big-fish

Release 0.6.2

Arthur Imbert

Jul 06, 2023

PREPROCESSING

CHAPTER

ONE

GETTING STARTED

To avoid dependency conflicts, we recommend the the use of a dedicated virtual or conda environment. In a terminal run the command:

```
$ conda create -n bigfish_env python=3.6
$ source activate bigfish_env
```

We recommend two options to then install Big-FISH in your virtual environment.

1.1 Download the package from PyPi

Use the package manager pip to install Big-FISH. In a terminal run the command:

```
$ pip install big-fish
```

1.2 Clone package from Github

Clone the project's Github repository and install it manually with the following commands:

```
$ git clone git@github.com:fish-quant/big-fish.git
$ cd big-fish
$ pip install .
```

CHAPTER

TWO

EXAMPLES

Several examples are available as Jupyter notebooks:

1. Read and write images.
2. Normalize and filter images.
3. Project in two dimensions.
4. Segment nuclei and cells.
5. Detect spots.
6. Extract cell level results.
7. Analyze coordinates.

You can also run these example online with mybinder. The remote server can take a bit of time to start.

CHAPTER

THREE

API REFERENCE

3.1 I/O operations

Functions used to read data from various sources and store them in a numpy array.

3.1.1 Read files

Read image, video and numerical data as a numpy array:

- *bigfish.stack.read_image()*
- *bigfish.stack.read_dv()*
- *bigfish.stack.read_array()*

Read a zipped archive of files as a dictionary-like object:

- *bigfish.stack.read_uncompressed()*
- *bigfish.stack.read_cell_extracted()*

Read CSV file:

- *bigfish.stack.read_array_from_csv()*
- *bigfish.stack.read_dataframe_from_csv()*

bigfish.stack.**read_image**(*path*, *sanity_check=False*)

Read an image with `png`, `jpg`, `jpeg`, `tif` or `tiff` extension.

Parameters

path
[str] Path of the image to read.

sanity_check
[bool] Check if the array returned fits with bigfish pipeline.

Returns

image
[ndarray, np.uint or np.int] Image read.

bigfish.stack.**read_dv**(*path*, *sanity_check=False*)

Read a video file with `dv` extension.

Parameters

path
[str] Path of the file to read.

sanity_check
[bool] Check if the array returned fits with bigfish pipeline.

Returns

video
[ndarray] Video read.

bigfish.stack.**read_array**(*path*)

Read a numpy array with `npy` extension.

Parameters

path
[str] Path of the array to read.

Returns

array
[ndarray] Array read.

bigfish.stack.**read_uncompressed**(*path*, *verbose=False*)

Read a NpzFile object with `npz` extension.

Parameters

path
[str] Path of the file to read.

verbose
[bool] Return names of the different objects.

Returns

data
[NpzFile object] NpzFile read.

bigfish.stack.**read_cell_extracted**(*path*, *verbose=False*)

Read a NpzFile object with `npz` extension, previously written with *bigfish.stack.save_cell_extracted()*.

Parameters

path
[str] Path of the file to read.

verbose
[bool] Return names of the different objects.

Returns

cell_results
[Dict] Dictionary including information about the cell (image, masks, coordinates arrays). Minimal information are:

- *cell_id*: Unique id of the cell.
- *bbox*: bounding box coordinates with the order (*min_y*, *min_x*, *max_y*, *max_x*).
- *cell_coord*: boundary coordinates of the cell.
- *cell_mask*: mask of the cell.

bigfish.stack.**read_array_from_csv**(*path*, *dtype=None*, *delimiter=';'*, *encoding='utf-8'*, *skiprows=0*)

Read a numpy array saved in a csv file.

Parameters

path
[str] Path of the csv file to read.

dtype
[type, optional] Expected dtype to cast the array.

delimiter
[str, default=";"] Delimiter used to separate columns.

encoding
[str, default="utf-8"] Encoding to use.

skiprows
[int, default=0] Skip the first *skiprows* lines of the file. Useful to skip the first rows of a csv with header.

Returns

array
[ndarray] Array read.

bigfish.stack.**read_dataframe_from_csv**(*path*, *delimiter=';'*, *encoding='utf-8'*)

Read a numpy array or a pandas object saved in a csv file.

Parameters

path
[str] Path of the csv file to read.

delimiter
[str] Delimiter used to separate columns.

encoding
[str] Encoding to use.

Returns

df
[pd.DataFrame] Pandas object read.

3.1.2 Write files

Save numpy array:

- *bigfish.stack.save_image()*
- *bigfish.stack.save_array()*

Save cell-level results in a zipped archive of files:

- *bigfish.stack.save_cell_extracted()*

Save tabular data in a CSV file:

- *bigfish.stack.save_data_to_csv()*

bigfish.stack.**save_image**(*image*, *path*, *extension='tif'*)

Save an image.

The input image should have between 2 and 5 dimensions, with boolean, (unsigned) integer, or float.

The dimensions should be in the following order: (round, channel, z, y, x).

Parameters

image
[np.ndarray] Image to save.

path
[str] Path of the saved image.

extension
[str] Default extension to save the image (among `png`, `jpg`, `jpeg`, `tif` or `tiff`).

Notes

- If the image has more than 2 dimensions, `tif` and `tiff` extensions are required (`png` extension does not handle 3-d images other than (M, N, 3) or (M, N, 4) shapes).
- A 2-d boolean image can be saved in `png`, `jpg` or `jpeg` (cast in np.uint8).
- A multidimensional boolean image should be saved with *bigfish.stack.save_array()* or as a boolean images with `tif`/ `tiff` extension.

bigfish.stack.**save_array**(*array*, *path*)

Save an array in a `npy` extension file.

The input array should have between 2 and 5 dimensions, with boolean, (unsigned) integer, or float.

Parameters

array
[np.ndarray] Array to save.

path
[str] Path of the saved array.

bigfish.stack.**save_cell_extracted**(*cell_results*, *path*)

Save cell-level results from `bigfish.stack.extract_cell()` in a NpzFile object with `npz` extension.

Parameters

cell_results
[Dict] Dictionary including information about the cell (image, masks, coordinates arrays). Minimal information are:

- *cell_id*: Unique id of the cell.
- *bbox*: bounding box coordinates with the order (*min_y*, *min_x*, *max_y*, *max_x*).
- *cell_coord*: boundary coordinates of the cell.
- *cell_mask*: mask of the cell.

path
[str] Path of the saved array.

bigfish.stack.**save_data_to_csv**(*data*, *path*, *delimiter=';'*)

Save a numpy array or a pandas object into a `csv` file.

The input should be a pandas object (*Series* or *DataFrame*) or a numpy array with 2 dimensions and (unsigned) integer or float.

Parameters

data
[np.ndarray, pd.Series or pd.DataFrame] Data to save.

path
[str] Path of the saved `csv` file.

delimiter
[str] Delimiter used to separate columns.

3.2 Image preparation

Functions used to normalize, format, cast, project or filter images.

3.2.1 Normalize images

Rescale or contrast pixel intensity:

- *bigfish.stack.rescale()*
- *bigfish.stack.compute_image_standardization()*

bigfish.stack.**rescale**(*tensor*, *channel_to_stretch=None*, *stretching_percentile=99.9*)

Rescale tensor values up to its dtype range (unsigned/signed integers) or between 0 and 1 (float).

Each round and each channel is rescaled independently. Tensor has between 2 to 5 dimensions, in the following order: (round, channel, z, y, x).

By default, we rescale the tensor intensity range to its dtype range (or between 0 and 1 for float tensor). We can improve the contrast by stretching a smaller range of pixel intensity: between the minimum value of a channel and percentile value of the channel (cf. `stretching_percentile`).

To be consistent with skimage, 64-bit (unsigned) integer images are not supported.

Parameters

tensor
[np.ndarray] Tensor to rescale.

channel_to_stretch
[int, List[int] or Tuple[int]] Channel to stretch. If None, minimum and maximum of each channel are used as the intensity range to rescale.

stretching_percentile
[float or int] Percentile to determine the maximum intensity value used to rescale the image. If 1, the maximum pixel intensity is used to rescale the image.

Returns

tensor
[np.ndarray] Tensor rescaled.

bigfish.stack.**compute_image_standardization**(*image*)

Normalize image by computing its z score.

Parameters

image
[np.ndarray] Image to normalize with shape (y, x).

Returns

normalized_image
[np.ndarray] Normalized image with shape (y, x).

3.2.2 Format images

Resize and pad images:

- *bigfish.stack.resize_image()*
- *bigfish.stack.get_marge_padding()*

bigfish.stack.**resize_image**(*image*, *output_shape*, *method='bilinear'*)

Resize an image with bilinear interpolation or nearest neighbor method.

Parameters

image
[np.ndarray] Image to resize.

output_shape
[Tuple[int]] Shape of the resized image.

method
[str] Interpolation method to use.

Returns

image_resized
[np.ndarray] Resized image.

bigfish.stack.**get_marge_padding**(*height*, *width*, *x*)

Pad image to make its shape a multiple of *x*.

Parameters

height
[int] Original height of the image.

width
[int] Original width of the image.

x
[int] Padded image have a *height* and *width* multiple of *x*.

Returns

marge_padding
[List[List]] List of lists with the format [[*marge_height_t*, *marge_height_b*], [*marge_width_l*, *marge_width_r*]].

3.2.3 Cast images

Cast images to a specified dtype (with respect to the image range of values):

- *bigfish.stack.cast_img_uint8()*
- *bigfish.stack.cast_img_uint16()*
- *bigfish.stack.cast_img_float32()*
- *bigfish.stack.cast_img_float64()*

bigfish.stack.**cast_img_uint8**(*tensor*)

Cast the image in np.uint8 and scale values between 0 and 255.

Negative values are not allowed as the skimage method `img_as_ubyte` would clip them to 0. Positives values are scaled between 0 and 255, excepted if they fit directly in 8 bit (in this case values are not modified).

Parameters

tensor
[np.ndarray] Image to cast.

Returns

tensor
[np.ndarray, np.uint8] Image cast.

bigfish.stack.**cast_img_uint16**(*tensor*)

Cast the data in np.uint16.

Negative values are not allowed as the skimage method `img_as_uint` would clip them to 0. Positives values are scaled between 0 and 65535, excepted if they fit directly in 16 bit (in this case values are not modified).

Parameters

tensor
[np.ndarray] Image to cast.

Returns

tensor
[np.ndarray, np.uint16] Image cast.

bigfish.stack.**cast_img_float32**(*tensor*)

Cast the data in np.float32.

If the input data is in (unsigned) integer, the values are scaled between 0 and 1. When converting from a np.float dtype, values are not modified.

Parameters

tensor
[np.ndarray] Image to cast.

Returns

tensor
[np.ndarray, np.float32] image cast.

bigfish.stack.**cast_img_float64**(*tensor*)

Cast the data in np.float64.

If the input data is in (unsigned) integer, the values are scaled between 0 and 1. When converting from a np.float dtype, values are not modified.

Parameters

tensor
[np.ndarray] Tensor to cast.

Returns

tensor
[np.ndarray, np.float64] Tensor cast.

3.2.4 Filter images

Apply filtering transformations:

- *bigfish.stack.mean_filter()*
- *bigfish.stack.median_filter()*
- *bigfish.stack.gaussian_filter()*
- *bigfish.stack.maximum_filter()*
- *bigfish.stack.minimum_filter()*
- *bigfish.stack.dilation_filter()*
- *bigfish.stack.erosion_filter()*

Use Laplacian of Gaussian (LoG) filter to enhance peak signals and denoise the rest of the image:

- *bigfish.stack.log_filter()*

Use blurring filters with large kernel to estimate and remove background signal:

- *bigfish.stack.remove_background_mean()*
- *bigfish.stack.remove_background_gaussian()*

bigfish.stack.**mean_filter**(*image*, *kernel_shape*, *kernel_size*)

Apply a mean filter to a 2-d through convolution filter.

Parameters

image
[np.ndarray, np.uint or np.float] Image with shape (y, x).

kernel_shape
[str] Shape of the kernel used to compute the filter (*diamond*, *disk*, *rectangle* or *square*).

kernel_size
[int, Tuple(int) or List(int)] The size of the kernel. For the rectangle we expect two integers (*height*, *width*).

Returns

image_filtered
[np.ndarray, np.uint] Filtered 2-d image with shape (y, x).

bigfish.stack.**median_filter**(*image*, *kernel_shape*, *kernel_size*)

Apply a median filter to a 2-d image.

Parameters

image
[np.ndarray, np.uint] Image with shape (y, x).

kernel_shape
[str] Shape of the kernel used to compute the filter (*diamond*, *disk*, *rectangle* or *square*).

kernel_size
[int, Tuple(int) or List(int)] The size of the kernel. For the rectangle we expect two integers (*height*, *width*).

Returns

image_filtered
[np.ndarray, np.uint] Filtered 2-d image with shape (y, x).

bigfish.stack.**gaussian_filter**(*image*, *sigma*, *allow_negative=False*)

Apply a Gaussian filter to a 2-d or 3-d image.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

sigma
[int, float, Tuple(float, int) or List(float, int)] Standard deviation used for the gaussian kernel (one for each dimension). If it's a scalar, the same standard deviation is applied to every dimensions.

allow_negative
[bool] Allow negative values after the filtering or clip them to 0. Not compatible with unsigned integer images.

Returns

image_filtered
[np.ndarray] Filtered image.

bigfish.stack.**maximum_filter**(*image*, *kernel_shape*, *kernel_size*)

Apply a maximum filter to a 2-d image.

Parameters

image
[np.ndarray, np.uint] Image with shape (y, x).

kernel_shape
[str] Shape of the kernel used to compute the filter (*diamond*, *disk*, *rectangle* or *square*).

kernel_size
[int, Tuple(int) or List(int)] The size of the kernel. For the rectangle we expect two integers (*height*, *width*).

Returns

image_filtered
[np.ndarray, np.uint] Filtered 2-d image with shape (y, x).

bigfish.stack.**minimum_filter**(*image*, *kernel_shape*, *kernel_size*)

Apply a minimum filter to a 2-d image.

Parameters

image
[np.ndarray, np.uint] Image with shape (y, x).

kernel_shape
[str] Shape of the kernel used to compute the filter (*diamond*, *disk*, *rectangle* or *square*).

kernel_size
[int, Tuple(int) or List(int)] The size of the kernel. For the rectangle we expect two integers (*height*, *width*).

Returns

image_filtered
[np.ndarray, np.uint] Filtered 2-d image with shape (y, x).

bigfish.stack.**dilation_filter**(*image*, *kernel_shape=None*, *kernel_size=None*)

Apply a dilation to a 2-d image.

Parameters

image
[np.ndarray] Image with shape (y, x).

kernel_shape
[str] Shape of the kernel used to compute the filter (*diamond*, *disk*, *rectangle* or *square*). If None, use cross-shaped structuring element (`connectivity=1`).

kernel_size
[int, Tuple(int) or List(int)] The size of the kernel. For the rectangle we expect two integers (*height*, *width*). If None, use cross-shaped structuring element (`connectivity=1`).

Returns

image_filtered
[np.ndarray] Filtered 2-d image with shape (y, x).

bigfish.stack.**erosion_filter**(*image*, *kernel_shape=None*, *kernel_size=None*)

Apply an erosion to a 2-d image.

Parameters

image
[np.ndarray] Image with shape (y, x).

kernel_shape
[str] Shape of the kernel used to compute the filter (*diamond*, *disk*, *rectangle* or *square*). If None, use cross-shaped structuring element (`connectivity=1`).

kernel_size
[int, Tuple(int) or List(int)] The size of the kernel. For the rectangle we expect two integers (*height*, *width*). If None, use cross-shaped structuring element (`connectivity=1`).

Returns

image_filtered
[np.ndarray] Filtered 2-d image with shape (y, x).

bigfish.stack.**log_filter**(*image*, *sigma*)

Apply a Laplacian of Gaussian filter to a 2-d or 3-d image.

The function returns the inverse of the filtered image such that the pixels with the highest intensity from the original (smoothed) image have positive values. Those with a low intensity returning a negative value are clipped to zero.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

sigma
[int, float, Tuple(float, int) or List(float, int)] Standard deviation used for the gaussian kernel (one for each dimension). If it's a scalar, the same standard deviation is applied to every dimensions.

Returns

image_filtered
[np.ndarray] Filtered image.

bigfish.stack.**remove_background_mean**(*image*, *kernel_shape='disk'*, *kernel_size=200*)

Remove background noise from a 2-d image, subtracting a mean filtering.

Parameters

image
[np.ndarray, np.uint] Image to process with shape (y, x).

kernel_shape
[str] Shape of the kernel used to compute the filter (*diamond*, *disk*, *rectangle* or *square*).

kernel_size
[int, Tuple(int) or List(int)] The size of the kernel. For the rectangle we expect two integers (*height*, *width*).

Returns

image_without_back
[np.ndarray, np.uint] Image processed.

bigfish.stack.**remove_background_gaussian**(*image*, *sigma*)

Remove background noise from a 2-d or 3-d image, subtracting a gaussian filtering.

Parameters

image
[np.ndarray] Image to process with shape (z, y, x) or (y, x).

sigma
[int, float, Tuple(float, int) or List(float, int)] Standard deviation used for the gaussian kernel (one for each dimension). If it's a scalar, the same standard deviation is applied to every dimensions.

Returns

image_no_background
[np.ndarray] Image processed with shape (z, y, x) or (y, x).

3.2.5 Project images in 2D

Build a 2D projection by computing the maximum, mean or median values:

- *bigfish.stack.maximum_projection()*
- *bigfish.stack.mean_projection()*
- *bigfish.stack.median_projection()*

bigfish.stack.**maximum_projection**(*image*)

Project the z-dimension of an image, keeping the maximum intensity of each yx pixel.

Parameters

image
[np.ndarray] A 3-d image with shape (z, y, x).

Returns

projected_image
[np.ndarray] A 2-d image with shape (y, x).

bigfish.stack.**mean_projection**(*image*, *return_float=False*)

Project the z-dimension of a image, computing the mean intensity of each yx pixel.

Parameters

image
[np.ndarray] A 3-d tensor with shape (z, y, x).

return_float
[bool, default=False] Return a (potentially more accurate) float array.

Returns

projected_image
[np.ndarray] A 2-d image with shape (y, x).

bigfish.stack.**median_projection**(*image*)

Project the z-dimension of a image, computing the median intensity of each yx pixel.

Parameters

image
[np.ndarray] A 3-d image with shape (z, y, x).

Returns

projected_image
[np.ndarray] A 2-d image with shape (y, x).

3.2.6 Clean out-of-focus pixels

Compute a pixel-wise focus score:

- *bigfish.stack.compute_focus()*

Remove the out-of-focus z-slices of a 3D image:

- *bigfish.stack.in_focus_selection()*
- *bigfish.stack.get_in_focus_indices()*

Build a 2D projection by removing the out-of-focus z-slices/pixels:

- *bigfish.stack.focus_projection()*

bigfish.stack.**compute_focus**(*image*, *neighborhood_size=31*)

Helmli and Scherer's mean method is used as a focus metric.

For each pixel yx in a 2-d image, we compute the ratio:

$$R(y,x) = \begin{cases} \frac{I(y,x)}{\mu(y,x)} & \text{if} \quad I(y,x) \geq \mu(y,x) \\ \frac{\mu(y,x)}{I(y,x)} & \text{otherwise} \end{cases}$$

with $I(y,x)$ the intensity of the pixel yx and $\mu(y,x)$ the mean intensity of the pixels in its neighborhood.

For a 3-d image, we compute this metric for each z surface.

Parameters

image
[np.ndarray] A 2-d or 3-d image with shape (y, x) or (z, y, x).

neighborhood_size
[int or tuple or list, default=31] The size of the square used to define the neighborhood of each pixel. An odd value is preferred. To define a rectangular neighborhood, a tuple or a list with two elements (height, width) can be provided.

Returns

focus
[np.ndarray, np.float64] A 2-d or 3-d tensor with the R(y, x) computed for each pixel of the original image.

bigfish.stack.**in_focus_selection**(*image*, *focus*, *proportion*)

Select and keep the 2-d slices with the highest level of focus.

Helmli and Scherer's mean method is used as a focus metric.

Parameters

image
[np.ndarray] A 3-d tensor with shape (z, y, x).

focus
[np.ndarray, np.float64] A 3-d tensor with a focus metric computed for each pixel of the original image. See *bigfish.stack.compute_focus()*.

proportion
[float or int] Proportion of z-slices to keep (float between 0 and 1) or number of z-slices to keep (positive integer).

Returns

in_focus_image
[np.ndarray] A 3-d tensor with shape (z_in_focus, y, x), with out-of-focus z-slice removed.

bigfish.stack.**get_in_focus_indices**(*focus*, *proportion*)

Select the best in-focus z-slices.

Helmli and Scherer's mean method is used as a focus metric.

Parameters

focus
[np.ndarray, np.float] A 3-d tensor with a focus metric computed for each pixel of the original image. See *bigfish.stack.compute_focus()*.

proportion
[float or int] Proportion of z-slices to keep (float between 0 and 1) or number of z-slices to keep (positive integer).

Returns

indices_to_keep
[List[int]] Indices of slices with the best focus score.

bigfish.stack.**focus_projection**(*image*, *proportion=0.75*, *neighborhood_size=7*, *method='median'*)

Project the z-dimension of an image.

Inspired from Samacoits Aubin's thesis (part 5.3, strategy 5). Compare to the original algorithm we use the same focus measures to select the in-focus z-slices and project our image.

1. Compute a focus score for each pixel yx with a fixed neighborhood size.
2. We keep a proportion of z-slices with the highest average focus score.
3. Keep the median/maximum pixel intensity among the top 5 z-slices (at most) with the highest focus score.

Parameters

image
[np.ndarray] A 3-d image with shape (z, y, x).

proportion
[float or int, default=0.75] Proportion of z-slices to keep (float between 0 and 1) or number of z-slices to keep (positive integer).

neighborhood_size
[int or tuple or list, default=7] The size of the square used to define the neighborhood of each pixel. An odd value is preferred. To define a rectangular neighborhood, a tuple or a list with two elements (height, width) can be provided.

method
[{*median*, *max*}, default=`median`] Projection method applied on the selected pixel values.

Returns

projected_image
[np.ndarray] A 2-d image with shape (y, x).

3.3 Augmentation

Functions used to increase and diversify a dataset by duplicating and transforming images. Available transformations are:

- Identity
- Transpose
- Inverse transpose
- Horizontal flip
- Vertical flip
- 90° rotation
- 180° rotation
- 270° rotation

Apply a random transformation on a 2D image:

- *bigfish.stack.augment_2d()*
- *bigfish.stack.augment_2d_function()*

Apply all the possible transformations on a 2D image:

- *bigfish.stack.augment_8_times()*
- *bigfish.stack.augment_8_times_reversed()*

bigfish.stack.**augment_2d**(*image*)

Augment an image applying a random operation.

Parameters

image
[np.ndarray] Image to augment with shape (y, x, channels) or (y, x, channels).

Returns

image_augmented
[np.ndarray] Image augmented with shape (y, x, channels).

bigfish.stack.**augment_2d_function**(*identity=False*)

Choose a random operation to augment a 2-d image.

Parameters

identity
[bool] Return identity function instead of a random transformation.

Returns

random_operation
[callable] Function to transform a 2-d image.

bigfish.stack.**augment_8_times**(*image*)

Apply every transformation to a 2-d image.

Parameters

image
[np.ndarray] Image to augment with shape (y, x, channels).

Returns

images_augmented
[List[np.ndarray]] List of images augmented with shape (y, x, channels).

bigfish.stack.**augment_8_times_reversed**(*images_augmented*)

Apply every transformation back to return the original 2-d image.

Parameters

images_augmented
[List[np.ndarray]] List of images augmented with shape (y, x, channels).

Returns

images_original
[List[np.ndarray]] List of original images with shape (y, x, channels).

3.4 Automated spot detection

Functions used to detect spots in a 2D or 3D image. Detection is performed in three steps:

1. Image is denoised and spots are enhanced by using a Laplacian of Gaussian (LoG) filter.
2. Peaks are detected in the filtered image with a local maximum detection algorithm.
3. An intensity threshold is applied to discriminate actual spots from noisy background.

3.4.1 Detect spots

The main function for spot detection is:

- *bigfish.detection.detect_spots()*

It is also possible to perform the main steps of the spot detection separately:

- *bigfish.detection.local_maximum_detection()*
- *bigfish.detection.spots_thresholding()*

See an example of application here.

bigfish.detection.**detect_spots**(*images*, *threshold=None*, *remove_duplicate=True*, *return_threshold=False*, *voxel_size=None*, *spot_radius=None*, *log_kernel_size=None*, *minimum_distance=None*)

Apply LoG filter followed by a Local Maximum algorithm to detect spots in a 2-d or 3-d image.

1. We smooth the image with a LoG filter.
2. We apply a multidimensional maximum filter.
3. A pixel which has the same value in the original and filtered images is a local maximum.
4. We remove local peaks under a threshold.
5. We keep only one pixel coordinate per detected spot.

Parameters

images
[List[np.ndarray] or np.ndarray] Image (or list of images) with shape (z, y, x) or (y, x). If several images are provided, the same threshold is applied.

threshold
[int, float or None] A threshold to discriminate relevant spots from noisy blobs. If None, optimal threshold is selected automatically. If several images are provided, one optimal threshold is selected for all the images.

remove_duplicate
[bool] Remove potential duplicate coordinates for the same spots. Slow the running.

return_threshold
[bool] Return the threshold used to detect spots.

voxel_size
[int, float, Tuple(int, float), List(int, float) or None] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

spot_radius
[int, float, Tuple(int, float), List(int, float) or None] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

log_kernel_size
[int, float, Tuple(int, float), List(int, float) or None] Size of the LoG kernel. It equals the standard deviation (in pixels) used for the gaussian kernel (one for each dimension). One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same standard deviation is applied to every dimensions. If None, we estimate it with the voxel size and spot radius.

minimum_distance
[int, float, Tuple(int, float), List(int, float) or None] Minimum distance (in pixels) between two spots we want to be able to detect separately. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same distance is applied to every dimensions. If None, we estimate it with the voxel size and spot radius.

Returns

spots
[List[np.ndarray] or np.ndarray, np.int64] Coordinates (or list of coordinates) of the spots with shape (nb_spots, 3) or (nb_spots, 2), for 3-d or 2-d images respectively.

threshold
[int or float] Threshold used to discriminate spots from noisy blobs.

bigfish.detection.**local_maximum_detection**(*image*, *min_distance*)

Compute a mask to keep only local maximum, in 2-d and 3-d.

1. We apply a multidimensional maximum filter.
2. A pixel which has the same value in the original and filtered images is a local maximum.

Several connected pixels can have the same value. In such a case, the local maximum is not unique.

In order to make the detection robust, it should be applied to a filtered image (using *bigfish.stack.log_filter()* for example).

Parameters

image
[np.ndarray] Image to process with shape (z, y, x) or (y, x).

min_distance
[int, float, Tuple(int, float), List(int, float)] Minimum distance (in pixels) between two spots we want to be able to detect separately. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same distance is applied to every dimensions.

Returns

mask
[np.ndarray, bool] Mask with shape (z, y, x) or (y, x) indicating the local peaks.

bigfish.detection.**spots_thresholding**(*image*, *mask_local_max*, *threshold*, *remove_duplicate=True*)

Filter detected spots and get coordinates of the remaining spots.

In order to make the thresholding robust, it should be applied to a filtered image (using *bigfish.stack.log_filter()* for example). If the local maximum is not unique (it can happen if connected pixels have the same value), a connected component algorithm is applied to keep only one coordinate per spot.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

mask_local_max
[np.ndarray, bool] Mask with shape (z, y, x) or (y, x) indicating the local peaks.

threshold
[float, int or None] A threshold to discriminate relevant spots from noisy blobs. If None, detection is aborted with a warning.

remove_duplicate
[bool] Remove potential duplicate coordinates for the same spots. Slow the running.

Returns

spots
[np.ndarray, np.int64] Coordinate of the local peaks with shape (nb_peaks, 3) or (nb_peaks, 2) for 3-d or 2-d images respectively.

mask
[np.ndarray, bool] Mask with shape (z, y, x) or (y, x) indicating the spots.

3.4.2 Find a threshold (automatically)

The need to set an appropriate threshold for each image is a real bottleneck that limits the possibility to scale a spot detection. Our method includes a heuristic function to to automatically set this threshold:

- *bigfish.detection.automated_threshold_setting()*
- *bigfish.detection.get_breaking_point()*
- *bigfish.detection.get_elbow_values()*

bigfish.detection.**automated_threshold_setting**(*image*, *mask_local_max*)

Automatically set the optimal threshold to detect spots.

In order to make the thresholding robust, it should be applied to a filtered image (using *bigfish.stack.log_filter()* for example). The optimal threshold is selected based on the spots distribution. The latter should have an elbow curve discriminating a fast decreasing stage from a more stable one (a plateau).

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

mask_local_max
[np.ndarray, bool] Mask with shape (z, y, x) or (y, x) indicating the local peaks.

Returns

optimal_threshold
[int] Optimal threshold to discriminate spots from noisy blobs.

bigfish.detection.**get_breaking_point**(*x*, *y*)

Select the x-axis value where a L-curve has a kink.

Assuming a L-curve from A to B, the 'breaking_point' is the more distant point to the segment [A, B].

Parameters

x
[np.array] X-axis values.

y
[np.array] Y-axis values.

Returns

breaking_point
[float] X-axis value at the kink location.

x
[np.array] X-axis values.

y
[np.array] Y-axis values.

bigfish.detection.**get_elbow_values**(*images*, *voxel_size=None*, *spot_radius=None*, *log_kernel_size=None*, *minimum_distance=None*)

Get values to plot the elbow curve used to automatically set the threshold to detect spots.

Parameters

images
[List[np.ndarray] or np.ndarray] Image (or list of images) with shape (z, y, x) or (y, x). If several images are provided, the same threshold is applied.

voxel_size
[int, float, Tuple(int, float), List(int, float) or None] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

spot_radius
[int, float, Tuple(int, float), List(int, float) or None] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

log_kernel_size
[int, float, Tuple(int, float), List(int, float) or None] Size of the LoG kernel. It equals the standard deviation (in pixels) used for the gaussian kernel (one for each dimension). One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same standard deviation is applied to every dimensions. If None, we estimate it with the voxel size and spot radius.

minimum_distance
[int, float, Tuple(int, float), List(int, float) or None] Minimum distance (in pixels) between

two spots we want to be able to detect separately. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same distance is applied to every dimensions. If None, we estimate it with the voxel size and spot radius.

Returns

thresholds
[np.ndarray, np.float64] Candidate threshold values.

count_spots
[np.ndarray, np.float64] Spots count (log scale).

threshold
[float or None] Threshold automatically set.

3.4.3 Compute signal-to-noise ratio

Compute a signal-to-noise ratio (SNR) for the image, based on the detected spots:

bigfish.detection.**compute_snr_spots**(*image*, *spots*, *voxel_size*, *spot_radius*)

Compute signal-to-noise ratio (SNR) based on spot coordinates.

$$\text{SNR} = \frac{\max(\text{spot signal}) - \text{mean}(\text{background})}{\text{std}(\text{background})}$$

Background is a region twice larger surrounding the spot region. Only the y and x dimensions are taking into account to compute the SNR.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

spots
[np.ndarray] Coordinate of the spots, with shape (nb_spots, 3) or (nb_spots, 2). One coordinate per dimension (zyx or yx coordinates).

voxel_size
[int, float, Tuple(int, float), List(int, float) or None] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

spot_radius
[int, float, Tuple(int, float), List(int, float) or None] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

Returns

snr
[float] Median signal-to-noise ratio computed for every spots.

3.5 Dense regions decomposition

Functions to detect dense and bright regions (with potential clustered spots), then use gaussian simulations to correct a misdetection in these regions.

3.5.1 Decompose and simulate dense regions

The main function to decompose dense regions is:

- *bigfish.detection.decompose_dense()*

It is also possible to perform the main steps of this decomposition separately:

- *bigfish.detection.get_dense_region()*
- *bigfish.detection.simulate_gaussian_mixture()*

See an example of application here.

bigfish.detection.**decompose_dense**(*image*, *spots*, *voxel_size*, *spot_radius*, *kernel_size=None*, *alpha=0.5*, *beta=1*, *gamma=5*)

Detect dense and bright regions with potential clustered spots and simulate a more realistic number of spots in these regions.

1. We estimate image background with a large gaussian filter. We then remove the background from the original image to denoise it.
2. We build a reference spot by aggregating predetected spots.
3. We fit gaussian parameters on the reference spots.
4. We detect dense regions to decompose.
5. We simulate as many gaussians as possible in the candidate regions.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

spots
[np.ndarray] Coordinate of the spots with shape (nb_spots, 3) or (nb_spots, 2) for 3-d or 2-d images respectively.

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

spot_radius
[int, float, Tuple(int, float) or List(int, float)] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions.

kernel_size
[int, float, Tuple(float, int), List(float, int) or None] Standard deviation used for the gaussian kernel (one for each dimension), in pixel. If it's a scalar, the same standard deviation is applied to every dimensions. If None, we estimate the kernel size from 'spot_radius', 'voxel_size' and 'gamma'

alpha
[int or float] Intensity percentile used to compute the reference spot, between 0 and 1. The higher, the brighter are the spots simulated in the dense regions. Consequently, a high intensity score reduces the number of spots added. Default is 0.5, meaning the reference spot considered is the median spot.

beta
[int or float] Multiplicative factor for the intensity threshold of a dense region. Default is 1. Threshold is computed with the formula:

$$\text{threshold} = \beta * \max(\text{median spot})$$

With median spot the median value of all detected spot signals.

gamma
[int or float] Multiplicative factor use to compute the gaussian kernel size:

$$\text{kernel size} = \frac{\gamma * \text{spot radius}}{\text{voxel size}}$$

We perform a large gaussian filter with such scale to estimate image background and remove it from original image. A large gamma increases the scale of the gaussian filter and smooth the estimated background. To decompose very large bright areas, a larger gamma should be set.

Returns

spots
[np.ndarray] Coordinate of the spots detected, with shape (nb_spots, 3) or (nb_spots, 2). One coordinate per dimension (zyx or yx coordinates).

dense_regions
[np.ndarray, np.int64] Array with shape (nb_regions, 7) or (nb_regions, 6). One coordinate per dimension for the region centroid (zyx or yx coordinates), the number of RNAs detected in the region, the area of the region, its average intensity value and its index.

reference_spot
[np.ndarray] Reference spot in 3-d or 2-d.

Notes

If `gamma = 0` and `kernel_size = None`, image is not denoised.

`bigfish.detection.`**`get_dense_region`**(*image*, *spots*, *voxel_size*, *spot_radius*, *beta=1*)

Detect and filter dense and bright regions.

A candidate region has at least 2 connected pixels above a specific threshold.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

spots
[np.ndarray] Coordinate of the spots with shape (nb_spots, 3) or (nb_spots, 2).

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

spot_radius
[int, float, Tuple(int, float) or List(int, float)] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions.

beta
[int or float] Multiplicative factor for the intensity threshold of a dense region. Default is 1. Threshold is computed with the formula:

$$\text{threshold} = \beta * \max(\text{median spot})$$

With median spot the median value of all detected spot signals.

Returns

dense_regions
[np.ndarray] Array with selected `skimage.measure._regionprops._RegionProperties` objects.

spots_out_region
[np.ndarray] Coordinate of the spots detected out of dense regions, with shape (nb_spots, 3) or (nb_spots, 2). One coordinate per dimension (zyx or yx coordinates).

max_size
[int] Maximum size of the regions.

`bigfish.detection.`**`simulate_gaussian_mixture`**(*image*, *candidate_regions*, *voxel_size*, *sigma*, *amplitude=100*, *background=0*, *precomputed_gaussian=None*)

Simulate as many gaussians as possible in the candidate dense regions in order to get a more realistic number of spots.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

candidate_regions
[np.ndarray] Array with filtered skimage.measure._regionprops._RegionProperties.

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

sigma
[int, float, Tuple(int, float) or List(int, float)] Standard deviation of the gaussian, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

amplitude
[float] Amplitude of the gaussian.

background
[float] Background minimum value.

precomputed_gaussian
[Tuple[np.ndarray]] Tuple with tables of precomputed values for the erf, with shape (nb_value, 2). One table per dimension.

Returns

spots_in_regions
[np.ndarray, np.int64] Coordinate of the spots detected inside dense regions, with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx coordinates) plus the index of the region.

regions
[np.ndarray, np.int64] Array with shape (nb_regions, 7) or (nb_regions, 6). One coordinate per dimension for the region centroid (zyx or yx coordinates), the number of RNAs detected in the region, the area of the region, its average intensity value and its index.

3.5.2 Modelize a reference spot

To simulate additional spots in the dense regions it is necessary to:

1. Build a reference spot.
   - *bigfish.detection.build_reference_spot()*
2. Modelize this reference spot by fitting gaussian parameters.
   - *bigfish.detection.modelize_spot()*
3. Simulate gaussian signal.
   - *bigfish.detection.precompute_erf()*
   - *bigfish.detection.initialize_grid()*
   - *bigfish.detection.gaussian_2d()*
   - *bigfish.detection.gaussian_3d()*

bigfish.detection.**build_reference_spot**(*image*, *spots*, *voxel_size*, *spot_radius*, *alpha=0.5*)

Build a median or mean spot in 3 or 2 dimensions as reference.

Reference spot is computed from a sample of uncropped detected spots. If such sample is not possible, an empty frame is returned.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

spots
[np.ndarray] Coordinate of the spots with shape (nb_spots, 3) for 3-d images or (nb_spots, 2) for 2-d images.

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

spot_radius
[int, float, Tuple(int, float) or List(int, float)] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions.

alpha
[int or float] Intensity score of the reference spot, between 0 and 1. If 0, reference spot approximates the spot with the lowest intensity. If 1, reference spot approximates the brightest spot. Default is 0.5.

Returns

reference_spot
[np.ndarray] Reference spot in 3-d or 2-d.

bigfish.detection.**modelize_spot**(*reference_spot*, *voxel_size*, *spot_radius*, *return_coord=False*)

Fit a gaussian function on the reference spot.

Parameters

reference_spot
[np.ndarray] A 3-d or 2-d image with detected spot and shape (z, y, x) or (y, x).

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

spot_radius
[int, float, Tuple(int, float) or List(int, float)] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions.

return_coord
[bool] Return gaussian coordinates.

Returns

parameters_fitted
[Tuple[float]]

- **mu_z**
  [float (optional)] Coordinate of the gaussian center along the z axis, in pixel.
- **mu_y**
  [float (optional)] Coordinate of the gaussian center along the y axis, in pixel.
- **mu_x**
  [float (optional)] Coordinate of the gaussian center along the x axis, in pixel.
- **sigma_z**
  [float] Standard deviation of the gaussian along the z axis, in pixel. Available only for a 3-d modelization.
- **sigma_yx**
  [float] Standard deviation of the gaussian in the yx plan, in pixel.
- **amplitude**
  [float] Amplitude of the gaussian.
- **background**
  [float] Background minimum value of the image.

bigfish.detection.**precompute_erf**(*ndim*, *voxel_size*, *sigma*, *max_grid=200*)

Precompute different values for the erf with a nanometer resolution.

Parameters

ndim
[int] Number of dimensions to consider (2 or 3).

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per

spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

sigma
[int, float, Tuple(int, float) or List(int, float)] Standard deviation of the gaussian, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

max_grid
[int] Maximum size of the grid on which we precompute the erf, in pixel.

Returns

table_erf
[Tuple[np.ndarray]] Tuple with tables of precomputed values for the erf, with shape (nb_value, 2). One table per dimension. First column is the coordinate along the table dimension. Second column is the precomputed erf value.

bigfish.detection.**initialize_grid**(*image_spot*, *voxel_size*, *return_centroid=False*)

Build a grid in nanometer to compute gaussian function values over a full volume or surface.

Parameters

image_spot
[np.ndarray] An image with detected spot and shape (z, y, x) or (y, x).

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

return_centroid
[bool] Compute centroid estimation of the grid.

Returns

grid
[np.ndarray, np.float32] A grid with the shape (3, z * y * x) or (2, y * x), in nanometer.

centroid_coord
[Tuple[float]] Estimated centroid of the spot, in nanometer. One element per dimension.

bigfish.detection.**gaussian_2d**(*grid*, *mu_y*, *mu_x*, *sigma_yx*, *voxel_size_yx*, *amplitude*, *background*, *precomputed=None*)

Compute the gaussian function over the grid representing a surface S with shape (S_y, S_x).

Parameters

grid
[np.ndarray, np.float] Grid data to compute the gaussian function for different voxel within a surface S. In nanometer, with shape (2, S_y * S_x).

mu_y
[float] Estimated mean of the gaussian signal along y axis, in nanometer.

mu_x
[float] Estimated mean of the gaussian signal along x axis, in nanometer.

sigma_yx
[int or float] Standard deviation of the gaussian in the yx plan, in nanometer.

voxel_size_yx
[int or float] Size of a voxel in the yx plan, in nanometer.

amplitude
[float] Estimated pixel intensity of the gaussian signal.

background
[float] Estimated pixel intensity of the background.

precomputed
[Tuple[np.ndarray] or None] Tuple with tables of precomputed values for the erf, with shape (nb_value, 2). One table per dimension.

Returns

values
[np.ndarray, np.float] Value of each voxel within the surface S according to the 2-d gaussian parameters. Shape (S_y * S_x,).

bigfish.detection.**gaussian_3d**(*grid*, *mu_z*, *mu_y*, *mu_x*, *sigma_z*, *sigma_yx*, *voxel_size_z*, *voxel_size_yx*, *amplitude*, *background*, *precomputed=None*)

Compute the gaussian function over the grid representing a volume V with shape (V_z, V_y, V_x).

Parameters

grid
[np.ndarray, np.float] Grid data to compute the gaussian function for different voxel within a volume V. In nanometer, with shape (3, V_z * V_y * V_x).

mu_z
[float] Estimated mean of the gaussian signal along z axis, in nanometer.

mu_y
[float] Estimated mean of the gaussian signal along y axis, in nanometer.

mu_x
[float] Estimated mean of the gaussian signal along x axis, in nanometer.

sigma_z
[int or float] Standard deviation of the gaussian along the z axis, in nanometer.

sigma_yx
[int or float] Standard deviation of the gaussian in the yx plan, in nanometer.

voxel_size_z
[int or float] Size of a voxel along the z axis, in nanometer.

voxel_size_yx
[int or float] Size of a voxel in the yx plan, in nanometer.

amplitude
[float] Estimated pixel intensity of the gaussian signal.

background
[float] Estimated pixel intensity of the background.

precomputed
[Tuple[np.ndarray] or None] Tuple with tables of precomputed values for the erf, with shape (nb_value, 2). One table per dimension.

Returns

values
[np.ndarray, np.float] Value of each voxel within the volume V according to the 3-d gaussian parameters. Shape (V_z * V_y * V_x,).

3.6 Subpixel fitting

Function to detect individual spot coordinate with a subpixel accuracy, fitting a gaussian signal.

bigfish.detection.**fit_subpixel**(*image*, *spots*, *voxel_size*, *spot_radius*)

Fit gaussian signal on every spot to find a subpixel coordinates.

Parameters

image
[np.ndarray] Image with shape (z, y, x) or (y, x).

spots
[np.ndarray] Coordinate of the spots detected, with shape (nb_spots, 3) or (nb_spots, 2). One coordinate per dimension (zyx or yx coordinates).

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

spot_radius
[int, float, Tuple(int, float) or List(int, float)] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions.

Returns

spots_subpixel
[np.ndarray] Coordinate of the spots detected, with shape (nb_spots, 3) or (nb_spots, 2). One coordinate per dimension (zyx or yx coordinates).

3.7 Cluster detection

Function to cluster spots in point cloud and detect relevant aggregated structures. A DBSCAN algorithm is performed.

See an example of application here.

bigfish.detection.**detect_clusters**(*spots*, *voxel_size*, *radius=350*, *nb_min_spots=4*)

Cluster spots and detect relevant aggregated structures.

1. If two spots are distant within a specific radius, we consider they are related to each other.
2. A minimum number spots related to each others defines a cluster.

Parameters

spots
[np.ndarray] Coordinates of the detected spots with shape (nb_spots, 3) or (nb_spots, 2).

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

radius
[int] The maximum distance between two samples for one to be considered as in the neighborhood of the other. Radius expressed in nanometer.

nb_min_spots
[int] The number of spots in a neighborhood for a point to be considered as a core point (from which a cluster is expanded). This includes the point itself.

Returns

clustered_spots
[np.ndarray] Coordinates of the detected spots with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx coordinates) plus the index of the cluster assigned to the spot. If no cluster was assigned, value is -1.

clusters
[np.ndarray] Array with shape (nb_clusters, 5) or (nb_clusters, 4). One coordinate per dimension for the clusters centroid (zyx or yx coordinates), the number of spots detected in the clusters and its index.

3.8 Colocalization

Match colocalized spots over two different channels:

- *bigfish.multistack.detect_spots_colocalization()*

Visualize the impact of distance threshold to discriminate between the colocalized spots and the distant ones:

- *bigfish.multistack.get_elbow_value_colocalized()*

bigfish.multistack.**detect_spots_colocalization**(*spots_1*, *spots_2*, *voxel_size*, *threshold=None*, *return_indices=False*, *return_threshold=False*)

Detect colocalized spots between two arrays of spot coordinates 'spots_1' and 'spots_2'. Pairs of spots below a specific threshold are defined as colocalized.

Parameters

spots_1
[np.ndarray] Coordinates of the spots 1 with shape (nb_spots_1, 3) or (nb_spots_1, 2).

spots_2
[np.ndarray] Coordinates of the spots 2 with shape (nb_spots_2, 3) or (nb_spots_2, 2).

voxel_size
[int, float, Tuple(int, float), or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

threshold
[int, float or None] A threshold to discriminate colocalized spots from distant ones. If None, an optimal threshold is selected automatically.

return_indices
[bool] Return the indices of the colocalized spots within 'spots_1' and 'spots_2'.

return_threshold
[bool] Return the threshold used to detect colocalized spots.

Returns

spots_1_colocalized
[np.ndarray] Coordinates of the colocalized spots from 'spots_1' with shape (nb_colocalized_spots,).

spots_2_colocalized
[np.ndarray] Coordinates of the colocalized spots from 'spots_2'with shape (nb_colocalized_spots,).

distances
[np.ndarray, np.float64] Distance matrix between spots with shape (nb_colocalized_spots,).

indices_1
[np.ndarray, np.int64] Indices of the colocalized spots in 'spots_1' with shape (nb_colocalized_spots,). Optional.

indices_2
[np.ndarray, np.int64] Indices of the colocalized spots in 'spots_2' with shape (nb_colocalized_spots,). Optional.

threshold
[int or float] Threshold used to discriminate colocalized spots from distant ones. Optional.

`bigfish.multistack.`**`get_elbow_value_colocalized`**(*spots_1*, *spots_2*, *voxel_size*)

Get values to plot the elbow curve used to automatically set the threshold to detect colocalized spots.

Parameters

spots_1
[np.ndarray] Coordinates of the spots with shape (nb_spots, 3) or (nb_spots, 2).

spots_2
[np.ndarray] Coordinates of the spots with shape (nb_spots, 3) or (nb_spots, 2).

voxel_size
[int, float, Tuple(int, float), or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

Returns

thresholds
[np.ndarray, np.float64] Candidate threshold values.

nb_colocalized
[np.ndarray, np.float64] Colocalized spots count.

optimal_threshold
[float or None] Threshold automatically set.

3.9 Nucleus segmentation

Functions used to segment nuclei.

3.9.1 Apply thresholding

Thresholding is the most standard and direct binary segmentation method:

bigfish.segmentation.**thresholding**(*image*, *threshold*)

Segment a 2-d image to discriminate object from background applying a threshold.

Parameters

image
[np.ndarray] A 2-d image to segment with shape (y, x).

threshold
[int or float] Pixel intensity threshold used to discriminate foreground from background.

Returns

image_segmented
[np.ndarray, bool] Binary 2-d image with shape (y, x).

3.9.2 Apply a Unet-based model (3-classes)

Load a pretrained model:

- *bigfish.segmentation.unet_3_classes_nuc()*

Segment nuclei:

- *bigfish.segmentation.apply_unet_3_classes()*
- *bigfish.segmentation.from_3_classes_to_instances()*

See an example of application here.

bigfish.segmentation.**unet_3_classes_nuc**()

Load a pretrained Unet model to predict 3 classes from nucleus images: background, edge and foreground.

Returns

model
[`tensorflow.keras.model` object] Pretrained Unet model.

bigfish.segmentation.**apply_unet_3_classes**(*model*, *image*, *target_size=None*, *test_time_augmentation=False*)

Segment image with a 3-classes trained model.

Parameters

model
[`tensorflow.keras.model` object] Pretrained Unet model that predicts 3 classes from nucleus or cell images (background, edge and foreground).

image
[np.ndarray, np.uint] Original image to segment with shape (y, x).

target_size
[int] Resize image before segmentation. A squared image is resize to *target_size*. A rectangular image is resize such that its smaller dimension equals *target_size*.

test_time_augmentation
[bool] Apply test time augmentation or not. The image is augmented 8 times and the final segmentation is the average result over these augmentations.

Returns

image_label_pred
[np.ndarray, np.int64] Labelled image. Each instance is characterized by the same pixel value.

bigfish.segmentation.**from_3_classes_to_instances**(*label_3_classes*)

Extract instance labels from 3-classes Unet output.

Parameters

label_3_classes
[np.ndarray, np.float32] Model prediction about the nucleus surface and boundaries, with shape (y, x, 3).

Returns

label
[np.ndarray, np.int64] Labelled image. Each instance is characterized by the same pixel value.

3.9.3 Remove segmented nuclei

Remove segmented nucleus instances to perform a new segmentation on the residual image:

bigfish.segmentation.**remove_segmented_nuc**(*image*, *nuc_mask*, *size_nuclei=2000*)

Remove the nuclei we have already segmented in an image.

1. We start from the segmented nuclei with a light dilation. The missed nuclei and the background are set to 0 and removed from the original image.
2. We reconstruct the missing nuclei by small dilation. As we used the original image to set the maximum allowed value at each pixel, the background pixels remain unchanged. However, pixels from the missing nuclei are partially reconstructed by the dilation. The reconstructed image only differs from the original one where the nuclei have been missed.
3. We subtract the reconstructed image from the original one.
4. From the few missing nuclei kept and restored, we build a binary mask (dilation, small object removal).
5. We apply this mask to the original image to get the original pixel intensity of the missing nuclei.
6. We remove pixels with a too low intensity.

Parameters

image
[np.ndarray, np.uint] Original nuclei image with shape (y, x).

nuc_mask
[np.ndarray,] Result of the segmentation (with instance differentiation or not).

size_nuclei
[int] Threshold above which we detect a nuclei.

Returns

image_without_nuc
[np.ndarray] Image with shape (y, x) and the same dtype of the original image. Nuclei previously detected in the mask are removed.

3.10 Cell segmentation

Functions used to segment cells.

3.10.1 Apply watershed algorithm

Main function to segment cells with a watershed algorithm:

- *bigfish.segmentation.cell_watershed()*

Our segmentation using watershed algorithm can also be perform with two separated steps:

- *bigfish.segmentation.get_watershed_relief()*
- *bigfish.segmentation.apply_watershed()*

bigfish.segmentation.**cell_watershed**(*image*, *nuc_label*, *threshold*, *alpha=0.8*)

Apply watershed algorithm to segment cell instances.

In a watershed algorithm we consider cells as watershed to be flooded. The watershed relief is inversely proportional to both the pixel intensity and the closeness to nuclei. Pixels with a high intensity or close to labelled nuclei have a low watershed relief value. They will be flooded in priority. Flooding the watersheds allows to propagate nuclei labels through potential cytoplasm areas. The lines separating watershed are the final segmentation of the cells.

Parameters

image
[np.ndarray, np.uint] Cells image with shape (y, x).

nuc_label
[np.ndarray, np.int64] Result of the nuclei segmentation with shape (y, x) and nuclei instances labelled.

threshold
[int or float] Threshold to discriminate cells surfaces from background.

alpha
[float or int] Weight of the pixel intensity values to compute the watershed relief.

Returns

cell_label
[np.ndarray, np.int64] Segmentation of cells with shape (y, x).

bigfish.segmentation.**get_watershed_relief**(*image*, *nuc_label*, *alpha*)

Build a representation of cells as watershed.

In a watershed algorithm we consider cells as watershed to be flooded. The watershed relief is inversely proportional to both the pixel intensity and the closeness to nuclei. Pixels with a high intensity or close to labelled nuclei have a low watershed relief value. They will be flooded in priority. Flooding the watersheds allows to propagate nuclei labels through potential cytoplasm areas. The lines separating watershed are the final segmentation of the cells.

Parameters

image
[np.ndarray, np.uint] Cells image with shape (z, y, x) or (y, x).

nuc_label
[np.ndarray, np.int64] Result of the nuclei segmentation with shape (y, x) and nuclei instances labelled.

alpha
[float or int] Weight of the pixel intensity values to compute the relief.

Returns

watershed_relief
[np.ndarray, np.uint16] Watershed representation of cells with shape (y, x).

bigfish.segmentation.**apply_watershed**(*watershed_relief*, *nuc_label*, *cell_mask*)

Apply watershed algorithm to segment cell instances.

In a watershed algorithm we consider cells as watershed to be flooded. The watershed relief is inversely proportional to both the pixel intensity and the closeness to nuclei. Pixels with a high intensity or close to labelled nuclei have a low watershed relief value. They will be flooded in priority. Flooding the watersheds allows to propagate nuclei labels through potential cytoplasm areas. The lines separating watershed are the final segmentation of the cells.

Parameters

watershed_relief
[np.ndarray, np.uint or np.int] Watershed representation of cells with shape (y, x).

nuc_label
[np.ndarray, np.int64] Result of the nuclei segmentation with shape (y, x) and nuclei instances labelled.

cell_mask
[np.ndarray, bool] Binary image of cells surface with shape (y, x).

Returns

cell_label
[np.ndarray, np.int64] Segmentation of cells with shape (y, x).

3.10.2 Apply a Unet-based model (distance map)

Load a pretrained model:

- *bigfish.segmentation.unet_distance_edge_double()*

Segment cells:

- *bigfish.segmentation.apply_unet_distance_double()*
- *bigfish.segmentation.from_distance_to_instances()*

See an example of application here.

bigfish.segmentation.**unet_distance_edge_double**()

Load a pretrained Unet model to predict foreground and a distance map to edge from nucleus and cell images.

Returns

model
[`tensorflow.keras.model` object] Pretrained Unet model.

bigfish.segmentation.**apply_unet_distance_double**(*model*, *nuc*, *cell*, *nuc_label*, *target_size=None*, *test_time_augmentation=False*)

Segment cell with a pretrained model to predict distance map and use it with a watershed algorithm.

Parameters

model
[tensorflow.keras.model object] Pretrained Unet model that predict distance to edges and cell surface.

nuc
[np.ndarray, np.uint] Original nucleus image with shape (y, x).

cell
[np.ndarray, np.uint] Original cell image to segment with shape (y, x).

nuc_label
[np.ndarray, np.int64] Labelled nucleus image. Each nucleus is characterized by the same pixel value.

target_size
[int] Resize image before segmentation. A squared image is resize to *target_size*. A rectangular image is resize such that its smaller dimension equals *target_size*.

test_time_augmentation
[bool] Apply test time augmentation or not. The image is augmented 8 times and the final segmentation is the average result over these augmentations.

Returns

cell_label_pred
[np.ndarray, np.int64] Labelled cell image. Each cell is characterized by the same pixel value.

bigfish.segmentation.**from_distance_to_instances**(*label_x_nuc*, *label_2_cell*, *label_distance*, *nuc_3_classes=False*, *compute_nuc_label=False*)

Extract instance labels from a distance map and a binary surface prediction with a watershed algorithm.

Parameters

label_x_nuc
[np.ndarray, np.float32] Model prediction about the nucleus surface (and boundaries), with shape (y, x, 1) or (y, x, 3).

label_2_cell
[np.ndarray, np.float32] Model prediction about cell surface, with shape (y, x, 1).

label_distance
[np.ndarray, np.uint16] Model prediction about the distance to edges, with shape (y, x, 1).

nuc_3_classes
[bool] Nucleus image input is an output from a 3-classes Unet.

compute_nuc_label
[bool] Extract nucleus instance labels.

Returns

nuc_label
[np.ndarray, np.int64] Labelled nucleus image. Each nucleus is characterized by the same pixel value.

cell_label
[np.ndarray, np.int64] Labelled cell image. Each cell is characterized by the same pixel value.

3.11 Postprocessing

Functions used to clean and refine segmentation results.

3.11.1 Label and clean instances

Label disconnected instances:

- *bigfish.segmentation.label_instances()*
- *bigfish.segmentation.merge_labels()*

Clean segmentation results:

- *bigfish.segmentation.clean_segmentation()*
- *bigfish.segmentation.remove_disjoint()*

See an example of application here.

bigfish.segmentation.**label_instances**(*image_binary*)

Count and label the different instances previously segmented in an image.

Parameters

image_binary
[np.ndarray, bool] Binary segmented image with shape (z, y, x) or (y, x).

Returns

image_label
[np.ndarray, np.int64] Labelled image. Each instance is characterized by the same pixel value.

bigfish.segmentation.**merge_labels**(*image_label_1*, *image_label_2*)

Combine two partial labels of the same image.

To prevent merging conflict, labels should not be rescale.

Parameters

image_label_1
[np.ndarray, np.int64] Labelled image with shape (z, y, x) or (y, x).

image_label_2
[np.ndarray, np.int64] Labelled image with shape (z, y, x) or (y, x).

Returns

image_label
[np.ndarray, np.int64] Labelled image with shape (z, y, x) or (y, x).

bigfish.segmentation.**clean_segmentation**(*image*, *small_object_size=None*, *fill_holes=False*, *smoothness=None*, *delimit_instance=False*)

Clean segmentation results (binary masks or integer labels).

Parameters

image
[np.ndarray, np.int64 or bool] Labelled or masked image with shape (y, x).

small_object_size
[int or None] Areas with a smaller surface (in pixels) are removed.

fill_holes
[bool] Fill holes within a labelled or masked area.

smoothness
[int or None] Radius of a median kernel filter. The higher the smoother instance boundaries are.

delimit_instance
[bool] Delimit clearly instances boundaries by preventing contact between each others.

Returns

image_cleaned
[np.ndarray, np.int64 or bool] Cleaned image with shape (y, x).

bigfish.segmentation.**remove_disjoint**(*image*)

For each instances with disconnected parts, keep the larger one.

Parameters

image
[np.ndarray, np.int, np.uint or bool] Labelled image with shape (z, y, x) or (y, x).

Returns

image_cleaned
[np.ndarray, np.int or np.uint] Cleaned image with shape (z, y, x) or (y, x).

3.11.2 Compute instance statistics

Compute statistics for each segmented instance:

- *bigfish.segmentation.compute_mean_diameter()*
- *bigfish.segmentation.compute_mean_convexity_ratio()*
- *bigfish.segmentation.compute_surface_ratio()*
- *bigfish.segmentation.count_instances()*

bigfish.segmentation.**compute_mean_diameter**(*image_label*)

Compute the averaged size of the segmented instances.

For each instance, we compute the diameter of an object with an equivalent surface. Then, we average the diameters.

Parameters

image_label
[np.ndarray, np.int or np.uint] Labelled image with shape (y, x).

Returns

mean_diameter
[float] Averaged size of the segmented instances.

bigfish.segmentation.**compute_mean_convexity_ratio**(*image_label*)

Compute the averaged convexity ratio of the segmented instances.

For each instance, we compute the ratio between its area and the area of its convex hull. Then, we average the diameters.

Parameters

image_label
[np.ndarray, np.int or np.uint] Labelled image with shape (y, x).

Returns

mean_convexity_ratio
[float] Averaged convexity ratio of the segmented instances.

bigfish.segmentation.**compute_surface_ratio**(*image_label*)

Compute the averaged surface ratio of the segmented instances.

We compute the proportion of surface occupied by instances.

Parameters

image_label
[np.ndarray, np.int or np.uint] Labelled image with shape (y, x).

Returns

surface_ratio
[float] Surface ratio of the segmented instances.

bigfish.segmentation.**count_instances**(*image_label*)

Count the number of instances annotated in the image.

Parameters

image_label
[np.ndarray, np.int or np.uint] Labelled image with shape (y, x).

Returns

nb_instances
[int] Number of instances in the image.

3.11.3 Match cells and nuclei

Match nuclei and cells:

bigfish.multistack.**match_nuc_cell**(*nuc_label*, *cell_label*, *single_nuc*, *cell_alone*)

Match each nucleus instance with the most overlapping cell instance.

Parameters

nuc_label
[np.ndarray, np.int or np.uint] Labelled image of nuclei with shape (z, y, x) or (y, x).

cell_label
[np.ndarray, np.int or np.uint] Labelled image of cells with shape (z, y, x) or (y, x).

single_nuc
[bool] Authorized only one nucleus in a cell.

cell_alone
[bool] Authorized cell without nucleus.

Returns

new_nuc_label
[np.ndarray, np.int or np.uint] Labelled image of nuclei with shape (z, y, x) or (y, x).

new_cell_label
[np.ndarray, np.int or np.uint] Labelled image of cells with shape (z, y, x) or (y, x).

3.12 Single-cell identification

Functions to exploit detection and segmentation results, by identifying individual cells and their objects.

3.12.1 Identify and remove transcription sites

Define transcription sites as clustered RNAs detected inside nucleus:

- *bigfish.multistack.remove_transcription_site()*

More generally, identify detected objects within a specific cellular region:

- *bigfish.multistack.identify_objects_in_region()*

bigfish.multistack.**remove_transcription_site**(*rna*, *clusters*, *nuc_mask*, *ndim*)

Distinguish RNA molecules detected in a transcription site from the rest.

A transcription site is defined as as a foci detected within the nucleus.

Parameters

rna
[np.ndarray] Coordinates of the detected RNAs with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx coordinates) plus the index of the cluster assigned to the RNA. If no cluster was assigned, value is -1.

clusters
[np.ndarray] Array with shape (nb_clusters, 5) or (nb_clusters, 4). One coordinate per dimension for the clusters centroid (zyx or yx coordinates), the number of RNAs detected in the clusters and their index.

nuc_mask
[np.ndarray, bool] Binary mask of the nuclei region with shape (y, x).

ndim
[int] Number of spatial dimensions to consider (2 or 3).

Returns

rna_out_ts
[np.ndarray] Coordinates of the detected RNAs with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx coordinates) plus the index of the foci assigned to the RNA. If no foci was assigned, value is -1. RNAs from transcription sites are removed.

foci
[np.ndarray] Array with shape (nb_foci, 5) or (nb_foci, 4). One coordinate per dimension for the foci centroid (zyx or yx coordinates), the number of RNAs detected in the foci and its index.

ts
[np.ndarray] Array with shape (nb_ts, 5) or (nb_ts, 4). One coordinate per dimension for the transcription site centroid (zyx or yx coordinates), the number of RNAs detected in the transcription site and its index.

bigfish.multistack.**identify_objects_in_region**(*mask*, *coord*, *ndim*)

Identify cellular objects in specific region.

Parameters

mask
[np.ndarray, bool] Binary mask of the targeted region with shape (y, x).

coord
[np.ndarray] Array with two dimensions. One object per row, zyx or yx coordinates in the first 3 or 2 columns.

ndim
[int] Number of spatial dimensions to consider (2 or 3).

Returns

coord_in
[np.ndarray] Coordinates of the objects detected inside the region.

coord_out
[np.ndarray] Coordinates of the objects detected outside the region.

3.12.2 Define and export single-cell results

Extract detection and segmentation results and for every individual cell:

- *bigfish.multistack.extract_cell()*
- *bigfish.multistack.extract_spots_from_frame()*
- *bigfish.multistack.summarize_extraction_results()*

See an example of application here.

bigfish.multistack.**extract_cell**(*cell_label*, *ndim*, *nuc_label=None*, *rna_coord=None*, *others_coord=None*, *image=None*, *others_image=None*, *remove_cropped_cell=True*, *check_nuc_in_cell=True*)

Extract cell-level results for an image.

The function gathers different segmentation and detection results obtained at the image level and assigns each of them to the individual cells.

Parameters

cell_label
[np.ndarray, np.uint or np.int] Image with labelled cells and shape (y, x).

ndim
[int] Number of spatial dimensions to consider (2 or 3).

nuc_label
[np.ndarray, np.uint or np.int] Image with labelled nuclei and shape (y, x). If None, individual nuclei are not assigned to each cell.

rna_coord
[np.ndarray] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns. If None, RNAs are not assigned to individual cells.

others_coord
[Dict[np.ndarray]] Dictionary of coordinates arrays. For each array of the dictionary, the different elements are assigned to individual cells. Arrays should be organized the same way than spots: zyx or yx coordinates in the first 3 or 2 columns, np.int64 dtype, one element per row. Can be used to assign different detected elements to the segmented cells along with the spots. If None, no others elements are assigned to the individual cells.

image
[np.ndarray, np.uint] Image in 2-d. If None, image of the individual cells are not extracted.

others_image
[Dict[np.ndarray]] Dictionary of images to crop. If None, no others image of the individual cells are extracted.

remove_cropped_cell
[bool] Remove cells cropped by the FoV frame.

check_nuc_in_cell
[bool] Check that each nucleus is entirely localized within a cell.

Returns

fov_results
[List[Dict]] List of dictionaries, one per cell segmented in the image. Each dictionary includes information about the cell (image, masks, coordinates arrays). Minimal information are:

- *cell_id*: Unique id of the cell.
- *bbox*: bounding box coordinates with the order (*min_y*, *min_x*, *max_y*, *max_x*).
- *cell_coord*: boundary coordinates of the cell.
- *cell_mask*: mask of the cell.

bigfish.multistack.**extract_spots_from_frame**(*spots*, *ndim*, *z_lim=None*, *y_lim=None*, *x_lim=None*)

Get spots coordinates within a given frame.

Parameters

spots
[np.ndarray] Coordinate of the spots. One coordinate per dimension first (zyx coordinates or yx coordinates) plus additional dimensions if necessary.

ndim
[{2, 3}] Number of spatial dimension to consider.

z_lim
[tuple[int, int]] Minimum and maximum coordinate of the frame along the z axis.

y_lim
[tuple[int, int]] Minimum and maximum coordinate of the frame along the y axis.

x_lim
[tuple[int, int]] Minimum and maximum coordinate of the frame along the x axis.

Returns

extracted_spots
[np.ndarray] Coordinate of the spots. One coordinate per dimension first (zyx coordinates or yx coordinates) plus additional dimensions if necessary.

bigfish.multistack.**summarize_extraction_results**(*fov_results*, *ndim*, *path_output=None*, *delimiter=';'*)

Summarize results extracted from an image and store them in a dataframe.

Parameters

fov_results
[List[Dict]] List of dictionaries, one per cell segmented in the image. Each dictionary includes information about the cell (image, masks, coordinates arrays). Minimal information are:

- *cell_id*: Unique id of the cell.
- *bbox*: bounding box coordinates with the order (*min_y*, *min_x*, *max_y*, *max_x*).
- *cell_coord*: boundary coordinates of the cell.
- *cell_mask*: mask of the cell.

ndim
[int] Number of spatial dimensions to consider (2 or 3).

path_output
[str, optional] Path to save the dataframe in a csv file.

delimiter
[str, default=";"] Delimiter used to separate columns if the dataframe is saved in a csv file.

Returns

df
[pd.DataFrame] Dataframe with summarized results from the field of view, at the cell level. At least *cell_id* (Unique id of the cell) and 'cell_area' (2-d area of the cell, in pixel) are returned. Other indicators are summarized if available:

- *nuc_area*: 2-d area of the nucleus, in pixel.
- *nb_rna*: Number of detected rna in the cell.
- *nb_rna_in_nuc*: Number of detected rna inside the nucleus.
- *nb_rna_out_nuc*: Number of detected rna outside the nucleus.

Extra coordinates elements detected are counted in the cell and summarized as well.

3.12.3 Manipulate surfaces, coordinates and boundaries

Convert identified surfaces into coordinates, delimit boundaries and manipulates coordinates:

- *bigfish.multistack.center_mask_coord()*
- *bigfish.multistack.from_boundaries_to_surface()*
- *bigfish.multistack.from_surface_to_boundaries()*
- *bigfish.multistack.from_binary_to_coord()*
- *bigfish.multistack.complete_coord_boundaries()*
- *bigfish.multistack.from_coord_to_frame()*

- *bigfish.multistack.from_coord_to_surface()*

bigfish.multistack.**center_mask_coord**(*main*, *others=None*)

Center a 2-d binary mask (surface or boundaries) or a 2-d localization coordinates array and pad it.

One mask or coordinates array should be at least provided (*main*). If others masks or arrays are provided (*others*), they will be transformed like *main*. All the provided masks should have the same shape.

Parameters

main
[np.ndarray, np.uint or np.int or bool] Binary image with shape (y, x) or array of coordinates with shape (nb_points, 2).

others
[List(np.ndarray)] List of binary image with shape (y, x), array of coordinates with shape (nb_points, 2) or array of coordinates with shape (nb_points, 3).

Returns

main_centered
[np.ndarray, np.uint or np.int or bool] Centered binary image with shape (y, x).

others_centered
[List(np.ndarray)] List of centered binary image with shape (y, x), centered array of coordinates with shape (nb_points, 2) or centered array of coordinates with shape (nb_points, 3).

bigfish.multistack.**from_boundaries_to_surface**(*binary_boundaries*)

Fill in the binary matrix representing the boundaries of an object.

Parameters

binary_boundaries
[np.ndarray, np.uint or np.int or bool] Binary image with shape (y, x).

Returns

binary_surface
[np.ndarray, bool] Binary image with shape (y, x).

bigfish.multistack.**from_surface_to_boundaries**(*binary_surface*)

Convert the binary surface to binary boundaries.

Parameters

binary_surface
[np.ndarray, np.uint or np.int or bool] Binary image with shape (y, x).

Returns

binary_boundaries
[np.ndarray, np.uint or np.int or bool] Binary image with shape (y, x).

bigfish.multistack.**from_binary_to_coord**(*binary*)

Extract coordinates from a 2-d binary matrix.

As the resulting coordinates represent the external boundaries of the object, the coordinates values can be negative.

Parameters

binary
[np.ndarray, np.uint or np.int or bool] Binary image with shape (y, x).

Returns

coord
[np.ndarray, np.int] Array of boundaries coordinates with shape (nb_points, 2).

bigfish.multistack.**complete_coord_boundaries**(*coord*)

Complete a 2-d coordinates array, by generating/interpolating missing points.

Parameters

coord
[np.ndarray, np.int] Array of coordinates to complete, with shape (nb_points, 2).

Returns

coord_completed
[np.ndarray, np.int] Completed coordinates arrays, with shape (nb_points, 2).

bigfish.multistack.**from_coord_to_frame**(*coord*, *external_coord=True*)

Initialize a frame shape to represent coordinates values in 2-d matrix.

If coordinates represent the external boundaries of an object, we add 1 to the minimum coordinate and substract 1 to the maximum coordinate in order to build the frame. The frame centers the coordinates by default.

Parameters

coord
[np.ndarray, np.int] Array of cell boundaries coordinates with shape (nb_points, 2) or (nb_points, 3).

external_coord
[bool] Coordinates represent external boundaries of object.

Returns

frame_shape
[tuple] Shape of the 2-d matrix.

min_y
[int] Value tu substract from the y coordinate axis.

min_x
[int] Value tu substract from the x coordinate axis.

marge
[int] Value to add to the coordinates.

bigfish.multistack.**from_coord_to_surface**(*cell_coord*, *nuc_coord=None*, *rna_coord=None*, *external_coord=True*)

Convert 2-d coordinates to a binary matrix with the surface of the object.

If we manipulate the coordinates of the external boundaries, the relative binary matrix has two extra pixels in each dimension. We compensate by keeping only the inside pixels of the object surface.

If others coordinates are provided (nucleus and mRNAs), the relative binary matrix is built with the same shape as the main coordinates (cell).

Parameters

cell_coord
[np.ndarray, np.int] Array of cell boundaries coordinates with shape (nb_points, 2).

nuc_coord
[np.ndarray, np.int] Array of nucleus boundaries coordinates with shape (nb_points, 2).

rna_coord
[np.ndarray, np.int] Array of mRNAs coordinates with shape (nb_points, 2) or (nb_points, 3).

external_coord
[bool] Coordinates represent external boundaries of object.

Returns

cell_surface
[np.ndarray, bool] Binary image of cell surface with shape (y, x).

nuc_surface
[np.ndarray, bool] Binary image of nucleus surface with shape (y, x).

rna_binary
[np.ndarray, bool] Binary image of mRNAs localizations with shape (y, x).

new_rna_coord
[np.ndarray, np.int] Array of mRNAs coordinates with shape (nb_points, 2) or (nb_points, 3).

3.13 Features engineering

3.13.1 Prepare input coordinates

Format input coordinates and compute intermediary results to prepare features computation:

`bigfish.classification.`**`prepare_extracted_data`**(*cell_mask*, *nuc_mask=None*, *ndim=None*, *rna_coord=None*, *centrosome_coord=None*)

Prepare data extracted from images.

Parameters

cell_mask
[np.ndarray, np.uint, np.int or bool] Surface of the cell with shape (y, x).

nuc_mask: np.ndarray, np.uint, np.int or bool
Surface of the nucleus with shape (y, x).

ndim
[int] Number of spatial dimensions to consider (2 or 3). Mandatory if *rna_coord* is provided.

rna_coord
[np.ndarray, np.int] Coordinates of the detected spots with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx dimensions) plus the index of the cluster assigned to the spot. If no cluster was assigned, value is -1.

centrosome_coord
[np.ndarray, np.int] Coordinates of the detected centrosome with shape (nb_elements, 3) or (nb_elements, 2). One coordinate per dimension (zyx or yx dimensions).

Returns

cell_mask
[np.ndarray, bool] Surface of the cell with shape (y, x).

distance_cell
[np.ndarray, np.float32] Distance map from the cell with shape (y, x), in pixels.

distance_cell_normalized
[np.ndarray, np.float32] Normalized distance map from the cell with shape (y, x).

centroid_cell
[np.ndarray, np.int] Coordinates of the cell centroid with shape (2,).

distance_centroid_cell
[np.ndarray, np.float32] Distance map from the cell centroid with shape (y, x), in pixels.

nuc_mask
[np.ndarray, bool] Surface of the nucleus with shape (y, x).

cell_mask_out_nuc
[np.ndarray, bool] Surface of the cell (outside the nucleus) with shape (y, x).

distance_nuc
[np.ndarray, np.float32] Distance map from the nucleus with shape (y, x), in pixels.

distance_nuc_normalized
[np.ndarray, np.float32] Normalized distance map from the nucleus with shape (y, x).

centroid_nuc
[np.ndarray, np.int] Coordinates of the nucleus centroid with shape (2,).

distance_centroid_nuc
[np.ndarray, np.float32] Distance map from the nucleus centroid with shape (y, x), in pixels.

rna_coord_out_nuc
[np.ndarray, np.int] Coordinates of the detected spots with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx dimensions) plus the index of the cluster assigned to the spot. If no cluster was assigned, value is -1. Spots detected inside the nucleus are removed.

centroid_rna
[np.ndarray, np.int] Coordinates of the rna centroid with shape (2,) or (3,).

distance_centroid_rna
[np.ndarray, np.float32] Distance map from the rna centroid with shape (y, x), in pixels.

centroid_rna_out_nuc
[np.ndarray, np.int] Coordinates of the rna centroid (outside the nucleus) with shape (2,) or (3,).

distance_centroid_rna_out_nuc
[np.ndarray, np.float32] Distance map from the rna centroid (outside the nucleus) with shape (y, x), in pixels.

distance_centrosome
[np.ndarray, np.float32] Distance map from the centrosome with shape (y, x), in pixels.

3.13.2 Compute features

Functions to compute features about cell morphology and RNAs localization. There are two main functions to compute spatial and morphological features are:

- *bigfish.classification.compute_features()*
- *bigfish.classification.get_features_name()*

Group of features can be computed separately:

- *bigfish.classification.features_distance()*
- *bigfish.classification.features_in_out_nucleus()*
- *bigfish.classification.features_protrusion()*
- *bigfish.classification.features_dispersion()*
- *bigfish.classification.features_topography()*
- *bigfish.classification.features_foci()*
- *bigfish.classification.features_area()*
- *bigfish.classification.features_centrosome()*

See an example of application here.

bigfish.classification.**compute_features**(*cell_mask*, *nuc_mask*, *ndim*, *rna_coord*, *smfish=None*, *voxel_size_yx=None*, *foci_coord=None*, *centrosome_coord=None*, *compute_distance=False*, *compute_intranuclear=False*, *compute_protrusion=False*, *compute_dispersion=False*, *compute_topography=False*, *compute_foci=False*, *compute_area=False*, *compute_centrosome=False*, *return_names=False*)

Compute requested features.

Parameters

cell_mask
[np.ndarray, np.uint, np.int or bool] Surface of the cell with shape (y, x).

nuc_mask: np.ndarray, np.uint, np.int or bool
Surface of the nucleus with shape (y, x).

ndim
[int] Number of spatial dimensions to consider (2 or 3).

rna_coord
[np.ndarray, np.int] Coordinates of the detected spots with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx dimensions) plus the index of the cluster assigned to the spot. If no cluster was assigned, value is -1. If cluster id is not provided foci related features are not computed.

smfish
[np.ndarray, np.uint] Image of RNAs, with shape (y, x).

voxel_size_yx
[int, float or None] Size of a voxel on the yx plan, in nanometer.

foci_coord
[np.ndarray, np.int] Array with shape (nb_foci, 5) or (nb_foci, 4). One coordinate per dimension for the foci centroid (zyx or yx coordinates), the number of spots detected in the foci and its index.

centrosome_coord
[np.ndarray, np.int] Coordinates of the detected centrosome with shape (nb_elements, 3) or (nb_elements, 2). One coordinate per dimension (zyx or yx dimensions). These coordinates are mandatory to compute centrosome related features.

compute_distance
[bool] Compute distance related features.

compute_intranuclear
[bool] Compute nucleus related features.

compute_protrusion
[bool] Compute protrusion related features.

compute_dispersion
[bool] Compute dispersion indices.

compute_topography
[bool] Compute topographic features.

compute_foci
[bool] Compute foci related features.

compute_area
[bool] Compute area related features.

compute_centrosome
[bool] Compute centrosome related features.

return_names
[bool] Return features names.

Returns

features
[np.ndarray, np.float32] Array of features.

bigfish.classification.**get_features_name**(*names_features_distance=False*, *names_features_intranuclear=False*, *names_features_protrusion=False*, *names_features_dispersion=False*, *names_features_topography=False*, *names_features_foci=False*, *names_features_area=False*, *names_features_centrosome=False*)

Return the current list of features names.

Parameters

names_features_distance
[bool] Return names of features related to distances from nucleus or cell membrane.

names_features_intranuclear
[bool] Return names of features related to nucleus.

names_features_protrusion
[bool] Return names of features related to protrusions.

names_features_dispersion
[bool] Return names of features used to quantify mRNAs dispersion within the cell.

names_features_topography
[bool] Return names of topographic features of the cell.

names_features_foci
[bool] Return names of features related to foci.

names_features_area
[bool] Return names of features related to area of the cell.

names_features_centrosome
[bool] Return names of features related to centrosome.

Returns

features_name
[List[str]] A list of features name.

bigfish.classification.**features_distance**(*rna_coord*, *distance_cell*, *distance_nuc*, *cell_mask*, *ndim*, *check_input=True*)

Compute distance related features.

Parameters

rna_coord
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns.

distance_cell
[np.ndarray, np.float32] Distance map from the cell with shape (y, x).

distance_nuc
[np.ndarray, np.float32] Distance map from the nucleus with shape (y, x).

cell_mask
[np.ndarray, bool] Surface of the cell with shape (y, x).

ndim
[int] Number of spatial dimensions to consider.

check_input
[bool] Check input validity.

Returns

index_mean_dist_cell
[float] Normalized mean distance of RNAs to the cell membrane.

index_median_dist_cell
[float] Normalized median distance of RNAs to the cell membrane.

index_mean_dist_nuc
[float] Normalized mean distance of RNAs to the nucleus.

index_median_dist_nuc
[float] Normalized median distance of RNAs to the nucleus.

bigfish.classification.**features_in_out_nucleus**(*rna_coord*, *rna_coord_out_nuc*, *check_input=True*)

Compute nucleus related features.

Parameters

rna_coord
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns.

rna_coord_out_nuc
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns. Spots detected inside the nucleus are removed.

check_input
[bool] Check input validity.

Returns

proportion_rna_in_nuc
[float] Proportion of RNAs detected inside the nucleus.

nb_rna_out_nuc
[float] Number of RNAs detected outside the nucleus.

nb_rna_in_nuc
[float] Number of RNAs detected inside the nucleus.

bigfish.classification.**features_protrusion**(*rna_coord*, *cell_mask*, *nuc_mask*, *ndim*, *voxel_size_yx*, *check_input=True*)

Compute protrusion related features.

Parameters

rna_coord
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns.

cell_mask
[np.ndarray, bool] Surface of the cell with shape (y, x).

nuc_mask
[np.ndarray, bool] Surface of the nucleus with shape (y, x).

ndim
[int] Number of spatial dimensions to consider.

voxel_size_yx
[int or float] Size of a voxel on the yx plan, in nanometer.

check_input
[bool] Check input validity.

Returns

index_rna_protrusion
[float] Number of RNAs detected in a protrusion and normalized by the expected number of RNAs under random distribution.

proportion_rna_protrusion
[float] Proportion of RNAs detected in a protrusion.

protrusion_area
[float] Protrusion area (in pixels).

bigfish.classification.**features_dispersion**(*smfish*, *rna_coord*, *centroid_rna*, *cell_mask*, *centroid_cell*, *centroid_nuc*, *ndim*, *check_input=True*)

Compute RNA Distribution Index features (RDI) described in:

RDI Calculator: An analysis Tool to assess RNA distributions in cells, Stueland M., Wang T., Park H. Y., Mili, S., 2019.

Parameters

smfish
[np.ndarray, np.uint] Image of RNAs, with shape (y, x).

rna_coord
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns.

centroid_rna
[np.ndarray, np.int] Coordinates of the rna centroid with shape (2,) or (3,).

cell_mask
[np.ndarray, bool] Surface of the cell with shape (y, x).

centroid_cell
[np.ndarray, np.int] Coordinates of the cell centroid with shape (2,).

centroid_nuc
[np.ndarray, np.int] Coordinates of the nucleus centroid with shape (2,).

ndim
[int] Number of spatial dimensions to consider.

check_input
[bool] Check input validity.

Returns

index_polarization
[float] Polarization index (PI).

index_dispersion
[float] Dispersion index (DI).

index_peripheral_distribution
[float] Peripheral distribution index (PDI).

bigfish.classification.**features_topography**(*rna_coord*, *cell_mask*, *nuc_mask*, *cell_mask_out_nuc*, *ndim*, *voxel_size_yx*, *check_input=True*)

Compute topographic features.

Parameters

rna_coord
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns.

cell_mask
[np.ndarray, bool] Surface of the cell with shape (y, x).

nuc_mask
[np.ndarray, bool] Surface of the nucleus with shape (y, x).

cell_mask_out_nuc
[np.ndarray, bool] Surface of the cell (outside the nucleus) with shape (y, x).

ndim
[int] Number of spatial dimensions to consider.

voxel_size_yx
[int or float] Size of a voxel on the yx plan, in nanometer.

check_input
[bool] Check input validity.

Returns

index_rna_nuc_marge
[float] Number of RNAs detected in a specific region around nucleus and normalized by the expected number of RNAs under random distribution. Six regions are targeted (less than 500nm, 500-1000nm, 1000-1500nm, 1500-2000nm, 2000-2500nm and 2500-3000nm from the nucleus boundary).

proportion_rna_nuc_marge
[float] Proportion of RNAs detected in a specific region around nucleus. Six regions are targeted (less than 500nm, 500-1000nm, 1000-1500nm, 1500-2000nm, 2000-2500nm and 2500-3000nm from the nucleus boundary).

index_rna_cell_marge
[float] Number of RNAs detected in a specific region around cell membrane and normalized by the expected number of RNAs under random distribution. Six regions are targeted (0-500nm, 500-1000nm, 1000-1500nm, 1500-2000nm, 2000-2500nm and 2500-3000nm from the cell membrane).

proportion_rna_cell_marge
[float] Proportion of RNAs detected in a specific region around cell membrane. Six regions are targeted (0-500nm, 500-1000nm, 1000-1500nm, 1500-2000nm, 2000-2500nm and 2500-3000nm from the cell membrane).

`bigfish.classification.`**`features_foci`**`(`*rna_coord*, *foci_coord*, *ndim*, *check_input=True*`)`

Compute foci related features.

Parameters

rna_coord
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns.

foci_coord
[np.ndarray, np.int] Array with shape (nb_foci, 5) or (nb_foci, 4). One coordinate per dimension for the foci centroid (zyx or yx coordinates), the number of spots detected in the foci and its index.

ndim
[int] Number of spatial dimensions to consider.

check_input
[bool] Check input validity.

Returns

proportion_rna_in_foci
[float] Proportion of RNAs detected in a foci.

`bigfish.classification.`**`features_area`**`(`*cell_mask*, *nuc_mask*, *cell_mask_out_nuc*, *check_input=True*`)`

Compute area related features.

Parameters

cell_mask
[np.ndarray, bool] Surface of the cell with shape (y, x).

nuc_mask
[np.ndarray, bool] Surface of the nucleus with shape (y, x).

cell_mask_out_nuc
[np.ndarray, bool] Surface of the cell (outside the nucleus) with shape (y, x).

check_input
[bool] Check input validity.

Returns

nuc_relative_area
[float] Proportion of nucleus area in the cell.

cell_area
[float] Cell area (in pixels).

nuc_area
[float] Nucleus area (in pixels).

cell_area_out_nuc
[float] Cell area outside the nucleus (in pixels).

bigfish.classification.**features_centrosome**(*smfish*, *rna_coord*, *distance_centrosome*, *cell_mask*, *ndim*, *voxel_size_yx*, *check_input=True*)

Compute centrosome related features (in 2 dimensions).

Parameters

smfish
[np.ndarray, np.uint] Image of RNAs, with shape (y, x).

rna_coord
[np.ndarray, np.int] Coordinates of the detected RNAs with zyx or yx coordinates in the first 3 or 2 columns.

distance_centrosome
[np.ndarray, np.float32] Distance map from the centrosome with shape (y, x), in pixels.

cell_mask
[np.ndarray, bool] Surface of the cell with shape (y, x).

ndim
[int] Number of spatial dimensions to consider.

voxel_size_yx
[int or float] Size of a voxel on the yx plan, in nanometer.

check_input
[bool] Check input validity.

Returns

index_mean_dist_cent
[float] Normalized mean distance of RNAs to the closest centrosome.

index_median_dist_cent
[float] Normalized median distance of RNAs to the closest centrosome.

index_rna_centrosome
[float] Number of RNAs within a 2000nm radius from a centrosome, normalized by the expected number of RNAs under random distribution.

proportion_rna_centrosome
[float] Proportion of RNAs within a 2000nm radius from a centrosome.

index_centrosome_dispersion
[float] Centrosomal dispersion index. It quantify the dispersion of RNAs around centrosomes. The lower, the closer the RNAs are.

3.14 Field of view plot

Functions to visualize 2D and 3D images:

3.14.1 Plot images

Plot 2D images or slices of 3D images:

- *bigfish.plot.plot_yx()*
- *bigfish.plot.plot_images()*

`bigfish.plot.`**`plot_yx`**(*image*, *r=0*, *c=0*, *z=0*, *rescale=False*, *contrast=False*, *title=None*, *framesize=(10, 10)*, *remove_frame=True*, *path_output=None*, *ext='png'*, *show=True*)

Plot the selected yx plan of the selected dimensions of an image.

Parameters

image
[np.ndarray] A 2-d, 3-d, 4-d or 5-d image with shape (y, x), (z, y, x), (c, z, y, x) or (r, c, z, y, x) respectively.

r
[int, default=0] Index of the round to keep.

c
[int, default=0] Index of the channel to keep.

z
[int, default=0] Index of the z slice to keep.

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

title
[str, optional] Title of the image.

framesize
[tuple=(10, 10)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

remove_frame
[bool, default=True] Remove axes and frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

bigfish.plot.**plot_images**(*images*, *rescale=False*, *contrast=False*, *titles=None*, *framesize=(15, 10)*, *remove_frame=True*, *path_output=None*, *ext='png'*, *show=True*)

Plot or subplot of 2-d images.

Parameters

images
[np.ndarray or list] Image or list of images with shape (y, x).

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

titles
[str or list, optional] Titles of the subplots.

framesize
[tuple, default=(15, 10)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

remove_frame
[bool, default=True] Remove axes and frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

3.14.2 Plot quality measure of an image

Plot focus measures for a 3D image:

bigfish.plot.**plot_sharpness**(*focus_measures*, *labels=None*, *title=None*, *framesize=(5, 5)*, *size_title=20*, *size_axes=15*, *size_legend=15*, *path_output=None*, *ext='png'*, *show=True*)

Plot focus measures of a 3-d image, at the z-slice level.

A measure of focus for each z-slice can be computed by averaging the pixel-wise focus measure returned from *bigfish.stack.compute_focus()*.

Parameters

focus_measures
[np.ndarray or list] A list of 1-d arrays with the sharpness measure for each z-slices.

labels
[str or list, optional] List of labels for the different measures to compare.

title
[str, optional] Title of the plot.

framesize
[tuple, default=(5, 5)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

size_title
[int, default=20] Size of the title.

size_axes
[int, default=15] Size of the axes label.

size_legend
[int, default=15] Size of the legend.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

3.15 Detection plot

Functions to visualize detection results.

Visualize detected spots:

- *bigfish.plot.plot_detection()*

Visualize the reference spot computed for an image:

- *bigfish.plot.plot_reference_spot()*

Visualize the elbow curve used to automatically set a detection threshold:

- *bigfish.plot.plot_elbow()*
- *bigfish.plot.plot_elbow_colocalized()*

bigfish.plot.**plot_detection**(*image*, *spots*, *shape='circle'*, *radius=3*, *color='red'*, *linewidth=1*, *fill=False*, *rescale=False*, *contrast=False*, *title=None*, *framesize=(15, 10)*, *remove_frame=True*, *path_output=None*, *ext='png'*, *show=True*)

Plot detected spots and foci on a 2-d image.

Parameters

image
[np.ndarray] A 2-d image with shape (y, x).

spots
[list or np.ndarray] Array with coordinates and shape (nb_spots, 3) or (nb_spots, 2). To plot different kind of detected spots with different symbols, use a list of arrays.

shape
[list or str, default='circle'] List of symbols used to localized the detected spots in the image, among *circle*, *square* or *polygon*. One symbol per array in *spots*. If *shape* is a string, the same symbol is used for every elements of 'spots'.

radius
[list or int or float, default=3] List of yx radii of the detected spots, in pixel. One radius per array in *spots*. If *radius* is a scalar, the same value is applied for every elements of *spots*.

color
[list or str, default='red'] List of colors of the detected spots. One color per array in *spots*. If *color* is a string, the same color is applied for every elements of *spots*.

linewidth
[list or int, default=1] List of widths or width of the border symbol. One integer per array in *spots*. If *linewidth* is an integer, the same width is applied for every elements of *spots*.

fill
[list or bool, default=False] List of boolean to fill the symbol of the detected spots. If *fill* is a boolean, it is applied for every symbols.

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

title
[str, optional] Title of the image.

framesize
[tuple, default=(15, 10)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

remove_frame
[bool, default=True] Remove axes and frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

bigfish.plot.**plot_reference_spot**(*reference_spot*, *rescale=False*, *contrast=False*, *title=None*, *framesize=(5, 5)*, *remove_frame=True*, *path_output=None*, *ext='png'*, *show=True*)

Plot the selected yx plan of the selected dimensions of an image.

Parameters

reference_spot
[np.ndarray] Spot image with shape (z, y, x) or (y, x).

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

title
[str, optional] Title of the image.

framesize
[tuple, default=(5, 5)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

remove_frame
[bool, default=True] Remove axes and frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

`bigfish.plot.`**`plot_elbow`**(*images*, *voxel_size=None*, *spot_radius=None*, *log_kernel_size=None*, *minimum_distance=None*, *title=None*, *framesize=(5, 5)*, *size_title=20*, *size_axes=15*, *size_legend=15*, *path_output=None*, *ext='png'*, *show=True*)

Plot the elbow curve that allows an automated spot detection.

Parameters

images
[list] List of ndarrays with shape (z, y, x) or (y, x). The same threshold is applied to every images.

voxel_size
[int or float or tuple or list, optional] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

spot_radius
[int or float or tuple or list, optional] Radius of the spot, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions. Not used if 'log_kernel_size' and 'minimum_distance' are provided.

log_kernel_size
[int or float or tuple or list, optional] Size of the LoG kernel. It equals the standard deviation (in pixels) used for the gaussian kernel (one for each dimension). One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same standard deviation is applied to every dimensions. If None, we estimate it with the voxel size and spot radius.

minimum_distance
[int or float or tuple or list, optional] Minimum distance (in pixels) between two spots we want to be able to detect separately. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same distance is applied to every dimensions. If None, we estimate it with the voxel size and spot radius.

title
[str, optional] Title of the plot.

framesize
[tuple, default=(5, 5)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

size_title
[int, default=20] Size of the title.

size_axes
[int, default=15] Size of the axes label.

size_legend
[int, default=15] Size of the legend.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

bigfish.plot.**plot_elbow_colocalized**(*spots_1*, *spots_2*, *voxel_size*, *threshold_max=None*, *title=None*, *framesize=(5, 5)*, *size_title=20*, *size_axes=15*, *size_legend=15*, *path_output=None*, *ext='png'*, *show=True*)

Plot the elbow curve that allows an automated colocalized spot detection.

Parameters

spots_1
[np.ndarray] Coordinates of the spots with shape (nb_spots, 3) or (nb_spots, 2).

spots_2
[np.ndarray] Coordinates of the spots with shape (nb_spots, 3) or (nb_spots, 2).

voxel_size
[int or float or tuple or list] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

threshold_max
[int or float, optional] Maximum threshold value to consider.

title
[str, optional] Title of the plot.

framesize
[tuple, default=(5, 5)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

size_title
[int, default=20] Size of the title.

size_axes
[int, default=15] Size of the axes label.

size_legend
[int, default=15] Size of the legend.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool] Show the figure or not.

3.16 Segmentation plot

Functions to visualize segmentation results.

Visualize segmented instances:

- *bigfish.plot.plot_segmentation()*
- *bigfish.plot.plot_segmentation_boundary()*

Compare segmented instances with a ground truth:

- *bigfish.plot.plot_segmentation_diff()*

bigfish.plot.**plot_segmentation**(*image*, *mask*, *rescale=False*, *contrast=False*, *title=None*, *framesize=(15, 10)*, *remove_frame=True*, *path_output=None*, *ext='png'*, *show=True*)

Plot result of a 2-d segmentation, with labelled instances if available.

Parameters

image
[np.ndarray] A 2-d image with shape (y, x).

mask
[np.ndarray] A 2-d image with shape (y, x).

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

title
[str, optional] Title of the image.

framesize
[tuple, default=(15, 10)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

remove_frame
[bool, default=True] Remove axes and frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

bigfish.plot.**plot_segmentation_boundary**(*image*, *cell_label=None*, *nuc_label=None*, *boundary_size=1*, *rescale=False*, *contrast=False*, *title=None*, *framesize=(10, 10)*, *remove_frame=True*, *path_output=None*, *ext='png'*, *show=True*)

Plot the boundary of the segmented objects.

Parameters

image
[np.ndarray] A 2-d image with shape (y, x).

cell_label
[np.ndarray, optional] A 2-d image with shape (y, x).

nuc_label
[np.ndarray, optional] A 2-d image with shape (y, x).

boundary_size
[int, default=1] Width of the cell and nucleus boundaries, in pixel.

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

title
[str, optional] Title of the image.

framesize
[tuple, default=(10, 10)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

remove_frame
[bool, default=True] Remove axes and frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

bigfish.plot.**plot_segmentation_diff**(*image*, *mask_pred*, *mask_gt*, *rescale=False*, *contrast=False*, *title=None*, *framesize=(15, 10)*, *remove_frame=True*, *path_output=None*, *ext='png'*, *show=True*)

Plot segmentation results along with ground truth to compare.

Parameters

image
[np.ndarray] Image with shape (y, x).

mask_pred
[np.ndarray] Image with shape (y, x).

mask_gt
[np.ndarray] Image with shape (y, x).

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

title
[str, optional] Title of the plot.

framesize
[tuple, default=(15, 10)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

remove_frame
[bool, default=True] Remove axes and frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

3.17 Single-cell plot

Function to visualize coordinate representation of individual cells:

- *bigfish.plot.plot_cell()*
- *bigfish.plot.plot_cell_coordinates()*

bigfish.plot.**plot_cell**(*ndim*, *cell_coord=None*, *nuc_coord=None*, *rna_coord=None*, *foci_coord=None*, *other_coord=None*, *image=None*, *cell_mask=None*, *nuc_mask=None*, *boundary_size=1*, *title=None*, *remove_frame=True*, *rescale=False*, *contrast=False*, *framesize=(15, 10)*, *path_output=None*, *ext='png'*, *show=True*)

Plot image and coordinates extracted for a specific cell.

Parameters

ndim
[{2, 3}] Number of spatial dimensions to consider in the coordinates.

cell_coord
[np.ndarray, optional] Coordinates of the cell border with shape (nb_points, 2). If None, coordinate representation of the cell is not shown.

nuc_coord
[np.ndarray, optional] Coordinates of the nucleus border with shape (nb_points, 2).

rna_coord
[np.ndarray, optional] Coordinates of the detected spots with shape (nb_spots, 4) or (nb_spots, 3). One coordinate per dimension (zyx or yx dimensions) plus the index of the cluster assigned to the spot. If no cluster was assigned, value is -1. If only coordinates of spatial dimensions are available, only centroid of foci can be shown.

foci_coord
[np.ndarray, optional] Array with shape (nb_foci, 5) or (nb_foci, 4). One coordinate per dimension for the foci centroid (zyx or yx dimensions), the number of spots detected in the foci and its index.

other_coord
[np.ndarray, optional] Coordinates of the detected elements with shape (nb_elements, 3) or (nb_elements, 2). One coordinate per dimension (zyx or yx dimensions).

image
[np.ndarray, optional] Original image of the cell with shape (y, x). If None, original image of the cell is not shown.

cell_mask
[np.ndarray, optional] Mask of the cell.

nuc_mask
[np.ndarray, optional] Mask of the nucleus.

boundary_size
[int, default=1] Width of the cell and nucleus boundaries, in pixel.

title
[str, optional] Title of the image.

remove_frame
[bool, default=True] Remove axes and frame.

rescale
[bool, default=False] Rescale pixel values of the image (made by default in matplotlib).

contrast
[bool, default=False] Contrast image.

framesize
[tuple, default=(15, 10)] Size of the frame used to plot with `plt.figure(figsize=framesize)`.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

`bigfish.plot.`**`plot_cell_coordinates`**(*ndim*, *cell_coord*, *nuc_coord*, *rna_coord*, *titles=None*, *remove_frame=True*, *framesize=(10, 5)*, *path_output=None*, *ext='png'*, *show=True*)

Plot cell coordinates for one or several cells.

Parameters

ndim
[{2, 3}] Number of spatial dimensions to consider in the coordinates.

cell_coord
[np.ndarray or list] Coordinates or list of coordinates of the cell border with shape (nb_points, 2).

nuc_coord
[np.ndarray or list] Coordinates or list of coordinates of the nucleus border with shape (nb_points, 2).

rna_coord
[np.ndarray or list] Coordinates or list of coordinates of the detected spots with shape (nb_spots, 3) or (nb_spots, 2). One coordinate per dimension (zyx or yx dimensions).

titles
[str or list, optional] Title or list of titles.

remove_frame
[bool, default=True] Remove axes and frame.

framesize
[tuple, default=(10, 5)] Size of the frame.

path_output
[str, optional] Path to save the image (without extension).

ext
[str or list, default='png'] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

show
[bool, default=True] Show the figure or not.

3.18 Utility functions

3.18.1 Check input quality

- *bigfish.stack.check_array()*
- *bigfish.stack.check_df()*
- *bigfish.stack.check_parameter()*
- *bigfish.stack.check_range_value()*

bigfish.stack.**check_array**(*array*, *ndim=None*, *dtype=None*, *allow_nan=True*)

Full safety check of an array.

Parameters

array
[np.ndarray] Array to check.

ndim
[int or List[int]] Number of dimensions expected.

dtype
[type or List[type]] Types expected.

allow_nan
[bool] Allow NaN values or not.

Returns

_
[bool] Assert if the array is well formatted.

bigfish.stack.**check_df**(*df*, *features=None*, *features_without_nan=None*)

Full safety check of a dataframe.

Parameters

df
[pd.DataFrame or pd.Series] Dataframe or Series to check.

features
[List[str]] Names of the expected features.

features_without_nan
[List[str]] Names of the features to check for the missing values

Returns

_
[bool] Assert if the dataframe is well formatted.

bigfish.stack.**check_parameter**(***kwargs*)

Check dtype of the function's parameters.

Parameters

kwargs
[Type or Tuple[Type]] Map of each parameter with its expected dtype.

Returns

_
[bool] Assert if the array is well formatted.

bigfish.stack.**check_range_value**(*array*, *min_=None*, *max_=None*)

Check the support of the array.

Parameters

array
[np.ndarray] Array to check.

min_
[int] Minimum value allowed.

max_
[int] Maximum value allowed.

Returns

_
[bool] Assert if the array has the right range of values.

3.18.2 Get constant values

- *bigfish.stack.get_margin_value()*
- *bigfish.stack.get_eps_float32()*

bigfish.stack.**get_margin_value**()

Return the margin pixel around a cell coordinate used to define its bounding box.

Returns

_
[int] Margin value (in pixels).

bigfish.stack.**get_eps_float32**()

Return the epsilon value for a 32 bit float.

Returns

_
[np.float32] Epsilon value.

3.18.3 Load and check stored data

- *bigfish.stack.load_and_save_url()*
- *bigfish.stack.check_hash()*
- *bigfish.stack.compute_hash()*
- *bigfish.stack.check_input_data()*

bigfish.stack.**load_and_save_url**(*remote_url*, *directory*, *filename=None*)

Download remote data and save them

Parameters

remote_url
[str] Remote url of the data to download.

directory
[str] Directory to save the download content.

filename
[str] Filename of the object to save.

Returns

path
[str] Path of the downloaded file.

bigfish.stack.**check_hash**(*path*, *expected_hash*)

Check hash value of a file.

Parameters

path
[str] Path of the file to check.

expected_hash
[str] Expected hash value.

Returns

_
[bool] True if hash values match.

bigfish.stack.**compute_hash**(*path*)

Compute sha256 hash of a file.

Parameters

path
[str] Path to read the file.

Returns

sha256
[str] Hash value of the file.

bigfish.stack.**check_input_data**(*input_directory*, *input_segmentation=False*)

Check input images exists and download them if necessary.

Parameters

input_directory
[str] Path of the image directory.

input_segmentation
[bool] Check 2-d example images for segmentation.

3.18.4 Compute moving average

- *bigfish.stack.moving_average()*
- *bigfish.stack.centered_moving_average()*

bigfish.stack.**moving_average**(*array*, *n*)

Compute a trailing average.

Parameters

array
[np.ndarray] Array used to compute moving average.

n
[int] Window width of the moving average.

Returns

results
[np.ndarray] Moving average values.

bigfish.stack.**centered_moving_average**(*array*, *n*)

Compute a centered moving average.

Parameters

array
[np.ndarray] Array used to compute moving average.

n
[int] Window width of the moving average.

Returns

results
[np.ndarray] Centered moving average values.

3.18.5 Convert pixels and nanometers

- *bigfish.detection.convert_spot_coordinates()*
- *bigfish.detection.get_object_radius_pixel()*
- *bigfish.detection.get_object_radius_nm()*

bigfish.detection.**convert_spot_coordinates**(*spots*, *voxel_size*)

Convert spots coordinates from pixel to nanometer.

Parameters

spots
[np.ndarray] Coordinates of the detected spots with shape (nb_spots, 3) or (nb_spots, 2).

voxel_size
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

Returns

spots_nanometer
[np.ndarray] Coordinates of the detected spots with shape (nb_spots, 3) or (nb_spots, 3), in nanometer.

bigfish.detection.**get_object_radius_pixel**(*voxel_size_nm*, *object_radius_nm*, *ndim*)

Convert the object radius in pixel.

When the object considered is a spot this value can be interpreted as the standard deviation of the spot PSF, in pixel. For any object modelled with a gaussian signal, this value can be interpreted as the standard deviation of the gaussian.

Parameters

voxel_size_nm
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

object_radius_nm
[int, float, Tuple(int, float) or List(int, float)] Radius of the object, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions.

ndim
[int] Number of spatial dimension to consider.

Returns

object_radius_px
[Tuple[float]] Radius of the object in pixel, one element per dimension (zyx or yx dimensions).

bigfish.detection.**get_object_radius_nm**(*voxel_size_nm*, *object_radius_px*, *ndim*)

Convert the object radius in nanometer.

When the object considered is a spot this value can be interpreted as the standard deviation of the spot PSF, in nanometer. For any object modelled with a gaussian signal, this value can be interpreted as the standard deviation of the gaussian.

Parameters

voxel_size_nm
[int, float, Tuple(int, float) or List(int, float)] Size of a voxel, in nanometer. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same value is applied to every dimensions.

object_radius_px
[int, float, Tuple(int, float) or List(int, float)] Radius of the object, in pixel. One value per spatial dimension (zyx or yx dimensions). If it's a scalar, the same radius is applied to every dimensions.

ndim
[int] Number of spatial dimension to consider.

Returns

object_radius_nm
[Tuple[float]] Radius of the object in nanometer, one element per dimension (zyx or yx dimensions).

3.18.6 Extract a spot image

- *bigfish.detection.get_spot_volume()*
- *bigfish.detection.get_spot_surface()*

bigfish.detection.**get_spot_volume**(*image*, *spot_z*, *spot_y*, *spot_x*, *radius_z*, *radius_yx*)

Get a subimage of a detected spot in 3 dimensions.

Parameters

image
[np.ndarray] Image with shape (z, y, x).

spot_z
[int or float] Coordinate of the detected spot along the z axis.

spot_y
[int or float] Coordinate of the detected spot along the y axis.

spot_x
[int or float] Coordinate of the detected spot along the x axis.

radius_z
[int or float] Radius in pixel of the detected spot, along the z axis.

radius_yx
[int or float] Radius in pixel of the detected spot, on the yx plan.

Returns

image_spot
[np.ndarray] Reference spot in 3-d.

_
[Tuple[int]] Lower zyx coordinates of the crop.

bigfish.detection.**get_spot_surface**(*image*, *spot_y*, *spot_x*, *radius_yx*)

Get a subimage of a detected spot in 2 dimensions.

Parameters

image
[np.ndarray] Image with shape (y, x).

spot_y
[int or float] Coordinate of the detected spot along the y axis.

spot_x
[int or float] Coordinate of the detected spot along the x axis.

radius_yx
[int or float] Radius in pixel of the detected spot, on the yx plan.

Returns

image_spot
[np.ndarray] Reference spot in 2-d.

_
[Tuple[int]] Lower yx coordinates of the crop.

3.18.7 Format and save plots

- *bigfish.plot.save_plot()*
- *bigfish.plot.get_minmax_values()*
- *bigfish.plot.create_colormap()*

bigfish.plot.**save_plot**(*path_output*, *ext*)

Save the plot.

Parameters

path_output
[str] Path to save the image (without extension).

ext
[str or List[str]] Extension used to save the plot. If it is a list of strings, the plot will be saved several times.

bigfish.plot.**get_minmax_values**(*tensor*)

Get the minimum and maximum value of the image according to its dtype.

Parameters

tensor
[np.ndarray] A 2-d, 3-d or 5-d tensor with shape (y, x), (z, y, x) or (r, c, z, y, x) respectively.

Returns

vmin
[int] Minimum value display in the plot.

vmax
[int] Maximum value display in the plot.

bigfish.plot.**create_colormap**()

Create a shuffled colormap to display segmentation masks.

Returns

colormap
[ListedColormap object] Colormap for matplotlib.

3.19 Support

If you have any question relative to the package, please open an issue on Github.

3.20 Citation

If you exploit this package for your work, please cite:

```
Arthur Imbert, Wei Ouyang, Adham Safieddine, Emeline Coleno, Christophe
Zimmer, Edouard Bertrand, Thomas Walter, Florian Mueller. FISH-quant v2:
a scalable and modular analysis tool for smFISH image analysis. bioRxiv
(2021) https://doi.org/10.1101/2021.07.20.453024
```

INDEX

G

I

L

M

P

R

S

T

U