
IdaLib

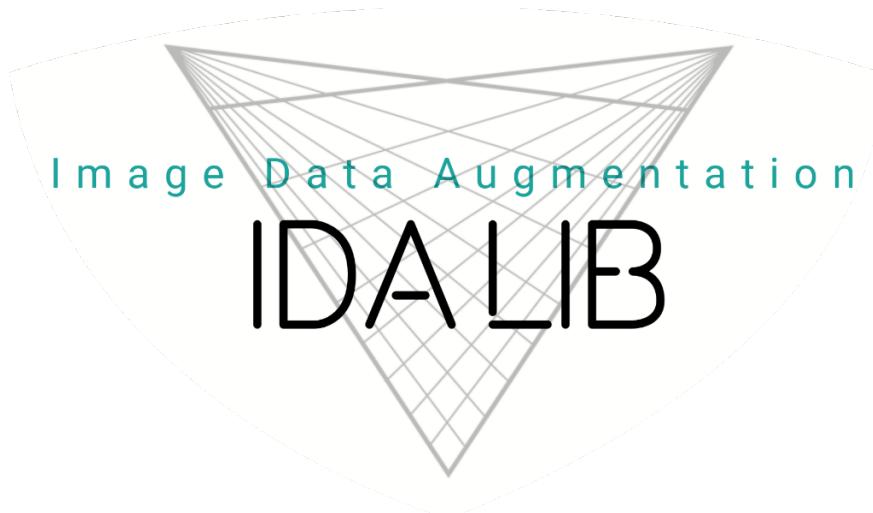
Release 0.0.1

Raquel Vilas

Jun 15, 2020

CONTENTS

1 Features	3
2 Instalation	5
3 Contents	7
3.1 Overview	7
3.1.1 Functionalities	7
3.1.2 Transformations	7
3.1.3 Visualization tool	12
3.2 API	13
3.2.1 API packages	13
3.2.2 Submodules	36
3.2.3 ida_lib.visualization module	36
3.3 Examples	36
3.3.1 Pipeline usage example	36
4 Indices and tables	43
Python Module Index	45
Index	47



IdaLib (Image Data Augmentation Library) is a library dedicated to the task of Image Data Augmentation in a fast, simple, efficient and flexible way

**CHAPTER
ONE**

FEATURES

- Multiple fast augmentations based on Kornia and OpenCV libraries
- Flexible
- Complete tool, includes support for tasks associated with Pytorch-based neural networks
- Supports multiple types of combined data (images, heat maps, segmentation maps, masks and keypoints)

**CHAPTER
TWO**

INSTALATION

You can use pip to install Ida-Lib

```
pip install ida-lib
```

If you want to get the latest version of the code before it is released on PyPI you can install the library from GitHub:

```
pip install -U git+https://github.com/raquelvilas18/ida_lib
```


CONTENTS

3.1 Overview

3.1.1 Functionalities

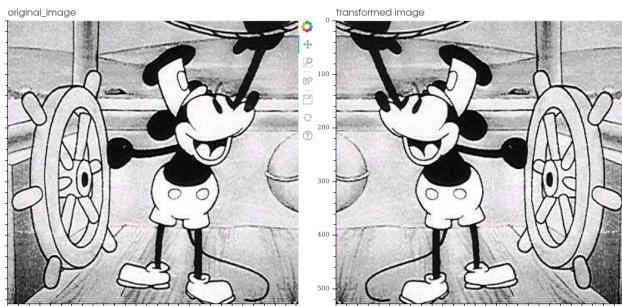
The functionalities on which IdaLib focuses are:

- Offering a **wide variety of image transformations**.
- Allow the **joint transformation** of:
 - image
 - heatmap
 - mask
 - segmaps
 - keypoints
 - any combination of them
- Offer an **interactive visualization tool** that facilitates the debugging of programs.
- Offer an **efficient operation composition pipeline** (parameterizable at probability and attribute level).
- Offer a **Dataloader** object (as a generator) that integrates the pipeline and allows the supply of any neural network.
- Offer a tool to perform **Image Data Augmentation directly to disk** in order to use the increased dataset directly in future situations

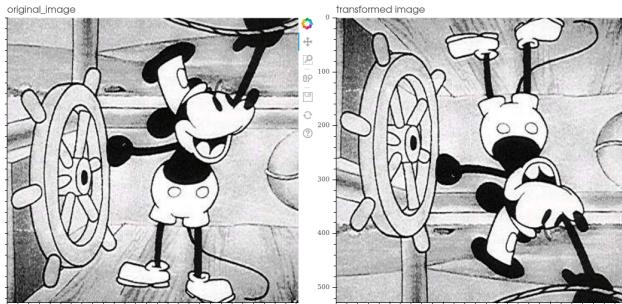
3.1.2 Transformations

The transformations included in the library are:

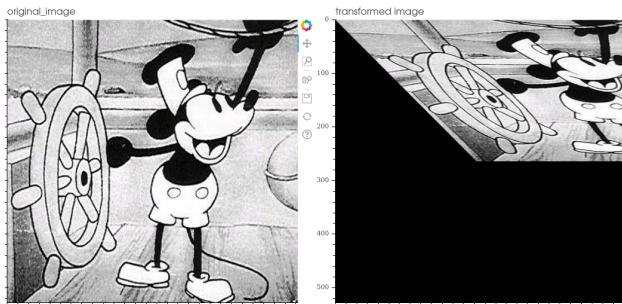
1. **hflip**: horizontal flipping the image



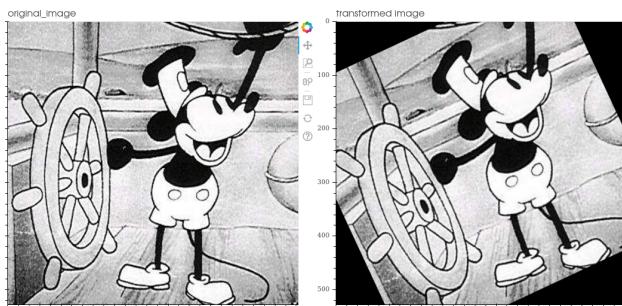
2. **vflip**: vertical flipping the image



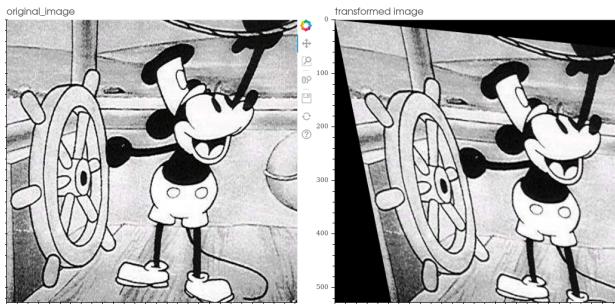
3. **Affine**: carry out the transformation expressed in the operation matrix



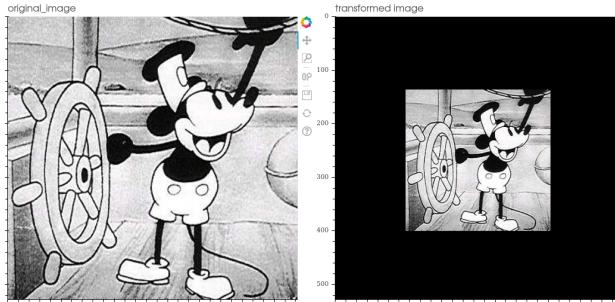
4. **Rotate**: rotate the image by the indicated degrees counterclockwise



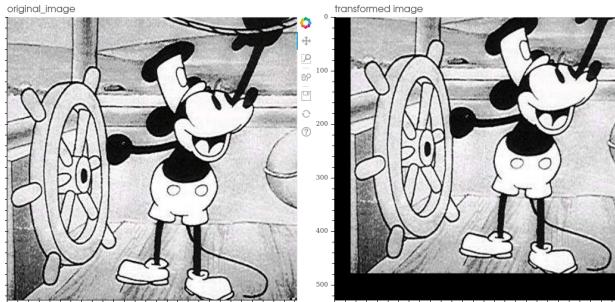
5. **Shear**: linear map that displaces each point in fixed direction, by an amount proportional to its signed distance from the line that is parallel to that direction and goes through the origin



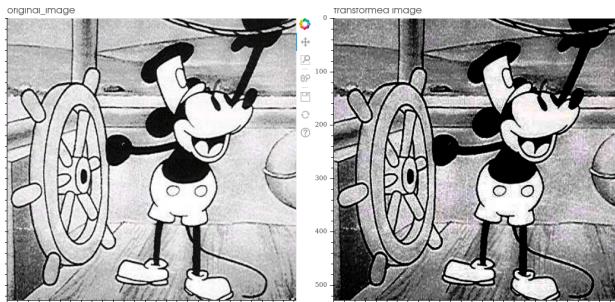
6. **Scale:** scale the image by making it smaller or larger (crop equivalent)



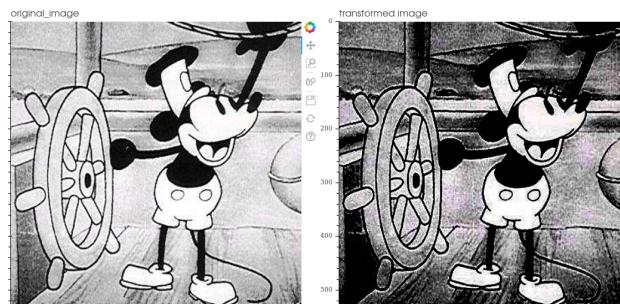
7. **Translate:** moves the image pixels to the positions indicated on each axis



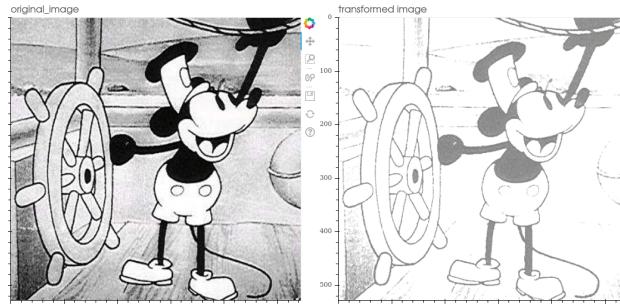
8. **Change gamma:** adjust image's gamma (luminance correction) .



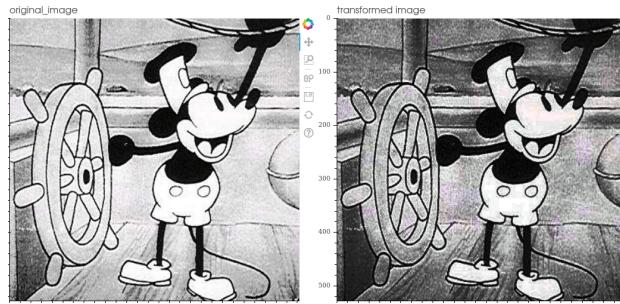
9. **Change contrast:** change the image contrast.



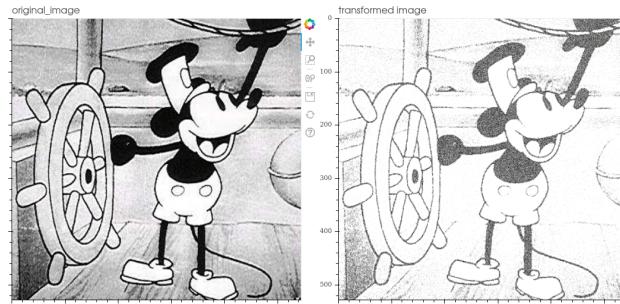
10. **Change brightness:** change the image brightness



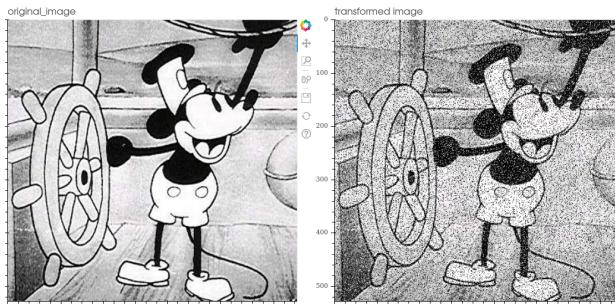
11. **Equalize histogram:** equalize the image histogram



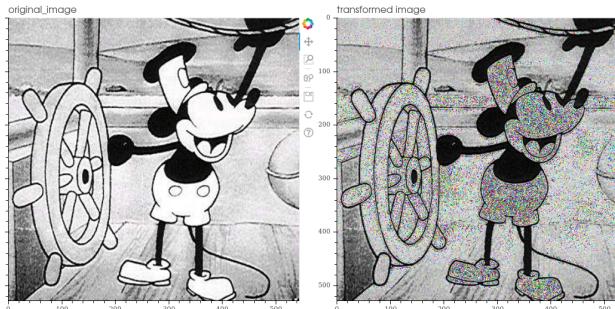
12. **Inject gaussian noise:** gaussian noise is a statistical noise having a probability density function (PDF) equal to that of the normal distribution



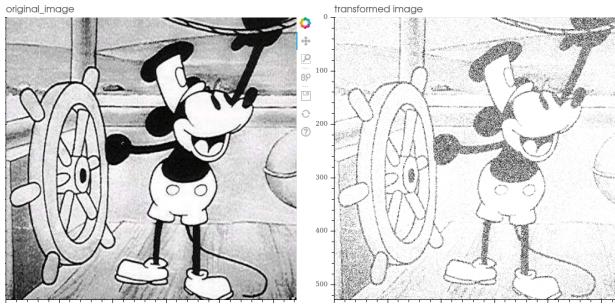
13. **Inject salt and pepper noise:** salt-and-pepper noise is a statistical noise composed of white (salt) and black (pepper) pixels



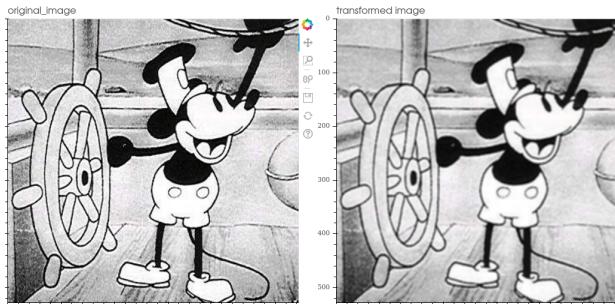
14. Inject spekple noise: Speckle is a granular interference that inherently exists in and degrades the quality of the active radar, synthetic aperture radar (SAR), medical ultrasound and optical coherence tomography images. It is applied by adding the image multiplied by the noise matrix -> $\text{img} + \text{img} * \text{uniform_noise}$



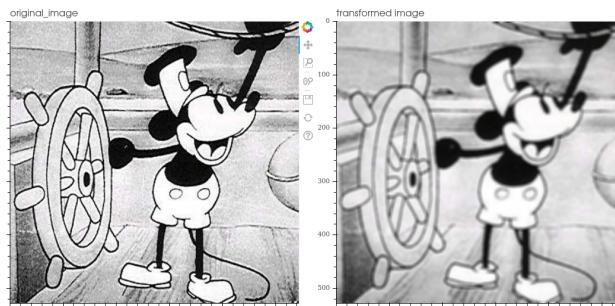
15. Inject poisson noise: It is applied by adding Poisson-distributed noise



16. Blur: blur image.



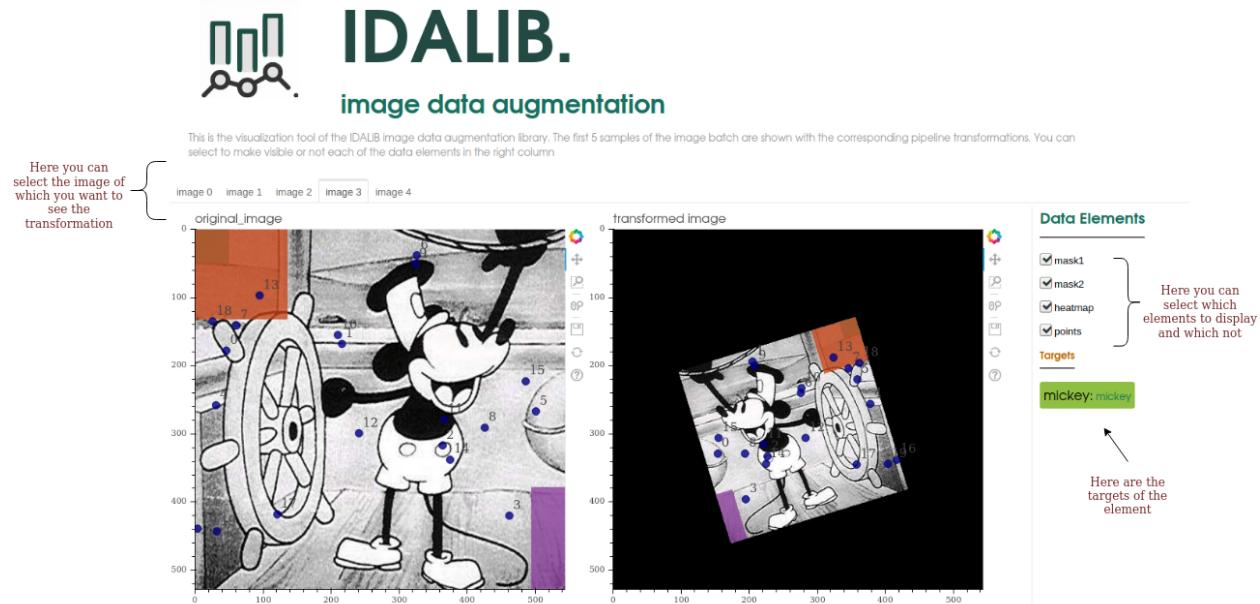
17. Gaussian blur: blurring an image by a Gaussian function.



3.1.3 Visualization tool

Ida Lib includes a tool to visualize the transformations to facilitate code debugging. It is an interactive tool developed with the bokeh framework and allows the selection of the data to be displayed in the image.

- The color code is used to differentiate each element and identify it in all the images.
- The dots are numbered in order to see their order
- Allows to compare different transformations obtained by the pipeline
- It also includes the targets in the visualization in order to have a complete view of the elements



Note: To test the visualization tool you can try this example: https://github.com/raquelvilas18/ida_lib/blob/master/examples/pipeline_usage.py

Warning: For the visualization tool a bokeh server is deployed; therefore it is only possible to have one open execution. It is important to close previous runs in order to open new windows

3.2 API

3.2.1 API packages

ida_lib.core package

Submodules

ida_lib.core.pipeline module

```
class ida_lib.core.pipeline.Pipeline(pipeline_operations: list, resize: tuple = None, interpolation: str = 'bilinear', padding_mode: str = 'zeros', output_format: str = 'dict', output_type: Optional[torch.dtype] = None)
```

Bases: object

The pipeline object represents the pipeline with data transformation operations (pictures, points). When executed, on a batch of images, it applies the necessary transformations (being different on each image based on the probabilities of each operation included).

Considerations: 1) The images must be of the same size, or the RESIZE operation must be included so that the transformations can be applied correctly 2) To run the pipeline, it accepts any type of input metadata named in the input dict. In particular it gives special treatment

to data named as:

- Mask: it is affected by geometric transformations and its output is discrete to values of 0-1
- Segmap: generalization of mask. Every value is discrete
- Image: affected by geometric and color transformations
- Keypoints: geometric transformations are applied to them as coordinates.
- Others: any other metadata will not be transformed (example: ‘tag’, ‘target’...)

Example:

```
pip = pipeline(resize = (25, 25), pipeline_operations=( translate_pipeline(probability=0.5,
    translation=(3, 0.05)), vflip_pipeline(probability=0.5), hflip_pipeline(probability=0.5),
    contrast_pipeline(probability=0.5, contrast_factor=1), random_brightness_pipeline(probability=0.2,
    brightness_range=(1.5, 1.6)), random_scale_pipeline(probability=1, scale_range=(0.5, 1.5), center_deviation=20),
    random_rotate_pipeline(probability=0.2, degrees_range=(-50, 50), center_deviation=20))
))
```

get_data_types () → tuple

Returns the tuple of data types identified on the input data

ida_lib.core.pipeline_functional module

```
ida_lib.core.pipeline_functional.get_compose_matrix(operations: list, data_info:  
                                                 Optional[dict] = None) →  
                                                 None_ VariableFunctions.tensor
```

Returns the transformation matrix composed by the multiplication in order of the input operations (according to their probability) If data_info is not None, go through the operations by entering the necessary information about the images (image center, shape..)

Parameters

- **operations** – list of pipeline operations
- **data_info** – dict with data info to configure operations parameters

Returns torch tensor of the transform matrix

```
ida_lib.core.pipeline_functional.get_compose_function(operations: list) →  
                                                 numpy.ndarray
```

returns the LUT table with the correspondence of each possible value according to the color operations to be implemented (according to their probability)

Parameters **operations** – list of pipeline operations

Returns compose function

```
ida_lib.core.pipeline_functional.preprocess_data(data: Union[list, dict], batch_info:  
                                                 Union[list, dict] = None, interpolation:  
                                                 str = None, resize: Optional[tuple] = None) → list
```

Combines the 2d information in a tensor and the points in a homogeneous coordinate matrix that allows applying the geometric operations in a single joint operation on the data and another on the points.

- Loads the data as tensor in GPU to prepare them as input to a neural network
- Analyze the data info required for the transformations (shape, bpp...)
- Resize the 2d data and keypoints to the new shape

Parameters

- **resize** – if it is wanted to resize the data, indicate the new size
- **data** – list of elements to be transformed through the pipe
- **batch_info** – dict with the required data info
- **interpolation** – desired interpolation mode to be applied

Returns preprocessed and resized data, and dict with batch info

```
ida_lib.core.pipeline_functional.split_operations_by_type(operations: list) → tu-  
                                                 ple
```

Split input operations into sub-lists of each transformation type the normalization operation is placed last to apply correctly the other operations

Parameters **operations** – list of pipeline operations

Returns tuple of lists of the operations separated into color, geometry and independent

```
ida_lib.core.pipeline_functional.postprocess_data(batch: list, batch_info: dict,
                                                data_original: Optional[list] =
                                                None, visualize: bool = False, ori-
                                                nal_type: torch.dtype = torch.uint8,
                                                output_format: str = 'dict') → list
```

Restores the data to the original form; separating the matrix into the different 2d input data and point coordinates.

Parameters

- **batch** – list of elements to be transformed through the pipe
- **batch_info** – dict with necessary information about the batch data
- **data_original** – original batch before transforms
- **visualize** – whether to run the visualization tool or not
- **original_type** – torch original type of the input data to do the conversion of the output data to this type
- **output_format** – whether to format the output element as a dict or as a tuple

Returns

```
ida_lib.core.pipeline_functional.switch_point_positions(point_matrix, input_list)
```

ida_lib.core.pipeline_geometric_ops module

```
class ida_lib.core.pipeline_geometric_ops.HflipPipeline(probability: float = 1, ex-
                                                               change_points: tuple =
                                                               None)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Horizontally flip the input image-mask-keypoints and 2d data

```
config_parameters(data_info: dict)
```

```
get_op_matrix() → None._VariableFunctions.tensor
```

```
need_data_info() → bool
```

```
switch_points()
```

```
class ida_lib.core.pipeline_geometric_ops.VflipPipeline(probability: float, ex-
                                                               change_points: tuple =
                                                               None)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Vertically flip the input image-mask-keypoints and 2d data

```
config_parameters(data_info: dict)
```

```
get_op_matrix() → None._VariableFunctions.tensor
```

```
need_data_info() → bool
```

```
switch_points()
```

```
class ida_lib.core.pipeline_geometric_ops.RotatePipeline(degrees: int, center:
                                                               None._VariableFunctions.tensor
                                                               = None, probability: float
                                                               = 1)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Rotate the input image-mask-keypoints and 2d data by the input degrees

```
config_parameters (data_info: dict)
get_op_matrix () → None._VariableFunctions.tensor
need_data_info () → bool
static switch_points ()

class ida_lib.core.pipeline_geometric_ops.ShearPipeline (shear: tuple, probability:
                                                               float = 1)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Shear the input image-mask-keypoints and 2d data by the input shear factor

get_op_matrix () → None._VariableFunctions.tensor
static need_data_info () → bool
static switch_points ()

class ida_lib.core.pipeline_geometric_ops.ScalePipeline (scale_factor: Union[float,
                                                               tuple], probability:
                                                               float = 1, center:
                                                               None._VariableFunctions.tensor
                                                               = None)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Scale the input image-mask-keypoints and 2d data by the input scaling value

config_parameters (data_info: dict)
get_op_matrix ()
need_data_info ()
static switch_points ()

class ida_lib.core.pipeline_geometric_ops.TranslatePipeline (translation: tuple,
                                                               probability: float =
                                                               1)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Translate the input image-mask-keypoints and 2d data by the input translation

get_op_matrix () → None._VariableFunctions.tensor
static need_data_info () → bool
static switch_points ()

class ida_lib.core.pipeline_geometric_ops.RandomScalePipeline (probability: float,
                                                               scale_range: tuple,
                                                               keep_aspect:
                                                               bool = True, center_deviation:
                                                               int
                                                               = None, center:
                                                               None._VariableFunctions.tensor
                                                               = None)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Scale the input image-mask-keypoints and 2d data by a random scaling value calculated within the input range

config_parameters (data_info: dict)
get_op_matrix () → None._VariableFunctions.tensor
need_data_info () → bool
```

```
static switch_points()

class ida_lib.core.pipeline_geometric_ops.RandomRotatePipeline(degrees_range:
    tuple, prob-
    ability: float
    = 1, cen-
    ter_deviation:
    int = None,
    center:
    None._VariableFunctions.tensor
    = None)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Rotate the input image-mask-keypoints and 2d data by a random scaling value calculated within the input range

```
config_parameters(data_info: dict)
get_op_matrix() → None._VariableFunctions.tensor
need_data_info() → bool
static switch_points()

class ida_lib.core.pipeline_geometric_ops.RandomShearPipeline(probability: float,
    shear_range: tu-
    ple)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Shear the input image-mask-keypoints and 2d data by a random shear value calculated within the input range

```
get_op_matrix() → None._VariableFunctions.tensor
static need_data_info() → bool
static switch_points()
```

```
class ida_lib.core.pipeline_geometric_ops.RandomTranslatePipeline(probability:
    float,
    transla-
    tion_range:
    tuple,
    same_translation_on_axis:
    bool      =
    False)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Translate the input image-mask-keypoints and 2d data by a random translation value calculated within the input range

```
get_op_matrix()
static need_data_info() → bool
static switch_points()
```

ida_lib.core.pipeline_local_ops module

```
class ida_lib.core.pipeline_local_ops.BlurPipeline(probability: float = 1, blur_size: tuple = 5, 5)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Blur input image ( non-weighted blur)

apply_to_image_if_probability(img: numpy.ndarray) → numpy.ndarray
get_op_matrix()

class ida_lib.core.pipeline_local_ops.GaussianBlurPipeline(probability: float = 1,
                                                               blur_size: tuple = 5,
                                                               5)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Blur input image by a Gaussian function

apply_to_image_if_probability(img: numpy.ndarray) → numpy.ndarray
get_op_matrix()

class ida_lib.core.pipeline_local_ops.GaussianNoisePipeline(probability: float =
                                                               1, var: float = 0.5)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Add gaussian noise to the input image (gaussian noise is a statistical noise having a probability density function (PDF) equal to that of the normal distribution)

apply_to_image_if_probability(img: numpy.ndarray) → numpy.ndarray
get_op_matrix()

class ida_lib.core.pipeline_local_ops.PoissonNoisePipeline(probability: float = 1)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Add poison noise to the input image (Speckle is a granular interference that inherently exists in and degrades the quality of the active radar, synthetic aperture radar (SAR), medical ultrasound and optical coherence tomography images. It is applied by adding Poisson-distributed noise)

apply_to_image_if_probability(img: numpy.ndarray) → numpy.ndarray
get_op_matrix()

class ida_lib.core.pipeline_local_ops.SaltAndPepperNoisePipeline(probability=1,
                                                               amount: Optional[float]
                                                               = 0.01,
                                                               s_vs_p: Optional[float]
                                                               = 0.5)
Bases: ida_lib.core.pipeline_operations.PipelineOperation
Add salt and pepper noise to the input image (salt-and-pepper noise is a statistical noise compose of white (salt) and black (pepper) pixels)

apply_to_image_if_probability(img: numpy.ndarray) → numpy.ndarray
get_op_matrix()
```

```
class ida_lib.core.pipeline_local_ops.SpeckleNoisePipeline(probability: float
= 1, mean: Optional[float] = 0, var: Optional[float] = 0.01)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Add speckle noise to the input image (Speckle is a granular interference that inherently exists in and degrades the quality of the active radar, synthetic aperture radar (SAR), medical ultrasound and optical coherence tomography images. It is applied by adding the image multiplied by the noise matrix -> img + img * uniform_noise)

```
apply_to_image_if_probability(img: numpy.ndarray) → numpy.ndarray  
get_op_matrix()
```

ida_lib.core.pipeline_operations module

```
class ida_lib.core.pipeline_operations.PipelineOperation(op_type: str, probability: float = 1)
```

Bases: abc.ABC

Abstract class of pipeline operations

```
apply_according_to_probability() → bool  
abstract get_op_matrix()  
get_op_type()
```

ida_lib.core.pipeline_pixel_ops module

```
class ida_lib.core.pipeline_pixel_ops.ContrastPipeline(contrast_factor: float, probability: float = 1)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change the contrast of the input image.

```
get_op_matrix()  
transform_function(x: int) → float
```

```
class ida_lib.core.pipeline_pixel_ops.RandomContrastPipeline(contrast_range: tuple, probability: float = 1)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change the contrast of the input image with a random contrast factor calculated within the input range

```
get_op_matrix()  
transform_function(x)
```

```
class ida_lib.core.pipeline_pixel_ops.BrightnessPipeline(brightness_factor: float, probability: float = 1)
```

Bases: *ida_lib.core.pipeline_operations.PipelineOperation*

Change brightness of the input image

```
get_op_matrix()  
get_op_type()  
transform_function(x: int) → float
```

```
class ida_lib.core.pipeline_pixel_ops.RandomBrightnessPipeline(probability:  
float, brightness_range:  
tuple)  
Bases: ida_lib.core.pipeline_operations.PipelineOperation  
Change brightness of the input image to random amount calculated within the input range  
get_op_matrix()  
get_op_type()  
transform_function(x: int) → float  
  
class ida_lib.core.pipeline_pixel_ops.GammaPipeline(gamma_factor: float, probability:  
float = 1)  
Bases: ida_lib.core.pipeline_operations.PipelineOperation  
Change the luminance of the input image  
get_op_matrix()  
transform_function(x: int) → float  
  
class ida_lib.core.pipeline_pixel_ops.RandomGammaPipeline(gamma_range: tuple,  
probability: float = 1)  
Bases: ida_lib.core.pipeline_operations.PipelineOperation  
Change the luminance of the input image by a random gamma factor calculated within the input range  
get_op_matrix()  
transform_function(x: int) → float  
  
class ida_lib.core.pipeline_pixel_ops.NormalizePipeline(probability: float = 1,  
old_range: tuple = 0, 255,  
new_range: tuple = 0, 1)  
Bases: ida_lib.core.pipeline_operations.PipelineOperation  
Change the pixels value to a normalize range  
get_op_matrix()  
static transform_function(x: int) → float  
  
class ida_lib.core.pipeline_pixel_ops.DenormalizePipeline(probability: float = 1,  
old_range: tuple = 0, 1,  
new_range: tuple = 0,  
255)  
Bases: ida_lib.core.pipeline_operations.PipelineOperation  
Denormalize pixel value  
get_op_matrix()  
transform_function(x: int) → float
```

Module contents

ida_lib.image_augmentation package

Submodules

ida_lib.image_augmentation.augment_to_disk module

```
class ida_lib.image_augmentation.augment_to_disk.AugmentToDisk(dataset:
                                                                torch.utils.data.dataset.Dataset,
                                                                samples_per_item:
                                                                Optional[int]
                                                                = 2, total_output_samples:
                                                                Optional[int]
                                                                = None, operations:
                                                                Optional[list] =
                                                                None, interpolation:
                                                                str
                                                                = 'bilinear',
                                                                padding_mode:
                                                                str = 'zeros',
                                                                resize: Optional[tuple]
                                                                = None, output_extension:
                                                                str = 'jpg', output_csv_path:
                                                                str = 'annotations.csv',
                                                                output_path:
                                                                str = './augmented')
```

Bases: object

The AugmentToDisk object allows to perform Data Image Augmentation directly to disk. That is, to save the images generated to disk to be used in future processes.

`final_save()`

This method can be overwritten to make a customized saving of the items according to the interests of the user

Method that runs only once, once all the images have been processed. Useful for writing csv with image annotations. By default the annotations of all images are saved in the same file. The csv file will have one row for each generated element, identified by its id. Each column will correspond with the labels associated to each generated element. In the case of coordinate lists, their coordinates are arranged in columns separating the x and y coordinates in each element (point0_x, point0_y, point1_x, ..., point_y)

`save_item(item: dict, index: int, output_path: str, types_2d: list, other_types: list, element: int)`

This method can be overwritten to make a customized saving of the items according to the interests of the user

Method that implements the way to save to disk each of the generated elements. By default it saves all the generated images in the specified path. The samples are organized by name following the form:

- images: <id_image>_<sample number> <extension>
- other two-dimensional types: <id_image>-<data_type>_<sample number> <extension>

Annotations on the data, such as labels, or point coordinates are stored in dictionaries that will be written when all the images have been processed.

Parameters

- **item** – input element to be saved to disk
- **element** – input element number to identify it
- **index** – sample number to which the input item corresponds
- **output_path** – path to the directory in which to save the generated data
- **types_2d** – list of types of two dimensional data of the input item
- **other_types** – list of types that are not two-dimensional elements

ida_lib.image_augmentation.data_loader module

```
class ida_lib.image_augmentation.data_loader.DataAugmentDataLoader(batch_size,
dataset:
torch.utils.data.dataset.Dataset,
shuffle=True,
pipeline_operations=None,
resize=None,
interpolation:
str = 'bilinear',
padding_mode:
str = 'zeros',
output_format:
str = 'dict', output_type: Optional[torch.dtype]
= None)
```

Bases: torch.utils.data.dataloader.DataLoader

The DataAugmentDataLoader class implements a Pytorch DataLoader but groups it into one class:

- The Dataset object that takes care of reading the data
- The iterative DataLoader object that will serve as an input system for a neural network.
- A pipeline that applies data image Augmentation operations over the input data.

To make use of this class, it is necessary to provide a dataset to make a personalized reading of your data.

```
class InnerDataset(pipeline, dataset)
Bases: torch.utils.data.dataset.Dataset
```

inner dataset is an internal class that uses the DataAugmentDataLoader to add the pipeline to the input dataset

Module contents

ida_lib.operations package

Submodules

ida_lib.operations.geometry_ops_functional module

```
ida_lib.operations.geometry_ops_functional.affine_compose_data(data: dict,
                                                               matrix:
                                                               None._VariableFunctions.tensor)
                                                               → dict
```

Parameters

- **data** – dict of elements to be transformed
- **matrix** – matrix of transformation

Returns

transformed data

```
ida_lib.operations.geometry_ops_functional.affine_coordinates_matrix(matrix_coordinates:
                                                               None._VariableFunctions.tensor)
                                                               ma-
                                                               trix_transformation:
                                                               None._VariableFunctions.tensor)
                                                               →
                                                               None._VariableFunctions.tensor
```

```
ida_lib.operations.geometry_ops_functional.affine_image(img:
                                                       None._VariableFunctions.tensor,
                                                       matrix:
                                                       None._VariableFunctions.tensor,
                                                       interpolation: str = 'bilinear',
                                                       padding_mode:
                                                       str = 'zeros') →
                                                       None._VariableFunctions.tensor
```

```
ida_lib.operations.geometry_ops_functional.config_scale_matrix(scale_factor,
                                                               center, matrix)
```

```
ida_lib.operations.geometry_ops_functional.get_rotation_matrix(center:
                                                               None._VariableFunctions.tensor,
                                                               degrees:
                                                               None._VariableFunctions.tensor)
```

```
ida_lib.operations.geometry_ops_functional.get_scale_matrix(center:
                                                       None._VariableFunctions.tensor,
                                                       scale_factor:
                                                       Union[float,
                                                       None._VariableFunctions.tensor])
```

```
ida_lib.operations.geometry_ops_functional.get_shear_matrix(shear_factor:
                                                       tuple) →
                                                       None._VariableFunctions.tensor
```

```
ida_lib.operations.geometry_ops_functional.get_squared_scale_matrix (center:  
    None._VariableFunctions.tensor,  
    scale_factor:  
    Union[float,  
    None._VariableFunctions.tensor])  
  
ida_lib.operations.geometry_ops_functional.get_squared_shear_matrix (shear_factor:  
    tuple)  
    →  
    None._VariableFunctions.tensor  
  
ida_lib.operations.geometry_ops_functional.get_translation_matrix (translation:  
    None._VariableFunctions.tensor)  
    →  
    None._VariableFunctions.tensor  
  
ida_lib.operations.geometry_ops_functional.hflip_compose_data (data: dict) →  
    dict
```

Parameters **data** – dict of elements to be transformed

Returns transformed data

```
ida_lib.operations.geometry_ops_functional.hflip_coordinates_matrix (matrix:  
    None._VariableFunctions.tensor,  
    width:  
    int) →  
    None._VariableFunctions.tensor
```

```
ida_lib.operations.geometry_ops_functional.hflip_image (img:  
    None._VariableFunctions.tensor)  
    →  
    None._VariableFunctions.tensor
```

```
ida_lib.operations.geometry_ops_functional.own_affine (tensor: torch.Tensor, ma-  
    trix: torch.Tensor, inter-  
    polation: str = 'bilinear',  
    padding_mode: str = 'bor-  
    der') → torch.Tensor
```

Apply an affine transformation to the image.

Parameters

- **tensor** – The image tensor to be warped.
- **matrix** – The 2x3 affine transformation matrix.
- **interpolation** – interpolation mode to calculate output values ‘bilinear’ | ‘nearest’. Default: ‘bilinear’.
- **padding_mode** – padding mode for outside grid values ‘zeros’ | ‘border’ | ‘reflection’. Default: ‘zeros’.

Returns The warped image.

```
ida_lib.operations.geometry_ops_functional.prepare_data (func)
```

Decorator that prepares the input data to apply the geometric transformation. For this purpose, it concatenates all the two-dimensional elements of the input data in the same tensor on which a single transformation is applied. If the input data contains point coordinates, they are grouped in a matrix as homogeneous coordinates, over which a single matrix multiplication is performed.

Parameters **func** – geometric function to be applied to the data

Returns processed data

```
ida_lib.operations.geometry_ops_functional.rotate_compose_data(data:      dict,
                                                               degrees:
                                                               None._VariableFunctions.tensor,
                                                               center:
                                                               None._VariableFunctions.tensor)
```

Parameters

- **data** – dict of elements to be transformed
- **degrees** – counterclockwise degrees of rotation
- **center** – center of rotation. Default, center of the image

Returns transformed data

```
ida_lib.operations.geometry_ops_functional.rotate_coordinates_matrix(matrix_coordinates:
                                                               None._VariableFunctions.tensor,
                                                               matrix:
                                                               None._VariableFunctions.tensor
                                                               →
                                                               None._VariableFunctions.tensor)
```

```
ida_lib.operations.geometry_ops_functional.rotate_image(img:
                                                               None._VariableFunctions.tensor,
                                                               degrees:
                                                               None._VariableFunctions.tensor,
                                                               center:
                                                               None._VariableFunctions.tensor)
                                                               →
                                                               None._VariableFunctions.tensor
```

mode

```
ida_lib.operations.geometry_ops_functional.scale_compose_data(data:      dict,
                                                               scale_factor:
                                                               Union[float,
                                                               None._VariableFunctions.tensor],
                                                               center:      Op-
                                                               tional[None._VariableFunctions.tensor]
                                                               = None) → dict
```

Parameters

- **data** – dict of elements to be transformed
- **scale_factor** – factor of scaling
- **center** – center of scaling. By default its taken the center of the image

Returns transformed data

```
ida_lib.operations.geometry_ops_functional.scale_coordinates_matrix(matrix_coordinates:
                                                               None._VariableFunctions.tensor,
                                                               matrix:
                                                               None._VariableFunctions.tensor
                                                               →
                                                               None._VariableFunctions.tensor)
```

```
ida_lib.operations.geometry_ops_functional.scale_image(img:  
    None._VariableFunctions.tensor,  
    scale_factor:  
        None._VariableFunctions.tensor,  
        center:  
            None._VariableFunctions.tensor)  
    →  
    None._VariableFunctions.tensor  
  
ida_lib.operations.geometry_ops_functional.shear_compose_data(data:      dict,  
                           shear_factor:  
                               tuple) → dict
```

Parameters

- **data** – dict of elements to be transformed
- **shear_factor** – pixels of shearing

Returns transformed data

```
ida_lib.operations.geometry_ops_functional.shear_coordinates_matrix(matrix_coordinates:  
    None._VariableFunctions.tensor,  
    matrix:  
        None._VariableFunctions.tensor)  
    →  
    None._VariableFunctions.tensor
```

```
ida_lib.operations.geometry_ops_functional.shear_image(img:  
    None._VariableFunctions.tensor,  
    shear_factor:  
        None._VariableFunctions.tensor)  
    →  
    None._VariableFunctions.tensor
```

```
ida_lib.operations.geometry_ops_functional.translate_compose_data(data:  dict,  
                           translation:  
                               Union[int,  
                                     None._VariableFunctions.tensor])  
    → dict
```

Parameters

- **data** – dict of elements to be transformed
- **translation** – number of pixels to translate

Returns transformed data

```
ida_lib.operations.geometry_ops_functional.translate_coordinates_matrix(matrix_coordinates:  
    None._VariableFunctions.tensor,  
    trans-  
    la-  
    tion:  
        None._VariableFunctions.tensor)  
    →  
    None._VariableFunctions.tensor
```

```

ida_lib.operations.geometry_ops_functional.translate_image(img:
    None._VariableFunctions.tensor,
    translation:
    None._VariableFunctions.tensor)
    →
    None._VariableFunctions.tensor

ida_lib.operations.geometry_ops_functional.vflip_compose_data(data: dict) →
dict

Parameters data – dict of elements to be transformed

Returns transformed data

ida_lib.operations.geometry_ops_functional.vflip_coordinates_matrix(matrix:
    None._VariableFunctions.tensor,
    height:
    int) →
None._VariableFunctions.tensor

ida_lib.operations.geometry_ops_functional.vflip_image(img:
    None._VariableFunctions.tensor)
    →
    None._VariableFunctions.tensor

```

ida_lib.operations.pixel_ops_functional module

```

ida_lib.operations.pixel_ops_functional.apply_blur(img, blur_size=5, 5)
ida_lib.operations.pixel_ops_functional.apply_gaussian_blur(img, blur_size=5, 5)

Parameters

```

- **img** – input image to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)

Returns returns the transformed image

```

ida_lib.operations.pixel_ops_functional.apply_gaussian_noise(image, var=20)
ida_lib.operations.pixel_ops_functional.apply_lut_by_pixel_function(function,
    image:
    numpy.ndarray)
    →
    numpy.ndarray

Applies the input operation to the image using a LUT

```

Parameters

- **function** – Mathematical function that represents the operation to carry out in each pixel of the image
- **image** – input image

Returns

```

ida_lib.operations.pixel_ops_functional.apply_poisson_noise(image)
ida_lib.operations.pixel_ops_functional.apply_salt_and_pepper_noise(image,
    amount=0.05,
    s_vs_p=0.5)

```

```
ida_lib.operations.pixel_ops_functional.apply_speckle_noise(image,      mean=0,  
                                         var=0.01)
```

```
ida_lib.operations.pixel_ops_functional.blur(img:           Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray], blur_size) → Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray]
```

Parameters

- **img** – input image to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.change_brightness(image:      Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray],  
                                         brightness: int)  
                                         → Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray]
```

Change the brightness of the input image.

Parameters

- **image** – input image to be normalized
- **brightness** – desired amount of brightness for the image 0 - no brightness 1 - same 2 - max brightness

Returns transformed image

```
ida_lib.operations.pixel_ops_functional.change_contrast(image:      Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray], contrast) → Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray]
```

:param image : input image to be transformed :param contrast: modification factor to be applied to the image contrast

- 0 - total contrast removal
- 1 - dont modify
- >1 - augment contrast

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.change_gamma(image:      Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray], gamma: float) → Union[dict,  
                                         None._VariableFunctions.tensor,  
                                         numpy.ndarray]
```

Parameters

- **image** – input image to be transformed

- **gamma** – desired factor gamma * gamma = 0 -> removes image luminance (black output image) * gamma = 1 -> remains unchanged * gamma > 1 -> increases luminance

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.gaussian_blur(img: Union[dict,
    None._VariableFunctions.tensor,
    numpy.ndarray], blur_size) → Union[dict,
    None._VariableFunctions.tensor,
    numpy.ndarray]
```

Parameters

- **img** – input image to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.gaussian_noise(image: Union[dict,
    None._VariableFunctions.tensor,
    numpy.ndarray], var=20) → Union[dict,
    None._VariableFunctions.tensor,
    numpy.ndarray]
```

Parameters

- **image** – input image to be transformed
- **var** – var of the gaussian distribution of noise

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.get_brightness_function(brightness: int)
```

Parameters **brightness** – brightness factor

Returns Return the lambda function of the brightness operation

```
ida_lib.operations.pixel_ops_functional.get_contrast_function(contrast: float)
```

Parameters **contrast** – modification factor to be applied to the image contrast

Returns Return the lambda function of the contrast operation

```
ida_lib.operations.pixel_ops_functional.get_gamma_function(gamma)
```

Parameters **gamma** – desired factor gamma

Returns Returns the lambda function of the gamma adjust operation

```
ida_lib.operations.pixel_ops_functional.histogram_equalization(img: Union[dict,
    None._VariableFunctions.tensor,
    numpy.ndarray]) → Union[dict,
    None._VariableFunctions.tensor,
    numpy.ndarray]
```

Parameters **img** – input image to be transformed

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.normalize_image(img:      numpy.ndarray,
                                                       norm_type: int = 32) →
                                                       numpy.ndarray
```

Normalize the input image

Parameters

- **img** – input image to be normalized
- **norm_type** – opencv normalization type ('cv2.NORM_MINMAX'
|cv2.NORM_HAMMING |cv2.NORM_HAMMING2 |cv2.NORM_INF
|cv2.NORM_RELATIVE ...)

Returns normalized image

```
ida_lib.operations.pixel_ops_functional.poisson_noise(image:      Union[dict,
                                                               None._VariableFunctions.tensor,
                                                               numpy.ndarray])
                                                               →      Union[dict,
                                                               None._VariableFunctions.tensor,
                                                               numpy.ndarray]
```

Parameters **image** – input image to be transformed

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.prepare_data_for_opencv(func)
```

Decorator that prepares the input data to apply the transformation that only affects the image (color). :param func: color function to be applied to the data :return: processed data

```
ida_lib.operations.pixel_ops_functional.salt_and_pepper_noise(image:
                                                               Union[dict,
                                                               None._VariableFunctions.tensor,
                                                               numpy.ndarray],
                                                               amount, s_vs_p)
                                                               →      Union[dict,
                                                               None._VariableFunctions.tensor,
                                                               numpy.ndarray]
```

Parameters

- **image** – input image to be transformed
- **amount** – percentage of image's pixels to be occupied by noise
- **s_vs_p** – percentage of salt respect total noise. Default same salt (white pixel) as pepper (black pixels)

Returns returns the transformed image

```
ida_lib.operations.pixel_ops_functional.speckle_noise(image:      Union[dict,
                                                               None._VariableFunctions.tensor,
                                                               numpy.ndarray], mean=0,
                                                               var=0.01) → Union[dict,
                                                               None._VariableFunctions.tensor,
                                                               numpy.ndarray]
```

Parameters

- **image** – input image to be transformed
- **mean** – mean of noise distribution
- **var** – variance of noise distribution

Returns returns the transformed image

ida_lib.operations.transforms module

ida_lib.operations.transforms.**hflip**(*data*: dict, *visualize*: bool = False) → dict

Horizontally flip the input data.

Parameters

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

ida_lib.operations.transforms.**vflip**(*data*: dict, *visualize*: bool = False) → dict

Vertically flip the input data.

Parameters

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

ida_lib.operations.transforms.**affine**(*data*: dict, *matrix*: None._VariableFunctions.tensor, *visualize*: bool = False) → dict

Applies affine transformation to the data

:param *data* dict of elements to be transformed :param *matrix*: matrix of transformation :param *visualize*: if true it activates the display tool to debug the transformation :return: transformed data

ida_lib.operations.transforms.**rotate**(*data*: dict, *degrees*: float, *visualize*: bool = False, *center*: Union[None, torch.Tensor] = None) → dict

Rotate each element of the input data by the indicated degrees counterclockwise

Parameters

- **data** – dict of elements to be transformed
- **degrees** – degrees of rotation
- **center** – center of rotation. If center is None, it is taken the center of the image to apply the rotation
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

ida_lib.operations.transforms.**shear**(*data*: dict, *shear_factor*: tuple, *visualize*: bool = False) → dict

Shear input data by the input shear factor

Parameters

- **data** – dict of elements to be transformed
- **shear_factor** – pixels to shear

- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

```
ida_lib.operations.transforms.scale(data: dict, scale_factor, visualize: bool = False, center: Union[None, torch.Tensor] = None) → dict
```

Scale each element of the input data by the input factor.

Parameters

- **data** – dict of elements to be transformed
- **scale_factor** – factor of scaling to be applied * scale_factor < 1 -> output image is smaller than input one * scale_factor = 1 -> output image is the same as the input image * scale_factor = 2 -> each original pixel occupies 2 pixels in the output image
- **center** – center of scaling. If center is None, it is taken the center of the image to apply the scale
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

```
ida_lib.operations.transforms.translate(data: dict, translation: tuple, visualize: bool = False) → dict
```

Translate input by the input translation.

Parameters

- **data** – dict of elements to be transformed
- **translation** – number of pixels to be translated
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

```
ida_lib.operations.transforms.change_gamma(data: Union[dict, torch.Tensor, numpy.ndarray], gamma, visualize: bool = False) → Union[dict, torch.Tensor, numpy.ndarray]
```

Adjust image's gamma (luminance correction). if the input data is a dictionary, only those corresponding

to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **gamma** – desired gamma factor (luminance of image)
 - gamma = 0 -> removes image luminance (black output image)
 - gamma = 1 -> remains unchanged
 - gamma > 1 -> increases luminance
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

```
ida_lib.operations.transforms.change_contrast (data: Union[dict, torch.Tensor, numpy.ndarray], contrast: float, visualize: bool = False) → Union[dict, torch.Tensor, numpy.ndarray]
```

Change the image contrast. if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **contrast** – desired contrast factor to the data * contrast = 0 -> removes image contrast (white output image) * contrast = 1 -> remains unchanged * contrast > 1 -> increases contrast

:param **visualize** : if true it activates the display tool to debug the transformation :return: transformed data

```
ida_lib.operations.transforms.change_brightness (data: Union[dict, torch.Tensor, numpy.ndarray], bright=1, visualize: bool = False) → Union[dict, torch.Tensor, numpy.ndarray]
```

Change the image brightness. if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **bright** – desired brightness amount for the data
 - brightness = 0 -> removes image brightness (black output image)
 - brightness = 1 -> remains unchanged
 - brightness > 1 -> increases brightness
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

```
ida_lib.operations.transforms.equalize_histogram (data: Union[dict, torch.Tensor, numpy.ndarray], visualize: bool = False) → Union[dict, torch.Tensor, numpy.ndarray]
```

Equalize image histogram. if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

Returns transformed data

```
ida_lib.operations.transforms.inject_gaussian_noise (data: Union[dict, torch.Tensor, numpy.ndarray], var=0.5, visualize: bool = False) → Union[dict, torch.Tensor, numpy.ndarray]
```

Inject gaussian noise. If the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **var** – variance of the noise distribution
- **visualize** – if true it activates the display tool to debug the transformation

Returns

 transformed data

```
ida_lib.operations.transforms.inject_poisson_noise(data: Union[dict, torch.Tensor,
                                                               numpy.ndarray], visualize:
                                                               bool = False) → Union[dict,
                                                               torch.Tensor, numpy.ndarray]
```

Inject poisson noise. if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **visualize** – if true it activates the display tool to debug the transformation

Returns

 transformed data

```
ida_lib.operations.transforms.inject_speckle_noise(data: Union[dict, torch.Tensor,
                                                               numpy.ndarray], mean=0,
                                                               var=0.01, visualize: bool =
                                                               False) → Union[dict, torch.Tensor,
                                                               numpy.ndarray]
```

Inject poisson noise. if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **mean** – mean of noise distribution
- **var** – variance of noise distribution
- **visualize** – if true it activates the display tool to debug the transformation

Returns

 transformed data

```
ida_lib.operations.transforms.inject_salt_and_pepper_noise(data: Union[dict,
                                                               torch.Tensor,
                                                               numpy.ndarray],
                                                               amount=0.05,
                                                               s_vs_p=0.5, visualize:
                                                               bool = False) → Union[dict,
                                                               torch.Tensor,
                                                               numpy.ndarray]
```

Inject salt and pepper noise if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **amount** – percentage of image's pixels to be occupied by noise

:param s_vs_p : noise type distribution. Default same salt (white pixel) as pepper (black pixels) :param visualize: if true it activates the display tool to debug the transformation :return: transformed data

```
ida_lib.operations.transforms.blur(data: Union[dict, torch.Tensor, numpy.ndarray], blur_size=5, 5, visualize: bool = False) → Union[dict, torch.Tensor, numpy.ndarray]
```

Blur image. if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)
- **visualize** – if true it activates the display tool to debug the transformation

Returns

transformed data

```
ida_lib.operations.transforms.gaussian_blur(data: Union[dict, torch.Tensor, numpy.ndarray], blur_size=5, 5, visualize: bool = False) → Union[dict, torch.Tensor, numpy.ndarray]
```

Blurring an image by a Gaussian function. if the input data is a dictionary, only those corresponding to an image are altered

Parameters

- **data** – dict of elements to be transformed
- **blur_size** – number of surrounding pixels affecting each output pixel. (pixels on axis X, pixels on axis y)
- **visualize** – if true it activates the display tool to debug the transformation

Returns

transformed data

ida_lib.operations.utils module

```
ida_lib.operations.utils.add_new_axis(arr: numpy.ndarray)
```

```
ida_lib.operations.utils.arrays_equal(arr1, arr2)
```

```
ida_lib.operations.utils.data_to_numpy(data)
```

```
ida_lib.operations.utils.dtype_to_torch_type(im_type: numpy.dtype)
```

Maps the numpy type to the equivalent torch.type

Parameters

im_type – numpy type

Returns

torch.type

```
ida_lib.operations.utils.element_to_dict_csv_format(item, name)
```

```
ida_lib.operations.utils.get_principal_type(data: dict)
```

```
ida_lib.operations.utils.get_torch_image_center(data)
```

```
ida_lib.operations.utils.homogeneous_points_to_list(keypoints)
```

```
ida_lib.operations.utils.homogeneous_points_to_matrix(keypoints)
```

```
ida_lib.operations.utils.is_a_normalized_image(image)
```

```
ida_lib.operations.utils.is_numpy_data(data)
```

```
ida_lib.operations.utils.keypoints_to_homogeneous_and_concatenate(keypoints,  
re-  
size_factor=None)  
ida_lib.operations.utils.keypoints_to_homogeneous_functional(keypoints)  
ida_lib.operations.utils.map_value(x, in_min, in_max, out_min, out_max)  
ida_lib.operations.utils.mask_change_to_01_functional(mask)  
ida_lib.operations.utils.remove_digits(label: str)  
ida_lib.operations.utils.round_torch(arr: None._VariableFunctions.tensor, n_digits: int =  
3)  
ida_lib.operations.utils.save_im(tensor, title)
```

Module contents

3.2.2 Submodules

3.2.3 ida_lib.visualization module

```
ida_lib.visualization.visualize(images: dict, images_originals: dict, max_images: int = 5)  
Generate the bokeh plot of the input batch transformation
```

Parameters

- **images** – list of transformed items (dict of image and other objects)
- **images_originals** – list of original items (dict of image and other objects)
- **max_images** – max number of tabs to be shown

```
ida_lib.visualization.plot_image_transformation(data, data_original)  
Generate the bokeh plot of the input batch transformation
```

Parameters

- **data** – input dict element
- **data_original** – original input element (before transforms)

3.3 Examples

You can find all the examples here: https://github.com/raquelvilas18/ida_lib/tree/master/examples

3.3.1 Pipeline usage example

Pipeline Usage example

```
"""In this file an example of how to use the idaLib pipeline is shown, in which you  
can see:  
* how to declare the pipeline  
* which format to use for the input elements  
* how to display or not the results  
* and how to execute it in general.
```

(continues on next page)

(continued from previous page)

```

For more information see the documentation
"""

from time import time

import numpy as np

from ida_lib.core.pipeline import *
from ida_lib.core.pipeline_geometric_ops import *
from ida_lib.core.pipeline_local_ops import *

data_type = np.uint8

# Read the example image
img: np.ndarray = cv2.imread('../micky.jpg', )
# opencv read in format BGR but IDALib works on RGB
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
#img = img.astype('float32') # Example of bits per pixel used

short_size = min(img.shape[0], img.shape[1])

# Generate an example of segmentation map over the image
segmap = np.zeros((img.shape[0], img.shape[1], 1), dtype=data_type)
segmap[28:171, 35:485, 0] = 1
segmap[10:25, 30:245, 0] = 2
segmap[10:25, 70:385, 0] = 3
segmap[10:110, 5:210, 0] = 4
segmap[18:223, 10:110, 0] = 5

# Generate 2 examples of masks
mask_example1 = np.zeros((img.shape[0], img.shape[1], 1), dtype=data_type)
mask_example1[0:50, 0:50] = 1
mask_example2 = np.zeros((img.shape[0], img.shape[1], 1), dtype=data_type)
mask_example2[-150:, -50:] = 1

# Generate an example of heatmap over the image
x = np.random.randn(img.shape[0] // 4 * img.shape[1] // 4)
y = np.random.randn(img.shape[0] // 4 * img.shape[1] // 4)
# Create heatmap
heatmap, xedges, yedges = np.histogram2d(x, y, bins=(img.shape[0] // 4, img.shape[1] / 
    ↪/ 4))
heatmap_complete = np.zeros((img.shape[0], img.shape[1], 1))
heatmap_complete[0:img.shape[0] // 4, 0:img.shape[1] // 4, 0] = 1

# Generate an example of coordinates of keypoints over the image
# the list of keypoints can be expressed as an array of nx2 dimensions or as a list
# of arrays (1x2) (of 2 coordinates)
number_of_points = 20
# generate 20 random coordinates (to make sure they don't go outside the image
# ↪boundaries set short_Size as the limit)
random_coordinates = np.random.randint(1, short_size, number_of_points * 2).
    ↪reshape(number_of_points, 2)

# Generate the input item of the pipeline. Its very important to name each element
# ↪with its data type so that the
# pipeline understands them. *If the item contains more than one element of each type,
# ↪just number them like mask1

```

(continues on next page)

(continued from previous page)

```

# and mask2
data = {'image': img, 'keypoints': random_coordinates, 'mask1': mask_example1, 'mask2'
        : mask_example2,
        'heatmap': heatmap_complete}

# For this example we are going to use the same identical input element but repeated
# n times to create a batch so we
# can see the different transformations
samples = 10
batch = [data.copy() for _ in range(samples)]

start_time = time() # time measurement

# Define the pipeline and operations.
pip = Pipeline(interpolation='nearest',
               pipeline_operations=(
                   ScalePipeline(probability=1, scale_factor=0.5),
                   ShearPipeline(probability=0, shear=(0.2, 0.2)),
                   TranslatePipeline(probability=0, translation=(10, 50)),
                   HflipPipeline(probability=0, exchange_points=[(0, 5), (1, 6)]),
                   RandomRotatePipeline(probability=0, degrees_range=(-20, 20)),
                   GaussianNoisePipeline(probability=0)))

# pass the batch through the pipeline and visualize the transformations
batch = pip(batch, visualize=True)

consumed_time = time() - start_time
# keep in mind that visualization is a significant overhead, so to take a good
# measure of
# performance set visualize=False
print("Total time consumed to process " + str(samples) + " samples: " + str(consumed_
    time))
print("Time per sample: :" + str(consumed_time / samples))

```

Dataloader Usage example

```

"""
This file has an example of how to use IDALib's own DataLoader which includes a
pipeline to perform image data
augmentation on your data.
This code follows the pytorch example of of using a dataloader
https://pytorch.org/tutorials/beginner/data_loading_tutorial.html but adapted to the
ida-lib dataloader
"""

import os

import kornia
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from skimage import io

from ida_lib.core.pipeline_geometric_ops import TranslatePipeline, VflipPipeline,
    HflipPipeline, RandomShearPipeline

```

(continues on next page)

(continued from previous page)

```

from ida_lib.core.pipeline_pixel_ops import ContrastPipeline
from ida_lib.image_augmentation.data_loader import *

# Firstly create custom dataset to read the input data
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir

    def __len__(self):
        return len(self.landmarks_frame)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = os.path.join(self.root_dir,
                               self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:]
        landmarks = np.array([landmarks])
        landmarks = landmarks.astype('float').reshape(-1, 2)
        sample = {'id': self.landmarks_frame.iloc[idx, 0], 'image': image, 'keypoints':
        landmarks}
        return sample

    # Auxiliar function to display elements
    def show_landmarks(image, landmarks):
        """Show image with landmarks"""
        img = kornia.tensor_to_image(image.byte())
        plt.imshow(img)
        landmarks = landmarks.cpu().numpy()
        plt.scatter(landmarks[:, 0], landmarks[:, 1], s=10, marker='o', c='r')
        plt.show()
        plt.pause(0.001)  # pause a bit so that plots are updated

    # initialize custom dataset
    face_dataset = FaceLandmarksDataset(csv_file='faces/face_landmarks.csv',
                                         root_dir='faces/')

    # initialize the custom dataloader
    dataloader = AugmentDataLoader(dataset=face_dataset,
                                   batch_size=4,
                                   shuffle=True,
                                   pipeline_operations=(
                                       TranslatePipeline(probability=1, translation=(30, -10)),
```

(continues on next page)

(continued from previous page)

```

        VflipPipeline(probability=0.5),
        HflipPipeline(probability=0.5),
        ContrastPipeline(probability=0.5, contrast_
→factor=1),
                    RandomShearPipeline(probability=0.5, shear_
→range=(0, 0.5))),
                    resize=(500, 300),
                    # we must indicate the size of the resize because the_
→images are not all the same size
                    interpolation='bilinear',
                    padding_mode='zeros'
                )

number_of_iterations = 3 # number of times the entire dataset is processed
for epoch in range(number_of_iterations - 1):
    for i_batch, sample_batched in enumerate(dataloader): # our dataloader works_
→like a normal dataloader
        print(i_batch, )
        keypoints = sample_batched['keypoints'][0, :, :]
        show_landmarks(sample_batched['image'][0], keypoints)
    print('all elements of the original dataset have been displayed and processed')

```

Image Augmentation to Disk example

```

import os

import numpy as np
import pandas as pd
import torch
from skimage import io
from torch.utils.data import Dataset

from ida_lib.core.pipeline_geometric_ops import RandomScalePipeline, HflipPipeline
from ida_lib.core.pipeline_pixel_ops import RandomContrastPipeline
from ida_lib.image_augmentation.augment_to_disk import AugmentToDisk

# Create custom dataset to read the input data to be augmented
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.landmarks_frame)

```

(continues on next page)

(continued from previous page)

```

def __getitem__(self, idx):
    if torch.is_tensor(idx):
        idx = idx.tolist()
    img_name = os.path.join(self.root_dir,
                           self.landmarks_frame.iloc[idx, 0])
    item_id = (self.landmarks_frame.iloc[idx, 0]).split('.')[0]
    image = io.imread(img_name)
    landmarks = self.landmarks_frame.iloc[idx, 1:]
    landmarks = np.array([landmarks])
    landmarks = landmarks.astype('float').reshape(-1, 2)
    sample = {'id': item_id, 'image': image, 'landmarks': landmarks}
    return sample

# Initialize the custom dataset

face_dataset = FaceLandmarksDataset(csv_file='faces/face_landmarks.csv',
                                     root_dir='faces/')

# parameter setting and initialization

augmentor = AugmentToDisk(dataset=face_dataset, # custom dataset that provides the
                           # input data
                           samples_per_item=5, # number of samples per input item
                           operations=(RandomScalePipeline(probability=0.6, scale_
                           range=(0.8, 1.2), center_deviation=20),
                           HflipPipeline(probability=0.5),
                           RandomContrastPipeline(probability=0.5, contrast_
                           range=(1, 1.5))),
                           interpolation='nearest',
                           padding_mode='zeros',
                           resize=(250, 250), # Here resizing is necessary because
                           # the input images have different sizes
                           output_extension='.jpg',
                           output_csv_path='annotations.csv',
                           output_path='./augmented_custom')

augmentor() # Run the augmentation

```

Pipeline Usage example

**CHAPTER
FOUR**

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

```
ida_lib.core, 21
ida_lib.core.pipeline, 13
ida_lib.core.pipeline_functional, 14
ida_lib.core.pipeline_geometric_ops, 15
ida_lib.core.pipeline_local_ops, 18
ida_lib.core.pipeline_operations, 19
ida_lib.core.pipeline_pixel_ops, 19
ida_lib.image_augmentation, 23
ida_lib.image_augmentation.augment_to_disk,
    21
ida_lib.image_augmentation.data_loader,
    22
ida_lib.operations, 36
ida_lib.operations.geometry_ops_functional,
    23
ida_lib.operations.pixel_ops_functional,
    27
ida_lib.operations.transforms, 31
ida_lib.operations.utils, 35
ida_lib.visualization, 36
```


INDEX

A

add_new_axis() (in module `ida_lib.operations.utils`), 35
affine() (in module `ida_lib.operations.transforms`), 31
affine_compose_data() (in module `ida_lib.operations.geometry_ops_functional`), 23
affine_coordinates_matrix() (in module `ida_lib.operations.geometry_ops_functional`), 23
affine_image() (in module `ida_lib.operations.geometry_ops_functional`), 23
apply_according_to_probability() (in module `ida_lib.core.pipeline_operations.PipelineOperation` method), 19
apply.blur() (in module `ida_lib.operations.pixel_ops_functional`), 27
apply_gaussian_blur() (in module `ida_lib.operations.pixel_ops_functional`), 27
apply_gaussian_noise() (in module `ida_lib.operations.pixel_ops_functional`), 27
apply_lut_by_pixel_function() (in module `ida_lib.operations.pixel_ops_functional`), 27
apply_poisson_noise() (in module `ida_lib.operations.pixel_ops_functional`), 27
apply_salt_and_pepper_noise() (in module `ida_lib.operations.pixel_ops_functional`), 27
apply_speckle_noise() (in module `ida_lib.operations.pixel_ops_functional`), 27
apply_to_image_if_probability() (in module `ida_lib.core.pipeline_local_ops.BlurPipeline` method), 18
apply_to_image_if_probability() (in module `ida_lib.core.pipeline_local_ops.GaussianBlurPipeline` method), 18

apply_to_image_if_probability() (in module `ida_lib.core.pipeline_local_ops.GaussianNoisePipeline` method), 18
apply_to_image_if_probability() (in module `ida_lib.core.pipeline_local_ops.PoissonNoisePipeline` method), 18
apply_to_image_if_probability() (in module `ida_lib.core.pipeline_local_ops.SaltAndPepperNoisePipeline` method), 18
apply_to_image_if_probability() (in module `ida_lib.core.pipeline_local_ops.SpeckleNoisePipeline` method), 19
arrays_equal() (in module `ida_lib.operations.utils`), 35
AugmentDataLoader (class in `ida_lib.image_augmentation.data_loader`), 22
AugmentDataLoader.InnerDataset (class in `ida_lib.image_augmentation.data_loader`), 22
AugmentToDisk (class in `ida_lib.image_augmentation.augment_to_disk`), 21

B

blur() (in module `ida_lib.operations.pixel_ops_functional`), 28
blur() (in module `ida_lib.operations.transforms`), 35
BlurPipeline (class in `ida_lib.core.pipeline_local_ops`), 18
BrightnessPipeline (class in `ida_lib.core.pipeline_pixel_ops`), 19

C

change_brightness() (in module `ida_lib.operations.pixel_ops_functional`), 28
change_brightness() (in module `ida_lib.operations.transforms`), 33
change_contrast() (in module `ida_lib.operations.pixel_ops_functional`), 28

change_contrast () (in module *ida_lib.operations.transforms*), 32
change_gamma () (in module *ida_lib.operations.pixel_ops_functional*), 28
change_gamma () (in module *ida_lib.operations.transforms*), 32
config_parameters () (in module *ida_lib.core.pipeline_geometric_ops.HflipPipeline*)
config_parameters () (in module *ida_lib.core.pipeline_geometric_ops.RandomRotatePipeline*)
config_parameters () (in module *ida_lib.core.pipeline_geometric_ops.RandomScalePipeline*)
config_parameters () (in module *ida_lib.core.pipeline_geometric_ops.VflipPipeline*)
config_scale_matrix () (in module *ida_lib.operations.geometry_ops_functional*), 23
ContrastPipeline (class in *ida_lib.core.pipeline_pixel_ops*), 19

D

data_to_numpy () (in module *ida_lib.operations.utils*), 35
DenormalizePipeline (class in *ida_lib.core.pipeline_pixel_ops*), 20
dtype_to_torch_type () (in module *ida_lib.operations.utils*), 35

E

element_to_dict_csv_format () (in module *ida_lib.operations.utils*), 35
equalize_histogram () (in module *ida_lib.operations.transforms*), 33

F

final_save () (*ida_lib.image_augmentation.augment_to_disk*.*AugmentToDisk*)
(method), 21

G

GammaPipeline (class in *ida_lib.core.pipeline_pixel_ops*), 20

gaussian_blur () (in module *ida_lib.operations.pixel_ops_functional*), 29
gaussian_blur () (in module *ida_lib.operations.transforms*), 35
gaussian_noise () (in module *ida_lib.operations.pixel_ops_functional*), 29
gaussianBlurPipeline (class in *ida_lib.core.pipeline_local_ops*), 18
GaussianNoisePipeline (class in *ida.lib.core.pipeline_local_ops*), 18
get_brightness_function () (in module *ida_lib.operations.pixel_ops_functional*), 29
get_compose_function () (in module *ida.lib.core.pipeline_functional*), 14
get_compose_matrix () (in module *ida.lib.core.pipeline_functional*), 14
get_contrast_function () (in module *ida.lib.operations.pixel_ops_functional*), 29
get_data_types () (*ida.lib.core.pipeline.Pipeline* method), 13
get_gamma_function () (in module *ida.lib.operations.pixel_ops_functional*), 29
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.HflipPipeline* method), 15
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.RandomRotatePipeline* method), 17
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.RandomScalePipeline* method), 16
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.RandomShearPipeline* method), 17
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.RandomTranslatePipeline* method), 17
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.RotatePipeline* method), 16
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.ScalePipeline* method), 16
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.ShearPipeline* method), 16
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.TranslatePipeline* method), 16
get_op_matrix () (*ida.lib.core.pipeline_geometric_ops.VflipPipeline* method), 15
get_op_matrix () (*ida.lib.core.pipeline_local_ops.BlurPipeline* method), 18
get_op_matrix () (*ida.lib.core.pipeline_local_ops.GaussianBlurPipeline* method), 18
get_op_matrix () (*ida.lib.core.pipeline_local_ops.GaussianNoisePipeline* method), 18
get_op_matrix () (*ida.lib.core.pipeline_local_ops.PoissonNoisePipeline* method), 18

```

        method), 18
get_op_matrix() (ida_lib.core.pipeline_local_ops.SaltAndPepperNoisePipeline, 24
    method), 18
get_op_matrix() (ida_lib.core.pipeline_local_ops.SpeckleNoisePipeline, 24
    method), 19
get_op_matrix() (ida_lib.core.pipeline_operations.PipelineOperation
    method), 19
        hflip_compose_data() (in module
            ida_lib.operations.geometry_ops_functional),
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.BrightnessPipeline, 15
    method), 19
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.ContrastPipeline, 15
    method), 19
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.DenormalizePipeline, 15
    method), 20
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.GammaPipeline, 20
    method), 20
        homogeneous_points_to_list() (in module
            ida_lib.core.pipeline_geometric_ops), 15
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.NormalizePipeline, 35
    method), 20
        homogeneous_points_to_matrix() (in module
            ida_lib.core.pipeline_operations.utils), 35
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.RandomBrightnessPipeline, 35
    method), 20
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.RandomContrastPipeline
    method), 19
get_op_matrix() (ida_lib.core.pipeline_pixel_ops.RandomGammaPipeline, 20
    method), 20
get_op_type() (ida_lib.core.pipeline_operations.PipelineOperation
    method), 19
get_op_type() (ida_lib.core.pipeline_pixel_ops.BrightnessPipeline, 14
    method), 19
get_op_type() (ida_lib.core.pipeline_pixel_ops.RandomBrightnessPipeline
    method), 20
get_principal_type() (in module
    ida_lib.operations.utils), 35
get_rotation_matrix() (in module
    ida_lib.operations.geometry_ops_functional), 23
get_scale_matrix() (in module
    ida_lib.operations.geometry_ops_functional), 23
get_shear_matrix() (in module
    ida_lib.operations.geometry_ops_functional), 23
get_squared_scale_matrix() (in module
    ida_lib.operations.geometry_ops_functional), 23
get_squared_shear_matrix() (in module
    ida_lib.operations.geometry_ops_functional), 24
get_torch_image_center() (in module
    ida_lib.operations.utils), 35
get_translation_matrix() (in module
    ida_lib.operations.geometry_ops_functional), 24
        hflip_image() (in module
            ida_lib.operations.geometry_ops_functional),
        inject_gaussian_noise() (in module
            ida_lib.operations.transforms), 33

```

H

hflip () (in module ida_lib.operations.transforms), 31

inject_poisson_noise() (in *ida_lib.operations.transforms*), 34
inject_salt_and_pepper_noise() (in *ida_lib.operations.transforms*), 34
inject_speckle_noise() (in *ida_lib.operations.transforms*), 34
is_a_normalized_image() (in *ida_lib.operations.utils*), 35
is_numpy_data() (in *ida_lib.operations.utils*), 35

K

keypoints_to_homogeneous_and_concatenate() (in module *ida_lib.operations.utils*), 35
keypoints_to_homogeneous_functional() (in module *ida_lib.operations.utils*), 36

M

map_value() (in module *ida_lib.operations.utils*), 36
mask_change_to_01_functional() (in module *ida_lib.operations.utils*), 36
module
 ida_lib.core, 21
 ida_lib.core.pipeline, 13
 ida_lib.core.pipeline_functional, 14
 ida_lib.core.pipeline_geometric_ops, 15
 ida_lib.core.pipeline_local_ops, 18
 ida_lib.core.pipeline_operations, 19
 ida_lib.core.pipeline_pixel_ops, 19
 ida_lib.image_augmentation, 23
 ida_lib.image_augmentation.augment_to_disk, 21
 ida_lib.image_augmentation.data_loader, 22
 ida_lib.operations, 36
 ida_lib.operations.geometry_ops_functional, 23
 ida_lib.operations.pixel_ops_functional, 27
 ida_lib.operations.transforms, 31
 ida_lib.operations.utils, 35
 ida_lib.visualization, 36

N

need_data_info() (*ida_lib.core.pipeline_geometric_ops.HflipPipeline method*), 15
need_data_info() (*ida_lib.core.pipeline_geometric_ops.RandomHflipPipeline pixel_ops*), 19
need_data_info() (*ida_lib.core.pipeline_geometric_ops.RandomRotatePipeline pixel_ops*), 19
need_data_info() (*ida_lib.core.pipeline_geometric_ops.RandomShearPipeline pixel_ops*), 20
need_data_info() (static method), 17

module
 need_data_info() (*ida_lib.core.pipeline_geometric_ops.RandomTranslatePipeline static method*), 17
 need_data_info() (*ida_lib.core.pipeline_geometric_ops.RotatePipeline method*), 16
 need_data_info() (*ida_lib.core.pipeline_geometric_ops.ScalePipeline method*), 16
 need_data_info() (*ida_lib.core.pipeline_geometric_ops.ShearPipeline static method*), 16
 need_data_info() (*ida_lib.core.pipeline_geometric_ops.TranslatePipeline static method*), 16
 need_data_info() (*ida_lib.core.pipeline_geometric_ops.VflipPipeline method*), 15
normalize_image() (in module *ida_lib.operations.pixel_ops_functional*), 29
NormalizePipeline (class in *ida_lib.core.pipeline_pixel_ops*), 20

O

own_affine() (in module *ida_lib.operations.geometry_ops_functional*), 24

P

Pipeline (class in *ida_lib.core.pipeline*), 13
PipelineOperation (class in *ida_lib.core.pipeline_operations*), 19
plot_image_transformation() (in module *ida_lib.visualization*), 36
poisson_noise() (in module *ida_lib.operations.pixel_ops_functional*), 30
PoissonNoisePipeline (class in *ida_lib.core.pipeline_local_ops*), 18
postprocess_data() (in module *ida_lib.core.pipeline_functional*), 14
prepare_data() (in module *ida_lib.operations.geometry_ops_functional*), 24
prepare_data_for_opencv() (in module *ida_lib.operations.pixel_ops_functional*), 30
preprocess_data() (in module *ida_lib.core.pipeline_functional*), 14

R

RandomBrightnessPipeline (class in *ida_lib.core.pipeline_pixel_ops*), 19
RandomContrastPipeline (class in *ida_lib.core.pipeline_pixel_ops*), 19
RandomGammaPipeline (class in *ida_lib.core.pipeline_pixel_ops*), 19
RandomRotatePipeline (class in *ida_lib.core.pipeline_geometric_ops*), 17

RandomScalePipeline (class in *ida_lib.core.pipeline_geometric_ops*), 16
 RandomShearPipeline (class in *ida_lib.core.pipeline_geometric_ops*), 17
 RandomTranslatePipeline (class in *ida_lib.core.pipeline_geometric_ops*), 17
 remove_digits() (in module *ida_lib.operations.utils*), 36
 rotate() (in module *ida_lib.operations.transforms*), 31
 rotate_compose_data() (in module *ida_lib.operations.geometry_ops_functional*), 24
 rotate_coordinates_matrix() (in module *ida_lib.operations.geometry_ops_functional*), 25
 rotate_image() (in module *ida_lib.operations.geometry_ops_functional*), 25
 RotatePipeline (class in *ida_lib.core.pipeline_geometric_ops*), 15
 round_torch() (in module *ida_lib.operations.utils*), 36

S

salt_and_pepper_noise() (in module *ida_lib.operations.pixel_ops_functional*), 30
 SaltAndPepperNoisePipeline (class in *ida_lib.core.pipeline_local_ops*), 18
 save_im() (in module *ida_lib.operations.utils*), 36
 save_item() (*ida_lib.image_augmentation.augment_to_disk*.~~AugmentToDisk~~ method), 21
 scale() (in module *ida_lib.operations.transforms*), 32
 scale_compose_data() (in module *ida_lib.operations.geometry_ops_functional*), 25
 scale_coordinates_matrix() (in module *ida_lib.operations.geometry_ops_functional*), 25
 scale_image() (in module *ida_lib.operations.geometry_ops_functional*), 25
 ScalePipeline (class in *ida_lib.core.pipeline_geometric_ops*), 16
 shear() (in module *ida_lib.operations.transforms*), 31
 shear_compose_data() (in module *ida_lib.operations.geometry_ops_functional*), 26
 shear_coordinates_matrix() (in module *ida_lib.operations.geometry_ops_functional*), 26
 shear_image() (in module *ida_lib.operations.geometry_ops_functional*), 26

ShearPipeline (class in *ida_lib.core.pipeline_geometric_ops*), 16
 spekle_noise() (in module *ida_lib.operations.pixel_ops_functional*), 30
 SpekleNoisePipeline (class in *ida_lib.core.pipeline_local_ops*), 18
 split_operations_by_type() (in module *ida_lib.core.pipeline_functional*), 14
 switch_point_positions() (in module *ida_lib.core.pipeline_functional*), 15
 switch_points() (*ida_lib.core.pipeline_geometric_ops.HflipPipeline* method), 15
 switch_points() (*ida_lib.core.pipeline_geometric_ops.RandomRotatePipeline* static method), 17
 switch_points() (*ida_lib.core.pipeline_geometric_ops.RandomScalePipeline* static method), 16
 switch_points() (*ida_lib.core.pipeline_geometric_ops.RandomShearPipeline* static method), 17
 switch_points() (*ida_lib.core.pipeline_geometric_ops.RandomTranslatePipeline* static method), 17
 switch_points() (*ida_lib.core.pipeline_geometric_ops.RotatePipeline* static method), 16
 switch_points() (*ida_lib.core.pipeline_geometric_ops.ScalePipeline* static method), 16
 switch_points() (*ida_lib.core.pipeline_geometric_ops.ShearPipeline* static method), 16
 switch_points() (*ida_lib.core.pipeline_geometric_ops.TranslatePipeline* static method), 16
 switch_points() (*ida_lib.core.pipeline_geometric_ops.VflipPipeline* static method), 15

T

transform_function() (*ida_lib.core.pipeline_pixel_ops.BrightnessPipeline* method), 19
 transform_function() (*ida_lib.core.pipeline_pixel_ops.ContrastPipeline* method), 19
 transform_function() (*ida_lib.core.pipeline_pixel_ops.DenormalizePipeline* method), 20
 transform_function() (*ida_lib.core.pipeline_pixel_ops.GammaPipeline* method), 20
 transform_function() (*ida_lib.core.pipeline_pixel_ops.NormalizePipeline* static method), 20
 transform_function() (*ida_lib.core.pipeline_pixel_ops.RandomBrightnessPipeline* method), 20
 transform_function() (*ida_lib.core.pipeline_pixel_ops.RandomContrastPipeline* method), 20

method), 19
transform_function()
 (*ida_lib.core.pipeline_pixel_ops.RandomGammaPipeline*
 method), 20
translate() (*in* *module*
 ida.lib.operations.transforms), 32
translate_compose_data() (*in* *module*
 ida.lib.operations.geometry_ops_functional),
 26
translate_coordinates_matrix() (*in module*
 ida.lib.operations.geometry_ops_functional),
 26
translate_image() (*in* *module*
 ida.lib.operations.geometry_ops_functional),
 26
TranslatePipeline (*class* *in*
 ida.lib.core.pipeline_geometric_ops), 16

V

vflip() (*in module ida.lib.operations.transforms*), 31
vflip_compose_data() (*in* *module*
 ida.lib.operations.geometry_ops_functional),
 27
vflip_coordinates_matrix() (*in* *module*
 ida.lib.operations.geometry_ops_functional),
 27
vflip_image() (*in* *module*
 ida.lib.operations.geometry_ops_functional),
 27
VflipPipeline (*class* *in*
 ida.lib.core.pipeline_geometric_ops), 15
visualize() (*in module ida.lib.visualization*), 36