shapedata Documentation

Release 0.2.0

Justus Schock

Jul 30, 2019

Contents:

CHAPTER 1

ShapeData

The ShapeData Package provides methods to transform images and the corrisponding landmarks in a similar way.

1.1 AbstractSingleImage

class AbstractSingleImage

Bases: object

Abstract Class to define a SingleImage-API

_crop(**args*, ***kwargs*)

Internal implementation of *AbstractSingleImage.crop()*

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

_crop_lmks(**args*, ***kwargs*)

Crops the landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

_crop_to_landmarks(**args*, ***kwargs*)

Internal implementation of *AbstractSingleImage.crop_to_landmarks()*

Parameters

- ***args** – positional arguments

- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

_normalize_rotation(**args*, ***kwargs*)
Internal implementation of *AbstractSingleImage.normalize_rotation()*

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

_save_landmarks(**args*, ***kwargs*)
Abstract internal Function to save landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

_transform_img(**args*, ***kwargs*)
Applies a given transformation to image

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

_transform_lmk(**args*, ***kwargs*)
Applies a given transformation to landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

apply_trafo(**args*, ***kwargs*)
Applies a given transformation to image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

cartesian_coordinates()
Transforms the landmarks into cartesian coordinates

Raises NotImplementedError – if not overwritten by subclass

crop(**args*, ***kwargs*)
Crops image and landmarks to given range

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

crop_to_landmarks(**args*, ***kwargs*)
Crops image and landmarks to bounding box specified by landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

classmethod from_files(**args*, ***kwargs*)
Creates a class instance from files

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError

get_landmark_bounds(**args*, ***kwargs*)
Calculates bounds of landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

homogeneous_coordinates()
Transforms the landmarks into homogeneous coordinates

Raises NotImplementedError – if not overwritten by subclass

img
Property to get the actual image pixels

Returns image pixels

Return type np.array

is_gray
Property returning whether the image is a grayscale image

Raises NotImplementedError – if not overwritten by subclass

is_homogeneous
Property returning whether the landmarks are in homogeneous coordinates

Raises NotImplementedError – if not overwritten by subclass

normalize_rotation(**args*, ***kwargs*)
Rotates image and landmarks in a way, that the vector between two given points is parallel to horizontal axis

Parameters

- ***args** – positional arguments

- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

rescale(**args*, ***kwargs*)
Rescales image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

resize(**args*, ***kwargs*)
Resizes image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

rotate(**args*, ***kwargs*)
Rotates image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

save(**args*, ***kwargs*)
Abstract Function to save image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

save_image(**args*, ***kwargs*)
Abstract Function to save image

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

save_landmarks(**args*, ***kwargs*)
Abstract Function to save landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

to_grayscale()
Converts image to grayscale

Raises NotImplementedError – if not overwritten by subclass

transform(**args*, ***kwargs*)
Applies a given transformation to image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

transform_about_centre(**args*, ***kwargs*)
Applies a given transformation to image and landmarks at image center (internally shifts image and landmarks center to origin, applies transformation and shifts back)

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

translate(**args*, ***kwargs*)
Translates image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

view(**args*, ***kwargs*)
Plots image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError – if not overwritten by subclass

1.2 BaseSingleImage

class BaseSingleImage(*img: <sphinx.ext.autodoc.importer._MockObject object at 0x7fad858c3198>*, **args*, ***kwargs*)
Bases: *shapedata.base_data_processing.AbstractSingleImage*

Holds Single Image

_crop(*min_y*, *min_x*, *max_y*, *max_x*)
Implements actual cropping inplace

Parameters

- **min_y** (*int*) – minimum y value

- **min_x** (*int*) – minimum x value
- **max_y** (*int*) – maximum y value
- **max_x** (*int*) – maximum x value

Raises NotImplementedError – if not overwritten by subclass

static _crop_lmks(*lmks*, *min_y*, *min_x*, *max_y*, *max_x*)
Crops landmarks to given values

Parameters

- **lmks** (*np.ndarray*) – landmarks to crop
- **min_y** (*int*) – minimum y value
- **min_x** (*int*) – minimum x value
- **max_y** (*int*) – maximum y value
- **max_x** (*int*) – maximum x value

Returns cropped landmarks

Return type np.ndarray

_crop_to_landmarks(*proportion=0.0*, ***kwargs*)
Crop to landmarks inplace

Parameters

- **proportion** (*float*) – boundary proportion of cropping
- ****kwargs** – additional keyword arguments

Raises NotImplementedError – if not overwritten by subclass

_normalize_rotation(*lmks*, *index_left*, *index_right*, ***kwargs*)
normalizes rotation based on two keypoints

Parameters

- **lmks** (*np.ndarray*) – landmarks for rotation normalization
- **index_left** (*int*) – index for left point
- **index_right** (*int*) – index for right point
- ****kwargs** – additional keyword arguments (passed to `skimage.transform.warp()`)

Returns transformed Image

Return type *BaseSingleImage*

_save_landmarks(*filepath*, *lmk_type*, ***kwargs*)
Saves landmarks to file

Parameters

- **filepath** (*str*) – path to file the landmarks should be saved to
- **lmk_type** (*str*) – specifies the type of landmark file
- ****kwargs** – additional keyword arguments passed to save function

Raises NotImplementedError – If not overwritten by subclass

_transform_img(*transformation: <sphinx.ext.autodoc.importer._MockObject object at 0x7fad858bee80>*, ***kwargs*)
Apply transformation inplace to image

Parameters

- **transformation** (skimage.transform.AffineTransform) – transformation to apply
- ****kwargs** – additional keyword arguments

Returns Transformed Image with original Landmarks

Return type *BaseSingleImage*

_transform_lmk(*transformation: <sphinx.ext.autodoc.importer._MockObject object at 0x7fad858bee80>*)
Apply transformation inplace to landmarks

Parameters **transformation** (skimage.transform.AffineTransform) – transformation to apply

Returns Image with Transformed Landmarks

Return type *BaseSingleImage*

apply_trafo(*transformation: <sphinx.ext.autodoc.importer._MockObject object at 0x7fad858bee80>*, ***kwargs*)
Apply transformation inplace to image and landmarks

Parameters

- **transformation** (skimage.transform.AffineTransform) – transformation to apply
- ****kwargs** – additional keyword arguments

Returns Transformed Image and Landmarks

Return type *BaseSingleImage*

cartesian_coordinates()
Transforms landmark coordinates inplace to cartesian coordinates

Returns Image with Landmarks in cartesian Coordinates

Return type *BaseSingleImage*

crop(*min_y*, *min_x*, *max_y*, *max_x*)
Crops Image by specified values

Parameters

- **min_y** (*int*) – minimum y value
- **min_x** (*int*) – minimum x value
- **max_y** (*int*) – maximum y value
- **max_x** (*int*) – maximum x value

Returns cropped image

Return type *BaseSingleImage*

crop_to_landmarks(*proportion=0.0*, ***kwargs*)
Crop image to landmarks

Parameters

- **proportion** (*float*) – image proportion to add to size of bounding box
- ****kwargs** – additional keyword arguments

Returns cropped image

Return type *BaseSingleImage*

classmethod from_files(*file*, *extension=None*, ***kwargs*)
Creates a class instance from files

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises NotImplementedError

classmethod from_ljson_files(*img_file*, ***kwargs*)
Create class from image or landmark file

Parameters **file** (*str*) – path to image or landmarkfile

Returns

Return type *BaseSingleImage*

classmethod from_npy_files(*file*, ***kwargs*)
Create class from image or landmark file

Parameters **file** (*str*) – path to image or landmarkfile

Returns

Return type *BaseSingleImage*

classmethod from_pts_files(*file*, ***kwargs*)
Create class from image or landmark file :param file: path to image or landmarkfile :type file: string

Returns

Return type *BaseSingleImage*

static get_landmark_bounds(*lmks*)
Function to calculate the landmark bounds

Parameters **lmks** (*np.ndarray*) – landmarks

Returns

- **int** (*min_y*)
- **int** (*min_x*)
- **int** (*max_y*)
- **int** (*max_x*)

homogeneous_coordinates()
Transforms landmark coordinates inplace to homogeneous coordinates

Returns Image with Landmarks in Homogeneous Coordinates

Return type *BaseSingleImage*

img
Property to get the actual image pixels

Returns image pixels

Return type np.array

is_gray
Property returning whether the image is a grayscale image

Raises `NotImplementedError` – if not overwritten by subclass

is_homogeneous
Property returning whether the landmarks are in homogeneous coordinates

Raises `NotImplementedError` – if not overwritten by subclass

normalize_rotation(**args*, ***kwargs*)
Rotates image and landmarks in a way, that the vector between two given points is parallel to horizontal axis

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises `NotImplementedError` – if not overwritten by subclass

rescale(*scale*, ***kwargs*)
Scale Image and landmarks

Parameters

- **scale** – scale parameter
- ****kwargs** – additional keyword arguments (passed to `skimage.transform.warp()`)

Returns transformed Image

Return type *BaseSingleImage*

resize(*target_shape*, ***kwargs*)
resize image and scale landmarks :param target_shape: target shape for resizing :type target_shape: tuple or list :param **kwargs: additional keyword arguments (passed to

`skimage.transform.warp()`)

Returns transformed Image

Return type *BaseSingleImage*

rotate(*angle*, *degree=True*, ***kwargs*)
Rotates the image and landmarks by given angle

Parameters

- **angle** (*float or int*) – rotation angle
- **degree** (*bool*) – whether the angle is given in degree or radiant
- ****kwargs** – additional keyword arguments (passed to `skimage.transform.warp()`)

Returns transformed Image

Return type *BaseSingleImage*

save(*directory*, *filename*, *lmk_type='LJSON'*, ***kwargs*)
Saves Image and optionally landmarks to files

Parameters

- **directory** (*str*) – string containing the directory to save
- **filename** (*str*) – string containing the filename (without the extension)
- **lmk_type** (*str or None*) – if None: no landmarks will be saved if str: specifies type of landmark file
- ****kwargs** – additional keyword arguments passed to save function for landmarks

save_image(*filepath*)
Saves Image to file

Parameters **filepath** (*str*) – file to save the image to

save_landmarks(*filepath*, *lmk_type='LJSON'*, ***kwargs*)
Saves landmarks to file

Parameters

- **filepath** (*str*) – path to file the landmarks should be saved to
- **lmk_type** (*str*) – specifies the type of landmark file
- ****kwargs** – additional keyword arguments passed to save function

to_grayscale()
Convert Image to grayscale

Returns Grayscale Image

Return type *BaseSingleImage*

transform(*transform=None*, *rotation=None*, *scale=None*, *translation=None*, *shear=None*, *trafo_matrix=None*, *return_matrix=False*, ***kwargs*)
transform image and landmarks by parameters or transformation matrix See skimage.transform.AffineTransform for a detailed parameter explanation

Parameters

- **transform** (skimage.transform.AffineTransform) – if transform is specified it overwrites all other arguments
- **rotation** (*float or None*) – rotation angle in radiant
- **scale** (*float or None*) – scale value
- **translation** – translation params
- **shear** – shear params
- **trafo_matrix** – transformation matrix
- **return_matrix** (*bool*) – whether to return the transformation matrix along the transformed object
- ****kwargs** – additional keyword arguments

Returns

- *BaseSingleImage* – transformed Image
- *[optional] np.ndarray* – transformation matrix

transform_about_centre(*transform=None*, *rotation=None*, *scale=None*, *translation=None*, *shear=None*, *trafo_matrix=None*, *return_matrix=False*, ***kwargs*)

Perform transformations about the image center. (internally shifting image to origin, perform transformation and shift it back)

Parameters

- **transform** (skimage.transform.AffineTransform) – if transform is specified it overwrites all other arguments
- **rotation** (*float*) – rotation angle in radiant
- **scale** (*float*) – scale value
- **translation** – translation params
- **shear** – shear params
- **trafo_matrix** – transformation matrix
- **return_matrix** (*bool*) – whether to return the transformation matrix along the transformed object
- ****kwargs** – additional keyword arguments

Returns

- *BaseSingleImage* – transformed Image
- *[optional] np.ndarray* – transformation matrix

translate(*translation*, *relative=False*, ***kwargs*)

translates image and landmarks

translation : translation parameters

relative [bool] whether translation parameters are relative to image size

****kwargs :**

additional keyword arguments (passed to `skimage.transform.warp()`)

BaseSingleImage transformed Image

view(**args*, ***kwargs*)

Plots image and landmarks

Parameters

- ***args** – positional arguments
- ****kwargs** – keyword arguments

Raises `NotImplementedError` – if not overwritten by subclass

1.3 Single Shape

The `single_shape` subpackage implements dataprocessing and image-landmark-transformations for 2D images. Furthermore it provides a `delira` compatible dataset.

1.3.1 ShapeDataset

class ShapeDataset(*data_path*, *img_size*, *crop=None*, *extension=None*, *rotate=None*, *cached=False*, *random_offset=False*, *random_scale=False*, *point_indices=None*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Dataset to load image and corresponding shape

_make_dataset(*path*)

1.3.2 default_loader

default_loader(*data: str*, *img_size: tuple*, *crop=None*, *extension=None*, *rotate=None*, *cached=False*, *random_offset=False*, *random_scale=False*, *point_indices=None*)

Helper Function to load single sample

Parameters

- **data** (str or `SingleImage2D`) – image file to load
- **img_size** (*tuple*) – image size for resizing
- **crop** (*None or float*) – if None: nor cropping will be applied if float: specifies boundary proportion for cropping
- **extension** (*str or None:*) – specifiying the extension
- **rotate** (*int or None*) – specifies to image rotation (in degrees)
- **cached** (*bool*) – whether or not the data is already cached
- **random_offset** (*bool or float*) – if bool: must be False -> No Random Shift is applied if float: specifies the maximal number of pixels to shift
- **random_scale** (*bool or float*) – if bool: must be False -> No random scaling is applied if float: specifies the maximum amount of scaling
- **point_indices** (*None or Iterable*) – if None: All landmarks are returned if Iterable: only landmarks corresponding to indices are returned

Returns

- *np.ndarray* – image
- *np.ndarray* – landmarks

1.3.3 preprocessing

preprocessing(*img: shapedata.single_shape.data_processing.SingleImage2D*, *img_size: tuple*, *crop=None*, *rotate=None*, *random_offset=False*, *random_scale=False*, *point_indices=None*)

Helper Function to preprocess a single sample

Parameters

- **img** (`SingleImage2D`) – image file to preprocess
- **img_size** (*tuple*) – image size for resizing
- **crop** (*None or float*) – if None: nor cropping will be applied if float: specifies boundary proportion for cropping
- **extension** (*str or None:*) – specifiying the extension

- **rotate** (*int or None*) – specifies to image rotation (in degrees)
- **random_offset** (*bool or float*) – if bool: must be False -> No Random Shift is applied if float: specifies the maximal number of pixels to shift
- **random_scale** (*bool or float*) – if bool: must be False -> No random scaling is applied if float: specifies the maximum amount of scaling
- **point_indices** (*None or Iterable*) – if None: All landmarks are returned if Iterable: only landmarks corresponding to indices are returned

Returns

- *np.ndarray* – image
- *np.ndarray* – landmarks

1.3.4 DataProcessing

class DataProcessing(*samples*, *dim=2*, ***kwargs*)

Bases: object

Process multiple SingleImages

See also:

SingleImage2D

static _get_files(*directory*, *extensions*)

return files with extensions

Parameters

- **directory** (*str*) – directory containing the files
- **extensions** (*list*) – list of strings specifying valid extensions

Returns valid files

Return type list

classmethod from_dir(*data_dir*, *verbose=True*, *n_jobs=None*, *n_dim=2*)

create class instance from directory

Parameters

- **data_dir** (*str*) – directory where shapedata is stored
- **verbose** (*bool*) – whether or not to print current progress
- **n_jobs** (*int*) – number of jobs for loading data (default: None -> all available CPUs are used)
- **n_dim** (*int*) – Integer indicating the dimensionality of the image (default: 2)

Returns class instance

Return type *DataProcessing*

images

get list of samples' pixels

Returns pixels

Return type list

landmarks
get list of samples' landmarks

Returns landmarks

Return type list

lmk_pca(*scale: bool*, *center: bool*, *pt_indices=[]*, **args*, ***kwargs*)
perform PCA on samples' landmarks

Parameters

- **scale** (*bool*) – whether or not to scale the principal components with the corresponding eigen value
- **center** (*bool*) – whether or not to substract mean before pca
- **pt_indices** (*int*) – indices to include into PCA (if empty: include all points)
- **args** (*list*) – additional positional arguments (passed to pca)
- ****kwargs** – additional keyword arguments (passed to pca)

Returns eigen_shapes

Return type np.array

resize(*img_size*)
resize all samples

Parameters **img_size** (*tuple*) – new image size

1.3.5 SingleImage2D

class SingleImage2D(*img*, *lmk=None*, ***kwargs*)
Bases: *shapedata.base_data_processing.BaseSingleImage*

Holds Single Image

_crop(*min_y*, *min_x*, *max_y*, *max_x*)
Implements actual cropping inplace

Parameters

- **min_y** (*int*) – minimum y value
- **min_x** (*int*) – minimum x value
- **max_y** (*int*) – maximum y value
- **max_x** (*int*) – maximum x value

Returns cropped image

Return type *SingleImage2D*

static _crop_lmks(*lmks*, *min_y*, *min_x*, *max_y*, *max_x*)
Crops landmarks to given values

Parameters

- **lmks** (*np.ndarray*) – landmarks to crop
- **min_y** (*int*) – minimum y value
- **min_x** (*int*) – minimum x value

- **max_y** (*int*) – maximum y value
- **max_x** (*int*) – maximum x value

Returns cropped landmarks

Return type np.ndarray

_crop_to_landmarks (*proportion=0.0*, ***kwargs*)
Crop to landmarks

Parameters

- **proportion** (*float*) – Cropping Proportion
- ****kwargs** – additional keyword arguments (ignored here)

Returns Cropped Image

Return type *SingleImage2D*

_normalize_rotation (*lmks*, *index_left*, *index_right*, ***kwargs*)
normalizes rotation based on two keypoints

Parameters

- **lmks** (*np.ndarray*) – landmarks for rotation normalization
- **index_left** (*int*) – index for left point
- **index_right** (*int*) – index for right point
- ****kwargs** – additional keyword arguments (passed to `skimage.transform.warp()`)

Returns transformed Image

Return type `BaseSingleImage`

_save_landmarks (*filepath*, *lmk_type*, ***kwargs*)
Saves landmarks to file

Parameters

- **filepath** (*str*) – path to file the landmarks should be saved to
- **lmk_type** (*str*) – specifies the type of landmark file
- ****kwargs** – additional keyword arguments passed to save function

Raises `ValueError` – no valid landmarktype is given

_transform_img (*transformation: <sphinx.ext.autodoc.importer._MockObject object at 0x7fad858bee80>*, ***kwargs*)
Apply transformation inplace to image

Parameters

- **transformation** (`skimage.transform.AffineTransform`) – transformation to apply
- ****kwargs** – additional keyword arguments

Returns Transformed Image with original Landmarks

Return type `BaseSingleImage`

_transform_lmk(*transformation: <sphinx.ext.autodoc.importer._MockObject object at 0x7fad858ad3c8>*)
Apply transformation inplace to landmarks

Parameters **transformation** (skimage.transform.AffineTransform) – transformation to apply

Returns Image with Transformed Landmarks

Return type BaseSingleImage

apply_trafo(*transformation: <sphinx.ext.autodoc.importer._MockObject object at 0x7fad858bee80>*, ***kwargs*)
Apply transformation inplace to image and landmarks

Parameters

- **transformation** (skimage.transform.AffineTransform) – transformation to apply
- ****kwargs** – additional keyword arguments

Returns Transformed Image and Landmarks

Return type BaseSingleImage

cartesian_coordinates()
Transforms landmark coordinates inplace to cartesian coordinates

Returns **class** – Image with Landmarks in cartesian Coordinates

Return type *SingleImage2D*

crop(*min_y*, *min_x*, *max_y*, *max_x*)
Crops Image by specified values

Parameters

- **min_y** (*int*) – minimum y value
- **min_x** (*int*) – minimum x value
- **max_y** (*int*) – maximum y value
- **max_x** (*int*) – maximum x value

Returns cropped image

Return type BaseSingleImage

crop_to_landmarks(*proportion=0.0*, ***kwargs*)
Crop image to landmarks

Parameters

- **proportion** (*float*) – image proportion to add to size of bounding box
- ****kwargs** – additional keyword arguments

Returns cropped image

Return type BaseSingleImage

classmethod from_files(*file*, *extension=None*, ***kwargs*)
Creates a class instance from files

Parameters

- ***args** – positional arguments

- ****kwargs** – keyword arguments

Raises NotImplementedError

classmethod from_ljson_files(*img_file*, ***kwargs*)
Creates class from menpo pts landmarks and image

Parameters

- **img_file** (*str*) – image file to load
- ****kwargs** – additional keyword arguments

Returns class instance

Return type *SingleImage2D*

classmethod from_npy_files(*file*, ***kwargs*)
Create class from image file :param file: path to image file :type file: str :param **kwargs: additional keyword arguments

Returns class instance

Return type *SingleImage2D*

classmethod from_pts_files(*img_file*, ***kwargs*)
Creates class from menpo ljson landmarks and image

Parameters

- **img_file** (*str*) – image file to load
- ****kwargs** – additional keyword arguments

Returns class instance

Return type *SingleImage2D*

static get_landmark_bounds(*lmks*)
Function to calculate the landmark bounds

Parameters **lmks** (*np.ndarray*) – landmarks

Returns

- **int** (*min_y*)
- **int** (*min_x*)
- **int** (*max_y*)
- **int** (*max_x*)

homogeneous_coordinates()
Transforms landmark coordinates inplace to homogeneous coordinates

Returns Image with Landmarks in Homogeneous Coordinates

Return type *SingleImage2D*

img
Property to get the actual image pixels

Returns image pixels

Return type np.array

is_gray
Property returning whether the image is a grayscale image

Raises NotImplementedError – if not overwritten by subclass

is_homogeneous
Property returning whether the landmarks are in homogeneous coordinates

Raises NotImplementedError – if not overwritten by subclass

normalize_rotation(*index_left*, *index_right*, ***kwargs*)
normalizes rotation based on two keypoints

index_left [int] landmark-index of the left point

index_right [int] landmark-index of the right point

****kwargs:** additional keyword arguments (passed to warp())

Returns normalized image

Return type *SingleImage2D*

rescale(*scale*, ***kwargs*)
Scale Image and landmarks

Parameters

- **scale** – scale parameter
- ****kwargs** – additional keyword arguments (passed to skimage.transform.warp())

Returns transformed Image

Return type BaseSingleImage

resize(*target_shape*, ***kwargs*)
resize image and scale landmarks :param target_shape: target shape for resizing :type target_shape: tuple or list :param **kwargs: additional keyword arguments (passed to

skimage.transform.warp())

Returns transformed Image

Return type BaseSingleImage

rotate(*angle*, *degree=True*, ***kwargs*)
Rotates the image and landmarks by given angle

Parameters

- **angle** (*float or int*) – rotation angle
- **degree** (*bool*) – whether the angle is given in degree or radiant
- ****kwargs** – additional keyword arguments (passed to skimage.transform.warp())

Returns transformed Image

Return type BaseSingleImage

save(*directory*, *filename*, *lmk_type='LJSON'*, ***kwargs*)
Saves Image and optionally landmarks to files

Parameters

- **directory** (*str*) – string containing the directory to save

- **filename** (*str*) – string containing the filename (without the extension)
- **lmk_type** (*str or None*) – if None: no landmarks will be saved if str: specifies type of landmark file
- ****kwargs** – additional keyword arguments passed to save function for landmarks

save_image(*filepath*)
Saves Image to file

Parameters **filepath** (*str*) – file to save the image to

save_landmarks(*filepath*, *lmk_type='LJSON'*, ***kwargs*)
Saves landmarks to file

Parameters

- **filepath** (*str*) – path to file the landmarks should be saved to
- **lmk_type** (*str*) – specifies the type of landmark file
- ****kwargs** – additional keyword arguments passed to save function

to_grayscale()
Convert Image to grayscale

Returns Grayscale Image

Return type BaseSingleImage

transform(*transform=None*, *rotation=None*, *scale=None*, *translation=None*, *shear=None*, *trafo_matrix=None*, *return_matrix=False*, ***kwargs*)
transform image and landmarks by parameters or transformation matrix See skimage.transform.AffineTransform for a detailed parameter explanation

Parameters

- **transform** (skimage.transform.AffineTransform) – if transform is specified it overwrites all other arguments
- **rotation** (*float or None*) – rotation angle in radiant
- **scale** (*float or None*) – scale value
- **translation** – translation params
- **shear** – shear params
- **trafo_matrix** – transformation matrix
- **return_matrix** (*bool*) – whether to return the transformation matrix along the transformed object
- ****kwargs** – additional keyword arguments

Returns

- BaseSingleImage – transformed Image
- *[optional] np.ndarray* – transformation matrix

transform_about_centre(*transform=None*, *rotation=None*, *scale=None*, *translation=None*, *shear=None*, *trafo_matrix=None*, *return_matrix=False*, ***kwargs*)
Perform transformations about the image center. (internally shifting image to origin, perform transformation and shift it back)

Parameters

- **transform** (skimage.transform.AffineTransform) – if transform is specified it overwrites all other arguments
- **rotation** (*float*) – rotation angle in radiant
- **scale** (*float*) – scale value
- **translation** – translation params
- **shear** – shear params
- **trafo_matrix** – transformation matrix
- **return_matrix** (*bool*) – whether to return the transformation matrix along the transformed object
- ****kwargs** – additional keyword arguments

Returns

- BaseSingleImage – transformed Image
- *[optional] np.ndarray* – transformation matrix

translate(*translation*, *relative=False*, ***kwargs*)

translates image and landmarks

translation : translation parameters

relative [bool] whether translation parameters are relative to image size

****kwargs :**

additional keyword arguments (passed to skimage.transform.warp())

BaseSingleImage transformed Image

view(*view_landmarks=False*, *create_fig=False*, ***kwargs*)

Shows image (and optional the landmarks)

Parameters

- **view_landmarks** (*bool*) – whether or not to show the landmarks
- ****kwargs** – additional keyword arguments (are passed to imshow)

Returns figure with plot

Return type Figure

1.4 ljson_importer

ljson_importer(*filepath*)

Importer for the Menpo JSON format. This is an n-dimensional landmark type for both images and meshes that encodes semantic labels in the format. Landmark set label: JSON Landmark labels: decided by file

Parameters **filepath** (*str*) – Absolute filepath of the file.

Returns loaded landmarks

Return type np.ndarray

1.5 pts_importer

pts_importer(*filepath*, *image_origin=True*, *z=False*, ***kwargs*)

Importer for the PTS file format. Assumes version 1 of the format. Implementations of this class should override the `_build_points()` which determines the ordering of axes. For example, for images, the *x* and *y* axes are flipped such that the first axis is *y* (height in the image domain). Note that PTS has a very loose format definition. Here we make the assumption (as is common) that PTS landmarks are 1-based. That is, landmarks on a 480x480 image are in the range [1-480]. As Menpo is consistently 0-based, we *subtract 1* off each landmark value automatically. If you want to use PTS landmarks that are 0-based, you will have to manually add one back on to landmarks post importing. Landmark set label: PTS

Parameters

- **filepath** (*str*) – Absolute filepath of the file.
- **image_origin** (*bool*, optional) – If `True`, assume that the landmarks exist within an image and thus the origin is the image origin.
- ****kwargs** (*dict*, optional) – Any other keyword arguments.

Returns imported points

Return type np.ndarray

1.6 ljson_exporter

ljson_exporter(*lmk_points*, *filepath*, ***kwargs*)

Given a file handle to write in to (which should act like a Python *file* object), write out the landmark data. No value is returned. Writes out the LJSON format which is a verbose format that closely resembles the labelled point graph format. It describes semantic labels and connectivity between labels. The first axis of the format represents the image y-axis and is consistent with ordering within Menpo.

Parameters

- **lmk_points** (*np.ndarray*) – The shape to write out.
- **filepath** (*str*) – The file to write in to

1.7 pts_exporter

pts_exporter(*pts*, *file_handle*, ***kwargs*)

Given a file handle to write in to (which should act like a Python *file* object), write out the landmark data. No value is returned. Writes out the PTS format which is a very simple format that does not contain any semantic labels. We assume that the PTS format has been created using Matlab and so use 1-based indexing and put the image x-axis as the first coordinate (which is the second axis within Menpo). Note that the PTS file format is only powerful enough to represent a basic pointcloud. Any further specialization is lost.

Parameters

- **pts** (*np.ndarray*) – points to save
- **file_handle** (*file*-like object) – The file to write in to

1.8 IMG_EXTENSIONS_2D

IMG_EXTENSIONS_2D
Contains the typical 2D file-extensions for images.

1.9 LMK_EXTENSIONS

LMK_EXTENSIONS
Contains all supported file-extensions for landmark files.

1.10 is_image_file

is_image_file(*filename*)
Helper Function to determine whether a file is an image file or not

Parameters **filename** (*str*) – the filename containing a possible image

Returns True if file is image file, False otherwise

Return type bool

1.11 is_landmark_file

is_landmark_file(*filename*)
Helper Function to determine whether a file is a landmark file or not

Parameters **filename** (*str*) – the filename containing possible landmarks

Returns True if file is landmark file, False otherwise

Return type bool

1.12 make_dataset

make_dataset(*dir*)
Helper Function to make a dataset containing all images in a certain directory

Parameters **dir**(*the directory containing the dataset*)–

Returns list of image paths

Return type list

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

C

D

F

G

H

I

L

M

N

P

R

S

T

V