset files. It can be called directly `python -m datadings.sets.MIT1003_write` or through *`datadings-write`*. Three files will be written:

- `MIT1003.msgpack` contains sample data
- `MIT1003.msgpack.index` contains index for random access
- `MIT1003.msgpack.md5` contains MD5 hashes of both files

Reading all samples sequentially, using a `MsgpackReader` as a context manager:

```
from datadings.reader import MsgpackReader
with MsgpackReader('MIT1003.msgpack') as reader:
    for sample in reader:
        # do dataset things!
```

This standard iterator returns dictionaries. Use the `rawiter()` method to get samples as messagepack encoded bytes instead.

Reading specific samples:

```
reader.seek_key('i14020903.jpeg')
print(reader.next()['key'])
reader.seek_index(100)
print(reader.next()['key'])
```

Reading samples as raw msgpacked bytes:

```
raw = reader.rawnext()
for raw in reader.rawiter():
    print(type(raw), len(raw))
```

Number of samples:

```
print(len(reader))
```

You can also change the order and selection of iterated samples with *`Augments`*. For example, to randomize the order of samples, wrap the reader in a *`Shuffler`*:

```
from datadings.reader import Shuffler
with Shuffler(MsgpackReader('MIT1003.msgpack')) as reader:
    for sample in reader:
        # do dataset things, but in random order!
```

A common use case is to iterate over the whole dataset multiple times. This can be done with the *Cycler*:

```
from datadings.reader import Cycler
with Cycler(MsgpackReader('MIT1003.msgpack')) as reader:
    for sample in reader:
        # do dataset things, but FOREVER!
```

2.2 Data conventions

datadings is built around two basic principles:

> Datasets are collections of individual data samples.
>
> Each sample is a dictionary with descriptive keys.

E.g., for supervised training with images each sample is a dictionary `{'key': unique_key, 'image': imagedata, 'label': label)`. Depending on the type of dataset different keys are used. There is no schema to dictate which keys are allowed and what they stand for, with the exception of the `"key"` which is always a unique identifier of that sample in the dataset. datadings follows the best-effort principle that the kind of data associated with a certain key remains the same across datasets. These are some common keys and their meanings:

- `"key"`: Unique identifier of this sample.
- `"image"`: Contents of an image file. Use any standard image library to get pixel data.
- `"label"`: Numerical label for the whole sample.
- `"*_image"`: Same as image, but different semantics. For example, `"label_image"` with per-pixel segmentation labels, or `"instance_image"` for instance segmentation.
- `"experiments"`: A list of experiments, usually saliency. Type depends on dataset.
- `"locations"`: List of (x,y) coordinates.
- `"map"`: Fixation map as image.
- `"fixations"`: List of (x,y) fixation points.

2.3 PyTorch integration

Warning: This functionality is highly experimental and subject to change in future version!

datadings provides experimental integration with PyTorch. There are two options:

1. *Dataset*
2. *IterableDataset*

These implement the respective PyTorch dataset classes and work as expected with the PyTorch `DataLoader`.

Note: `persistent_workers=True` must be used to let *`IterableDataset`* track the current epoch.

Warning: *`Dataset`* can be significantly slower than *`IterableDataset`*. If shuffling is necessary consider using *`QuasiShuffler`* instead.

Example usage with the PyTorch `DataLoader`:

```
from datadings.reader import MsgpackReader
from datadings.torch import IterableDataset
from datadings.torch import CompressedToPIL
from datadings.torch import dict2tuple
from datadings.torch import Compose

from tqdm import tqdm
from torch.utils.data import DataLoader
from torchvision.transforms import ToTensor
from torchvision.transforms import RandomResizedCrop
from torchvision.transforms import RandomHorizontalFlip


def main():
    path = '.../train.msgpack'
    batch_size = 256
    transforms = {'image': Compose(
        CompressedToPIL(),
        RandomResizedCrop((224, 224)),
        RandomHorizontalFlip(),
        ToTensor(),
    )}
    reader = MsgpackReader(path)
    ds = IterableDataset(
        reader,
        transforms=transforms,
        batch_size=batch_size,
    )
    train = DataLoader(
        dataset=ds,
        batch_size=batch_size,
        num_workers=4,
        persistent_workers=True,
    )
    for epoch in range(3):
        print('Epoch', epoch)
        for x, y in dict2tuple(tqdm(train)):
            pass


if __name__ == "__main__":
    main()
```

In our example `transforms` is a dictionary with one key `'image'`. That means the given transformation is applied to the value with this key. You can add more keys and transforms to apply functions to different keys.

Note: There will be warnings that transforms only accept varargs when using non-functional torchvision transforms due to their opaque call signatures. This is fine, since these transforms only need the value as input. Other transforms may not work though.

If you need to share randomness between transformations (e.g. to synchronize augmentation steps between image and mask in semantic segmentation) you can use functions that accept randomness as parameters, like functional transforms from torchvision. Datasets accept a callable `rng` parameter with signature `rng(sample: dict) -> dict`. `sample` is the sample that is going to be transformed and the returned dictionary must contain all positional parameters required by the transform functions. The *`Compose`* object reads the function signatures of your transforms to determine which parameters are required. A minimal example for how this works:

```
import random
from datadings.torch import Compose
from datadings.torch import Dataset
from datadings.reader import ListReader

def add(v, number):
    return v + number

def sub(x, value):
    return x - value

def rng(_):
    return {
        'number': random.randrange(1, 10),
        'value': random.randrange(1, 10),
    }

samples = [{'a': 0, 'b': 0, 'c': 0} for _ in range(10)]
reader = ListReader(samples)
transforms = {
    'a': Compose(add),
    'b': Compose(sub),
    'c': Compose((add, sub)),
}
dataset = Dataset(reader, transforms=transforms, rng=rng)
for i in range(len(dataset)):
    print(dataset[i])
```

Note: You can use `functools.partial()` (or similar) to set constant values for parameters and change defaults for keyword arguments instead of including them in your `rng` dictionary.

Warning: Transform functions will receive the same value if they share parameter names. If this is not intended you must wrap one of those functions in another function with and change on of the parameter names.

Alternatively `transforms` may be a custom function with signature `t(sample: dict) -> dict`. This allows you to use multiple values from the sample for a transform, create new values based on the sample, etc.

2.4 File format

The file format datadings uses is simple. A dataset is made up of six files:

- `.msgpack` main data file
- `.msgpack.offsets` sample start offset file
- `.msgpack.keys` key file
- `.msgpack.key_hashes` key hash file
- `.msgpack.filter` Bloom filter file
- `.msgpack.md5` for integrity checks

2.4.1 Data file

Each sample is a key-value map with a string `key` that is unique for the dataset. In Python notation:

```
{"key": <unique key>, ... }
```

Each sample is represented by one msgpack message. The main dataset file has the extension `.msgpack` and contains a sequence of these messages:

```
<sample 1><sample 2><sample 3> ...
```

Note that no additional data is stored in this file. The msgpack format does not require the length of the message to be known for unpacking, so this single file is sufficient for sequential access.

Arrays & complex numbers

msgpack on its own does not support densely packed arrays or complex numbers. While it may be sufficient to use lists for heterogeneous data types or few values, datadings uses msgpack-numpy to support storing arbitrary numpy arrays efficiently. This introduces a limitation on the keys that can be present in samples.

Reserved keys

The following keys are reserved by datadings for internal use and thus cannot be used in samples:

- `"key"`: used to uniquely identify samples in a dataset
- `"nd"`: used by msgpack-numpy for array decoding
- `"complex"`: used by msgpack-numpy for complex number decoding

Using these keys results in undefined behavior.

2.4.2 Index

Datasets are indexed to enable fast sequential and random access. Previous versions of datadings created a monolithic index file that contained both keys and read offsets of samples. New-style indexes are made up of 4 separate files:

1. `.msgpack.offsets`: uint64 start offsets for samples in the data file stored in network byte order, where *offset[i]* corresponds to the ith sample.
2. `.msgpack.keys`: msgpacked list of keys.
3. `.msgpack.key_hashes`: 8 byte salt, followed by 8 byte blake2s hashes of all keys. The salt is chosen to avoid hash collisions.
4. `.msgpack.filter`: A Bloom filter for all keys in simplebloom format. It is setup to provide very low false-positive probabilities.

The advantage of this new style of index is that is allows for fast and lazy loading of elements as they are required. For typical datasets the keys file is several times larger than both offsets and key hashes, and both are several timers larger than the bloom filter. To check whether the dataset contains a key, only the filter and key hashes are required. The larger keys file itself is only loaded whenever a method returns the sample keys. Thus upon initialization the reader only checks for the presence of index files and warns if they are missing.

2.4.3 MD5 file

Finally, every dataset comes with a `.msgpack.md5` file with hashes for the data and index files, so their integrity can be verified.

2.4.4 Limitations

Since msgpack is used datadings inherits its limitations.

- Maps and lists cannot have more than 2^{32}-1 entries.
- Strings and binary data cannot be longer than 2^{32}-1 bytes.
- Integers (signed or unsigned) are limited to 64 bits.
- Floats are limited to single or double precision.

This means each dataset is limited to less than 2^{32} samples (since the index uses a map) and around 2^{64} bytes total file size (the largest possible byte offset is 2^{64}-1). The same applies to each individual sample regarding the number of keys present and its packed size.

2.4.5 Legacy index file

Previous versions of datadings used a different index arrangement. An index file with the `.msgpack.index` extension contained a map of key-offset pairs. In Python notation:

```
{"sample 1": 0, "sample 2": 1234, ... }
```

For every `key` in the dataset it gives the offset in bytes of the respective sample from the beginning of the file. Index entries are stored with offsets in ascending order.

2.5 Creating a custom dataset

Follow this guide if the dataset you want to use is not yet part of datadings. If the data is publicly available, please consider *contributing* it to datadings.

2.5.1 A basic example

Typically the process of converting individual samples into the datadings format is divided into locating/loading/preprocessing samples and writing them to the dataset file. Here's a ready-to-run example that illustrates this:

```
import random
from datadings.writer import FileWriter


def generate_samples():
    for i in range(1000):
        data = i.to_bytes(10000, 'big')
        label = random.randrange(10)
        yield {'key': str(i), 'data': data, 'label': label}


def main():
    with FileWriter('dummy.msgpack') as writer:
        for sample in generate_samples():
            writer.write(sample)
```

The `FileWriter` should be used as a context manager to ensure that it is closed properly. Its `write` method accepts samples as dictionaries with a unique string `key`.

2.5.2 Converting directory trees

Apart from the featured `MsgpackReader` datadings also provides the `DirectoryReader` class to read samples from directory trees. Let's assume your dataset is currently stored in a directory tree like this:

```
yourdataset / a / a1
                / a2
            / b / b3
                / b4
```

You can now simply replace the `generate_samples` function above with a `DirectoryReader`:

```
def main():
    with FileWriter('yourdataset.msgpack') as writer:
        for sample in DirectoryReader('yourdataset/{LABEL}/**'):
            writer.write(sample)
```

The names of the directories at the level marked by `{LABEL}` are used as `label`, the path to the file from the label onwards is used as the `key`, and the file contents are loaded into `data`:

```
{'key': 'a/a1', 'label': 0, 'path': 'yourdataset/a/a1',
 '_additional_info': [], '_label': 'a', 'data': b'content of a1'}
{'key': 'a/a2', 'label': 0, 'path': 'yourdataset/a/a2',
```

(continues on next page)

(continued from previous page)

```
 '_additional_info': [], '_label': 'a', 'data': b'content of a2'}
{'key': 'b/b1', 'label': 1, 'path': 'yourdataset/b/b1',
 '_additional_info': [], '_label': 'b', 'data': b'content of b1'}
{'key': 'b/b2', 'label': 1, 'path': 'yourdataset/b/b2',
 '_additional_info': [], '_label': 'b', 'data': b'content of b2'}
```

You can now make any additional changes to the samples before handing them off to the writer. Check the *reference* for more details on how you can influence its behavior.

If your dataset is not a directory tree, but stored in a ZIP file you can use the *ZipFileReader* instead.

2.5.3 More complex datasets

If your dataset consists of multiple files per sample, needs additional metadata, or it is stored in an unusual way (like a single large TAR file that you don't want to extract), you will need to write additional code to provide the samples. You can take a look at the source code of the *included datasets* like *MIT1003* for pointers.

2.6 Contributing a dataset

To contribute a new dataset to datadings, please follow the steps below and create a merge request in our Gitlab repository.

Each dataset defines modules to read and write in the *datadings.sets* package. Typically the read module contains additional meta-data that is common for all samples, like class labels or distributions. The convention is that for a dataset called `FOO`, these modules are called `FOO` and `FOO_write`.

2.6.1 Metadata

Small amounts of data or data that is not available for download should be added to datadings directly. Examples for this are class labels/distributions/weights/colors, file lists, etc. Anything more than a 1 kiB should be included as zopfli or xz compressed text, JSON or msgpack files to reduce the size of the repository and distributed wheels. zopfli may give slightly better compression for very small files, while xz is vastly superior for larger files. Keep in mind that higher xz levels can require considerable amounts of memory for decompression. This shell script will try `gzip -9`, `zopfli` (if available), and all `xz` levels 0 to 9e and report file size and memory used (kiB) for decompression:

```
#!/bin/bash
set -e

echo -e "comp\tmemory\tsize"
bytes=$(stat -c %s "$FILE")
echo -e "none\t-\t$bytes"

gzip -9 -k -f "$FILE"
bytes=$(ls -l "$FILE.gz" | cut -d " " -f 5)
mem=$( 2>&1 /usr/bin/time -f "%M" gunzip -f -k "$FILE.gz")
echo -e "gzip -9\t$mem\t$bytes"

if command -v zopfli &> /dev/null; then
    zopfli -k -f "$FILE"
    bytes=$(ls -l "$FILE.gz" | cut -d " " -f 5)
```

(continues on next page)

(continued from previous page)

```
    mem=$( 2>&1 /usr/bin/time -f "%M" gunzip -f -k "$FILE.gz")
    echo -e "zopfli\t$mem\t$bytes"
fi

for LEVEL in 0 0e 1 1e 2 2e 3 3e 4 4e 5 5e 6 6e 7 7e 8 8e 9 9e; do
    xz -$LEVEL -k -f "$FILE"
    bytes=$(ls -l "$FILE.xz" | cut -d " " -f 5)
    mem=$( 2>&1 /usr/bin/time -f "%M" unxz -f -k "$FILE.xz")
    echo -e "xz -$LEVEL\t$mem\t$bytes"
done
```

For example here's the sorted output for `ILSVRC2012_val.txt`:

```
$ FILE=ILSVRC2012_val.txt ./testcomp.sh | sort -n -t $'\t' -k 3
comp    memory  size
xz -6   5068    123436
xz -7   4616    123436
xz -8   5140    123436
xz -9   3824    123436
xz -0e  2652    123540
xz -1e  3432    123540
xz -2e  4020    123540
xz -4e  4496    123540
xz -6e  5576    123540
xz -7e  5760    123540
xz -8e  6008    123540
xz -9e  5092    123540
xz -3e  4388    123680
xz -5e  4012    123680
xz -5   4320    125460
xz -2   3952    166608
xz -3   4016    167436
xz -1   3224    167828
xz -0   2720    168588
xz -4   3796    168924
zopfli  3120    201604
gzip -9 3292    229789
none    -       1644500
```

Surprisingly `xz -0e` is the clear winner here, giving excellent compression ratio with very low memory requirements.

2.6.2 Read module

Add a module called `FOO` to the *`datadings.sets`* package and add/load available meta-data, if any. With less complex datasets this module may be empty, but should be added anyway.

More complex datasets

For some datasets it simply does not make sense to convert them to the datadings file format. Typically conversion requires (at least temporarily) roughly twice the space as the original data. With the *`YFCC100m`* dataset for example the conversion would take a very long time and take up many TiB additional space. Other examples are be large video files that should really be streamed while decoding instead of loading all of the data at once, which datadings does not yet support. For these and similar cases it (at least currently) does not make sense to use the datadings msgpack format with the *`MsgpackReader`*. Instead, we recommend the `FOO` module provide a `FOOReader` class that extends `datadings.reader.Reader` or one of its subclasses. An effort should be made to reduce processing times. The `FOOReader` should read directly from the source files of the dataset and perform limited pre-processing. For example, datadings includes a list of samples from the *`YFCC100m`* dataset that are not useful, because they were either damaged or blank images. This slow process of analyzing every image was performed offline to speed up subsequent iterations of the dataset.

2.6.3 Write module

Now add another module called `FOO_write`. This will be an executable that writes dataset files. There are generally four steps to the writing process:

- Argument parsing.
- Download and verify source files.
- Locate and load sample data.
- Convert and write samples to dataset.

If you prefer to learn from code, the *`CAT2000_write`* module is a relatively simple, yet full-featured example.

Argument parsing

Scripts typically lean heavily on the `datadings.argparse` module to parse command line arguments. It provides utility functions like `make_parser` to create argument parsers with sensible default settings and a lot of commonly used arguments already added. For example, most datasets need an `indir` argument, where source files are located, as well as an optional `outdir` argument, which is where the dataset files will be written. The `datadings.argparse` module also provides functions to add a lesser-used arguments in a single line, including descriptive help text. By convention a function called `argument_indir` adds the `indir` argument to the given parser, including additional configuration and help text.

For a simple dataset with no additional arguments, a main function might begin like this:

```
def main():
    from ..argparse import make_parser
    from ..argparse import argument_threads
    from ..tools import prepare_indir

    parser = make_parser(__doc__)
    argument_threads(parser)
```

(continues on next page)

(continued from previous page)

```
    args = parser.parse_args()
    outdir = args.outdir or args.indir
```

Download and verify

If possible, datadings should download source files. This is not possible for all datasets, because data might only be available on request or after registration. If that is the case, add a description to the docstring on how to download the data. Manual pre-processing steps, like unpacking archives, should be avoided if at all possible. The only exception to this rule is if Python is ill-equipped to handle the source file format, e.g., some unusual compression scheme like 7zip.

If downloading is possible, datadings provides some convenient tools to do so. First, define which files are required in a global variable called `FILES`, for example for the Pascal VOC 2012 dataset:

```
BASE_URL = 'http://saliency.mit.edu/'
FILES = {
    'train': {
        'path': 'trainSet.zip',
        'url': BASE_URL+'trainSet.zip',
        'md5': '56ad5c77e6c8f72ed9ef2901628d6e48',
    },
    'test': {
        'path': 'testSet.zip',
        'url': BASE_URL+'testSet.zip',
        'md5': '903ec668df2e5a8470aef9d8654e7985',
    }
}
```

Our example defines `"train"` and `"test"` files, with a relative path, a URL to download them from and and MD5 hash to verify their integrity. The verification step is especially important, since we want to support the reuse of previously downloaded files. So we need to make sure that the file we are using is actually what we expect to find.

This dictionary of file definitions can now be given to helper functions from the *`datadings.tools`* module. Most convenient is *`prepare_indir`*, which first attempts to download (if URL is given) and verify each file. If successful, it then returns a dict where all paths are replaced with the true location of each file.

Our main function now looks like this:

```
def main():
    from ..argparse import make_parser
    from ..argparse import argument_threads
    from ..tools import prepare_indir

    parser = make_parser(__doc__)
    argument_threads(parser)
    args = parser.parse_args()
    outdir = args.outdir or args.indir

    files = prepare_indir(FILES, args)
```

Locate, load, and write data

These steps heavily depend on the structure of the dataset and this guide can only provide general guidelines. We recommended to first define a generator, which loads and yields one sample at a time:

```
def yield_samples(stuff):
    samples = []  # find samples in stuff
    for sample in samples:
        # load data from source file
        yield SampleType(data, metadata, etc)
```

Instead of returning individual values it is recommended to use one of the provided type functions from *datadings.sets.types*. New types can be added if none of them fits your dataset. Type functions are generated by the `generate_types.py` script from the definitions in `generate_types.json`.

The generator is used by a `write_set` function, which is called once per split of the dataset. Here, create a *FileWriter* with the desired output path and pass samples to it:

```
def write_set(split, stuff, outdir, args):
    gen = yield_samples(stuff)
    outfile = pt.join(outdir, split + '.msgpack')
    writer = FileWriter(outfile, total=num_samples, overwrite=args.no_confirm)
    with writer:
        for sample in gen:
            writer.write(sample)
```

Important: Samples must have a unique `"key"`. An exception will be raised if keys are repeated.

Note: If the `overwrite` parameter of the writer is `False`, the user will be prompted to overwrite an existing file. The user can now:

- Accept to overwrite the file.
- Decline, which raises a `FileExistsError`. The program should continue as if writing had finished.
- Abort, which raises a `KeyboardInterrupt`. The program should abort immediately.

The default argument parser accepts a `no_confirm` argument, which is passed to the `overwrite` parameter.

The final function `write_sets` will call `write_set` once per split of the dataset:

```
def write_sets(files, outdir, args):
    for split in ('train', 'test'):
        try:
            write_set(split, files[split]['path'], outdir)
        except FileExistsError:
            continue
        except KeyboardInterrupt:
            break
```

Note: We catch the `FileExistsError` and `KeyboardInterrupt`, which may be raised by the writer.

The final main function now looks like this. We call `write_sets` and wrap the main function itself to catch keyboard interrupts by the user:

```
def main():
    from ..argparse import make_parser
    from ..tools import prepare_indir

    parser = make_parser(__doc__)
    args = parser.parse_args()
    outdir = args.outdir or args.indir

    files = prepare_indir(FILES, args)

    write_sets(files, outdir, args)


if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        pass
    finally:
        print()
```

2.6.4 Writing faster

Since datadings is all about speed and convenience, which are high related when it comes to writing datasets, you may want to optimize your program to increase the write speed. Two relatively simple optimizations are recommended to speed up the process.

First, the generator can be wrapped with the *`datadings.tools.yield_threaded()`* function, which runs the generator in a background thread. This effectively decouples filesystem read and write operations, but does not help if the CPU is the bottleneck.

In those cases where the bottleneck is neither reading nor writing, but a costly conversion (e.g., transcoding images), a thread or process pool can be used to parallelize this step:

```
def write_set(split, stuff, outdir, args):
    gen = yield_threaded(yield_samples(stuff))

    def costly_conversion(sample):
        # do something that you want parallelized
        return sample

    outfile = pt.join(outdir, split + '.msgpack')
    writer = FileWriter(outfile, total=num_samples, overwrite=args.no_confirm)
    pool = ThreadPool(args.threads)
    with writer:
        for sample in pool.imap_unordered(create_sample, gen):
            writer.write(sample)
```

Note: Add `datadings.tools.argument_threads()` to the parser to allow users to control the number of threads.

Note: `imap_unordered` makes no guarantees about the order of the returned samples. If the order is important, consider using a different method like `imap`. Beware though that this may use substantially more memory, as samples are stored in memory until they can be returned in the correct order.

2.7 API reference

2.7.1 datadings.commands package

A number of useful tools are installed with datadings. These will be accessible on the command line as `datadings-*` where * is replaced with one of the submodule names,

The main tool to interact with datadings in this way is *`datadings-write`*. It finds available datasets and runs their writing scripts.

These are the available tools:

- *`datadings-write`* creates new dataset files.
- *`datadings-cat`* prints the (abbreviated) contents of a dataset file.
- *`datadings-shuffle`* shuffles an existing dataset file.
- *`datadings-merge`* merges two or more dataset files.
- *`datadings-split`* splits a dataset file into two or more subsets.
- *`datadings-bench`* runs some basic read performance benchmarks.

You can either call them directly or run them as modules with `python -m datadings.commands.*`, again with star replaced by the name the command, e.g., `write`.

Submodules

datadings.commands.bench module

usage: bench.py [-h] [-r] [-s] [-b BUFFERING] [–separator SEPARATOR] [–root-dir ROOT_DIR] [–include INCLUDE [INCLUDE . . .]] [–exclude EXCLUDE [EXCLUDE . . .]] infile

Run a read benchmark on a given dataset.

Support reading from msgpack files or directory trees.

Note: For directory trees, please refer to the DirectoryReader documentation for details on how to specify the dataset structure.

Positional arguments

infile Input file.

Optional arguments

-h, --help show this help message and exit

-r, --raw Do not decode samples. This is representative of performance in a multi-process environment, where workers take over decoding.

-s, --shuffle Shuffle the reader.

-b BUFFERING, --buffering BUFFERING MsgpackReader only: Buffer size of reader.

--separator SEPARATOR DirectoryReader only: separator for file input mode.

--root-dir ROOT_DIR DirectoryReader only: root directory for file input mode.

–include INCLUDE [INCLUDE ...] DirectoryReader only: Include patterns.

–exclude EXCLUDE [EXCLUDE ...] DirectoryReader only: Exclude patterns.

`datadings.commands.bench.`**`bench`**`(`*readerfun*, *args*`)`

`datadings.commands.bench.`**`entry`**`()`

`datadings.commands.bench.`**`main`**`()`

`datadings.commands.bench.`**`reader_directory`**`(`*args*`)`

`datadings.commands.bench.`**`reader_msgpack`**`(`*args*`)`

datadings.commands.cat module

usage: cat.py [-h] [-s MAXSTRING] infile

Cat msgpack files.

Positional arguments

infile File to cat.

Optional arguments

-h, --help show this help message and exit

-s MAXSTRING, --maxstring MAXSTRING max length of strings

`datadings.commands.cat.`**`cat`**`(`*infile*, *maxstring*`)`

`datadings.commands.cat.`**`entry`**`()`

`datadings.commands.cat.`**`main`**`()`

datadings.commands.convert_index module

usage: convert_index.py [-h] [-o PATH] infile

Convert the legacy index of dataset a to the new style.

Positional arguments

infile Input file.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

`datadings.commands.convert_index.`**`convert_index`**`(`*path*, *outdir*`)`

`datadings.commands.convert_index.`**`entry`**`()`

`datadings.commands.convert_index.`**`main`**`()`

datadings.commands.merge module

usage: merge.py [-h] [-s {concat,random}] [–shuffle] infile [infile . . .] outfile

Merge two or more dataset files.

Available strategies:

- concat: Concat input files in the order they are given.
- random: Choose input file to read next sample from randomly, with probability depending on the relative size of datasets.

Positional arguments

infile Files to merge. outfile Output file.

Optional arguments

-h, --help show this help message and exit

-s {concat,random}, –strategy {concat,random} Merging strategy to use.

--shuffle Shuffle each dataset before merging.

`datadings.commands.merge.`**`cumsum`**`(`*it*`)`

`datadings.commands.merge.`**`entry`**`()`

`datadings.commands.merge.`**`main`**`()`

`datadings.commands.merge.`**`merge_concat`**`(`*infiles*, *outfile*, *shuffle*`)`

`datadings.commands.merge.`**`merge_random`**`(`*infiles*, *outfile*, *shuffle=False*`)`

`datadings.commands.merge.select_set(ranges)`

`datadings.commands.merge.setup_ranges(lens)`

datadings.commands.sample module

usage: sample.py [-h] [-s {sequential,random}] infile outfile number

Extract samples from a dataset.

Positional arguments

infile Input file. outfile Output file. number Number of samples to extract.

Optional arguments

-h, --help show this help message and exit

-s {sequential,random}, –strategy {sequential,random} Sampling strategy to use.

`datadings.commands.sample.entry()`

`datadings.commands.sample.main()`

`datadings.commands.sample.sample(infile, outfile, number, strategy)`

datadings.commands.shuffle module

usage: shuffle.py [-h] [-y] [–true-shuffle] [–buf-size BUF_SIZE] [–chunk-size CHUNK_SIZE] infile outfile

Shuffle an existing dataset file.

Positional arguments

infile Input file. outfile Output file.

Optional arguments

-h, --help show this help message and exit

-y, --no-confirm Don't require user interaction.

--true-shuffle Use slower but more random shuffling algorithm

--buf-size BUF_SIZE size of the shuffling buffer for fast shuffling; values less than 1 are interpreted as fractions of the dataset length; bigger values improve randomness, but use more memory

--chunk-size CHUNK_SIZE size of chunks read by the fast shuffling algorithm; bigger values improve performance, but reduce randomness

`datadings.commands.shuffle.entry()`

`datadings.commands.shuffle.main()`

`datadings.commands.shuffle.shuffle(infile, outfile, args)`

datadings.commands.split module

usage: split.py [-h] [-o PATH [PATH . . .]] [-y] infile split [split . . .]

Splits one dataset into several smaller ones.

For example two split positions A and B produce three output files:

[. . . . .file1. . . . .|A|. . . . .file2. . . . .|B|. . . . .file3. . . . .]

File 1 contains samples 0 to A-1. File 2 contains samples A to B-1. File 3 contains samples B to end of input.

Positional arguments

infile File to split. split Index where infile is split.

Optional arguments

-h, --help	show this help message and exit
-o PATH [PATH . . .], –outfiles PATH [PATH . . .]	Output files.
-y, --no-confirm	Don't require user interaction.

`datadings.commands.split.entry()`

`datadings.commands.split.main()`

`datadings.commands.split.split_dataset(infile, outfiles, splits, overwrite)`

datadings.commands.write module

usage: write.py [-h] [dataset]

Create dataset files. The following datasets are supported:

A: *ADE20k*, *ANP460*

C: *CAMVID*, *CAT2000*, *CIFAR10*, *CIFAR100*, *Cityscapes*, *Coutrot1*

F: *FIGRIMFixation*

I: *ILSVRC2012*, *ImageNet21k*, *InriaBuildings*

M: *MIT300*, *MIT1003*

P: *Places365*, *Places2017*

R: *RIT18*

S: *SALICON2015*, *SALICON2017*

V: *VOC2012*, *Vaihingen*

The help text and documentation page for each dataset contains more information about requirements and possible options. Run

```
python -m datadings.commands.write [dataset] -h
```

or

`datadings-write [dataset] -h`

to view them.

Positional arguments

dataset Dataset to write.

Optional arguments

-h, --help	show this help message and exit

`datadings.commands.write.`**`entry`**`()`

`datadings.commands.write.`**`find_writers`**`()`

`datadings.commands.write.`**`format_writers`**`(`*writers*`)`

`datadings.commands.write.`**`main`**`()`

`datadings.commands.write.`**`writer_link`**`(`*w*`)`

2.7.2 datadings.index package

`datadings.index.`**`hash_keys`**`(`*keys: Sequence[str], max_tries: int = 1000*`)` → Tuple[bytes, Sequence[int]]

Apply the `hash_string()` function to the given list of keys, so the returned hashes are 64 bit integers. All hashes are salted and guaranteed collision free. If necessary this method will try different salt values

Parameters

- **`keys`** – list of keys
- **`max_tries`** – how many different salt values to try to find collision-free hashes

Returns used salt and list of key hashes

`datadings.index.`**`keys_len`**`(`*path: pathlib.Path*`)` → int

Read the dataset length from the keys file.

Correct suffix is appended if path ends with a different suffix.

Parameters **`path`** – path to data or keys file

Returns length of dataset

`datadings.index.`**`legacy_index_len`**`(`*path: pathlib.Path*`)` → int

Read the dataset length from the legacy index file.

Correct suffix is appended if path ends with a different suffix.

Parameters **`path`** – path to data or index file

Returns length of dataset

`datadings.index.`**`legacy_load_index`**`(`*path: pathlib.Path*`)` → Tuple[Sequence[str], Sequence[int]]

Load legacy index as two lists of keys and offsets. Semantics of the returned lists are the same as for `load_keys` and `load_offsets`.

Correct suffix is appended if path ends with a different suffix.

Parameters **path** – Path to dataset or index file

Returns keys and offsets list

`datadings.index.load_filter`(*path: pathlib.Path*) → simplebloom.bloom.BloomFilter

Load a Bloom filter from file.

Correct suffix is appended if path ends with a different suffix.

Parameters **path** – path to data or filter file

Returns the Bloom filter

`datadings.index.load_key_hashes`(*path: pathlib.Path*) → Tuple[bytes, Sequence[int]]

Load key hashes from file.

Correct suffix is appended if path ends with a different suffix.

Parameters **path** – path to data or key hashes file

Returns hash salt and list of key hashes

`datadings.index.load_keys`(*path: pathlib.Path*) → Sequence[str]

Load keys from file.

Correct suffix is appended if path ends with a different suffix.

Parameters **path** – path to data or keys file

Returns list of keys

`datadings.index.load_offsets`(*path: pathlib.Path*) → Sequence[int]

Load sample offsets from file. First value is always 0 and last is size of data file in bytes, so `len(offsets) = len(dataset) + 1`.

Correct suffix is appended if path ends with a different suffix.

Parameters **path** – path to data or offsets file

Returns sample offsets in data file

`datadings.index.write_filter`(*keys: Sequence[str], path: pathlib.Path*) → pathlib.Path

Create a Bloom filter for the given keys and write result to file.

Correct suffix is appended if path ends with a different suffix.

Parameters

- **keys** – list of keys
- **path** – path to data or filter file

Returns Path that was written to

`datadings.index.write_key_hashes`(*keys: Sequence[str], path: pathlib.Path*) → pathlib.Path

Hash list of keys and write result to file.

See `hash_keys` for details on hash method.

Correct suffix is appended if path ends with a different suffix.

Parameters

- **keys** – list of keys
- **path** – path to data or offsets file

Returns Path that was written to

`datadings.index.write_keys`(*keys: Sequence[str]*, *path: pathlib.Path*) → pathlib.Path

Write list of offsets to file.

Correct suffix is appended if path ends with a different suffix.

Parameters

- **keys** – list of keys
- **path** – path to data or keys file

Returns Path that was written to

`datadings.index.write_offsets`(*offsets: Sequence[int]*, *path: pathlib.Path*) → pathlib.Path

Write list of offsets to file.

Correct suffix is appended if path ends with a different suffix.

Parameters

- **offsets** – list of offsets
- **path** – path to data or offsets file

Returns Path that was written to

2.7.3 datadings.reader package

Submodules

datadings.reader.augment module

An Augment wraps a `Reader <datadings.reader.reader.Reader` and changes how samples are iterated over. How readers are used is largely unaffected.

class `datadings.reader.augment.Cycler`(*reader*)

Bases: `datadings.reader.augment.Augment`

Infinitely cycle a `Reader <datadings.reader.reader.Reader`.

> **Warning:** Augments are not thread safe!

iter(*yield_key=False*, *raw=False*, *copy=True*, *chunk_size=16*)

Create an iterator.

Parameters

- **yield_key** – if True, yields (key, sample) pairs.
- **raw** – if True, yields samples as msgpacked messages.
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`
- **chunk_size** – number of samples read at once; bigger values can increase throughput, but also memory

Returns Iterator

seek(*index*)

class datadings.reader.augment.**QuasiShuffler**(*reader*, *buf_size=0.01*, *chunk_size=16*, *seed=None*)

Bases: datadings.reader.augment.Augment

A slightly less random than a true Reader <datadings.reader.augment.Shuffler but much faster.

The dataset is divided into equal-size chunks that are read in random order. Shuffling follows these steps:

1. Fill the buffer with chunks.
2. Read the next chunk.
3. Select a random sample from the buffer and yield it.
4. Replace the sample with the next sample from the current chunk.
5. If there are chunks left, goto 2.

This means there are typically more samples from the current chunk in the buffer than there would be if a true shuffle was used. This effect is more pronounced for smaller fractions $\frac{B}{C}$ where C is the chunk size and B the buffer size. As a rule of thumb it is sufficient to keep $\frac{B}{C}$ roughly equal to the number of classes in the dataset.

Note: Seeking and resuming iteration with a new iterator are relatively costly operations. If possible create one iterator and use it repeatedly.

Parameters

- **reader** – the reader to wrap
- **buf_size** – size of the buffer; values less than 1 are interpreted as fractions of the dataset length; bigger values improve randomness, but use more memory
- **chunk_size** – size of each chunk; bigger values improve performance, but reduce randomness
- **seed** – random seed to use; defaults to len(reader) * self.buf_size * chunk_size

iter(*yield_key=False*, *raw=False*, *copy=True*, *chunk_size=None*)

Create an iterator.

Parameters

- **yield_key** – if True, yields (key, sample) pairs.
- **raw** – if True, yields samples as msgpacked messages.
- **copy** – if False, allow the reader to return data as memoryview objects instead of bytes
- **chunk_size** – number of samples read at once; bigger values can increase throughput, but also memory

Returns Iterator

seek(*index*)

class datadings.reader.augment.**Range**(*reader*, *start=0*, *stop=None*)

Bases: datadings.reader.augment.Augment

Extract a range of samples from a given reader.

start and stop behave like the parameters of the range function.

Parameters

- **reader** – reader to sample from

- **start** – start of range
- **stop** – stop of range

iter(*yield_key=False*, *raw=False*, *copy=True*, *chunk_size=16*)
Create an iterator.

Parameters

- **yield_key** – if True, yields (key, sample) pairs.
- **raw** – if True, yields samples as msgpacked messages.
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`
- **chunk_size** – number of samples read at once; bigger values can increase throughput, but also memory

Returns Iterator

seek(*index*)

class datadings.reader.augment.**Repeater**(*reader*, *times*)
Bases: `datadings.reader.augment.Augment`

Repeat a `Reader <datadings.reader.reader.Reader` a fixed number of times.

> **Warning:** Augments are not thread safe!

iter(*yield_key=False*, *raw=False*, *copy=True*, *chunk_size=16*)
Create an iterator.

Parameters

- **yield_key** – if True, yields (key, sample) pairs.
- **raw** – if True, yields samples as msgpacked messages.
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`
- **chunk_size** – number of samples read at once; bigger values can increase throughput, but also memory

Returns Iterator

seek(*index*)

class datadings.reader.augment.**Shuffler**(*reader*, *seed=None*)
Bases: `datadings.reader.augment.Augment`

Iterate over a `Reader <datadings.reader.reader.Reader` in random order.

Parameters

- **reader** – The reader to augment.
- **seed** – optional random seed; defaults to len(reader)

> **Warning:** Augments are not thread safe!

iter(*yield_key=False*, *raw=False*, *copy=True*, *chunk_size=16*)
Create an iterator.

Parameters

- **yield_key** – if True, yields (key, sample) pairs.
- **raw** – if True, yields samples as msgpacked messages.
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`
- **chunk_size** – number of samples read at once; bigger values can increase throughput, but also memory

Returns Iterator

seek(*index*)

datadings.reader.directory module

class datadings.reader.directory.**DirectoryReader**(*patterns: Sequence[Union[str, pathlib.Path]], labels: Optional[Union[Iterable, pathlib.Path]] = None, numeric_labels=True, initfun: Callable = <function noop>, convertfun: Callable = <function noop>, include: Sequence[str] = (), exclude: Sequence[str] = (), separator='\t', root_dir=''*)

Bases: *datadings.reader.list.ListReader*

Reader that loads samples from one or multiple filesystem directories.

One or more search patterns must be given to tell the reader where to look for samples. Each search pattern can either be:

- A glob pattern to a filesystem directory. Use the special `{LABEL}` string to define which directory in the path to use as a label.
- A path to a CSV-like file (with the given `separator` string) where each line contains the path to a sample file. Paths can be relative and optionally prefixed with a `root_dir`. A label as well as additional information can be included besides the path in additional columns. They will be stored as `"label"` and `"_additional_info"`.

Example glob pattern: `some_dir/{LABEL}/**`

This patterns loads a dataset with a typical directory tree structure where samples from each class are located in separate subdirectories. The name of the directory at the level of `{LABEL}` is used as the label.

You can further narrow down which files to include with additional `fnmatch.fnmatch()` glob patterns. These are applied as follows:

- If no inclusion patterns are given, all files are included.
- If inclusion patterns are given, a file must match at least one.
- A file is excluded if it matches any exclusion patterns.

Note: Please refer to the *ListReader* documentation for a more detailed explanation on how labels are handled.

Parameters

- **patterns** – One or more search patterns.
- **labels** – Optional. List of labels in desired order, or path to file with one label per line. If `None`, get `"label"` keys from samples, if any, and sort.

- **numeric_labels** – If true, convert labels to numeric index to list of all labels.
- **initfun** – Callable `initfun(sample: dict)` to modify samples in-place during initialization.
- **convertfun** – Callable `convertfun(sample: dict)` to modify samples in-place before they are returned.
- **include** – Set of inclusion patterns.
- **exclude** – Set of exclusion patterns.
- **separator** – Separator string for file patterns.
- **root_dir** – Prefix for relative paths.

`datadings.reader.directory.`**`check_included`**(*filename*, *include*, *exclude*)

`datadings.reader.directory.`**`glob_pattern`**(*pattern*, *prefix*)

`datadings.reader.directory.`**`yield_directory`**(*patterns*, *separator*)

`datadings.reader.directory.`**`yield_file`**(*infile*, *prefix*, *separator*)

datadings.reader.list module

class `datadings.reader.list.`**`ListReader`**(*samples: Sequence[dict], labels: Optional[Union[Iterable, pathlib.Path]] = None, numeric_labels=True, initfun: Callable = <function noop>, convertfun: Callable = <function noop>*)

Bases: `datadings.reader.reader.Reader`

Reader that holds a list of samples. Functions can be given to load data on the fly and/or perform further conversion steps.

Two special keys `"key"` and `"label"` in samples are used:

- `"key"` is a unique identifier for samples. Sample index is added to samples if it is missing.
- `"label"` holds an optional label. Replaced by a numeric index to the list of labels if `numeric_labels` is true. The original label is retained as `"_label"`.

Note: If `labels` argument is not given, the list of all labels will be extracted from all samples. The list of all labels is `natsorted` to determine numerical labels.

Note: `initfun` is applied to the given samples during initialization and thus remain for the life of the reader. `convertfun` is applied to a shallow copy of the sample every time before it is returned.

Important: Since `None` is not sortable, the `labels` argument must be given to use `None` as a label.

Parameters

- **samples** – Sequence of samples. Must be indexable, so no generators or one-time iterators.
- **labels** – Optional. List of labels in desired order, or path to file with one label per line. If `None`, get `"label"` keys from samples, if any, and sort.

- **numeric_labels** – If true, convert labels to numeric index to list of all labels.
- **initfun** – Callable `initfun(sample: dict)` to modify samples in-place during initialization.
- **convertfun** – Callable `convertfun(sample: dict)`. Applied to shallow copies of samples before they are returned.
- **convertfun** – Callable `convertfun(sample: dict)` to modify a shallow copy of samples in-place before they are returned.

find_index(*key*)
Returns the index of the sample with the given key.

find_key(*index*)
Returns the key of the sample with the given index.

get(*index*, *yield_key=False*, *raw=False*, *copy=True*)
Returns sample at given index.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **index** – Index of the sample
- **yield_key** – If True, returns (key, sample)
- **raw** – If True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Sample as index.

slice(*start*, *stop=None*, *yield_key=False*, *raw=False*, *copy=True*)
Returns a generator of samples selected by the given slice.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **start** – start index of slice
- **stop** – stop index of slice
- **yield_key** – if True, yield (key, sample)
- **raw** – if True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Iterator of selected samples

datadings.reader.list.**load_lines**(*path*)

datadings.reader.list.**noop**(_)

datadings.reader.list.**sorted_labels**(*samples*)

datadings.reader.msgpack module

class datadings.reader.msgpack.**MsgpackReader**(*path: Union[str, pathlib.Path], buffering=0*)

Bases: datadings.reader.reader.Reader

Reader for msgpack files in the *datadings format description*.

Needs at least data and index file. For example, if the dataset file is `some_dir/dataset.msgpack`, then the reader will attempt to load the index from `some_dir/dataset.msgpack.index`.

Can optionally verify the integrity of data and index files if the md5 file `some_dir/dataset.msgpack.md5` is present.

Parameters

- **path** – Dataset file to load.
- **buffering** – Read buffer size in bytes.

Raises IOError – If dataset or index cannot be loaded.

find_index(*key*)

Returns the index of the sample with the given key.

find_key(*index*)

Returns the key of the sample with the given index.

get(*index, yield_key=False, raw=False, copy=True*)

Returns sample at given index.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **index** – Index of the sample
- **yield_key** – If True, returns (key, sample)
- **raw** – If True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Sample as index.

slice(*start, stop=None, yield_key=False, raw=False, copy=True*)

Returns a generator of samples selected by the given slice.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **start** – start index of slice
- **stop** – stop index of slice
- **yield_key** – if True, yield (key, sample)
- **raw** – if True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Iterator of selected samples

verify_data(*read_size=524288*, *progress=False*)
Hash the dataset file and verify against the md5 file.

Parameters

- **read_size** – Read-ahead size in bytes.
- **progress** – display progress

Returns True if verification was successful.

verify_index(*read_size=524288*, *progress=False*)
Hash the index file and verify against the md5 file.

Parameters

- **read_size** – Read-ahead size in bytes.
- **progress** – display progress

Returns True if verification was successful.

datadings.reader.reader module

class datadings.reader.reader.**Reader**
Bases: object

Abstract base class for dataset readers.

Readers should be used as context managers:

```
with Reader(...) as reader:
    for sample in reader:
        [do dataset things]
```

Subclasses must implement the following methods:

- __exit__
- __len__
- __contains__
- find_key
- find_index
- get
- slice

abstract find_index(*key*)
Returns the index of the sample with the given key.

abstract find_key(*index*)
Returns the key of the sample with the given index.

abstract get(*index*, *yield_key=False*, *raw=False*, *copy=True*)
Returns sample at given index.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **index** – Index of the sample
- **yield_key** – If True, returns (key, sample)
- **raw** – If True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Sample as index.

iter(*start=None*, *stop=None*, *yield_key=False*, *raw=False*, *copy=True*, *chunk_size=16*)

Iterate over the dataset.

`start` and `stop` behave like the parameters of the `range` function0.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **start** – start of range; if None, current index is used
- **stop** – stop of range
- **yield_key** – if True, yields (key, sample) pairs.
- **raw** – if True, yields samples as msgpacked messages.
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`
- **chunk_size** – number of samples read at once; bigger values can increase throughput, but require more memory

Returns Iterator

next()

Returns the next sample.

This can be slow for file-based readers if a lot of samples are to be read. Consider using iter instead:

```
it = iter(reader)
while 1:
    next(it)
    ...
```

Or simply loop over the reader:

```
for sample in reader:
    ...
```

rawiter(*yield_key=False*)

Create an iterator that yields samples as msgpacked messages.

Included for backwards compatibility and may be deprecated and subsequently removed in the future.

Parameters **yield_key** – If True, yields (key, sample) pairs.

Returns Iterator

rawnext() → bytes

Return the next sample msgpacked as raw bytes.

This can be slow for file-based readers if a lot of samples are to be read. Consider using iter instead:

```
it = iter(reader)
while 1:
    next(it)
    ...
```

Or simply loop over the reader:

```
for sample in reader:
    ...
```

Included for backwards compatibility and may be deprecated and subsequently removed in the future.

seek(*index*)
Seek to the given index. Alias for `seek_index`.

seek_index(*index*)
Seek to the given index.

seek_key(*key*)
Seek to the sample with the given key.

abstract slice(*start*, *stop=None*, *yield_key=False*, *raw=False*, *copy=True*)
Returns a generator of samples selected by the given slice.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **start** – start index of slice
- **stop** – stop index of slice
- **yield_key** – if True, yield (key, sample)
- **raw** – if True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Iterator of selected samples

datadings.reader.sharded module

class datadings.reader.sharded.**ShardedReader**(*shards: Union[str, pathlib.Path, Iterable[Union[str, pathlib.Path,* datadings.reader.reader.Reader*]]]*)

Bases: *datadings.reader.reader.Reader*

A Reader that combines several shards into one. Shards can be specified either as a glob pattern `dir/*.msgpack` for msgpack files, or an iterable of individual shards. Each shard can be a string, `Path`, or *`Reader`*.

Parameters **shards** – glob pattern or a list of strings, Path objects or Readers

find_index(*key*)
Returns the index of the sample with the given key.

find_key(*index*)
Returns the key of the sample with the given index.

get(*index*, *yield_key=False*, *raw=False*, *copy=True*)
Returns sample at given index.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **index** – Index of the sample
- **yield_key** – If True, returns (key, sample)
- **raw** – If True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Sample as index.

slice(*start*, *stop=None*, *yield_key=False*, *raw=False*, *copy=True*)
Returns a generator of samples selected by the given slice.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **start** – start index of slice
- **stop** – stop index of slice
- **yield_key** – if True, yield (key, sample)
- **raw** – if True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Iterator of selected samples

datadings.reader.zipfile module

class datadings.reader.zipfile.**ZipFileReader**(*path: Union[str, pathlib.Path], patterns: Sequence[Union[str, pathlib.Path]] = '{LABEL}/**', labels=None, numeric_labels=True, initfun: Callable = <function noop>, convertfun: Callable = <function noop>, include=(), exclude=(), separator='\t'*)

Bases: *datadings.reader.list.ListReader*

Reader that reads files from a ZIP file.

Note: Functionally identical to the `DirectoryReader`, expect it reads from ZIP files instead of filesystem directories. Please refer to its documentation for more detailed explanations.

Parameters

- **patterns** – One or more search patterns.
- **labels** – Optional. List of labels in desired order, or path to file with one label per line. If `None`, get `"label"` keys from samples, if any, and sort.

- **numeric_labels** – If true, convert labels to numeric index to list of all labels.
- **initfun** – Callable `initfun(sample: dict)` to modify samples in-place during initialization.
- **convertfun** – Callable `convertfun(sample: dict)` to modify samples in-place before they are returned.
- **include** – Set of inclusion patterns.
- **exclude** – Set of exclusion patterns.
- **separator** – Separator string for file patterns.

`datadings.reader.zipfile.`**`glob_pattern`**(*infos*, *pattern*)

`datadings.reader.zipfile.`**`yield_zipfile`**(*zipfile_*, *patterns*, *separator*)

2.7.4 datadings.sets package

This package contains modules to create dataset files for all supported datasets, as well as metadata required to work with them.

Submodules

datadings.sets.ADE20k module

`datadings.sets.ADE20k.`**`index_to_color`**(*array*, *_index_array=array([0, 1, 3, ..., 3143, 3144, 3145], dtype=uint16)*)

`datadings.sets.ADE20k.`**`load_statistics`**(*name*)

datadings.sets.ADE20k_write module

usage: ADE20k_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] [-t 0-2] [–calculate-weights] [–scenelabels] INDIR

Create ADE20k data set files.

This tool will look for the following files in the input directory and download them if necessary:

- ADE20K_2016_07_26.zip

See also:

http://groups.csail.mit.edu/vision/datasets/ADE20K

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label"`
- `"label_image"`
- `"parts_images"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes) -t 0-2, –threads 0-2

Number of threads for conversion. 0 uses all available CPUs (default 2).

--calculate-weights Calculate median-frequency class weights.

--scenelabels extract list of scene labels

`datadings.sets.ADE20k_write.`**`array_to_image`**(*array*, *format*, *dtype*, *mode*, ***kwargs*)

`datadings.sets.ADE20k_write.`**`array_to_png16`**(*array*)

`datadings.sets.ADE20k_write.`**`calculate_weights`**(*files*, *outdir*)

`datadings.sets.ADE20k_write.`**`extract_scenelabels`**(*files*, *outdir*)

`datadings.sets.ADE20k_write.`**`imagedata_to_segpng`**(*data*)

`datadings.sets.ADE20k_write.`**`instance_map`**(*im*)

`datadings.sets.ADE20k_write.`**`load_index`**(*imagezip*, **keys*)

`datadings.sets.ADE20k_write.`**`main`**()

`datadings.sets.ADE20k_write.`**`segmentation_map`**(*im*)

`datadings.sets.ADE20k_write.`**`write_set`**(*imagezip*, *outdir*, *split*, *scenelabels*, *args*)

`datadings.sets.ADE20k_write.`**`write_sets`**(*files*, *outdir*, *args*)

`datadings.sets.ADE20k_write.`**`yield_images`**(*names*)

datadings.sets.ANP460 module

datadings.sets.ANP460_write module

usage: ANP460_write.py [-h] [-o PATH] INDIR

Create ANP460 data set files.

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"experiments"`
- `"anp"`

- "type"

Each experiment has the following keys:

- "locations"
- "map"
- "answer"
- "duration"

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

`datadings.sets.ANP460_write.main()`

`datadings.sets.ANP460_write.write_image(imagezip, datazip, anp_list, txt_files, stimuluspath, writer)`

`datadings.sets.ANP460_write.write_sets(indir, outdir, shuffle=True)`

datadings.sets.CAMVID module

`datadings.sets.CAMVID.index_to_color(array)`

datadings.sets.CAMVID_write module

usage: CAMVID_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] [–calculate-weights] INDIR

Create CamVid data set files.

This tool will look for the following files in the input directory and download them if necessary:

- CamVid.zip (https://github.com/alexgkendall/SegNet-Tutorial/archive/fcaf7c4978dd8d091ec67db7cb7fdd225f5051c5.zip)

See also:

- http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid/
- https://github.com/alexgkendall/SegNet-Tutorial

Important: Samples have the following keys:

- "key"
- "image"
- "label_image"

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes) –calculate-weights Calculate median-frequency class weights.

`datadings.sets.CAMVID_write.`**`calculate_weights`**(*files*)

`datadings.sets.CAMVID_write.`**`imagedata_to_array`**(*data*)

`datadings.sets.CAMVID_write.`**`main`**()

`datadings.sets.CAMVID_write.`**`write_image`**(*imagezip*, *writer*, *inpath*, *outpath*)

`datadings.sets.CAMVID_write.`**`write_set`**(*imagezip*, *outdir*, *split*, *files*, *args*)

`datadings.sets.CAMVID_write.`**`write_sets`**(*files*, *outdir*, *args*)

datadings.sets.CAT2000 module

datadings.sets.CAT2000_write module

usage: CAT2000_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] [-t 0-2] INDIR

Create CAT2000 data set files.

This tool will look for the following files in the input directory and download them if necessary:

- trainSet.zip
- testSet.zip

See also:

http://saliency.mit.edu/results_cat2000.html

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"experiments"`

Each experiment has the following keys:

- `"locations"`
- `"map"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes) -t 0-2, –threads 0-2

Number of threads for conversion. 0 uses all available CPUs (default 1).

`datadings.sets.CAT2000_write.`**`create_sample`**`(`*item*`)`

`datadings.sets.CAT2000_write.`**`filter_invalid_fixpoints`**`(`*points*, *size*`)`

`datadings.sets.CAT2000_write.`**`find_fixpoints`**`(`*arr*`)`

`datadings.sets.CAT2000_write.`**`main`**`()`

`datadings.sets.CAT2000_write.`**`transform_points`**`(`*points*, *offset*, *scale_factor*`)`

`datadings.sets.CAT2000_write.`**`write_set`**`(`*imagezip*, *outdir*, *split*, *args*`)`

`datadings.sets.CAT2000_write.`**`write_sets`**`(`*files*, *outdir*, *args*`)`

`datadings.sets.CAT2000_write.`**`yield_samples`**`(`*imagezip*, *names*`)`

datadings.sets.CIFAR10 module

datadings.sets.CIFAR100 module

datadings.sets.CIFAR100_write module

usage: CIFAR100_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] INDIR

Create CIFAR 100 data set files.

This tool will look for the following files in the input directory and download them if necessary:

- cifar-100-python.tar.gz

See also:

https://www.cs.toronto.edu/~kriz/cifar.html

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label"`
- `"coarse_label"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes)

`datadings.sets.CIFAR100_write.main()`

`datadings.sets.CIFAR100_write.write_set(tar, outdir, split, args)`

`datadings.sets.CIFAR100_write.write_sets(files, outdir, args)`

`datadings.sets.CIFAR100_write.yield_rows(files)`

datadings.sets.CIFAR10_write module

usage: CIFAR10_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] INDIR

Create CIFAR 10 data set files.

This tool will look for the following files in the input directory and download them if necessary:

- cifar-10-python.tar.gz

See also:

https://www.cs.toronto.edu/~kriz/cifar.html

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes)

`datadings.sets.CIFAR10_write.`**`get_files`**(*tar*, *prefix*, *names*)

`datadings.sets.CIFAR10_write.`**`main`**()

`datadings.sets.CIFAR10_write.`**`row2image`**(*row*)

`datadings.sets.CIFAR10_write.`**`write_set`**(*tar*, *outdir*, *split*, *args*)

`datadings.sets.CIFAR10_write.`**`write_sets`**(*files*, *outdir*, *args*)

`datadings.sets.CIFAR10_write.`**`yield_rows`**(*files*)

datadings.sets.Cityscapes module

datadings.sets.Cityscapes_write module

usage: Cityscapes_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] [-t 0-2] INDIR

Create CityScapes data set files.

This tool will look for the following files in the input directory:

- disparity_trainvaltest.zip
- gtFine_trainvaltest.zip
- leftImg8bit_trainvaltest.zip

Note: Registration is required to download this dataset. Please visit the website to download it.

See also:

https://www.cityscapes-dataset.com/

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label_image"`
- `"disparity_image"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes) -t 0-2, –threads 0-2

Number of threads for conversion. 0 uses all available CPUs (default 1).

`datadings.sets.Cityscapes_write.`**`get_keys`**(*leftzip*, *split*)

`datadings.sets.Cityscapes_write.`**`main`**()

`datadings.sets.Cityscapes_write.`**`write_set`**(*outdir*, *split*, *gen*, *total*)

`datadings.sets.Cityscapes_write.`**`write_sets`**(*files*, *outdir*, *args*)

`datadings.sets.Cityscapes_write.`**`yield_samples`**(*keys*, *leftzip*, *disparityzip*, *gtzip*)

datadings.sets.Coutrot1 module

datadings.sets.Coutrot1_write module

usage: Coutrot1_write.py [-h] [-o PATH] [-y] [-s] INDIR

Create Coutrot 1 data set files.

This tool will look for the following files in the input directory and download them if necessary:

- coutrot_database1.mat
- ERB3_Stimuli.zip

See also:

http://antoinecoutrot.magix.net/public/databases.html

Note: ERB3_Stimuli.zip must be downloaded manually as it is hosted on mega.nz.

Warning: OpenCV 2.4+ is required to create this dataset. If you are unsure how to install OpenCV you can use pip:

```
pip install opencv-python
```

Important: Samples are extracted from video frames and thus NOT SHUFFLED! If this is not desirable the datadings-shuffle command can be used to create a shuffled copy.

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"experiments"`

Each experiment has the following keys:

- `"locations"`
- `"map"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

class `datadings.sets.Coutrot1_write.`**`LucasKanade`**

Bases: `object`

`track`(*xy*, *pointid*)

`update`(*frame*)

Update the tracking result for a new frame.

Parameters **`frame`** – Next frame.

Returns Dict of (point, (x, y)) pairs.

`datadings.sets.Coutrot1_write.`**`iter_frames_with_fixpoints`**(*frame_gen*, *experiments*)

`datadings.sets.Coutrot1_write.`**`iter_video_frames_opencv`**(*path*)

`datadings.sets.Coutrot1_write.`**`main`**()

`datadings.sets.Coutrot1_write.`**`write_sets`**(*files*, *indir*, *outdir*, *args*)

`datadings.sets.Coutrot1_write.`**`write_video`**(*name_prefix*, *frame_gen*, *experiments*, *writer*, *min_fixpoints=30*, *write_delta=10*, *max_fixpoint_age=60*)

datadings.sets.FIGRIMFixation module

datadings.sets.FIGRIMFixation_write module

usage: FIGRIMFixation_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] INDIR

Create FIGRIM Fixation data set files.

This tool will look for the following files in the input directory and download them if necessary:

- Targets.zip
- allImages_release.mat
- Fillers.zip
- allImages_fillers.mat

See also:

http://figrim.mit.edu/index_eyetracking.html

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"experiments"`

Each experiment has the following keys:

- `"locations"`
- `"map"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help	show this help message and exit
-o PATH, --outdir PATH	Output directory. Defaults to indir.
-y, --no-confirm	Don't require user interaction.
-s, --skip-verification	Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes)

`datadings.sets.FIGRIMFixation_write.`**`main`**`()`

`datadings.sets.FIGRIMFixation_write.`**`write_set`**`(`*split*, *files*, *outdir*, *args*`)`

`datadings.sets.FIGRIMFixation_write.`**`write_sets`**`(`*files_target*, *files_filler*, *outdir*, *args*`)`

datadings.sets.ILSVRC2012 module

datadings.sets.ILSVRC2012_synsets module

datadings.sets.ILSVRC2012_write module

usage: ILSVRC2012_write.py [-h] [-o PATH] [-y] [-s] [-t 0-2] [–compress [quality 0-100]] [–subsampling {444,422,420,440,411,Gray}] INDIR

Create ILSVRC 2012 challenge data set files.

This tool will look for the following files in the input directory:

- ILSVRC2012_img_train.tar
- ILSVRC2012_img_val.tar

See also:

http://image-net.org/challenges/LSVRC/2012/index

Note: Registration is required to download this dataset. Please visit the website to download it.

Important: For performance reasons samples are read in same order as they are stored in the source tar files. It is recommended to use the datadings-shuffle command to create a shuffled copy.

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

-t 0-2, –threads 0-2 Number of threads for conversion. 0 uses all available CPUs (default 1).

–compress [quality 0-100] Use JPEG compression with optional quality. Default quality is 85. Big images are resized to roughly fit 500x375.

–subsampling {444,422,420,440,411,Gray} Color subsampling factor used with compress option. 444 is forced for small images to preserve details.

`datadings.sets.ILSVRC2012_write.argument_compress(parser)`

`datadings.sets.ILSVRC2012_write.argument_subsampling(parser)`

`datadings.sets.ILSVRC2012_write.main()`

`datadings.sets.ILSVRC2012_write.verify_image(data, quality=None, short_side=375, long_side=500, colorsubsampling='422')`

`datadings.sets.ILSVRC2012_write.write_set(split, outdir, gen, args)`

`datadings.sets.ILSVRC2012_write.write_sets(files, outdir, args)`

`datadings.sets.ILSVRC2012_write.yield_samples(split, tar)`

`datadings.sets.ILSVRC2012_write.yield_train(tar)`

`datadings.sets.ILSVRC2012_write.yield_val(tar)`

datadings.sets.ImageNet21k_create_meta module

This module is used to convert/create required metadata for the ImageNet21k dataset (winter release) to be included with datadings.

This tool will look for the following files in the input directory:

- imagenet21k_miil_tree.pth
- winter21_whole.tar.gz

Important: PyTorch is required to load imagenet21k_miil_tree.pth. The more lightweight CPU-only version is sufficient.

See also:

http://image-net.org/download https://github.com/Alibaba-MIIL/ImageNet21K

Note: Registration is required to download winter21_whole.tar.gz. Please visit the website to download it. If you experience issues downloading you may consider using bittorrent: https://academictorrents.com/details/8ec0d8df0fbb507594557bce993920442f4f6477

`datadings.sets.ImageNet21k_create_meta.convert_semantic_tree(infile)`

`datadings.sets.ImageNet21k_create_meta.extract_synset_counts(infile)`

`datadings.sets.ImageNet21k_create_meta.main()`

datadings.sets.ImageNet21k_synsets module

datadings.sets.ImageNet21k_write module

usage: ImageNet21k_write.py [-h] [-o PATH] [-y] [-s] [-t 0-2] [–compress [quality 0-100]] [–subsampling {444,422,420,440,411,Gray}] INDIR

Create ImageNet21k winter release data set files.

This tool will look for the following files in the input directory:

- winter21_whole.tar.gz

See also:

http://image-net.org/download https://github.com/Alibaba-MIIL/ImageNet21K

Note: Registration is required to download this dataset. Please visit the website to download it. If you experience issues downloading you may consider using bittorrent: https://academictorrents.com/details/8ec0d8df0fbb507594557bce993920442f4f6477

Important: For performance reasons samples are read in same order as they are stored in the source tar files. It is recommended to use the datadings-shuffle command to create a shuffled copy.

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label"`
- `"label_tree"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

-t 0-2, –threads 0-2 Number of threads for conversion. 0 uses all available CPUs (default 1).

–compress [quality 0-100] Use JPEG compression with optional quality. Default quality is 85. Big images are resized to roughly fit 500x375.

–subsampling {444,422,420,440,411,Gray} Color subsampling factor used with compress option. 444 is forced for small images to preserve details.

`datadings.sets.ImageNet21k_write.main()`

`datadings.sets.ImageNet21k_write.write_sets(files, outdir, args)`

`datadings.sets.ImageNet21k_write.yield_samples(infile)`

datadings.sets.InriaBuildings module

datadings.sets.InriaBuildings_write module

usage: InriaBuildings_write.py [-h] [-o PATH] [-y] INDIR

Create InriaBuildings data set files.

This tool will look for the unpacked "AerialImageDataset" directory in the input directory.

See also:

https://project.inria.fr/aerialimagelabeling/contest/

Note: Registration is required to download this dataset. Please visit the website and follow the instructions to download and decompress it.

Important: Samples are NOT SHUFFLED! It is recommended to use the datadings-shuffle command to create a shuffled copy.

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label_image"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

`datadings.sets.InriaBuildings_write.array2imagedata(array)`

`datadings.sets.InriaBuildings_write.images_and_labels_iter(img_dir, label_dir, locations, ids)`

`datadings.sets.InriaBuildings_write.main()`

`datadings.sets.InriaBuildings_write.write(writer, img, labels, filename='')`

`datadings.sets.InriaBuildings_write.write_sets(indir, outdir, args, crop_size=(384, 384))`

datadings.sets.MIT1003 module

datadings.sets.MIT1003_write module

usage: MIT1003_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] INDIR

Create MIT1003 data set files.

This tool will look for the following files in the input directory and download them if necessary:

- ALLSTIMULI.zip
- DATA.zip

See also:

http://people.csail.mit.edu/tjudd/WherePeopleLook/index.html

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"experiments"`

Each experiment has the following keys:

- `"locations"`
- `"map"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes)

`datadings.sets.MIT1003_write.main()`

`datadings.sets.MIT1003_write.write_image(imagezip, datazip, mat_files, stimuluspath, writer)`

`datadings.sets.MIT1003_write.write_sets(files, outdir, args)`

datadings.sets.MIT300 module

datadings.sets.MIT300_write module

usage: MIT300_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] INDIR

Create MIT300 data set files.

This tool will look for the following files in the input directory and download them if necessary:

- BenchmarkIMAGES.zip

See also:

http://saliency.mit.edu/results_mit300.html

Important: Samples have the following keys:

- `"key"`
- `"image"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes)

`datadings.sets.MIT300_write.`**`main`**`()`

`datadings.sets.MIT300_write.`**`write_image`**`(`*imagezip*, *stimuluspath*, *writer*`)`

`datadings.sets.MIT300_write.`**`write_sets`**`(`*files*, *outdir*, *args*`)`

datadings.sets.Places2017 module

`datadings.sets.Places2017.`**`index_to_color`**`(`*array*`)`

datadings.sets.Places2017_write module

usage: Places2017_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] [–class-counts] [–color-map] [–classes] [–scenelabels] INDIR

Create Places2017 data set files.

This tool will look for all or parts of the following files in the input directory, depending on the given command, and download them if necessary:

- images.tar
- sceneparsing.tar
- annotations_instance.tar
- boundaries.tar
- objectInfo150.txt
- color150.mat

Additionally, ADE20K_2016_07_26.zip from the ADE20k dataset is required to extract scene labels with the `--scenelabels` option.

See also:

https://github.com/CSAILVision/placeschallenge

> **Warning:** OpenCV 2.4+ is required to create this dataset. If you are unsure how to install OpenCV you can use pip:
>
> pip install opencv-python

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label"`
- `"label_image"`
- `"instance_image"`
- `"boundary"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes) –class-counts Count pixels per class and write to

Places2017_counts.json.

--color-map Create color map and write to Places2017_colors.json.

--classes Print the class list.

--scenelabels Extract the scene labels from ADE20k dataset and write to Places2017_scenelabels.json. This will look for ADE20K_2016_07_26.zip in indir and download if necessary.

`datadings.sets.Places2017_write.`**`array_to_image`**(*array*, *format*, *dtype*, *mode*, ***kwargs*)

`datadings.sets.Places2017_write.`**`array_to_png`**(*array*)

`datadings.sets.Places2017_write.`**`create_color_map`**(*files*, *outdir*)

`datadings.sets.Places2017_write.`**`create_counts`**(*files*, *outdir*)

`datadings.sets.Places2017_write.`**`extract_boundary`**(*boundarytar*, *name*)

`datadings.sets.Places2017_write.`**`extract_class`**(*classtar*, *name*)

`datadings.sets.Places2017_write.`**`extract_instance`**(*instancetar*, *name*)

`datadings.sets.Places2017_write.`**`extract_scene`**(*scenes*, *name*)

`datadings.sets.Places2017_write.`**`extract_scenelabels`**(*files*, *outdir*)

`datadings.sets.Places2017_write.`**`find_sets`**(*tarfp*)

`datadings.sets.Places2017_write.`**`main`**()

`datadings.sets.Places2017_write.`**`print_classes`**(*files*)

`datadings.sets.Places2017_write.`**`write_set`**(*imagetar*, *classtar*, *instancetar*, *boundarytar*, *outdir*, *name*, *members*, *scenes*, *overwrite*)

`datadings.sets.Places2017_write.`**`write_sets`**(*files*, *outdir*, *args*)

datadings.sets.Places365 module

datadings.sets.Places365_write module

usage: Places365_write.py [-h] [-o PATH] [-y] [-s] [-l] [-c] INDIR

Create Places365 data set files. You can choose between any combination of

- high or low resolution images and
- standard or challenge (extended) training data set

This tool will look for the following files in the input directory depending on the chosen version and download them if necessary:

- standard (large)
  - train_large_places365standard.tar (105 GB)
  - val_large.tar (2.1 GB)
  - test_large.tar (19 GB)
- standard (small)
  - train_256_places365standard.tar (24 GB)
  - val_256.tar (501 MB)
  - test_256.tar (4.4 GB)
- challenge (large)
  - train_large_places365challenge.tar (476 GB)
  - val_large.tar (2.1 GB)
  - test_large.tar (19 GB)
- challenge (small)
  - train_256_places365challenge.tar (108 GB)
  - val_256.tar (501 MB)
  - test_256.tar (4.4 GB)

See also:

http://places2.csail.mit.edu/index.html

Important: For performance reasons shuffling is not available. It is recommended to use the datadings-shuffle command to create a shuffled copy.

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help	show this help message and exit
-o PATH, --outdir PATH	Output directory. Defaults to indir.
-y, --no-confirm	Don't require user interaction.
-s, --skip-verification	Skip verification of source files.
-l, --low-res	Download the resized and cropped images (256x256). Default are images with minimum dimension of 512 and preserved aspect ratio.
-c, --challenge	Download the extended challenge training dataset. Validation and testing are the same.

`datadings.sets.Places365_write.`**`file_spec`**(*path*, *md5*, *total=None*)

`datadings.sets.Places365_write.`**`get_files`**(*challenge*, *low_res*)

`datadings.sets.Places365_write.`**`main`**()

`datadings.sets.Places365_write.`**`write_sets`**(*indir*, *outdir*, *args*)

datadings.sets.RIT18 module

datadings.sets.RIT18_write module

usage: RIT18_write.py [-h] [-o PATH] [-y] [-s] INDIR

Create RIT18 data set files.

This tool will look for the following files in the input directory and download them if necessary:

- rit18_data.mat

See also:

https://github.com/rmkemker/RIT-18

Warning: Images, labels, and masks are numpy arrays, not images!

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label_image"`
- `"mask"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

`datadings.sets.RIT18_write.main()`

`datadings.sets.RIT18_write.write(writer, img, labels, mask, filename)`

`datadings.sets.RIT18_write.write_set(outdir, split, labels, data, args, crop_size)`

`datadings.sets.RIT18_write.write_sets(files, outdir, args, crop_size=(384, 384))`

`datadings.sets.RIT18_write.write_train(outdir, dataset, args)`

datadings.sets.SALICON2015 module

datadings.sets.SALICON2015_write module

usage: SALICON2015_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] INDIR

Create SALICON 2015 challenge data set files.

This tool will look for the following files in the input directory and download them if necessary:

- image.zip
- fixations.zip

See also:

http://salicon.net/challenge-2015/

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"experiments"`

Each experiment has the following keys:

- `"locations"`
- `"map"`
- `"timestamps"`
- `"fixations"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes)

`datadings.sets.SALICON2015_write.`**`get_keys`**(*imagezip*, *split*)

`datadings.sets.SALICON2015_write.`**`main`**()

`datadings.sets.SALICON2015_write.`**`write_set`**(*split*, *gen*, *outdir*, *total*)

`datadings.sets.SALICON2015_write.`**`write_sets`**(*files*, *outdir*, *args*)

`datadings.sets.SALICON2015_write.`**`yield_samples`**(*keys*, *imagezip*, *fixationzip*)

datadings.sets.SALICON2017 module

datadings.sets.SALICON2017_write module

usage: SALICON2017_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] INDIR

Create SALICON 2015 challenge data set files.

This tool will look for the following files in the input directory and download them if necessary:

- image.zip
- fixations.zip

See also:

http://salicon.net/challenge-2015/

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"experiments"`

Each experiment has the following keys:

- `"locations"`
- `"map"`
- `"timestamps"`
- `"fixations"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes)

datadings.sets.VOC2012 module

`datadings.sets.VOC2012.`**`bitget`**(*byteval*, *idx*)

`datadings.sets.VOC2012.`**`class_color_map`**(*n=256*)

Create class colors as per VOC devkit.

Adapted from: https://gist.github.com/wllhf/a4533e0adebe57e3ed06d4b50c8419ae

Parameters **n** – Number of classes.

Returns Numpy array of shape (n, 3).

`datadings.sets.VOC2012.`**`index_to_color`**(*array*)

`datadings.sets.VOC2012.`**`median_frequency_weights`**(*counts*)

datadings.sets.VOC2012_write module

usage: VOC2012_write.py [-h] [-o PATH] [-y] [-s] [–shuffle {yes,no}] [–calculate-weights] INDIR

Create Pascal VOC 2012 dataset files.

This tool will look for the following files in the input directory and download them if necessary:

- VOCtrainval_11-May-2012.tar

See also:

http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label_image"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help show this help message and exit

-o PATH, --outdir PATH Output directory. Defaults to indir.

-y, --no-confirm Don't require user interaction.

-s, --skip-verification Skip verification of source files.

–shuffle {yes,no} Write samples in random order. (default: yes) –calculate-weights Calculate median-frequency class weights.

`datadings.sets.VOC2012_write.`**`array_to_imagedata`**(*im*, *format*, ***kwargs*)

`datadings.sets.VOC2012_write.`**`calculate_set_weights`**(*files*)

`datadings.sets.VOC2012_write.`**`class_counts`**(*gen*, ***kwargs*)

`datadings.sets.VOC2012_write.`**`extract`**(*tar*, *path*)

`datadings.sets.VOC2012_write.`**`extractmember`**(*tar*, *member*)

`datadings.sets.VOC2012_write.`**`get_imageset`**(*tar*, *path*)

`datadings.sets.VOC2012_write.`**`imagedata_to_array`**(*data*)

`datadings.sets.VOC2012_write.`**`main`**()

`datadings.sets.VOC2012_write.`**`map_color_image`**(*im*)

`datadings.sets.VOC2012_write.`**`print_values`**(*prefix*, *values*)

`datadings.sets.VOC2012_write.`**`single_value`**(*a*)

`datadings.sets.VOC2012_write.`**`sorted_values`**(*d*)

`datadings.sets.VOC2012_write.`**`write_image`**(*writer*, *filename*, *im*, *seg*)

`datadings.sets.VOC2012_write.`**`write_set`**(*tar*, *split*, *outdir*, *args*)

`datadings.sets.VOC2012_write.`**`write_sets`**(*files*, *outdir*, *args*)

datadings.sets.Vaihingen module

datadings.sets.Vaihingen_write module

usage: Vaihingen_write.py [-h] [-o PATH] [-y] INDIR

Create Vaihingen data set files.

This tool will look for the ISPRS_semantic_labeling_Vaihingen directory in the input directory.

See also:

http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-vaihingen.html

Note: Requests to download the dataset can be made by filling out the form on the website.

Warning: Images, labels, and masks are numpy arrays, not images!

Important: Samples have the following keys:

- `"key"`
- `"image"`
- `"label_image"`
- `"mask"`

Positional arguments

INDIR Directory that contains dataset source files.

Optional arguments

-h, --help	show this help message and exit
-o PATH, --outdir PATH	Output directory. Defaults to indir.
-y, --no-confirm	Don't require user interaction.

`datadings.sets.Vaihingen_write.`**`images_label_dsm_iter`**(*indir*, *image_ids*)

`datadings.sets.Vaihingen_write.`**`main`**()

`datadings.sets.Vaihingen_write.`**`map_color_values_to_class_indices`**(*img*)

`datadings.sets.Vaihingen_write.`**`write`**(*writer*, *img*, *labels*, *mask*, *filename*)

`datadings.sets.Vaihingen_write.`**`write_sets`**(*indir*, *outdir*, *args*, *crop_size=(384, 384)*)

datadings.sets.YFCC100m module

The Yahoo Flickr Creative Commons 100 Million (YFCC100m) dataset.

Important: Only images are included. No videos or metadata.

See also:

https://multimediacommons.wordpress.com/yfcc100m-core-dataset/

Warning: This code is intended to load a pre-release version of the YFCC100m dataset. Please complain if you want to use the release version available from amazon: https://multimediacommons.wordpress.com/yfcc100m-core-dataset/

Important: Samples have the following keys:

- `"key"`
- `"image"`

class `datadings.sets.YFCC100m.`**`DevNull`**

Bases: `object`

`close`()

`read`(*_)

`write`(*_)

class `datadings.sets.YFCC100m.`**`YFCC100mReader`**(*image_packs_dir*, *validator=<function noop>*, *reject_file_paths=('/home/docs/checkouts/readthedocs.org/user_builds/datadi*
), error_file=None, *error_file_mode='a'*)

Bases: `datadings.reader.reader.Reader`

Special reader for the YFCC100m dataset only. It reads images from 10000 ZIP files of roughly 10000 images each.

One pass over the whole dataset was made to filter out irrelevant images if one of the following conditions is met:

- Image is damaged/incomplete.
- Less than 2600 bytes.
- Exactly 9218 bytes - a placeholder image from Flickr.
- Less than 20000 bytes and less than 5% of lines in the image have a variance less than 50.

Which images are rejected is controlled by the files given as `reject_file_paths`. Set this to None or empty list to iterate over the whole dataset.

Parameters

- **`image_packs_dir`** – Path to directory with image ZIP files.
- **`validator`** – Callable `validator(data: bytes) -> Union[bytes, None]`. Validates images before they are returned. Receives image data and returns data or `None`.

Warning: A validating reader cannot be copied and it is strongly discourages to copy readers with `error_file` paths.

Warning: Methods``get``, `slice`, `find_index`, `find_key`, `seek_index`, and `seek_key` are considerably slower for this reader compared to others. Use iterators and large `slice` ranges instead.

`find_index`(*key*)

Returns the index of the sample with the given key.

`find_key`(*index*)

Returns the key of the sample with the given index.

`get`(*index*, *yield_key=False*, *raw=False*, *copy=True*)

Returns sample at given index.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **index** – Index of the sample
- **yield_key** – If True, returns (key, sample)
- **raw** – If True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Sample as index.

next()

Returns the next sample.

This can be slow for file-based readers if a lot of samples are to be read. Consider using iter instead:

```
it = iter(reader)
while 1:
    next(it)
    ...
```

Or simply loop over the reader:

```
for sample in reader:
    ...
```

open_error_file_()

rawnext()

Return the next sample msgpacked as raw bytes.

This can be slow for file-based readers if a lot of samples are to be read. Consider using iter instead:

```
it = iter(reader)
while 1:
    next(it)
    ...
```

Or simply loop over the reader:

```
for sample in reader:
    ...
```

Included for backwards compatibility and may be deprecated and subsequently removed in the future.

seek_index(*index*)

Seek to the given index.

seek_key(*key*)

Seek to the sample with the given key.

slice(*start*, *stop=None*, *yield_key=False*, *raw=False*, *copy=True*)

Returns a generator of samples selected by the given slice.

`copy=False` allows the reader to use zero-copy mechanisms. Data may be returned as `memoryview` objects rather than `bytes`. This can improve performance, but also drastically increase memory consumption, since one sample can keep the whole slice in memory.

Parameters

- **start** – start index of slice
- **stop** – stop index of slice
- **yield_key** – if True, yield (key, sample)
- **raw** – if True, returns sample as msgpacked message
- **copy** – if False, allow the reader to return data as `memoryview` objects instead of `bytes`

Returns Iterator of selected samples

`datadings.sets.YFCC100m.decode_fast(data)`

`datadings.sets.YFCC100m.main()`

`datadings.sets.YFCC100m.noop(data)`

`datadings.sets.YFCC100m.validate_image(data)`

datadings.sets.YFCC100m_counts module

datadings.sets.tools module

`datadings.sets.tools.load_json(name)`

Load a JSON file contained in the sets package. Supports gzip, xz, zip, or uncompressed files.

Parameters **name** – file name

Returns decoded JSON

`datadings.sets.tools.open_comp(name, mode='rb', level=None, encoding=None)`

Returns a file object contained in the sets package. Supports gzip, xz, or uncompressed files.

Parameters

- **name** – file name
- **mode** – open mode
- **level** – compression level/preset for writing
- **encoding** – encoding for text mode

Returns open file object

datadings.sets.types module

This module contains helper functions to create dictionaries with specific keys.

It is auto-generated by the `generate_types.py` script. Do not edit it directly. Instead add new types to `generate_types.json`.

`datadings.sets.types.ADE20kData(key, image, label, label_image, parts_images)`

Returns a dictionary:

```
{
    'key': key,
    'image': image,
```

(continues on next page)

(continued from previous page)

```
    'label': label,
    'label_image': label_image,
    'parts_images': parts_images
}
```

datadings.sets.types.**ANP460Data**(*key*, *image*, *experiments*, *anp*, *type*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'experiments': experiments,
    'anp': anp,
    'type': type
}
```

datadings.sets.types.**ANP460Experiment**(*locations*, *map*, *answer*, *duration*)

Returns a dictionary:

```
{
    'locations': locations,
    'map': map,
    'answer': answer,
    'duration': duration
}
```

datadings.sets.types.**ImageClassificationData**(*key*, *image*, *label*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'label': label
}
```

datadings.sets.types.**ImageCoarseClassificationData**(*key*, *image*, *label*, *coarse_label*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'label': label,
    'coarse_label': coarse_label
}
```

datadings.sets.types.**ImageData**(*key*, *image*)

Returns a dictionary:

```
{
    'key': key,
    'image': image
}
```

datadings.sets.types.**ImageDisparitySegmentationData**(*key*, *image*, *label_image*, *disparity_image*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'label_image': label_image,
    'disparity_image': disparity_image
}
```

datadings.sets.types.**ImageInstanceSegmentationData**(*key*, *image*, *label_image*, *instance_image*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'label_image': label_image,
    'instance_image': instance_image
}
```

datadings.sets.types.**ImageNet21kData**(*key*, *image*, *label*, *label_tree*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'label': label,
    'label_tree': label_tree
}
```

datadings.sets.types.**ImageSegmentationData**(*key*, *image*, *label_image*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'label_image': label_image
}
```

datadings.sets.types.**MaskedImageSegmentationData**(*key*, *image*, *label_image*, *mask*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'label_image': label_image,
    'mask': mask
}
```

datadings.sets.types.**Places2017Data**(*key*, *image*, *label*, *label_image*, *instance_image*, *boundary*)

Returns a dictionary:

```
{
    'key': key,
```

(continues on next page)

(continued from previous page)

```
    'image': image,
    'label': label,
    'label_image': label_image,
    'instance_image': instance_image,
    'boundary': boundary
}
```

datadings.sets.types.**SaliencyData**(*key*, *image*, *experiments*)

Returns a dictionary:

```
{
    'key': key,
    'image': image,
    'experiments': experiments
}
```

datadings.sets.types.**SaliencyExperiment**(*locations*, *map*)

Returns a dictionary:

```
{
    'locations': locations,
    'map': map
}
```

datadings.sets.types.**SaliencyTimeseriesExperiment**(*locations*, *map*, *timestamps*, *fixations*)

Returns a dictionary:

```
{
    'locations': locations,
    'map': map,
    'timestamps': timestamps,
    'fixations': fixations
}
```

2.7.5 datadings.tools package

class datadings.tools.**ProgressPrinter**(*_, **__)

Bases: tqdm.std.tqdm

monitor_interval = 0

class datadings.tools.**SentinelEnd**

Bases: object

class datadings.tools.**SentinelError**

Bases: object

class datadings.tools.**Yielder**(*gen*, *queue*, *end*, *error*)

Bases: threading.Thread

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

stop()

class datadings.tools.**YielderProc**(*gen*, *queue*)

Bases: multiprocessing.context.Process

run()

Method to be run in sub-process; can be overridden in sub-class

stop()

datadings.tools.**document_keys**(*typefun*, *block='Important:'*, *prefix='Samples have the following keys:'*, *postfix=''*)

Extract the keys that samples created by a type function have create a documentation string that lists them. For example, it produces the following documentation for *ImageClassificationData*:

```
{block}
    {prefix}

    - ``"key"``
    - ``"image"``
    - ``"label"``

    {postfix}
```

Parameters

- **typefun** – Type function to analyze.
- **block** – Type of block to use. Defaults to "Important:".
- **prefix** – Text before parameter list.
- **postfix** – Text after parameter list.

datadings.tools.**download_files_if_not_found**(*files*, *indir*)

Run :py:func:download_if_not_found for multiple files.

See also:

datadings.tools.prepare_indir()

datadings.tools.**download_if_not_found**(*url*, *path*)

Check if path is a file, otherwise download from url to path.

datadings.tools.**hash_md5hex**(*path*, *read_size=65536*, *progress=False*)

Calculate the (hexadecimal) MD5 hash of a file.

Parameters

- **path** – File to hash.
- **read_size** – Read-ahead size.
- **progress** – If True, display progress.

Returns Hexadecimal MD5 hash as string.

datadings.tools.**hash_string**(*s: str*, *salt: bytes = b''*, *__struct=<Struct object>*) → int

Hash a string using the blake2s algorithm.

Parameters

- **s** – the string
- **salt** – optional salt, max 8 bytes

Returns first 8 bytes of the hash, interpreted as big-endian uint64

datadings.tools.**hash_string_bytes**(*s: str*, *salt: bytes = b''*, *__struct=<Struct object>*) → bytes

Hash a string using the blake2s algorithm.

Parameters

- **s** – the string
- **salt** – optional salt, max 8 bytes

Returns first 8 bytes of the hash

datadings.tools.**load_md5file**(*path*)

Load a text files of MD5 hashes.

Parameters **path** – Path to MD5 file.

Returns Dict of (file, hash) pairs.

datadings.tools.**locate_files**(*files*, *indir*)

Returns a copy of `files` where paths are replaced with concrete paths located in `indir`.

See also:

datadings.tools.prepare_indir()

datadings.tools.**make_printer**(*bar_format='{desc} {percentage:3.0f}% {elapsed}<{remaining}, {rate_fmt}{postfix}'*, *miniters=0*, *mininterval=0.5*, *smoothing=0.1*, ***kwargs*)

Convenience function to create tqdm objects with some default arguments.

Returns tqdm.tqdm object.

datadings.tools.**path_append**(*path: pathlib.Path*, *string: str*)

Append a string to the name of a pathlib Path.

Parameters

- **path** – the path
- **string** – the bit to append

Returns Path with stuff appended

Raises

- **ValueError** –
- **e.g., root /.** –

datadings.tools.**path_append_suffix**(*path: pathlib.Path*, *suffix: str*)

Appends the given suffix to the path if the path does not end with said suffix:

```
>>> path_append_suffix(Path('some.file'), '.file')
>>> Path('some.file')
>>> path_append_suffix(Path('some.file'), '.txt')
>>> Path('some.file.txt')
```

Behaves like `path_append` if suffix does not startwith `'.'` (dot):

```
>>> path_append_suffix(Path('some.file'), 'txt')
>>> Path('some.filetxt')
```

Parameters

- **path** – the base path
- **suffix** – suffix to append if necessary

Returns Path that ends with suffix.

`datadings.tools.`**`prepare_indir`**(*files*, *args*)

Prepare a directory for dataset creation. `files` specifies with files need be downloaded and/or integrity checked. It is a dict of file descriptions like these:

```
files = {
    'train': {
        'path': 'dataset.zip',
        'url': 'http://cool.dataset/dataset.zip',
        'md5': '56ad5c77e6c8f72ed9ef2901628d6e48',
    }
}
```

Once downloads and/or verification have finished, the relative paths are replaced with concrete paths in `args.indir`.

Parameters

- **files** – Dict of file descriptions.
- **args** – Parsed argparse arguments object with `indir` and `skip_verification` arguments.

Returns Files with paths located in args.indir.

`datadings.tools.`**`print_over`**(**args*, ***kwargs*)

Wrapper around print that replaces the current line. It prints from the start of the line and clears remaining characters. Accepts the same kwargs as the print function.

Parameters **flush** – If True, flush after printing.

`datadings.tools.`**`query_user`**(*question*, *default='yes'*, *answers=('yes', 'no', 'abort')*)

Ask user a question via input() and return their answer.

Adapted from http://code.activestate.com/recipes/577097/

Parameters

- **question** – String that is presented to the user.
- **default** – Presumed answer if the user just hits <Enter>. Must be one of `prompts` or `None` (meaning an answer is required of the user).
- **answers** – Answers the user can give.

Returns One of `prompts`.

`datadings.tools.`**`split_array`**(*img*, *v_pixels*, *h_pixels*, *indices=(1, 2)*)

Split/tile an image/numpy array in horizontal and vertical direction.

Parameters

- **img** – The image to split.

- **h_pixels** – Width of each tile in pixels.
- **v_pixels** – Height of each tile in pixels.
- **indices** – 2-tuple of indices used to calculate number of tiles.

Returns Yields single tiles from the image as arrays.

`datadings.tools.tiff_to_nd_array(file_path, type=<class 'numpy.uint8'>)`

Decode a TIFF image and returns all contained subimages as numpy array. The first dimension of the array indexes the subimages.

> **Warning:** Requires geo (GDAL) extra!

Parameters

- **file_path** – Path to TIFF file.
- **type** – Output dtype.

Returns TIFF image as numpy array.

`datadings.tools.verify_file(meta, indir)`

`datadings.tools.verify_files(files, indir)`

Verify the integrity of the given files.

See also:

datadings.tools.prepare_indir()

`datadings.tools.yield_process(gen)`

Run a generator in a background thread and yield its output in the current thread.

Parameters **gen** – Generator to yield from.

`datadings.tools.yield_threaded(gen)`

Run a generator in a background thread and yield its output in the current thread.

Parameters **gen** – Generator to yield from.

Submodules

datadings.tools.argparse module

A collection of useful helper functions to create argument parsers. Using pre-defined arguments ensures that arguments are consistent across different tools in datadings.

All helper functions in this module follow the convention that a function called `argument_indir` adds the `indir` argument to the given parser, including additional configuration and help text.

Note: Any of the default arguments given by the `argument_*` functions can be overwritten. For example, `argument_shuffle` defines `choices = ['yes', 'no']`. If those are too formal for your liking, you can also use `argument_shuffle(parser, choices=['yeah', 'naw']`. But please don't.

---

class datadings.tools.argparse.**MinMaxAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse.Action

Action to clamp value between given min and max values. Create subclass to set min_value and max_value:

```
class Action(MinMaxAction):
    min_value = 1
    max_value = 7

parser.add_argument('onetoseven', action=Action)
```

max_value = None

min_value = None

class datadings.tools.argparse.**YesNoAction**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Bases: argparse.Action

Action like store_true that checks if value == yes.

datadings.tools.argparse.**argument_calculate_weights**(**args*, ***kwargs*)

Add the following argument to the given ArgumentParser:

```
parser.add_argument(
    '--calculate-weights',
    action='store_true',
    help='Calculate median-frequency class weights.',
)
```

Parameters

- **parser** – Call add_argument on this ArgumentParser.
- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_indir**(**args*, ***kwargs*)

Add the following argument to the given ArgumentParser:

```
parser.add_argument(
    'indir',
    type=<class 'str'>,
    default='.',
    metavar='INDIR',
    help='Directory that contains dataset source files.',
)
```

Parameters

- **parser** – Call add_argument on this ArgumentParser.
- **args** – Additional positional arguments for add_argument.

- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_infile**(**args*, ***kwargs*)

Add the following argument to the given ArgumentParser:

```
parser.add_argument(
    'infile',
    type=<class 'str'>,
    default=None,
    help='Input file.',
)
```

Parameters

- **parser** – Call add_argument on this ArgumentParser.
- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_no_confirm**(**args*, ***kwargs*)

Add the following argument to the given ArgumentParser:

```
parser.add_argument(
    '-y', '--no-confirm',
    dest='no_confirm',
    action='store_true',
    help='Don't require user interaction.',
)
```

Parameters

- **parser** – Call add_argument on this ArgumentParser.
- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_outdir**(**args*, ***kwargs*)

Add the following argument to the given ArgumentParser:

```
parser.add_argument(
    '-o', '--outdir',
    type=<class 'str'>,
    default=None,
    metavar='PATH',
    help='Output directory. Defaults to indir.',
)
```

Parameters

- **parser** – Call add_argument on this ArgumentParser.
- **args** – Additional positional arguments for add_argument.

- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_outfile**(**args*, ***kwargs*)

Add the following argument to the given `ArgumentParser`:

```
parser.add_argument(
    '-o', '--outfile',
    type=<class 'str'>,
    default=None,
    metavar='PATH',
    help='Output file.',
)
```

Parameters

- **parser** – Call add_argument on this `ArgumentParser`.
- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_outfile_positional**(**args*, ***kwargs*)

Add the following argument to the given `ArgumentParser`:

```
parser.add_argument(
    'outfile',
    type=<class 'str'>,
    help='Output file.',
)
```

Parameters

- **parser** – Call add_argument on this `ArgumentParser`.
- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_outfiles**(**args*, ***kwargs*)

Add the following argument to the given `ArgumentParser`:

```
parser.add_argument(
    '-o', '--outfiles',
    type=<class 'str'>,
    default=None,
    metavar='PATH',
    help='Output files.',
    nargs='+',
)
```

Parameters

- **parser** – Call add_argument on this `ArgumentParser`.

- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_shuffle**(**args*, ***kwargs*)

Add the following argument to the given `ArgumentParser`:

```
parser.add_argument(
    '--shuffle',
    default='yes',
    choices=['yes', 'no'],
    action=<class 'datadings.tools.argparse.YesNoAction'>,
    help='Write samples in random order. (default: yes)',
)
```

Parameters

- **parser** – Call add_argument on this `ArgumentParser`.
- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_skip_verification**(**args*, ***kwargs*)

Add the following argument to the given `ArgumentParser`:

```
parser.add_argument(
    '-s', '--skip-verification',
    action='store_true',
    help='Skip verification of source files.',
)
```

Parameters

- **parser** – Call add_argument on this `ArgumentParser`.
- **args** – Additional positional arguments for add_argument.
- **kwargs** – Additional keyword arguments for add_argument. Can override keyword arguments specified above.

datadings.tools.argparse.**argument_threads**(*parser*, *default=1*, *max_threads=0*)

Add threads argument to parser.

Parameters

- **parser** – Argument is added here.
- **default** – Default number of threads.
- **max_threads** – Maximum number of threads. If >0, use given number. If 0 use `cpu_count()`. if <0, use `-max_threads*cpu_count()`

datadings.tools.argparse.**make_parser**(*description*, *indir=True*, *outdir=True*, *no_confirm=True*, *skip_verification=True*, *shuffle=True*, *formatter_class=<class 'argparse.RawDescriptionHelpFormatter'>*, ***kwargs*) → argparse.ArgumentParser

Create an `ArgumentParser` with a set of common arguments.

Parameters

- **description** – Description text displayed before arguments. Usually `__doc__` is fine.
- **indir** – If True, add `indir` argument.
- **outdir** – If True, add `outdir` argument.
- **no_confirm** – If True, add `no_confirm` argument.
- **skip_verification** – If True, add `skip_verification` argument.
- **shuffle** – If True, add `shuffle` argument.
- **formatter_class** – Description formatter, defaults to raw.
- **kwargs** – kwargs given to `ArgumentParser`.

Returns `ArgumentParser`.

datadings.tools.argparse.**make_parser_simple**(*description*, *indir=False*, *outdir=False*, *no_confirm=False*, *skip_verification=False*, *shuffle=False*, *formatter_class=<class 'argparse.RawDescriptionHelpFormatter'>*, ***kwargs*) → argparse.ArgumentParser

Same as *make_parser()*, but add no arguments by default.

datadings.tools.cached_property module

MIT License

Copyright (c) 2020 Alexey Stepanov

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Backport of python 3.8 functools.cached_property.

cached_property() - computed once per instance, cached as attribute

class datadings.tools.cached_property.**cached_property**(*func: Callable[[Any], datadings.tools.cached_property._T]*)

Bases: `object`

Cached property implementation.

Transform a method of a class into a property whose value is computed once and then cached as a normal attribute for the life of the instance. Similar to property(), with the addition of caching. Useful for expensive computed properties of instances that are otherwise effectively immutable.

__init__(*func: Callable[[Any], datadings.tools.cached_property._T]*) → None
Cached property implementation.

datadings.tools.compression module

datadings.tools.compression.**open_comp**(*path*, *mode='rb'*, *level=None*, *encoding=None*)
Returns a file object contained in the sets package. Supports gzip, xz, or uncompressed files.

Parameters

- **path** – file path
- **mode** – open mode
- **level** – compression level/preset for writing
- **encoding** – encoding for text mode

Returns open file object

datadings.tools.matlab module

datadings.tools.matlab.**iter_fields**(*arr*, *ignore=()*)
Iterate over the fields of a structured numpy array (i.e., an array with a complex data type). Each iteration yields (fieldname, value).

Parameters

- **arr** – A structured array.
- **ignore** – Fields to ignore.

Returns Yields individual fields from the array.

datadings.tools.matlab.**loadmat**(*mat*)
Load a Matlab "mat" file.

Parameters **mat** – Path, file-like object, or data.

Returns Contents of the Matlab file.

datadings.tools.msgpack module

Compatibility functions that wrap msgpack to seamlessly support both version 1.0.0 and earlier versions. Encoding is always UTF-8 and bin type is enabled and `strict_map_key=False`.

All helpers are setup to use `msgpack_numpy` for transparent packing and unpacking of numpy arrays enabled by default.

datadings.tools.msgpack.**make_packer = functools.partial(<class 'msgpack._cmsgpack.Packer'>, default=<function encode>)**
Create a packer with default arguments.

datadings.tools.msgpack.**make_unpacker = functools.partial(<class 'msgpack._cmsgpack.Unpacker'>, strict_map_key=False, object_hook=<function decode>)**
Create a unpacker with default arguments.

datadings.tools.msgpack.**pack**(*o*, *stream*, ***kwargs*)
Pack object to stream.

datadings.tools.msgpack.**packb**(*o*, ***kwargs*)
Pack object to bytes.

datadings.tools.msgpack.**unpack**(*stream*, ***kwargs*)
Unpack object from stream.

datadings.tools.msgpack.**unpackb**(*packed*, **, object_hook=None*, *list_hook=None*, *bool use_list=True*, *bool raw=False*, *int timestamp=0*, *bool strict_map_key=True*, *unicode_errors=None*, *object_pairs_hook=None*, *ext_hook=ExtType*, *Py_ssize_t max_str_len=-1*, *Py_ssize_t max_bin_len=-1*, *Py_ssize_t max_array_len=-1*, *Py_ssize_t max_map_len=-1*, *Py_ssize_t max_ext_len=-1*)
Unpack object from bytes.

2.7.6 datadings.torch package

class datadings.torch.**Compose**(**transforms*, *prefix=''*)
Bases: object

Compose a sequence of transform functions. Functions must accept the intended value from samples as first argument. They may have an arbitrary number of positional and keyword arguments.

Example usage with *Dataset*:

```
import random
from datadings.torch import Compose
from datadings.torch import Dataset
from datadings.reader import ListReader

def add(v, number):
    return v + number

def sub(x, value):
    return x - value

def rng(_):
    return {
        'number': random.randrange(1, 10),
        'value': random.randrange(1, 10),
    }

samples = [{'a': 0, 'b': 0, 'c': 0} for _ in range(10)]
reader = ListReader(samples)
transforms = {
    'a': Compose(add),
    'b': Compose(sub),
    'c': Compose((add, sub)),
}
```

(continues on next page)

(continued from previous page)

```
dataset = Dataset(reader, transforms=transforms, rng=rng)
for i in range(len(dataset)):
    print(dataset[i])
```

Parameters

- **transforms** – sequence of transform functions, either one iterable or varargs
- **prefix** – string prefix for parameter names, i.e., if a function normally requires parameter `size` given `prefix='mask_'` the parameter `'mask_size'`

class datadings.torch.**CompressedToPIL**

Bases: `object`

Compatible torchvision transform that takes a compressed image as bytes (or similar) and returns a PIL image.

class datadings.torch.**Dataset**(**args: Any*, ***kwargs: Any*)

Bases: *datadings.torch.DatasetBase*, `torch.utils.data.`

Implementation of `torch.utils.data.Dataset`.

> **Warning:** *Dataset* can be significantly slower than *IterableDataset*. If shuffling is necessary consider using *QuasiShuffler* instead.

Example usage with the PyTorch `DataLoader`:

```
path = '.../train.msgpack'
batch_size = 256
reader = MsgpackReader(path)
transforms = {'image': Compose(
    CompressedToPIL(),
    ...,
    ToTensor(),
)}
ds = Dataset(reader, transforms=transforms)
train = DataLoader(dataset=ds, batch_size=batch_size)
for epoch in range(3):
    for x, y in dict2tuple(tqdm(train)):
        pass
```

Parameters

- **reader** – the datadings reader instance
- **transforms** – Transforms applied to samples before they are returned. Either a dict of transform functions or callable with signature `f(sample: dict) -> dict` that is applied directly to samples. In the dict form keys correspond to keys in the sample and values are callables with signature `t(value: any, params: dict) -> any` (e.g., an instance of *Compose*) with `params` the value returned by the `rng` callable.
- **rng** – callable with signature `rng(params: dict) -> dict` that returns a dict of parameters applied to transforms

class datadings.torch.**DatasetBase**(*reader:* datadings.reader.reader.Reader, *transforms=None*, *rng=None*)

Bases: `object`

class datadings.torch.**IterableDataset**(**args: Any*, ***kwargs: Any*)

Bases: *datadings.torch.DatasetBase*, torch.utils.data.

Implementation of `torch.utils.data.IterableDataset` to use with datadings readers.

With distributed training the reader is divided into `world_size * num_workers` shards. Each dataloader worker of each rank iterates over a different shard. The final batch delivered by a worker may be smaller than the batch size if the length of the reader is not divisible by `batch_size * num_shards`.

Note: Set `persistent_workers=True` for the `DataLoader` to let the dataset object track the current epoch. It then cycles through shards This makes ranks cycle through shards of the dataset Without this option torch may create new worker processes at any time, which resets the dataset to its initial state.

Warning: Raises `RuntimeError` if `0 < len(shard) % batch_size < 1`, since this may lead to an uneven number of batches generated by each worker. This can lead to crashes if it happens between rank workers, or deadlock if ranks receive different a number of batches. Change `num_workers`, `batch_size`, or `world_size` to avoid this.

Example usage with the PyTorch `DataLoader`:

```
path = '.../train.msgpack'
batch_size = 256
reader = MsgpackReader(path)
transforms = {'image': Compose(
    CompressedToPIL(),
    ...,
    ToTensor(),
)}
ds = IterableDataset(
    reader,
    transforms=transforms,
    batch_size=batch_size,
)
train = DataLoader(
    dataset=ds,
    batch_size=batch_size,
    num_workers=4,
    persistent_workers=True,
)
for epoch in range(3):
    print('Epoch', epoch)
    for x, y in dict2tuple(tqdm(train)):
        pass
```

Parameters

- **reader** – the datadings reader instance
- **transforms** – Transforms applied to samples before they are returned. Either a dict of transform functions or callable with signature `f(sample: dict) -> dict` that is applied directly to samples. In the dict form keys correspond to keys in the sample and values are callables with signature `t(value: any, params: dict) -> any` (e.g., an instance of *Compose*) with `params` the value returned by the `rng` callable.

- **rng** – callable with signature `rng(params: dict) -> dict` that returns a dict of parameters applied to transforms
- **batch_size** – same batch size as given to the `DataLoader`
- **epoch** – starting epoch, zero indexed; only relevant when resuming
- **copy** – see *datadings.reader.reader.Reader.iter()*
- **chunk_size** – see *datadings.reader.reader.Reader.iter()*
- **group** – distributed process group to use (if not using the default)

`datadings.torch.dict2tuple(it, keys=('image', 'label'))`
Utility function that extracts and yields the given keys from each sample in the given iterator.

`datadings.torch.no_rng(_)`

2.7.7 datadings.writer package

`class datadings.writer.FileWriter(outfile, buffering=0, overwrite=False, **kwargs)`
Bases: *datadings.writer.Writer*

Writer for file-based datasets. Requires sample dicts with a unique `"key"` value.

`write(sample)`
Write a sample to the dataset file.

Parameters **args** – Sample data to write.

`class datadings.writer.RawWriter(outfile, buffering=0, overwrite=False, **kwargs)`
Bases: *datadings.writer.Writer*

Writer for raw data. No packing is done. *write()* requires `key` and `data` as arguments.

`write(key, data)`
Write a sample to the dataset file.

Parameters **args** – Sample data to write.

`class datadings.writer.Writer(outfile, buffering=0, overwrite=False, **kwargs)`
Bases: `object`

Writers can be used to create dataset files along with index and MD5 hash.

Writer is an abstract class. It cannot be instantiated. Subclasses must implement the abstract write method.

It is recommended to use writers as context manager in "with" statements:

with Writer('dataset.msgpack') as writer:

for sample in samples: writer.write(sample)

The writer is then automatically closed and index and md5 files are written.

Important: If `overwrite` is `False`, the user will be prompted to overwrite an existing file. The user can now:

- Accept to overwrite the file.
- Decline, which raises a `FileExistsError`. The program should continue as if writing had finished.
- Abort, which raises a `KeyboardInterrupt`. The program should abort immediately.

Parameters

- **outfile** – Path to the dataset file.
- **overwrite** – If outfile exists, force overwriting.
- **kwargs** – Keyword arguments for *datadings.tools.make_printer()*.

close()

Flush and close the dataset file and write index and MD5 files.

abstract write(**args*)

Write a sample to the dataset file.

Parameters **args** – Sample data to write.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

datadings.commands, 18
datadings.commands.bench, 18
datadings.commands.cat, 19
datadings.commands.convert_index, 20
datadings.commands.merge, 20
datadings.commands.sample, 21
datadings.commands.shuffle, 21
datadings.commands.split, 22
datadings.commands.write, 22
datadings.index, 23
datadings.reader, 25
datadings.reader.augment, 25
datadings.reader.directory, 28
datadings.reader.list, 29
datadings.reader.msgpack, 31
datadings.reader.reader, 32
datadings.reader.sharded, 34
datadings.reader.zipfile, 35
datadings.sets, 36
datadings.sets.ADE20k, 36
datadings.sets.ADE20k_write, 36
datadings.sets.ANP460, 37
datadings.sets.ANP460_write, 37
datadings.sets.CAMVID, 38
datadings.sets.CAMVID_write, 38
datadings.sets.CAT2000, 39
datadings.sets.CAT2000_write, 39
datadings.sets.CIFAR10, 40
datadings.sets.CIFAR100, 40
datadings.sets.CIFAR100_write, 40
datadings.sets.CIFAR10_write, 41
datadings.sets.Cityscapes, 42
datadings.sets.Cityscapes_write, 42
datadings.sets.Coutrot1, 43
datadings.sets.Coutrot1_write, 43
datadings.sets.FIGRIMFixation, 45
datadings.sets.FIGRIMFixation_write, 45
datadings.sets.ILSVRC2012, 46
datadings.sets.ILSVRC2012_synsets, 46
datadings.sets.ILSVRC2012_write, 46
datadings.sets.ImageNet21k_create_meta, 47
datadings.sets.ImageNet21k_synsets, 48
datadings.sets.ImageNet21k_write, 48
datadings.sets.InriaBuildings, 49
datadings.sets.InriaBuildings_write, 49
datadings.sets.MIT1003, 50
datadings.sets.MIT1003_write, 50
datadings.sets.MIT300, 51
datadings.sets.MIT300_write, 51
datadings.sets.Places2017, 51
datadings.sets.Places2017_write, 52
datadings.sets.Places365, 53
datadings.sets.Places365_write, 53
datadings.sets.RIT18, 55
datadings.sets.RIT18_write, 55
datadings.sets.SALICON2015, 56
datadings.sets.SALICON2015_write, 56
datadings.sets.SALICON2017, 57
datadings.sets.SALICON2017_write, 57
datadings.sets.tools, 63
datadings.sets.types, 63
datadings.sets.Vaihingen, 59
datadings.sets.Vaihingen_write, 59
datadings.sets.VOC2012, 58
datadings.sets.VOC2012_write, 58
datadings.sets.YFCC100m, 60
datadings.sets.YFCC100m_counts, 63
datadings.tools, 66
datadings.tools.argparse, 70
datadings.tools.cached_property, 75
datadings.tools.compression, 76
datadings.tools.matlab, 76
datadings.tools.msgpack, 76
datadings.torch, 77
datadings.writer, 80

INDEX

Symbols

__init__() (*datadings.tools.cached_property.cached_property method*), 76

A

ADE20kData() (*in module datadings.sets.types*), 63
ANP460Data() (*in module datadings.sets.types*), 64
ANP460Experiment() (*in module datadings.sets.types*), 64
argument_calculate_weights() (*in module datadings.tools.argparse*), 71
argument_compress() (*in module datadings.sets.ILSVRC2012_write*), 47
argument_indir() (*in module datadings.tools.argparse*), 71
argument_infile() (*in module datadings.tools.argparse*), 72
argument_no_confirm() (*in module datadings.tools.argparse*), 72
argument_outdir() (*in module datadings.tools.argparse*), 72
argument_outfile() (*in module datadings.tools.argparse*), 73
argument_outfile_positional() (*in module datadings.tools.argparse*), 73
argument_outfiles() (*in module datadings.tools.argparse*), 73
argument_shuffle() (*in module datadings.tools.argparse*), 74
argument_skip_verification() (*in module datadings.tools.argparse*), 74
argument_subsampling() (*in module datadings.sets.ILSVRC2012_write*), 47
argument_threads() (*in module datadings.tools.argparse*), 74
array2imagedata() (*in module datadings.sets.InriaBuildings_write*), 49
array_to_image() (*in module datadings.sets.ADE20k_write*), 37
array_to_image() (*in module datadings.sets.Places2017_write*), 53
array_to_imagedata() (*in module datadings.sets.VOC2012_write*), 59
array_to_png() (*in module datadings.sets.Places2017_write*), 53
array_to_png16() (*in module datadings.sets.ADE20k_write*), 37

B

bench() (*in module datadings.commands.bench*), 19
bitget() (*in module datadings.sets.VOC2012*), 58

C

cached_property (*class in datadings.tools.cached_property*), 75
calculate_set_weights() (*in module datadings.sets.VOC2012_write*), 59
calculate_weights() (*in module datadings.sets.ADE20k_write*), 37
calculate_weights() (*in module datadings.sets.CAMVID_write*), 39
cat() (*in module datadings.commands.cat*), 19
check_included() (*in module datadings.reader.directory*), 29
class_color_map() (*in module datadings.sets.VOC2012*), 58
class_counts() (*in module datadings.sets.VOC2012_write*), 59
close() (*datadings.sets.YFCC100m.DevNull method*), 61
close() (*datadings.writer.Writer method*), 81
Compose (*class in datadings.torch*), 77
CompressedToPIL (*class in datadings.torch*), 78
convert_index() (*in module datadings.commands.convert_index*), 20
convert_semantic_tree() (*in module datadings.sets.ImageNet21k_create_meta*), 47
create_color_map() (*in module datadings.sets.Places2017_write*), 53
create_counts() (*in module datadings.sets.Places2017_write*), 53
create_sample() (*in module datadings.sets.CAT2000_write*), 40

cumsum() (*in module datadings.commands.merge*), 20
Cycler (*class in datadings.reader.augment*), 25

D

datadings.commands
    module, 18
datadings.commands.bench
    module, 18
datadings.commands.cat
    module, 19
datadings.commands.convert_index
    module, 20
datadings.commands.merge
    module, 20
datadings.commands.sample
    module, 21
datadings.commands.shuffle
    module, 21
datadings.commands.split
    module, 22
datadings.commands.write
    module, 22
datadings.index
    module, 23
datadings.reader
    module, 25
datadings.reader.augment
    module, 25
datadings.reader.directory
    module, 28
datadings.reader.list
    module, 29
datadings.reader.msgpack
    module, 31
datadings.reader.reader
    module, 32
datadings.reader.sharded
    module, 34
datadings.reader.zipfile
    module, 35
datadings.sets
    module, 36
datadings.sets.ADE20k
    module, 36
datadings.sets.ADE20k_write
    module, 36
datadings.sets.ANP460
    module, 37
datadings.sets.ANP460_write
    module, 37
datadings.sets.CAMVID
    module, 38
datadings.sets.CAMVID_write
    module, 38
datadings.sets.CAT2000
    module, 39
datadings.sets.CAT2000_write
    module, 39
datadings.sets.CIFAR10
    module, 40
datadings.sets.CIFAR100
    module, 40
datadings.sets.CIFAR100_write
    module, 40
datadings.sets.CIFAR10_write
    module, 41
datadings.sets.Cityscapes
    module, 42
datadings.sets.Cityscapes_write
    module, 42
datadings.sets.Coutrot1
    module, 43
datadings.sets.Coutrot1_write
    module, 43
datadings.sets.FIGRIMFixation
    module, 45
datadings.sets.FIGRIMFixation_write
    module, 45
datadings.sets.ILSVRC2012
    module, 46
datadings.sets.ILSVRC2012_synsets
    module, 46
datadings.sets.ILSVRC2012_write
    module, 46
datadings.sets.ImageNet21k_create_meta
    module, 47
datadings.sets.ImageNet21k_synsets
    module, 48
datadings.sets.ImageNet21k_write
    module, 48
datadings.sets.InriaBuildings
    module, 49
datadings.sets.InriaBuildings_write
    module, 49
datadings.sets.MIT1003
    module, 50
datadings.sets.MIT1003_write
    module, 50
datadings.sets.MIT300
    module, 51
datadings.sets.MIT300_write
    module, 51
datadings.sets.Places2017
    module, 51
datadings.sets.Places2017_write
    module, 52
datadings.sets.Places365
    module, 53

datadings.sets.Places365_write
module, 53
datadings.sets.RIT18
module, 55
datadings.sets.RIT18_write
module, 55
datadings.sets.SALICON2015
module, 56
datadings.sets.SALICON2015_write
module, 56
datadings.sets.SALICON2017
module, 57
datadings.sets.SALICON2017_write
module, 57
datadings.sets.tools
module, 63
datadings.sets.types
module, 63
datadings.sets.Vaihingen
module, 59
datadings.sets.Vaihingen_write
module, 59
datadings.sets.VOC2012
module, 58
datadings.sets.VOC2012_write
module, 58
datadings.sets.YFCC100m
module, 60
datadings.sets.YFCC100m_counts
module, 63
datadings.tools
module, 66
datadings.tools.argparse
module, 70
datadings.tools.cached_property
module, 75
datadings.tools.compression
module, 76
datadings.tools.matlab
module, 76
datadings.tools.msgpack
module, 76
datadings.torch
module, 77
datadings.writer
module, 80
Dataset (*class in datadings.torch*), 78
DatasetBase (*class in datadings.torch*), 78
decode_fast() (*in module datadings.sets.YFCC100m*), 63
DevNull (*class in datadings.sets.YFCC100m*), 61
dict2tuple() (*in module datadings.torch*), 80
DirectoryReader (*class in datadings.reader.directory*), 28
document_keys() (*in module datadings.tools*), 67
download_files_if_not_found() (*in module datadings.tools*), 67
download_if_not_found() (*in module datadings.tools*), 67

E

entry() (*in module datadings.commands.bench*), 19
entry() (*in module datadings.commands.cat*), 19
entry() (*in module datadings.commands.convert_index*), 20
entry() (*in module datadings.commands.merge*), 20
entry() (*in module datadings.commands.sample*), 21
entry() (*in module datadings.commands.shuffle*), 21
entry() (*in module datadings.commands.split*), 22
entry() (*in module datadings.commands.write*), 23
extract() (*in module datadings.sets.VOC2012_write*), 59
extract_boundary() (*in module datadings.sets.Places2017_write*), 53
extract_class() (*in module datadings.sets.Places2017_write*), 53
extract_instance() (*in module datadings.sets.Places2017_write*), 53
extract_scene() (*in module datadings.sets.Places2017_write*), 53
extract_scenelabels() (*in module datadings.sets.ADE20k_write*), 37
extract_scenelabels() (*in module datadings.sets.Places2017_write*), 53
extract_synset_counts() (*in module datadings.sets.ImageNet21k_create_meta*), 47
extractmember() (*in module datadings.sets.VOC2012_write*), 59

F

file_spec() (*in module datadings.sets.Places365_write*), 55
FileWriter (*class in datadings.writer*), 80
filter_invalid_fixpoints() (*in module datadings.sets.CAT2000_write*), 40
find_fixpoints() (*in module datadings.sets.CAT2000_write*), 40
find_index() (*datadings.reader.list.ListReader method*), 30
find_index() (*datadings.reader.msgpack.MsgpackReader method*), 31
find_index() (*datadings.reader.reader.Reader method*), 32
find_index() (*datadings.reader.sharded.ShardedReader method*), 34

find_index() (*datadings.sets.YFCC100m.YFCC100mReader method*), 61
find_key() (*datadings.reader.list.ListReader method*), 30
find_key() (*datadings.reader.msgpack.MsgpackReader method*), 31
find_key() (*datadings.reader.reader.Reader method*), 32
find_key() (*datadings.reader.sharded.ShardedReader method*), 34
find_key() (*datadings.sets.YFCC100m.YFCC100mReader method*), 61
find_sets() (*in module datadings.sets.Places2017_write*), 53
find_writers() (*in module datadings.commands.write*), 23
format_writers() (*in module datadings.commands.write*), 23

G

get() (*datadings.reader.list.ListReader method*), 30
get() (*datadings.reader.msgpack.MsgpackReader method*), 31
get() (*datadings.reader.reader.Reader method*), 32
get() (*datadings.reader.sharded.ShardedReader method*), 34
get() (*datadings.sets.YFCC100m.YFCC100mReader method*), 61
get_files() (*in module datadings.sets.CIFAR10_write*), 42
get_files() (*in module datadings.sets.Places365_write*), 55
get_imageset() (*in module datadings.sets.VOC2012_write*), 59
get_keys() (*in module datadings.sets.Cityscapes_write*), 43
get_keys() (*in module datadings.sets.SALICON2015_write*), 57
glob_pattern() (*in module datadings.reader.directory*), 29
glob_pattern() (*in module datadings.reader.zipfile*), 36

H

hash_keys() (*in module datadings.index*), 23
hash_md5hex() (*in module datadings.tools*), 67
hash_string() (*in module datadings.tools*), 67
hash_string_bytes() (*in module datadings.tools*), 68

I

ImageClassificationData() (*in module datadings.sets.types*), 64
ImageCoarseClassificationData() (*in module datadings.sets.types*), 64
ImageData() (*in module datadings.sets.types*), 64
imagedata_to_array() (*in module datadings.sets.CAMVID_write*), 39
imagedata_to_array() (*in module datadings.sets.VOC2012_write*), 59
imagedata_to_segpng() (*in module datadings.sets.ADE20k_write*), 37
ImageDisparitySegmentationData() (*in module datadings.sets.types*), 64
ImageInstanceSegmentationData() (*in module datadings.sets.types*), 65
ImageNet21kData() (*in module datadings.sets.types*), 65
images_and_labels_iter() (*in module datadings.sets.InriaBuildings_write*), 49
images_label_dsm_iter() (*in module datadings.sets.Vaihingen_write*), 60
ImageSegmentationData() (*in module datadings.sets.types*), 65
index_to_color() (*in module datadings.sets.ADE20k*), 36
index_to_color() (*in module datadings.sets.CAMVID*), 38
index_to_color() (*in module datadings.sets.Places2017*), 51
index_to_color() (*in module datadings.sets.VOC2012*), 58
instance_map() (*in module datadings.sets.ADE20k_write*), 37
iter() (*datadings.reader.augment.Cycler method*), 25
iter() (*datadings.reader.augment.QuasiShuffler method*), 26
iter() (*datadings.reader.augment.Range method*), 27
iter() (*datadings.reader.augment.Repeater method*), 27
iter() (*datadings.reader.augment.Shuffler method*), 27
iter() (*datadings.reader.reader.Reader method*), 33
iter_fields() (*in module datadings.tools.matlab*), 76
iter_frames_with_fixpoints() (*in module datadings.sets.Coutrot1_write*), 44
iter_video_frames_opencv() (*in module datadings.sets.Coutrot1_write*), 44
IterableDataset (*class in datadings.torch*), 79

K

keys_len() (*in module datadings.index*), 23

L

legacy_index_len() (*in module datadings.index*), 23
legacy_load_index() (*in module datadings.index*), 23
ListReader (*class in datadings.reader.list*), 29
load_filter() (*in module datadings.index*), 24

load_index() (*in module datadings.sets.ADE20k_write*), 37
load_json() (*in module datadings.sets.tools*), 63
load_key_hashes() (*in module datadings.index*), 24
load_keys() (*in module datadings.index*), 24
load_lines() (*in module datadings.reader.list*), 30
load_md5file() (*in module datadings.tools*), 68
load_offsets() (*in module datadings.index*), 24
load_statistics() (*in module datadings.sets.ADE20k*), 36
loadmat() (*in module datadings.tools.matlab*), 76
locate_files() (*in module datadings.tools*), 68
LucasKanade (*class in datadings.sets.Coutrot1_write*), 44

M

main() (*in module datadings.commands.bench*), 19
main() (*in module datadings.commands.cat*), 19
main() (*in module datadings.commands.convert_index*), 20
main() (*in module datadings.commands.merge*), 20
main() (*in module datadings.commands.sample*), 21
main() (*in module datadings.commands.shuffle*), 21
main() (*in module datadings.commands.split*), 22
main() (*in module datadings.commands.write*), 23
main() (*in module datadings.sets.ADE20k_write*), 37
main() (*in module datadings.sets.ANP460_write*), 38
main() (*in module datadings.sets.CAMVID_write*), 39
main() (*in module datadings.sets.CAT2000_write*), 40
main() (*in module datadings.sets.CIFAR100_write*), 41
main() (*in module datadings.sets.CIFAR10_write*), 42
main() (*in module datadings.sets.Cityscapes_write*), 43
main() (*in module datadings.sets.Coutrot1_write*), 44
main() (*in module datadings.sets.FIGRIMFixation_write*), 45
main() (*in module datadings.sets.ILSVRC2012_write*), 47
main() (*in module datadings.sets.ImageNet21k_create_meta*), 47
main() (*in module datadings.sets.ImageNet21k_write*), 49
main() (*in module datadings.sets.InriaBuildings_write*), 49
main() (*in module datadings.sets.MIT1003_write*), 50
main() (*in module datadings.sets.MIT300_write*), 51
main() (*in module datadings.sets.Places2017_write*), 53
main() (*in module datadings.sets.Places365_write*), 55
main() (*in module datadings.sets.RIT18_write*), 56
main() (*in module datadings.sets.SALICON2015_write*), 57
main() (*in module datadings.sets.Vaihingen_write*), 60
main() (*in module datadings.sets.VOC2012_write*), 59
main() (*in module datadings.sets.YFCC100m*), 63
make_packer (*in module datadings.tools.msgpack*), 76
make_parser() (*in module datadings.tools.argparse*), 74
make_parser_simple() (*in module datadings.tools.argparse*), 75
make_printer() (*in module datadings.tools*), 68
make_unpacker (*in module datadings.tools.msgpack*), 76
map_color_image() (*in module datadings.sets.VOC2012_write*), 59
map_color_values_to_class_indices() (*in module datadings.sets.Vaihingen_write*), 60
MaskedImageSegmentationData() (*in module datadings.sets.types*), 65
max_value (*datadings.tools.argparse.MinMaxAction attribute*), 71
median_frequency_weights() (*in module datadings.sets.VOC2012*), 58
merge_concat() (*in module datadings.commands.merge*), 20
merge_random() (*in module datadings.commands.merge*), 20
min_value (*datadings.tools.argparse.MinMaxAction attribute*), 71
MinMaxAction (*class in datadings.tools.argparse*), 70
module
    datadings.commands, 18
    datadings.commands.bench, 18
    datadings.commands.cat, 19
    datadings.commands.convert_index, 20
    datadings.commands.merge, 20
    datadings.commands.sample, 21
    datadings.commands.shuffle, 21
    datadings.commands.split, 22
    datadings.commands.write, 22
    datadings.index, 23
    datadings.reader, 25
    datadings.reader.augment, 25
    datadings.reader.directory, 28
    datadings.reader.list, 29
    datadings.reader.msgpack, 31
    datadings.reader.reader, 32
    datadings.reader.sharded, 34
    datadings.reader.zipfile, 35
    datadings.sets, 36
    datadings.sets.ADE20k, 36
    datadings.sets.ADE20k_write, 36
    datadings.sets.ANP460, 37
    datadings.sets.ANP460_write, 37
    datadings.sets.CAMVID, 38
    datadings.sets.CAMVID_write, 38
    datadings.sets.CAT2000, 39
    datadings.sets.CAT2000_write, 39
    datadings.sets.CIFAR10, 40
    datadings.sets.CIFAR100, 40

datadings.sets.CIFAR100_write, 40
datadings.sets.CIFAR10_write, 41
datadings.sets.Cityscapes, 42
datadings.sets.Cityscapes_write, 42
datadings.sets.Coutrot1, 43
datadings.sets.Coutrot1_write, 43
datadings.sets.FIGRIMFixation, 45
datadings.sets.FIGRIMFixation_write, 45
datadings.sets.ILSVRC2012, 46
datadings.sets.ILSVRC2012_synsets, 46
datadings.sets.ILSVRC2012_write, 46
datadings.sets.ImageNet21k_create_meta, 47
datadings.sets.ImageNet21k_synsets, 48
datadings.sets.ImageNet21k_write, 48
datadings.sets.InriaBuildings, 49
datadings.sets.InriaBuildings_write, 49
datadings.sets.MIT1003, 50
datadings.sets.MIT1003_write, 50
datadings.sets.MIT300, 51
datadings.sets.MIT300_write, 51
datadings.sets.Places2017, 51
datadings.sets.Places2017_write, 52
datadings.sets.Places365, 53
datadings.sets.Places365_write, 53
datadings.sets.RIT18, 55
datadings.sets.RIT18_write, 55
datadings.sets.SALICON2015, 56
datadings.sets.SALICON2015_write, 56
datadings.sets.SALICON2017, 57
datadings.sets.SALICON2017_write, 57
datadings.sets.tools, 63
datadings.sets.types, 63
datadings.sets.Vaihingen, 59
datadings.sets.Vaihingen_write, 59
datadings.sets.VOC2012, 58
datadings.sets.VOC2012_write, 58
datadings.sets.YFCC100m, 60
datadings.sets.YFCC100m_counts, 63
datadings.tools, 66
datadings.tools.argparse, 70
datadings.tools.cached_property, 75
datadings.tools.compression, 76
datadings.tools.matlab, 76
datadings.tools.msgpack, 76
datadings.torch, 77
datadings.writer, 80
monitor_interval (*datadings.tools.ProgressPrinter attribute*), 66
MsgpackReader (*class in datadings.reader.msgpack*), 31

N

next() (*datadings.reader.reader.Reader method*), 33
next() (*datadings.sets.YFCC100m.YFCC100mReader method*), 62
no_rng() (*in module datadings.torch*), 80
noop() (*in module datadings.reader.list*), 30
noop() (*in module datadings.sets.YFCC100m*), 63

O

open_comp() (*in module datadings.sets.tools*), 63
open_comp() (*in module datadings.tools.compression*), 76
open_error_file_() (*datadings.sets.YFCC100m.YFCC100mReader method*), 62

P

pack() (*in module datadings.tools.msgpack*), 77
packb() (*in module datadings.tools.msgpack*), 77
path_append() (*in module datadings.tools*), 68
path_append_suffix() (*in module datadings.tools*), 68
Places2017Data() (*in module datadings.sets.types*), 65
prepare_indir() (*in module datadings.tools*), 69
print_classes() (*in module datadings.sets.Places2017_write*), 53
print_over() (*in module datadings.tools*), 69
print_values() (*in module datadings.sets.VOC2012_write*), 59
ProgressPrinter (*class in datadings.tools*), 66

Q

QuasiShuffler (*class in datadings.reader.augment*), 25
query_user() (*in module datadings.tools*), 69

R

Range (*class in datadings.reader.augment*), 26
rawiter() (*datadings.reader.reader.Reader method*), 33
rawnext() (*datadings.reader.reader.Reader method*), 33
rawnext() (*datadings.sets.YFCC100m.YFCC100mReader method*), 62
RawWriter (*class in datadings.writer*), 80
read() (*datadings.sets.YFCC100m.DevNull method*), 61
Reader (*class in datadings.reader.reader*), 32
reader_directory() (*in module datadings.commands.bench*), 19
reader_msgpack() (*in module datadings.commands.bench*), 19
Repeater (*class in datadings.reader.augment*), 27
row2image() (*in module datadings.sets.CIFAR10_write*), 42
run() (*datadings.tools.Yielder method*), 66
run() (*datadings.tools.YielderProc method*), 67

S

SaliencyData() (*in module datadings.sets.types*), 66

SaliencyExperiment() (*in module datadings.sets.types*), 66
SaliencyTimeseriesExperiment() (*in module datadings.sets.types*), 66
sample() (*in module datadings.commands.sample*), 21
seek() (*datadings.reader.augment.Cycler method*), 25
seek() (*datadings.reader.augment.QuasiShuffler method*), 26
seek() (*datadings.reader.augment.Range method*), 27
seek() (*datadings.reader.augment.Repeater method*), 27
seek() (*datadings.reader.augment.Shuffler method*), 28
seek() (*datadings.reader.reader.Reader method*), 34
seek_index() (*datadings.reader.reader.Reader method*), 34
seek_index() (*datadings.sets.YFCC100m.YFCC100mReader method*), 62
seek_key() (*datadings.reader.reader.Reader method*), 34
seek_key() (*datadings.sets.YFCC100m.YFCC100mReader method*), 62
segmentation_map() (*in module datadings.sets.ADE20k_write*), 37
select_set() (*in module datadings.commands.merge*), 20
SentinelEnd (*class in datadings.tools*), 66
SentinelError (*class in datadings.tools*), 66
setup_ranges() (*in module datadings.commands.merge*), 21
ShardedReader (*class in datadings.reader.sharded*), 34
shuffle() (*in module datadings.commands.shuffle*), 21
Shuffler (*class in datadings.reader.augment*), 27
single_value() (*in module datadings.sets.VOC2012_write*), 59
slice() (*datadings.reader.list.ListReader method*), 30
slice() (*datadings.reader.msgpack.MsgpackReader method*), 31
slice() (*datadings.reader.reader.Reader method*), 34
slice() (*datadings.reader.sharded.ShardedReader method*), 35
slice() (*datadings.sets.YFCC100m.YFCC100mReader method*), 62
sorted_labels() (*in module datadings.reader.list*), 30
sorted_values() (*in module datadings.sets.VOC2012_write*), 59
split_array() (*in module datadings.tools*), 69
split_dataset() (*in module datadings.commands.split*), 22
stop() (*datadings.tools.Yielder method*), 67
stop() (*datadings.tools.YielderProc method*), 67

T

tiff_to_nd_array() (*in module datadings.tools*), 70
track() (*datadings.sets.Coutrot1_write.LucasKanade method*), 44
transform_points() (*in module datadings.sets.CAT2000_write*), 40

U

unpack() (*in module datadings.tools.msgpack*), 77
unpackb() (*in module datadings.tools.msgpack*), 77
update() (*datadings.sets.Coutrot1_write.LucasKanade method*), 44

V

validate_image() (*in module datadings.sets.YFCC100m*), 63
verify_data() (*datadings.reader.msgpack.MsgpackReader method*), 31
verify_file() (*in module datadings.tools*), 70
verify_files() (*in module datadings.tools*), 70
verify_image() (*in module datadings.sets.ILSVRC2012_write*), 47
verify_index() (*datadings.reader.msgpack.MsgpackReader method*), 32

W

write() (*datadings.sets.YFCC100m.DevNull method*), 61
write() (*datadings.writer.FileWriter method*), 80
write() (*datadings.writer.RawWriter method*), 80
write() (*datadings.writer.Writer method*), 81
write() (*in module datadings.sets.InriaBuildings_write*), 50
write() (*in module datadings.sets.RIT18_write*), 56
write() (*in module datadings.sets.Vaihingen_write*), 60
write_filter() (*in module datadings.index*), 24
write_image() (*in module datadings.sets.ANP460_write*), 38
write_image() (*in module datadings.sets.CAMVID_write*), 39
write_image() (*in module datadings.sets.MIT1003_write*), 50
write_image() (*in module datadings.sets.MIT300_write*), 51
write_image() (*in module datadings.sets.VOC2012_write*), 59
write_key_hashes() (*in module datadings.index*), 24
write_keys() (*in module datadings.index*), 24
write_offsets() (*in module datadings.index*), 25
write_set() (*in module datadings.sets.ADE20k_write*), 37
write_set() (*in module datadings.sets.CAMVID_write*), 39

write_set() (*in module datadings.sets.CAT2000_write*), 40
write_set() (*in module datadings.sets.CIFAR100_write*), 41
write_set() (*in module datadings.sets.CIFAR10_write*), 42
write_set() (*in module datadings.sets.Cityscapes_write*), 43
write_set() (*in module datadings.sets.FIGRIMFixation_write*), 45
write_set() (*in module datadings.sets.ILSVRC2012_write*), 47
write_set() (*in module datadings.sets.Places2017_write*), 53
write_set() (*in module datadings.sets.RIT18_write*), 56
write_set() (*in module datadings.sets.SALICON2015_write*), 57
write_set() (*in module datadings.sets.VOC2012_write*), 59
write_sets() (*in module datadings.sets.ADE20k_write*), 37
write_sets() (*in module datadings.sets.ANP460_write*), 38
write_sets() (*in module datadings.sets.CAMVID_write*), 39
write_sets() (*in module datadings.sets.CAT2000_write*), 40
write_sets() (*in module datadings.sets.CIFAR100_write*), 41
write_sets() (*in module datadings.sets.CIFAR10_write*), 42
write_sets() (*in module datadings.sets.Cityscapes_write*), 43
write_sets() (*in module datadings.sets.Coutrot1_write*), 44
write_sets() (*in module datadings.sets.FIGRIMFixation_write*), 45
write_sets() (*in module datadings.sets.ILSVRC2012_write*), 47
write_sets() (*in module datadings.sets.ImageNet21k_write*), 49
write_sets() (*in module datadings.sets.InriaBuildings_write*), 50
write_sets() (*in module datadings.sets.MIT1003_write*), 50
write_sets() (*in module datadings.sets.MIT300_write*), 51
write_sets() (*in module datadings.sets.Places2017_write*), 53
write_sets() (*in module datadings.sets.Places365_write*), 55
write_sets() (*in module datadings.sets.RIT18_write*), 56
write_sets() (*in module datadings.sets.SALICON2015_write*), 57
write_sets() (*in module datadings.sets.Vaihingen_write*), 60
write_sets() (*in module datadings.sets.VOC2012_write*), 59
write_train() (*in module datadings.sets.RIT18_write*), 56
write_video() (*in module datadings.sets.Coutrot1_write*), 44
Writer (*class in datadings.writer*), 80
writer_link() (*in module datadings.commands.write*), 23

Y

YesNoAction (*class in datadings.tools.argparse*), 71
YFCC100mReader (*class in datadings.sets.YFCC100m*), 61
yield_directory() (*in module datadings.reader.directory*), 29
yield_file() (*in module datadings.reader.directory*), 29
yield_images() (*in module datadings.sets.ADE20k_write*), 37
yield_process() (*in module datadings.tools*), 70
yield_rows() (*in module datadings.sets.CIFAR100_write*), 41
yield_rows() (*in module datadings.sets.CIFAR10_write*), 42
yield_samples() (*in module datadings.sets.CAT2000_write*), 40
yield_samples() (*in module datadings.sets.Cityscapes_write*), 43
yield_samples() (*in module datadings.sets.ILSVRC2012_write*), 47
yield_samples() (*in module datadings.sets.ImageNet21k_write*), 49
yield_samples() (*in module datadings.sets.SALICON2015_write*), 57
yield_threaded() (*in module datadings.tools*), 70
yield_train() (*in module datadings.sets.ILSVRC2012_write*), 47
yield_val() (*in module datadings.sets.ILSVRC2012_write*), 47
yield_zipfile() (*in module datadings.reader.zipfile*), 36
Yielder (*class in datadings.tools*), 66
YielderProc (*class in datadings.tools*), 67

Z

ZipFileReader (*class in datadings.reader.zipfile*), 35