BFPy

Release July 17, 2022

Nicolas Francois Bader

Jul 17, 2022

CONTENTS:

CHAPTER

ONE

DOCUMENTATION

1.1 BFPy module

class BFPy.**Analysis**(*img: str*, *out_format: Optional[str] = 'png'*)

Bases: `FourierCorrelationAnalysis`

Enables radial intensity distribution analysis. An instanciated object of this class has full access to all methods and attributes, since this class is child to all other classes

Parameters

- **img** (*str*) – Name of the image to be loaded. Name must correspond to name of original csv file
- **out_format** (*Optional[str]*) – Format of saved image (png, jpeg, tif, pdf)

get_radial_intensities(*only_rad_plot: Optional[bool] = True*, *show: Optional[bool] = True*, *save: Optional[bool] = True*) → Tuple[List, List]

Calculates radial intensities and yields the corresponding polar representation

Parameters

- **only_rad_plot** (*Optional[bool]*) – If only the radial intensity profile (black) is to be plotted or a version of the image with highlightes contour (color map) alongside the colored radial intensity profile
- **show** (*Optional[bool]*) – If the plot is to be shown
- **save** (*Optional[bool]*) – If the plot is to be saved

Returns

List of angles with list of corresponding radial intensities

Return type

List, List

get_radial_slice_intensities(*threshold: float*, *only_rad_plot: Optional[bool] = True*, *show: Optional[bool] = True*, *save: Optional[bool] = True*)

Calculates radial slice intensities and yields the corresponding polar representation. Based on radial intensity algorithm.

Parameters

- **threshold** (*float*) – Threshold in percent of total radial line length, at which the slices should start

- **only_rad_plot** (*Optional[bool]*) – If only the radial intensity profile (black) is to be plotted or a version of the image with highlightes contour (color map) alongside the colored radial intensity profile
- **show** (*Optional[bool]*) – If the plot is to be shown
- **save** (*Optional[bool]*) – If the plot is to be saved

Returns
List of angles with list of corresponding radial intensities

Return type
List, List

class BFPy.**CsvToImg**(*csv_file: str*, *csv_reference: str*, *out_format: Optional[str] = 'png'*, *show_init: Optional[bool] = False*)

Bases: `object`

Creates a corrected BFP image of specified format. Not required for main functionalities, just add-on.

Parameters

- **csv_file** (*str*) – Name of the csv file containing the non-corrected BFP image
- **csv_reference** (*str*) – Name of the csv file containing the reference BFP image
- **out_format** (*Optional[str]*) – Format ouf image to be written (png, jpg, pdf, tif)
- **show_init** (*Optional[bool]*) – If the image should be displayed

class BFPy.**FourierCorrelationAnalysis**(*img: str*)

Bases: *Image*

Algorithm to conduct Fourier correlation analysis

Parameters
img (*str*) – Name of the image to be loaded. Name must correspond to name of original csv file

get_center_coords() → Tuple[float, float]

Heart of the FCA algorithm, as it combines all methods and yields the center coordinates of the structure of interest

Returns
Center coordinates of the structure of interest

Return type
int, int

get_correlation_coeff(*img2: ndarray*) → ndarray

Calculates Fourier correlation coefficient map

Parameters
img2 (*np.ndarray*) – Gray scale image

Returns
Fourier correlation coefficient map (power spectrum)

Return type
np.ndarray

get_correlation_coeff_alternative(*img1: ndarray*, *img2: ndarray*) → ndarray

Alternative: Calculates Fourier correlation coefficient map

Parameters

- **img1** (*np.ndarray*) – Gray scale image
- **img2** (*np.ndarray*) – Gray scale image

Returns
Fourier correlation coefficient map (power spectrum)

Return type
np.ndarray

get_max_coords(*img: ndarray*) → list[int, int]

Coordinates of maximum pixel in an image

Parameters
img (*np.ndarray*) – Gray scale image

Returns
Coordinates

Return type
List[int, int]

plot_fca() → None

Visualization of FCA: original image, both flipped version, self correlation and both flip correlations

class BFPy.**Image**(*img: str*)

Bases: object

Toolbox for basic image manipulation. Prefers csv files over image files, meaning: The constructor deduces the name of the original csv file and csv reference, based on specified input image name. If these csv files exist, the constructor loads their data. Otherwise, the image file itself is taken as source of data.

Parameters
img (*str*) – Name of the image to be loaded. Name must correspond to name of original csv file

get_contours(*binary: ndarray*, *show: Optional[bool] = True*) → Tuple[ndarray, float]

Contour detection algorithm

Parameters

- **binary** (*np.ndarray*) – The image as binary version
- **show** (*Optional[bool]*) – If image with highlighted identified contours should be displayed

Returns
Contours and hierarchies

Return type
np.ndarray, float

get_coords_from_contours(*contour_list: ndarray*) → Tuple[List[int], List[int]]

Extracts x and y coordinates from provided list of contours (e.g. return value of method 'get_longest_contour')

Parameters
contour_list (*List or np.ndarray*) – Previously identified contours

Returns
List of x coordinates and list of corresponding y coordinates

Return type
List, List

get_flipped(*img: ndarray*) → Tuple[ndarray, ndarray]

Flips image over horizontal and vertical

Parameters
img (*np.ndarray*) – The image

Returns
Version of image flipped over x-axis and image flipped over y-axis

Return type
np.ndarray, np.ndarray

get_fourier_transform(*img: ndarray*) → ndarray

Centered Fourier transformation of image

Parameters
img (*np.ndarray*) – The image

Returns
Centered Fourier transform

Return type
np.ndarray

get_inverse_fourier_transform(*ft_img: ndarray*) → ndarray

Inverse Fourier transformation

Parameters
ft_img (*np.ndarray*) – The Fourier transform of an image

Returns
Inverse Fourier transform, a real space image

Return type
np.ndarray

get_longest_contour(*contours: ndarray*) → List[ndarray]

Returns longest contour out of list of multiple contours (e.g. return value of method 'get_contours'). Longest contour corresponds to structure of interest usually, shorter contours to noise/…

Parameters
contours (*np.ndarray*) – Previously identified contours

Returns
List containing longest contours as only element

Return type
List[np.ndarray]

get_super_res(*img_gray: ndarray*, *model: Optional[str] = 'EDSR_x3.pb'*, *blur: Optional[bool] = False*, *show: Optional[bool] = False*, *save: Optional[bool] = True*) → Tuple[ndarray, str]

Upscaling of grayscale image to superresolution image, using the specified machine learning model. This function is based on https://towardsdatascience.com/deep-learning-based-super-resolution-with-opencv-4fd736678066. Refer to this website for further information.

Parameters

- **model** (*str*) – Machine learning model to be used
- **blur** (*Optional[bool]*) – If resulting image should be blurred using a 3x3 Gaussian kernel

- **show** (*Optional[bool]*) – If resulting he image should be displayed
- **save** (*Optional[bool]*) – If resulting image should be saved

Returns
Upscaled superresolution 8-bit image and filename of saved image

Return type
np.ndarray, str

make_false_color_img(*img_gray: ndarray*, *img_name: str*) → None

Creates false color image, as pdf and as same format of input image

Parameters

- **img_gray** (*np.ndarray*) – Gray scale image
- **img_name** (*str*) – Name of image. Used to create name for output image

mask_background(*img_gray: ndarray*, *lower_thres: Optional[int] = 12*, *upper_thres: Optional[int] = 242*, *pad: Optional[float] = 0.1*, *show: Optional[bool] = False*, *save: Optional[bool] = True*, *show_hist: Optional[bool] = False*) → Tuple[ndarray, str]

Takes an input gray image and masks the background, returning the masked image. This is done in two steps: 1. All pixel values smaller than the lower and larger than the upper threshold are set to zero, using a histogram. This step increases the chance of the contour of the BFP being identified correctly. 2. Using the identified BFP, this part of the image is cropped and put atop of a black canvas (all pixel values equal zero). The space around the BFP crop out and the border of the black canvas are set via the pad paramater. The combination of both steps ensures a uniform black background, up to a certain degeree indepednent of the S/N ratio. Nonetheless, pixel contained inside the BFP might be set to zero, too.

Parameters

- **img_gray** (*np.ndarray*) – gray scale 8-bit image
- **lower_thres** (*Optional[int]*) – Specifies the lower threshold
- **upper_thres** (*Optional[int]*) – Specifies the upper threshold
- **pad** (*Optional[float]*) – In percent relative to longest main axis through BFP crop put. The pad is added on both sides of the crop out, setting the size of the resulting square black canvas and function return
- **show** (*Optional[bool]*) – If the image should be displayed
- **save** (*Optional[bool]*) – If the image should be saved
- **show_hist** (*Optional[bool]*) – If the histogram used in the first step as well as the thresholds should be displayed. Via user prompt, the thresholds can be set manually then and the thresholds set during the function call will be overriden

Returns
Masked image (= BFP crop out ontop of black canvas, 8-bit square image) and filename of saved image

Return type
np.ndarray, str

normalize_for_cbar(*img: ndarray*) → ndarray

Normalizes an image to pixel values [0, 1]

Parameters
img (*np.ndarray*) – The image

Returns
Normalized version of image

Return type
np.ndarray

`rotate_180_deg`(*img: ndarray*) → ndarray

Rotates image by 180deg

Parameters
img (*np.ndarray*) – The image

Returns
Rotated image

Return type
np.ndarray

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX