
epic Documentation

Release 1.6.3

Will Price, Michael Wray

Feb 16, 2019

Contents

1	epic_kitchens package	3
1.1	epic_kitchens.dataset	3
1.2	epic_kitchens.dataset.epic_dataset module	3
1.3	epic_kitchens.dataset.video_dataset module	5
1.4	epic_kitchens.gulp	5
1.5	epic_kitchens.gulp.adapter	5
1.6	epic_kitchens.gulp.visualisation	7
1.7	epic_kitchens.meta	8
1.8	epic_kitchens.preprocessing	13
1.9	epic_kitchens.preprocessing.split_segments	13
1.10	epic_kitchens.labels	13
1.11	epic_kitchens.time	14
1.12	epic_kitchens.video	17
2	CLI tools	19
2.1	Action segmentation	19
2.2	Gulp data ingestor	19
	Python Module Index	21

[EPIC Kitchens](#) is an egocentric vision dataset, visit the website to find out more.

In addition to the dataset, we provide a python library for performing common tasks on the dataset.

Have a look at [EPIC-mini](#) for example usage.

epic-kitchens

This library contains a variety of useful classes and functions for performing common operations on the [EPIC Kitchens](#) dataset.

1.1 epic_kitchens.dataset

1.2 epic_kitchens.dataset.epic_dataset module

```
class epic_kitchens.dataset.epic_dataset.EpicVideoDataset (gulp_path,  
                                                         class_type, *,  
                                                         with_metadata=False,  
                                                         class_getter=None,  
                                                         segment_filter=None,  
                                                         sample-  
                                                         transform=None)
```

Bases: *epic_kitchens.dataset.video_dataset.VideoDataset*

VideoDataset for gulped RGB frames

```
__init__ (gulp_path, class_type, *, with_metadata=False, class_getter=None, segment_filter=None,  
          sample_transform=None)
```

Parameters

- **gulp_path** (`Union[Path, str]`) – Path to gulp directory containing the gulped EPIC RGB or flow frames
- **class_type** (`str`) – One of verb, noun, verb+noun, None, determines what label the segment returns. None should be used for loading test datasets.
- **with_metadata** (`bool`) – When True the segments will yield a tuple (metadata, class) where the class is defined by the class getter and the metadata is the raw dictionary stored in the gulp file.

- **class_getter** (Optional[Callable[[Dict[str, Any]], Any]]) – Optionally provide a callable that takes in the gulp dict representing the segment from which you should return the class you wish the segment to have.
- **segment_filter** (Optional[Callable[[VideoSegment], bool]]) – Optionally provide a callable that takes a segment and returns True if you want to keep the segment in the dataset, or False if you wish to exclude it.
- **sample_transform** (Optional[Callable[[List[Image]], List[Image]]) – Optionally provide a sample transform function which takes a list of PIL images and transforms each of them. This is applied on the frames just before returning from `load_frames()`.

Return type None

load_frames (*segment*, *indices=None*)

Load frame(s) from gulp directory.

Parameters

- **segment** (*VideoSegment*) – Video segment to load
- **indices** (Optional[Iterable[int]]) – Frames indices to read

Return type List[Image]

Returns Frames indexed by indices from the segment.

video_segments

List of video segments that are present in the dataset. They describe the start and stop times of the clip and its class.

Return type List[VideoSegment]

class epic_kitchens.dataset.epic_dataset.**EpicVideoFlowDataset** (*gulp_path*,
class_type, *,
with_metadata=False,
class_getter=None,
segment_filter=None,
sample_transform=None)

Bases: *epic_kitchens.dataset.epic_dataset.EpicVideoDataset*

VideoDataset for loading gupled flow. The loader assumes that flow u, v frames are stored alternately in a flat manner: $[u_0, v_0, u_1, v_1, \dots, u_n, v_n]$

class epic_kitchens.dataset.epic_dataset.**GulpVideoSegment** (*gulp_metadata_dict*,
class_getter)

Bases: *epic_kitchens.dataset.video_dataset.VideoSegment*

SegmentRecord for a video segment stored in a gulp file.

Assumes that the video segment has the following metadata in the gulp file:

- id
- num_frames

id

ID of video segment

Return type str

label

Return type `Any`

num_frames

Number of video frames

Return type `int`

1.3 epic_kitchens.dataset.video_dataset module

```
class epic_kitchens.dataset.video_dataset.VideoDataset(class_count, segment_filter=None, sample_transform=None)
```

Bases: `abc.ABC`

A dataset interface for use with `TsnDataset`. Implement this interface if you wish to use your dataset with TSN.

We cannot use `torch.utils.data.Dataset` because we need to yield information about the number of frames per video, which we can't do with the standard `torch.utils.data.Dataset`.

load_frames (*segment*, *idx=None*)

Return type `List[Image]`

video_segments

Return type `List[VideoSegment]`

```
class epic_kitchens.dataset.video_dataset.VideoSegment
```

Bases: `abc.ABC`

Represents a video segment with an associated label.

id

label

num_frames

Return type `int`

1.4 epic_kitchens.gulp

Dataset Adapters for GulpIO.

This module contains two adapters for 'gulping' both RGB and flow frames which can then be used with the `EpicVideoDataset` classes.

1.5 epic_kitchens.gulp.adapter

```
class epic_kitchens.gulp.adapter.EpicDatasetAdapter(video_segment_dir, annotations_df, frame_size=-1, extension='jpg', labelled=True)
```

Bases: `gulpio.adapters.AbstractDatasetAdapter`

Gulp Dataset Adapter for Gulping RGB frames extracted from the EPIC-KITCHENS dataset

`__init__(video_segment_dir, annotations_df, frame_size=-1, extension='jpg', labelled=True)`
 Gulp all action segments in `annotations_df` reading the dumped frames from `video_segment_dir`

Parameters

- **video_segment_dir** (*str*) – Root directory containing segmented frames:

```

frame-segments/
├── P01
│   ├── P01_01
│   │   ├── P01_01_0_open-door
│   │   │   ├── frame_0000000008.jpg
│   │   │   ├── ...
│   │   │   └── frame_0000000202.jpg
│   │   ├── ...
│   │   └── P01_01_329_put-down-plate
│   │       ├── frame_0000098424.jpg
│   │       ├── ...
│   │       └── frame_0000098501.jpg
│   └── ...
└── ...
  
```

- **annotations_df** (*DataFrame*) – *DataFrame* containing labels to be gulped.
- **frame_size** (*int*) – Size of shortest edge of the frame, if not already this size then it will be resized.
- **extension** (*str*) – Extension of dumped frames.

Return type *None*

iter_data (*slice_element=None*)

Get frames and metadata corresponding to segment

Parameters **slice_element** (*optional*) – If not specified all frames for the segment will be returned

Yields *dict* – dictionary with the fields

- **meta**: All metadata corresponding to the segment, this is the same as the data in the labels csv
- **frames**: list of `PIL.Image.Image` corresponding to the frames specified in `slice_element`
- **id**: UID corresponding to segment

Return type `Iterator[Dict[str, Any]]`

class `epic_kitchens.gulp.adapter.EpicFlowDatasetAdapter` (*video_segment_dir, annotations_df, frame_size=-1, extension='jpg', labelled=True*)

Bases: `epic_kitchens.gulp.adapter.EpicDatasetAdapter`

Gulp Dataset Adapter for Gulping flow frames extracted from the EPIC-KITCHENS dataset

iter_data (*slice_element=None*)

Get frames and metadata corresponding to segment

Parameters **slice_element** (*optional*) – If not specified all frames for the segment will be returned

Yields *dict* – dictionary with the fields

- `meta`: All metadata corresponding to the segment, this is the same as the data in the labels csv
- `frames`: list of `PIL.Image.Image` corresponding to the frames specified in `slice_element`
- `id`: UID corresponding to segment

exception `epic_kitchens.gulp.adapter.MissingDataException`

Bases: `Exception`

1.6 epic_kitchens.gulp.visualisation

class `epic_kitchens.gulp.visualisation.FlowVisualiser` (*dataset*)

Bases: `epic_kitchens.gulp.visualisation.Visualiser`

Visualiser for video dataset containing optical flow (u, v) frames

class `epic_kitchens.gulp.visualisation.RgbVisualiser` (*dataset*)

Bases: `epic_kitchens.gulp.visualisation.Visualiser`

Visualiser for video dataset containing RGB frames

class `epic_kitchens.gulp.visualisation.Visualiser` (*dataset*)

Bases: `abc.ABC`

show (*uid, **kwargs*)

Show the given video corresponding to `uid` in a HTML5 video element.

Parameters

- `uid` (`Union[int, str]`) – UID of video segment
- `fps` (`float, optional`) – FPS of video sequence

Return type `HTML`

`epic_kitchens.gulp.visualisation.clipify_flow` (*frames, *, fps=30.0*)

Destack flow frames, join them side by side and then create a clip for display

Parameters

- `frames` (`List[Image]`) – A list of alternating u, v flow frames to join into a video. Even indices should be u flow frames, and odd indices, v flow frames.
- `fps` (`float`) – float, optional FPS of generated `moviepy.editor.ImageSequenceClip`

Return type `ImageSequenceClip`

`epic_kitchens.gulp.visualisation.clipify_rgb` (*frames, *, fps=60.0*)

Parameters

- `frames` (`List[Image]`) – A list of frames
- `fps` (`float`) – FPS of clip

Returns Frames concatenated into clip

Return type `moviepy.editor.ImageSequenceClip`

`epic_kitchens.gulp.visualisation.combine_flow_uv_frames` (*uv_frames*, *method='hstack'*, *width_axis=2*), *

Destack (u, v) frames and concatenate them side by side for display purposes

Return type `ndarray`

`epic_kitchens.gulp.visualisation.hstack_frames` (**frame_sequences*, *width_axis=2*)

Return type `ndarray`

1.7 epic_kitchens.meta

class `epic_kitchens.meta.Action` (*verb*, *noun*)

Bases: `tuple`

noun

Alias for field number 1

verb

Alias for field number 0

class `epic_kitchens.meta.ActionClass` (*verb_class*, *noun_class*)

Bases: `tuple`

noun_class

Alias for field number 1

verb_class

Alias for field number 0

`epic_kitchens.meta.class_to_noun` (*cls*)

Parameters `cls` (`int`) – numeric noun class

Return type `str`

Returns Canonical noun representing the class

Raises `IndexError` – if `cls` is an invalid noun class

`epic_kitchens.meta.class_to_verb` (*cls*)

Parameters `cls` (`int`) – numeric verb class

Return type `str`

Returns Canonical verb representing the class

Raises `IndexError` – if `cls` is an invalid verb class

`epic_kitchens.meta.get_datadir` ()

Return type `Path`

Returns Directory under which any downloaded files are stored, defaults to current working directory

`epic_kitchens.meta.is_many_shot_action` (*action_class*)

Parameters `action_class` (`ActionClass`) – (`verb_class`, `noun_class`) tuple

Return type `bool`

Returns Whether `action_class` is many shot or not

`epic_kitchens.meta.is_many_shot_noun(noun_class)`

Parameters `noun_class` (`int`) – numeric noun class

Return type `bool`

Returns Whether noun class is many shot or not

`epic_kitchens.meta.is_many_shot_verb(verb_class)`

Parameters `verb_class` (`int`) – numeric verb class

Return type `bool`

Returns Whether verb_class is many shot or not

`epic_kitchens.meta.many_shot_actions()`

Return type `Set[ActionClass]`

Returns The set of actions classes that are many shot (`verb_class` appears more than 100 times in training, `noun_class` appears more than 100 times in training, and the action appears at least once in training).

`epic_kitchens.meta.many_shot_nouns()`

Return type `Set[int]`

Returns The set of noun classes that are many shot (appear more than 100 times in training).

`epic_kitchens.meta.many_shot_verbs()`

Return type `Set[int]`

Returns The set of verb classes that are many shot (appear more than 100 times in training).

`epic_kitchens.meta.noun_classes()`

Get dataframe containing the mapping between numeric noun classes, the canonical noun of that class and nouns clustered into the class.

Returns

Column Name	Type	Example	Description
<code>noun_id</code>	<code>int</code>	2	ID of the noun class.
<code>class_key</code>	<code>string</code>	<code>pan:dust</code>	Key of the noun class.
<code>nouns</code>	list of string (1 or more)	<code>"['pan:dust', 'dustpan']"</code>	All nouns within the class (includes the key).

Return type Dataframe with the columns

`epic_kitchens.meta.noun_to_class(noun)`

Parameters `noun` (`str`) – A noun from a narration

Return type `int`

Returns The corresponding numeric class of the noun if it exists

Raises `IndexError` – If the noun doesn't belong to any of the noun classes

`epic_kitchens.meta.set_datadir(dir_)`

Set download directory

Parameters `dir` – Path to directory in which to store all downloaded metadata files

`epic_kitchens.meta.set_version(version)`

`epic_kitchens.meta.test_timestamps(split)`

Parameters `split` (*str*) – ‘seen’, ‘unseen’, or ‘all’ (loads both with a ‘split’)

Return type `DataFrame`

Returns

Dataframe with the columns

Column Name	Type	Example	Description
<code>uid</code>	<code>int</code>	1924	Unique ID of the segment.
<code>participant</code>	<code>string</code>	P01	ID of the participant.
<code>video_id</code>	<code>string</code>	P01_11	Video the segment is in.
<code>start_time</code>	<code>string</code>	00:00:00 000	Start time in HH:mm:ss.SSS of the action.
<code>stop_time</code>	<code>string</code>	00:00:01 890	End time in HH:mm:ss.SSS of the action.
<code>start_frame</code>	<code>int</code>	1	Start frame of the action (WARNING only for frames extracted as detailed in annotations README).
<code>stop_frame</code>	<code>int</code>	93	End frame of the action (WARNING only for frames extracted as detailed in annotations README).

`epic_kitchens.meta.training_labels()`

Return type `DataFrame`

Returns

Dataframe with the columns

Column Name	Type	Example	Description
uid	int	6374	Unique ID of the segment.
video_id	string	P03_01	Video the segment is in.
narration	string	close fridge	English description of the action provided by the participant.
start_time	string	00:23:43 847	Start time in HH:mm:ss.SSS of the action.
stop_time	string	00:23:47 212	End time in HH:mm:ss.SSS of the action.
start_frame	int	85430	Start frame of the action (WARNING only for frames extracted as detailed in annotations README)
stop_frame	int	85643	End frame of the action (WARNING only for frames extracted as detailed in annotations README)
participant	string	P03	ID of the participant.
verb	string	close	Parsed verb from the narration.
noun	string	fridge	First parsed noun from the narration.
verb_class	int	3	Numeric ID of the parsed verb's class.
noun_class	int	10	Numeric ID of the parsed noun's class.
all_nouns	list of string (1 or more)	['fridge']	List of all parsed nouns from the narration.
all_nouns_classes	list of int (1 or more)	[10]	List of numeric IDs corresponding to all of the parsed nouns' classes from the narration.

```
epic_kitchens.meta.training_narrations()
```

Return type DataFrame

Returns

Dataframe with the columns

Column Name	Type	Example	Description
participant_id	string	P03	ID of the participant.
video_id	string	P03_01	Video the segment is in.
start_timestamp	string	00:23:43. 847	Start time in HH:mm:ss.SSS of the narration.
stop_timestamp	string	00:23:47. 212	End time in HH:mm:ss.SSS of the narration.
narration	string	close fridge	English description of the action provided by the participant.

```
epic_kitchens.meta.training_object_labels()
```

Return type DataFrame

Returns

Dataframe with the columns

Column Name	Type	Example	Description
noun_class	int	20	Integer value representing the class in noun-classes.csv.
noun	string	bag	Original string name for the object.
participant	stringid	P01	ID of participant.
video_id	string	P01_01	Video the object was annotated in.
frame	int	056581	Frame number of the annotated object.
bounding_boxes	list of 4-tuple (0 or more)	"[(76, 1260, 462, 186)]"	Annotated boxes with format (<top:int>, <left:int>, <height:int>, <width:int>).

`epic_kitchens.meta.verb_classes()`

Get dataframe containing the mapping between numeric verb classes, the canonical verb of that class and verbs clustered into the class.

Return type DataFrame

Returns

Dataframe with the columns

Column Name	Type	Example	Description
verb_id	int	3	ID of the verb class.
class_key	string	close	Key of the verb class.
verbs	list of string (1 or more)	"['close', 'close-off', 'shut']"	All verbs within the class (includes the key).

`epic_kitchens.meta.verb_to_class(verb)`

Parameters `verb` (str) – A noun from a narration

Return type int

Returns The corresponding numeric class of the verb if it exists

Raises `IndexError` – If the verb doesn't belong to any of the verb classes

`epic_kitchens.meta.video_descriptions()`

Return type DataFrame

Returns

High level description of the task trying to be accomplished in a video

Column Name	Type	Example	Description
video_id	string	P01_01	ID of the video.
date	string	30/04/2017	Date on which the video was shot.
time	string	13:49:00	Local recording time of the video.
description	string	prepared breakfast with soy milk and cereals	Description of the activities contained in the video.


```
epic_kitchens.meta.video_info()
```

Return type DataFrame

Returns

Technical information stating the resolution, duration and FPS of each video.

Column Name	Type	Example	Description
video	string	P01_01	Video ID
resolution	string	1920x1080	Resolution of the video, format is WIDTHxHEIGHT
duration	float	1652.152817	Duration of the video, in seconds
fps	float	59. 9400599400599	Frame rate of the video

1.8 epic_kitchens.preprocessing

Pre-processing tools to munge data into a format suitable for training

1.9 epic_kitchens.preprocessing.split_segments

Program for splitting frames into action segments See *Action segmentation* for usage details

```
epic_kitchens.preprocessing.split_segments.main(args)
```

1.10 epic_kitchens.labels

Column names present in a labels dataframe.

Rather than accessing column names directly, we suggest you import these constants and use them to access the data in case the names change at any point.

```
epic_kitchens.labels.NARRATION_COL = 'narration'
```

Start timestamp column name, the timestamp of the start of the action segment

e.g. "00:23:43.847"

```
epic_kitchens.labels.NOUNS_CLASS_COL = 'all_noun_classes'
```

The noun class corresponding to an action without a noun, consider the narration “stir” where no object is specified.

```
epic_kitchens.labels.NOUNS_COL = 'all_nouns'
```

Nouns class column name, the classes corresponding to each noun extracted from the narration

e.g. [10]

```
epic_kitchens.labels.NOUN_CLASS_COL = 'noun_class'
```

Nouns column name, all nouns extracted from the narration

e.g. ["fridge"]

`epic_kitchens.labels.NOUN_COL = 'noun'`

Noun class column name, the class corresponding to the first noun extracted from the narration

e.g. 10

`epic_kitchens.labels.PARTICIPANT_ID_COL = 'participant_id'`

Verb column name, the first verb extracted from the narration

e.g. "close"

`epic_kitchens.labels.START_F_COL = 'start_frame'`

Stop frame column name, the frame corresponding to the starting timestamp

e.g. 85643

`epic_kitchens.labels.START_TS_COL = 'start_timestamp'`

Stop timestamp column name, the timestamp of the end of the action segment

e.g. "00:23:47.212"

`epic_kitchens.labels.STOP_F_COL = 'stop_frame'`

Participant ID column name, the identifier corresponding to an individual

e.g. 85643

`epic_kitchens.labels.STOP_TS_COL = 'stop_timestamp'`

Start frame column name, the frame corresponding to the starting timestamp

e.g. 85430

`epic_kitchens.labels.UID_COL = 'uid'`

Video column name, an identifier for a specific video of the form Pdd_dd, the first two digits are the participant ID, and the last two digits the video ID

e.g. "P03_01"

`epic_kitchens.labels.VERB_CLASS_COL = 'verb_class'`

Noun column name, the first noun extracted from the narration

e.g. "fridge"

`epic_kitchens.labels.VERB_COL = 'verb'`

Verb class column name, the class corresponding to the verb extracted from the narration.

e.g. 3

`epic_kitchens.labels.VIDEO_ID_COL = 'video_id'`

Narration column name, the original narration by the participant about the action performed

e.g. "close fridge"

1.11 epic_kitchens.time

Functions for converting between frames and timestamps

`epic_kitchens.time.flow_frame_count` (*rgb_frame*, *stride*, *dilation*)

Get the number of frames in a optical flow segment given the number of frames in the corresponding rgb segment from which the flow was extracted with parameters (*stride*, *dilation*)

Parameters

- **rgb_frame** (*int*) – RGB Frame number
- **stride** (*int*) – Stride used in extracting optical flow

- **dilation** (*int*) – Dilation used in extracting optical flow

Return type `int`

Returns The number of optical flow frames

Examples

```
>>> flow_frame_count(6, 1, 1)
5
>>> flow_frame_count(6, 2, 1)
3
>>> flow_frame_count(6, 1, 2)
4
>>> flow_frame_count(6, 2, 2)
2
>>> flow_frame_count(6, 3, 1)
2
>>> flow_frame_count(6, 1, 3)
3
>>> flow_frame_count(7, 1, 1)
6
>>> flow_frame_count(7, 2, 1)
3
>>> flow_frame_count(7, 1, 2)
5
>>> flow_frame_count(7, 2, 2)
3
>>> flow_frame_count(7, 3, 1)
2
>>> flow_frame_count(7, 1, 3)
4
```

`epic_kitchens.time.seconds_to_timestamp` (*total_seconds*)

Convert seconds into a timestamp

Parameters `total_seconds` (*float*) – time in seconds

Return type `str`

Returns timestamp representing `total_seconds`

Examples

```
>>> seconds_to_timestamp(1)
'00:00:1.000'
>>> seconds_to_timestamp(1.1)
'00:00:1.100'
>>> seconds_to_timestamp(60)
'00:01:0.000'
>>> seconds_to_timestamp(61)
'00:01:1.000'
>>> seconds_to_timestamp(60 * 60 + 1)
'01:00:1.000'
>>> seconds_to_timestamp(60 * 60 + 60 + 1)
'01:01:1.000'
```

(continues on next page)

(continued from previous page)

```
>>> seconds_to_timestamp(1225.78500002)
'00:20:25.785'
```

`epic_kitchens.time.timestamp_to_frame` (*timestamp*, *fps*)

Convert timestamp to frame number given the FPS of the extracted frames

Parameters

- **timestamp** (*str*) – formatted as HH:MM:SS[.FractionalPart]
- **fps** (*float*) – frames per second

Return type `int`

Returns frame corresponding timestamp

Examples

```
>>> timestamp_to_frame("00:00:00", 29.97)
1
>>> timestamp_to_frame("00:00:01", 29.97)
29
>>> timestamp_to_frame("00:00:01", 59.94)
59
>>> timestamp_to_frame("00:01:00", 60)
3600
>>> timestamp_to_frame("01:00:00", 60)
216000
```

`epic_kitchens.time.timestamp_to_seconds` (*timestamp*)

Convert a timestamp into total number of seconds

Parameters **timestamp** (*str*) – formatted as HH:MM:SS[.FractionalPart]

Return type `float`

Returns timestamp converted to seconds

Examples

```
>>> timestamp_to_seconds("00:00:00")
0.0
>>> timestamp_to_seconds("00:00:05")
5.0
>>> timestamp_to_seconds("00:00:05.5")
5.5
>>> timestamp_to_seconds("00:01:05.5")
65.5
>>> timestamp_to_seconds("01:01:05.5")
3665.5
```

1.12 epic_kitchens.video

class epic_kitchens.video.**FlowModalityIterator** (*dilation=1, stride=1, bound=20, rgb_fps=59.94*)

Bases: *epic_kitchens.video.ModalityIterator*

Iterator for optical flow (*u, v*) frames

__init__ (*dilation=1, stride=1, bound=20, rgb_fps=59.94*)

Parameters

- **dilation** – Dilation that optical flow was extracted with
- **stride** – Stride that optical flow was extracted with
- **bound** – Bound that optical flow was extracted with
- **rgb_fps** – FPS of RGB video flow was computed from

frame_iterator (*start, stop*)

Parameters

- **start** (*str*) – start time (timestamp: HH:MM:SS[.FractionalPart])
- **stop** (*str*) – stop time (timestamp: HH:MM:SS[.FractionalPart])

Yields Frame indices iterator corresponding to segment from *start* to *stop*

Return type `Iterable[int]`

class epic_kitchens.video.**ModalityIterator**

Bases: *abc.ABC*

Interface that a modality extracted from video must implement

frame_iterator (*start, stop*)

Parameters

- **start** (*str*) – start time (timestamp: HH:MM:SS[.FractionalPart])
- **stop** (*str*) – stop time (timestamp: HH:MM:SS[.FractionalPart])

Yields Frame indices iterator corresponding to segment from *start* to *stop*

Return type `Iterable[int]`

class epic_kitchens.video.**RGBModalityIterator** (*fps*)

Bases: *epic_kitchens.video.ModalityIterator*

Iterator for RGB frames

frame_iterator (*start, stop*)

Parameters

- **start** (*str*) – start time (timestamp: HH:MM:SS[.FractionalPart])
- **stop** (*str*) – stop time (timestamp: HH:MM:SS[.FractionalPart])

Yields Frame indices iterator corresponding to segment from *start* to *stop*

Return type `Iterable[int]`

epic_kitchens.video.get_narration (*annotation*)

Get narration from annotation row, defaults to "unnarrated" if row has no narration column.

`epic_kitchens.video.iterate_frame_dir` (*root*)

Iterate over a directory of video dirs with the hierarchy `root/P01/P01_01/`

Parameters `root` (`Path`) – Root directory with person directory children, then each person directory has video directory children e.g. `root -> P01 -> P01_01`

Yields (`person_dir`, `video_dir`)

Return type `Iterator[Tuple[Path, Path]]`

`epic_kitchens.video.split_dataset_frames` (*modality_iterator*, *frames_dir*, *segment_root_dir*, *annotations*, *frame_format*='frame%06d.jpg', *pattern*=`re.compile('.*')`)

Split dumped video frames from `frames_dir` into directories within `segment_root_dir` for each video segment defined in `annotations`.

Parameters

- **modality_iterator** (`ModalityIterator`) – Modality iterator of frames
- **frames_dir** (`Path`) – Directory containing dumped frames
- **segment_root_dir** (`Path`) – Directory to write split segments to
- **annotations** (`DataFrame`) – Dataframe containing segment information
- **frame_format** (`str`, *optional*) – Old style string format that must contain a single `%d` formatter describing file name format of the dumped frames.
- **pattern** (`re.Pattern`, *optional*) – Regexp to match video directories

Return type `None`

`epic_kitchens.video.split_video_frames` (*modality_iterator*, *frame_format*, *video_annotations*, *segment_root_dir*, *video_dir*)

Split frames from a single video file stored in `video_dir` into segment directories stored in `segment_root_dir`.

Parameters

- **modality_iterator** (`ModalityIterator`) – Modality iterator
- **frame_format** (`str`) – Old style string format that must contain a single `%d` formatter describing file name format of the dumped frames.
- **video_annotations** (`DataFrame`) – Dataframe containing rows only corresponding to video frames stored in **:param:'video_dir'**
- **segment_root_dir** (`Path`) – Directory to write split segments to
- **video_dir** (`Path`) – Directory containing dumped frames for a single video

Return type `None`

2.1 Action segmentation

A script is provided to segment frames from a video into a set of action-segments.

```
$ python -m epic_kitchens.preprocessing.split_segments \  
  P03 \  
  path/to/frames \  
  path/to/frame-segments \  
  path/to/labels.pkl \  
  RGB \  
  --fps 60 \  
  --frame-format 'frame_%010d.jpg' \  
  --of-stride 2 \  
  --of-dilation 3
```

2.2 Gulp data ingestor

```
$ python -m epic_kitchens.gulp \  
  path/to/frame-segments \  
  path/to/gulp-dir \  
  path/to/labels.pkl \  
  RGB \  
  --num-workers $(nproc) \  
  --segments-per-chunk 100 \  
  --
```


e

- `epic_kitchens`, 3
- `epic_kitchens.dataset`, 3
- `epic_kitchens.dataset.epic_dataset`, 3
- `epic_kitchens.dataset.video_dataset`, 5
- `epic_kitchens.gulp`, 5
- `epic_kitchens.gulp.adapter`, 5
- `epic_kitchens.gulp.visualisation`, 7
- `epic_kitchens.labels`, 13
- `epic_kitchens.meta`, 8
- `epic_kitchens.preprocessing`, 13
- `epic_kitchens.preprocessing.split_segments`, 13
- `epic_kitchens.time`, 14
- `epic_kitchens.video`, 17

Symbols

- `__init__()` (*epic_kitchens.dataset.epic_dataset.EpicVideoDataset* method), 3
`__init__()` (*epic_kitchens.gulp.adapter.EpicDatasetAdapter* method), 5
`__init__()` (*epic_kitchens.video.FlowModalityIterator* method), 17
- A**
- Action (class in *epic_kitchens.meta*), 8
 ActionClass (class in *epic_kitchens.meta*), 8
- C**
- `class_to_noun()` (in module *epic_kitchens.meta*), 8
`class_to_verb()` (in module *epic_kitchens.meta*), 8
`clipify_flow()` (in module *epic_kitchens.gulp.visualisation*), 7
`clipify_rgb()` (in module *epic_kitchens.gulp.visualisation*), 7
`combine_flow_uv_frames()` (in module *epic_kitchens.gulp.visualisation*), 7
- E**
- epic_kitchens* (module), 3
epic_kitchens.dataset (module), 3
epic_kitchens.dataset.epic_dataset (module), 3
epic_kitchens.dataset.video_dataset (module), 5
epic_kitchens.gulp (module), 5
epic_kitchens.gulp.adapter (module), 5
epic_kitchens.gulp.visualisation (module), 7
epic_kitchens.labels (module), 13
epic_kitchens.meta (module), 8
epic_kitchens.preprocessing (module), 13
epic_kitchens.preprocessing.split_segments (module), 13
epic_kitchens.time (module), 14
epic_kitchens.video (module), 17
EpicDatasetAdapter (class in *epic_kitchens.gulp.adapter*), 5
EpicFlowDatasetAdapter (class in *epic_kitchens.gulp.adapter*), 6
EpicVideoDataset (class in *epic_kitchens.dataset.epic_dataset*), 3
EpicVideoFlowDataset (class in *epic_kitchens.dataset.epic_dataset*), 4
- F**
- `flow_frame_count()` (in module *epic_kitchens.time*), 14
FlowModalityIterator (class in *epic_kitchens.video*), 17
FlowVisualiser (class in *epic_kitchens.gulp.visualisation*), 7
`frame_iterator()` (*epic_kitchens.video.FlowModalityIterator* method), 17
`frame_iterator()` (*epic_kitchens.video.ModalityIterator* method), 17
`frame_iterator()` (*epic_kitchens.video.RGBModalityIterator* method), 17
- G**
- `get_datadir()` (in module *epic_kitchens.meta*), 8
`get_narration()` (in module *epic_kitchens.video*), 17
GulpVideoSegment (class in *epic_kitchens.dataset.epic_dataset*), 4
- H**
- `hstack_frames()` (in module *epic_kitchens.gulp.visualisation*), 8
- I**
- `id` (*epic_kitchens.dataset.epic_dataset.GulpVideoSegment* attribute), 4

I
 id (*epic_kitchens.dataset.video_dataset.VideoSegment attribute*), 5
 is_many_shot_action() (in module *epic_kitchens.meta*), 8
 is_many_shot_noun() (in module *epic_kitchens.meta*), 8
 is_many_shot_verb() (in module *epic_kitchens.meta*), 9
 iter_data() (*epic_kitchens.gulp.adapter.EpicDatasetAdapter method*), 6
 iter_data() (*epic_kitchens.gulp.adapter.EpicFlowDataAdapter method*), 6
 iterate_frame_dir() (in module *epic_kitchens.video*), 17
L
 label (*epic_kitchens.dataset.epic_dataset.GulpVideoSegment attribute*), 4
 label (*epic_kitchens.dataset.video_dataset.VideoSegment attribute*), 5
 load_frames() (*epic_kitchens.dataset.epic_dataset.EpicVideoDataset method*), 4
 load_frames() (*epic_kitchens.dataset.video_dataset.VideoDataset method*), 5
M
 main() (in module *epic_kitchens.preprocessing.split_segments*), 13
 many_shot_actions() (in module *epic_kitchens.meta*), 9
 many_shot_nouns() (in module *epic_kitchens.meta*), 9
 many_shot_verbs() (in module *epic_kitchens.meta*), 9
 MissingDataException, 7
 ModalityIterator (class in *epic_kitchens.video*), 17
N
 NARRATION_COL (in module *epic_kitchens.labels*), 13
 noun (*epic_kitchens.meta.Action attribute*), 8
 noun_class (*epic_kitchens.meta.ActionClass attribute*), 8
 NOUN_CLASS_COL (in module *epic_kitchens.labels*), 13
 noun_classes() (in module *epic_kitchens.meta*), 9
 NOUN_COL (in module *epic_kitchens.labels*), 13
 noun_to_class() (in module *epic_kitchens.meta*), 9
 NOUNS_CLASS_COL (in module *epic_kitchens.labels*), 13
 NOUNS_COL (in module *epic_kitchens.labels*), 13
 num_frames (*epic_kitchens.dataset.epic_dataset.GulpVideoSegment attribute*), 5
 num_frames (*epic_kitchens.dataset.video_dataset.VideoSegment attribute*), 5
P
 PARTICIPANT_ID_COL (in module *epic_kitchens.labels*), 14
R
 RGBModalityIterator (class in *epic_kitchens.video*), 17
 RgbVisualiser (class in *epic_kitchens.gulp.visualisation*), 7
S
 seconds_to_timestamp() (in module *epic_kitchens.time*), 15
 set_datadir() (in module *epic_kitchens.meta*), 9
 set_version() (in module *epic_kitchens.meta*), 9
 show() (*epic_kitchens.gulp.visualisation.Visualiser method*), 7
 split_dataset_frames() (in module *epic_kitchens.video*), 18
 split_video_frames() (in module *epic_kitchens.video*), 18
 START_F_COL (in module *epic_kitchens.labels*), 14
 START_TS_COL (in module *epic_kitchens.labels*), 14
 STOP_F_COL (in module *epic_kitchens.labels*), 14
 STOP_TS_COL (in module *epic_kitchens.labels*), 14
T
 test_timestamps() (in module *epic_kitchens.meta*), 10
 timestamp_to_frame() (in module *epic_kitchens.time*), 16
 timestamp_to_seconds() (in module *epic_kitchens.time*), 16
 training_labels() (in module *epic_kitchens.meta*), 10
 training_narrations() (in module *epic_kitchens.meta*), 11
 training_object_labels() (in module *epic_kitchens.meta*), 11
U
 UID_COL (in module *epic_kitchens.labels*), 14
V
 verb (*epic_kitchens.meta.Action attribute*), 8
 verb_class (*epic_kitchens.meta.ActionClass attribute*), 8
 VERB_CLASS_COL (in module *epic_kitchens.labels*), 14
 verb_classes() (in module *epic_kitchens.meta*), 12
 VERB_COL (in module *epic_kitchens.labels*), 14
 verb_to_class() (in module *epic_kitchens.meta*), 12

`video_descriptions()` (in module `epic_kitchens.meta`), 12

`VIDEO_ID_COL` (in module `epic_kitchens.labels`), 14

`video_info()` (in module `epic_kitchens.meta`), 13

`video_segments` (`epic_kitchens.dataset.epic_dataset.EpicVideoDataset` attribute), 4

`video_segments` (`epic_kitchens.dataset.video_dataset.VideoDataset` attribute), 5

`VideoDataset` (class in `epic_kitchens.dataset.video_dataset`), 5

`VideoSegment` (class in `epic_kitchens.dataset.video_dataset`), 5

`Visualiser` (class in `epic_kitchens.gulp.visualisation`), 7