`contrast_brightness_modify(img, alpha, beta)`

Saturated contrast/brightness modification.

$$g\,(i,j) = \boldsymbol{\alpha}.f\,(i,j) + \boldsymbol{\beta}$$

With f(i,j) the original pixel value, **α** the desired contrast, **β** the desired brightness, g(i,j) the adjusted pixel value.

Parameters:

img: ***String like***
Direction/path to input image.

alpha: ***float, [1,2)***
Gain for each pixel intensity.

beta*: float*
Offet for each pixel intensity.

Returns:

new_img*: ndarray*
Output image with modified contrast and brightness.

`automatic_brightness_and_contrast(image,clip_hist_percent=1, inverse_color=False)`

Histogram equalization algorithm. This function clip a percentage of the histogram from the bottom.

Parameters:

image: ***array_like***
Input image.

clip_hist_percent: ***float, Optional***
Clipped percentage of the maximum of grayscale histogram.

inverse_color*: bool, Optional*
Inverse the color of the image.

Returns:

auto_result*: ndarray*
Output image with modified contrast and brightness.

alpha*: float*
Gain for each pixel intensity.

beta*: float*

Offet for each pixel intensity.

`adjust_gamma(image, gamma=1.0)`

`Gamma correction algorithm.`

Parameters:

image: ***array_like***

Input image.

gamma*: float, optional*

gamma coefficient for the algorithm.

Returns:

cv2.LUT(image, table)*: ndarray*

Output image with modified gamma coefficient.

`image_treatment(name_img,inverse_color=False,kernel_morpho=5,open_iter=1,close_iter=1,clear_bder=False)`

`Image process algorithm.`

Parameters:

name_img: ***string like***

Direction to the input image.

inverse_color*: bool, optional*

Inverse the color of the image.

kernel_morpho*: int, optional*

Size of the kernel for morphological operations.

open_iter*: int, optional*

Number of iterations of morphological opening.

close_iter*: int, optional*

Number of iterations of morphological closing.

clear_bder*: bool, optional*

Clear object in contact with border.

Returns:

binary*: ndarray*

Output binarized image.

alpha*: float*

Gain for each pixel intensity.

beta*: float*

Offset for each pixel intensity.

`image_treatment_manuel(name_img, inverse_color=False, kernel_morpho=5, open_iter=1, close_iter=1, clear_bder=False, alpha=1, beta=0)`

Image process algorithm with manual input of contrast and brightness.

Parameters:

name_img: *string like*

Direction to the input image.

inverse_color: *bool, optional*

Inverse the color of the image.

kernel_morpho*: int, optional*

Size of the kernel for morphological operations.

open_iter*: int, optional*

Number of iterations of morphological opening.

close_iter*: int, optional*

Number of iterations of morphological closing.

clear_bder*: bool, optional*

Clear object in contact with border.

alpha*: float, optional*

Gain for each pixel intensity.

beta*: float, optional*

Offset for each pixel intensity.

Returns:

binary*: ndarray*

Output binarized image.

alpha*: float*

Gain for each pixel intensity.

beta*: float*

Offset for each pixel intensity.

`detect_scale_bar(image_path, physical_length, inverse_color=False)`

Detect and return the nm/pixel ratio of the scale bar.

Parameters:

Image_path: ***string like***

Direction/path to the input image.

physical_length: ***float, Optional***

The physical length in nanometer of the scale bar.

inverse_color: ***bool, Optional***

Inverse the color of the image.

Returns:

scale_bar_ratio: ***float***

Ratio of the scale bar in nm/pixel.

`NP_segmentation_local_max(name_img, min_distance, dist_max_threshold=0.4, erode_iter=1, open_iter=0, kernel_size=3)`

Watershed segmentation algorithm using local extremum.

Parameters:

name_img: ***string like***

Direction/path to the input image.

min_distance: ***float, Optional***

Minimum value of the pixel in the distance map to be considered as extremum.

dist_max_threshold: ***bool, Optional***

Percentage of the threshold in the distance map.

erode_iter: ***int, Optional***

Number of iterations of morphological erosion.

open_iter: ***bool, Optional***

Number of iterations of morphological opening.

kernel_size: ***int, Optional***

Size of the kernel for morphological operations.

Returns:

labels_ws: ***ndarray***

Segmented image as a 2D array.

`NP_segmentation_fg_bg(name_img,dist_max_threshold=0.4, erode_iter=1,open_iter=0,kernel_size=3)`

`Watershed segmentation algorithm using true background/foreground extraction.`

Parameters:

name_img: ***string like***
Direction/path to the input image.

dist_max_threshold*: bool, Optional*
Percentage of the threshold in the distance map.

erode_iter*: int, Optional*
Number of iterations of morphological erosion.

open_iter*: bool, Optional*
Number of iterations of morphological opening.

kernel_size*: int, Optional*
Size of the kernel for morphological operations.

Returns:

labels_ws*: ndarray*
Segmented image as a 2D array.

`size_histogram(labels_ws, pixel_to_nm, name_img,bins = 100)`

`Size histogram construction.`

Parameters:

labels_ws: ***ndarray***
Segmented image as a 2D array.

pixel_to_nm*: float*
Pixel/nm ratio.

name_img*: string like*
Direction/path to the input image.

bins*: int, Optional*
Number of intervals of the histogram.

Returns:

radius*: np.array*
Array of size of segmented objects.

`divide_histogram`(*radius, edge_radius*)

Divide the size histogram construction.

Parameters:

radius*: np.array*
Input array of size of segmented objects.

edge_radius*: float*
Value to divide the array.

Returns:

radius1*: np.array*
Array of size < edge_radius of segmented objects.

radius2*: np.array*
Array of size > edge_radius of segmented objects.

`Gaussian_fit` (*radius, bins*)

Gaussian curve fit algorithm.

Parameters:

radius*: np.array*
Input array of size of segmented objects.

bins*: int*
Number of intervals of the histogram.

Returns:

param_optimised*: np.array*
Array of optimized values.

param_covariance_matrix*: ndarray*
Covariance matrix of optimized values.

x_hist*: np.array*
Array of x value of the histogram.

y_hist*: np.array*
Array of y value of the histogram.

`plot_Gaussian_fit(radius, bins)`

Plot the gaussian curve fit on the size histogram.

Parameters:

radius*: np.array*

Input array of size of segmented objects.

bins*: int*

Number of intervals of the histogram.

Returns:

None

`extract_np(i,img, labels_ws,black_bg_color = False)`

Extract an object from the orignal image.

Parameters:

i*: int*

Integer numerating the nanoparticle.

img*: ndarray*

Original image.

labels_ws*: ndarray*

Segmented mask.

black_bg_color*: bool, optional*

Indicate the color of the background.

Returns:

None

`extract_binary_np(i,img, labels_ws,black_bg_color = False)`

Extract an object from the orignal image in form of a binary mask.

Parameters:

i*: int*

Integer numerating the nanoparticle.

img*: ndarray*

Original image.

labels_ws*: ndarray*

Segmented mask.

black_bg_color*: bool, optional*

Indicate the color of the background.

Returns:

calibrated_image: Image

Image of the extracted object.

`testing_image(img,model,target_size=(256,256), color_mode='L')`

Classify an image using a model.

Parameters:

img*: ndarray*

Original image.

model*: keras model*

Model used for classification.

target_size*: (int,int), optional*

Target size of the input image corresponding to the model input.

color_mode*: string, optional*

Color mode of the image ('L' for grayscale, 'RGB' for RGB).

Returns:

result: np.array

Output vector of probability.

test_image: ndarray

Original image with expanded dimension for classification label.

`show_xplique(model,img,label,total_label,alpha,method)`

AI explainable via Xplique. This Xplique function is only compatible for this whole process and notebook "Example of usage". In case of using Xplique for a specific image, please check out the examples of Xplique notebook.

Parameters:

model*: keras model*

Model used for classification.

img*: ndarray*

Original image.

label*: int*

Classified label (usually np.argmax(result)).

total_label*: int*

Total number of classes.

alpha*: float [0,1]*

Intensity of the explication image on original image.

method*: string*

Explanation method (GradientInput, GradCAM, Saliency…).

Returns:

None

```
Classification(img_name,model,total_label,labels_ws,
target_size=(256,256),color_mode='L',black_bg_color = False)
```

Classification plus showing extracted classes. Return a list of extracted mask of each class.

Parameters:

Img_name*: ndarray*

Original image.

model*: keras model*

Model used for classification.

total_label*: int*

Total number of classes.

labels_ws*: ndarray*

Segmented mask.

target_size*: (int,int), optional*

Target size of the input image corresponding to the model input.

color_mode*: string, optional*

Color mode of the image (‘L’ for grayscale, ‘RGB’ for RGB).

black_bg_color*: bool, optional*

Indicate the color of the background (corresponding to the model).

Returns:

labels*: list*

List of extracted labels correspond to different classes.