Package 'cytomapper'

August 9, 2022

Version 1.8.0

Title Visualization of highly multiplexed imaging data in R

Description Highly multiplexed imaging acquires the single-cell expression of selected proteins in a spatially-resolved fashion. These measurements can be visualised across multiple length-scales. First, pixel-level intensities represent the spatial distributions of feature expression with highest resolution. Second, after segmentation, expression values or cell-level metadata (e.g. cell-type information) can be visualised on segmented cell areas. This package contains functions for the visualisation of multiplexed read-outs and cell-level information obtained by multiplexed imaging technologies. The main functions of this package allow 1. the visualisation of pixel-level information across multiple channels, 2. the display of cell-level information (expression and/or metadata) on segmentation masks and 3. gating and visualisation of single cells.

License GPL (>= 2)

Depends R (>= 4.0), EBImage, SingleCellExperiment, methods

Imports S4Vectors, BiocParallel, HDF5Array, DelayedArray, RColorBrewer, viridis, utils, SummarizedExperiment, tools, graphics, raster, grDevices, stats, ggplot2, ggbeeswarm, svgPanZoom, svglite, shiny, shinydashboard, matrixStats, rhdf5, nnls

Suggests BiocStyle, knitr, rmarkdown, markdown, cowplot, testthat, shinytest

biocViews ImmunoOncology, Software, SingleCell, OneChannel, TwoChannel, MultipleComparison, Normalization, DataImport

VignetteBuilder knitr

URL https://github.com/BodenmillerGroup/cytomapper

BugReports https://github.com/BodenmillerGroup/cytomapper/issues

RoxygenNote 7.1.2

Encoding UTF-8

git_url https://git.bioconductor.org/packages/cytomapper

git_branch RELEASE_3_15

git_last_commit d6546b7

git_last_commit_date 2022-04-26

Date/Publication 2022-08-09

Author Nils Eling [aut, cre] (<https://orcid.org/0000-0002-4711-1176>),
Nicolas Damond [aut] (<https://orcid.org/0000-0003-3027-8989>),
Tobias Hoch [ctb]

Maintainer Nils Eling <nils.eling@dqbm.uzh.ch>

R topics documented:

`compImage` *Performs channel compensation on multi-channel images*

Description

Corrects the intensity spillover between neighbouring channels of multi-channel images using a non-negative least squares approach.

Usage

```
compImage(object, sm, overwrite = FALSE, BPPARAM = SerialParam())
```

Arguments

object	a CytoImageList object containing pixel intensities for all channels. The channelNames must be in the form of (mt)(mass)Di (e.g. Sm152Di for Samarium isotope with the atomic mass 152) and match with the column names in sm.
sm	numeric matrix containing the spillover estimated between channels. The column names must be of the form (mt)(mass)Di (e.g. Sm152Di for Samarium isotope with the atomic mass 152) and match to the channelNames of object.
overwrite	(for images stored on disk) should the original image array be overwritten by the compensated image array? By default (overwrite = FALSE), a new entry called "XYZ_comp" will be written to the .h5 file (see below).
BPPARAM	parameters for parallelised processing.

Value

returns the compensated pixel intensities in form of a CytoImageList object.

The input object

The channelNames of object need to match the column names of sm. To adapt the spillover matrix accordingly, please use the adaptSpillmat function.

Images stored on disk

Image compensation also works for images stored on disk. By default, the compensated images are stored as a second entry called "XYZ_comp" in the .h5 file. Here "XYZ" specifies the name of the original entry. By storing the compensated next to the original images on disk, space usage increases. To avoid storing duplicated data, one can specify overwrite = TRUE, therefore deleting the original images and only storing the compensated images. However, the original images cannot be accessed anymore after compensation.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

References

Chevrier, S. et al., Compensation of Signal Spillover in Suspension and Imaging Mass Cytometry., Cell Systems 2018 6(5):612-620.e5

See Also

nnls, for the underlying algorithm

compCytof, for how to compensate single-cell data

Examples

```
data("pancreasImages")

# Generate example spillover matrix
metals <- c("Dy161Di", "Dy162Di", "Dy163Di", "Dy164Di", "Ho165Di")
sm <- matrix(c(1, 0.033, 0.01, 0.007, 0,
               0.016, 1, 0.051, 0.01, 0,
               0.004, 0.013, 1, 0.023, 0,
               0.005, 0.008, 0.029, 1, 0.006,
               0, 0, 0, 0.001, 1), byrow = TRUE,
             ncol = 5, nrow = 5,
             dimnames = list(metals, metals))

# Rename channels - just used as example
channelNames(pancreasImages) <- metals

# Perform channel spillover
comp_images <- compImage(pancreasImages, sm)
```

CytoImageList-class *S4 class for list of images*

Description

This class facilitates the handling of multiple one- or multi-channel images. It inherits from SimpleList setting elementType="Image". Therefore, each slot contains an either one- or multi-dimensional array in form of an Image object.

Usage

```
CytoImageList(
  ...,
  on_disk = FALSE,
  h5FilesPath = NULL,
  BPPARAM = SerialParam()
)
```

Arguments

...	A list of images (or coercible to a list) or individual images
on_disk	Logical indicating if images in form of HDF5Array objects (as .h5 files) should be stored on disk rather than in memory.
h5FilesPath	path to where the .h5 files for on disk representation are stored. This path needs to be defined when on_disk = TRUE. When files should only temporarily be stored on disk, please set h5FilesPath = getHDF5DumpDir()
BPPARAM	parameters for parallelised processing. This is only recommended for very large images. See MulticoreParam for information on how to use multiple cores for parallelised processing.

Details

Similar to the `Image` class, the first two dimensions of each entry indicate the spatial dimension of the image. These can be different for each entry. The third dimension indicates the number of channels per Image. Each entry in the CytoImageList class object must contain the same number of channels. Here, each channel represents pixel values indicating measurement intensities or in case of segmentation masks the cells' ID. The CytoImageList class therefore only supports a Grayscale colormode (see `colormode`) representation of each individual image.

The class further contains an `elementMetadata` slot that stores image-level meta information. This slot should be accessed using the `mcols` accessor function.

Value

A CytoImageList object

Restrictions on entry names

The CytoImageList class only supports unique entry names to avoid duplicated images. Names of a CytoImageList object can be get and set via `names(x)`, where `x` is a CytoImageList object. Furthermore, only named or unnamed CytoImageList objects are allowed. Partially named objects causing empty or NA names return an error.

Coercion

Coercion to and from list, `SimpleList` and `List`:

as.list(x), as(x, "SimpleList"), as(x, "SimpleList"): Coercion from a CytoImageList object `x`

as(x, "CytoImageList"): Coercion from a list, SimpleList or List object `x` to anCytoImageList object

Looping

While `lapply` and `mapply` return regular list objects, `endoapply` and `mendoapply` return CytoImageList objects.

On disk representation

When setting `on_disk = TRUE` and specifying the `h5FilesPath`, images are stored on disk. To convert back to an in-memory `CytoImageList` object, one can call `CytoImageList(on_disk_IL, on_disk = FLASE)`.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

See Also

Image, for further image analysis tools.

SimpleList, for basics functions to handle SimpleList objects

?loadImages, for reading images into a CytoImageList object

?"CytoImageList-naming", for setting and getting image and channel names

?"CytoImageList-subsetting", for subsetting and accessor functions

Examples

```
# Creation of CytoImageList
u <- matrix(rbinom(100, 10, 0.5), ncol=10, nrow=10)
v <- matrix(rbinom(100, 10, 0.5), ncol=10, nrow=10)
IL1 <- CytoImageList(image1 = Image(u), image2 = Image(v))

# Coercion
as.list(IL1)
as(IL1, "SimpleList")
as(list(image1 = Image(u), image2 = Image(v)), "CytoImageList")

# On disk representation
IL1 <- CytoImageList(image1 = Image(u), image2 = Image(v),
                     on_disk = TRUE,
                     h5FilesPath = HDF5Array::getHDF5DumpDir())
```

CytoImageList-manipulation

Manipulating CytoImageList objects

Description

Methods to change pixel values in CytoImageList objects. In the following sections, `object` is a CytoImageList object containing one or multiple channels.

Value

A CytoImageList object containing the manipulated Images.

Image scaling

In some cases, images need to be scaled by a constant (e.g. 2^16-1 = 65535) `value` to revert them back to the original pixel values after reading them in.

`scaleImages(object, value)` Scales all images in the CytoImageList `object` by `value`. Here `value` needs to be a single numeric or a numeric vector of the same length as `object`.

Image normalization

Linear scaling of the intensity values of each Image contained in a CytoImageList `object` to a specific range. Images can either be scaled to the minimum/maximum value per channel or across all channels (default `separateChannels = TRUE`). Also, images can be scaled to the minimum/maximum value per image or across all images (default `separateImages = FALSE`). The latter allows the visual comparison of intensity values across images.

To clip the images before normalization, the `inputRange` can be set. The `inputRange` either takes NULL (default), a vector of length 2 specifying the clipping range for all channels or a list where each named entry contains a channel-specific clipping range.

Image normalization also works for images stored on disk. By default, the normalized images are stored as a second entry called "XYZ_norm" in the .h5 file. Here "XYZ" specifies the name of the original entry. By storing the normalized next to the original images on disk, space usage increases. To avoid storing duplicated data, one can specify `overwrite = TRUE`, therefore deleting the original images and only storing the normalized images. However, the original images cannot be accessed anymore after normalisation.

`normalize(object, separateChannels = TRUE, separateImages = FALSE,ft = c(0, 1), inputRange = NULL, overwrite = FALSE)`:

`object`: A CytoImageList object

`separateChannels`: Logical if pixel values should be normalized per channel (default) or across all channels.

`separateImages`: Logical if pixel values should be normalized per image or across all images (default).

`ft`: Numeric vector of 2 values, target minimum and maximum intensity values after normalization (see `normalize`).

`inputRange`: Numeric vector of 2 values, sets the absolute clipping range of the input intensity values (see `normalize`). Alternatively a names list where each entry corresponds to a channel-specific clipping range.

`overwrite`: Only relevant when images are kept on disk. By specifying `overwrite = TRUE`, the normalized images will overwrite the original images in the .h5 file, therefore reducing space on disk. However, the original images cannot be accessed anymore after normalization. If `overwrite = FALSE` (default), the normalized images are added as a new entry called "XYZ_norm" to the .h5 file.)

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

See Also

`normalize` for details on Image normalization

Examples

```
data(pancreasImages)
```

```
# Scale images to create segmentation masks
cur_files <- list.files(system.file("extdata", package = "cytomapper"),
                        pattern = "mask.tiff", full.names = TRUE)
x <- loadImages(cur_files)
# Error when running plotCells(x)
# Therefore scale to account for 16 bit encoding
x <- scaleImages(x, 2^16 - 1)
plotCells(x)

# Default normalization
x <- normalize(pancreasImages)
plotPixels(x, colour_by = c("H3", "CD99"))

# Setting the clipping range
x <- normalize(x, inputRange = c(0, 0.9))
plotPixels(x, colour_by = c("H3", "CD99"))

# Setting the clipping range per channel
x <- normalize(pancreasImages,
               inputRange = list(H3 = c(0, 70), CD99 = c(0, 100)))
plotPixels(x, colour_by = c("H3", "CD99"))

# Normalizing per image
x <- normalize(pancreasImages, separateImages = TRUE)
plotPixels(x, colour_by = c("H3", "CD99"))
```

`CytoImageList-naming` *Getting and setting the channel and image names*

Description

Methods to get and set the names of individual channels or the names of individual images.

Setting and getting the channel names

In the following code, `x` is a CytoImageList object containing one or multiple channels. The channel names can be replaced by `value`, which contains a character vector of the same length as the number of channels in the images.

`channelNames(x)` Returns the names of all channels stored in `x`

`channelNames(x) <- value` Replaces the channel names of `x` with `values`. For this, `value` needs to have the same length as the number of channels in `x`

Setting and getting the image names

Here, `x` is a CytoImageList object. The element names can be replaced by `value`, which contains a character vector of the same length as the number of images. In case of the CytoImageList object, elements are always images.

`names(x)` Returns the names of all images stored in `x`

`names(x) <- value` Replaces the image names of `x` with `value`. For this, `value` needs to have the same length as `x`

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

Examples

```
data("pancreasImages")

# Get channel and image names
channelNames(pancreasImages)
names(pancreasImages)

# Set channel and image names
channelNames(pancreasImages) <- paste0("marker", 1:5)
names(pancreasImages) <- paste0("image", 1:3)
```

CytoImageList-subsetting

General subsetting methods for CytoImageList objects

Description

These getter and setter functions are used to extract, replace and merge entries in a `CytoImageList` object.

Arguments

`x, y`	`CytoImageList` objects
`i`	integer, logical, character or vector of such indicating which element(s) to replace or extract
`value`	a `CytoImageList` or `Image` object

Value

A CytoImageList object

Setting and getting images

Functions to extract and replace elements (= images) of a CytoImageList object. In the following code, `x` is a CytoImageList object. The parameter `i` indicates the element(s) of `x` that should be returned or replaced. Replacement is done by `value`, which takes a CytoImageList or Image object. If `length(i) > 0`, `value` has to be a CytoImageList object of `length(i)`, otherwise `value` allows a CytoImageList object of length 1 or an Image object. If an Image object is provided, only the image entry in the CytoImageList object is replaced, not the corresponding elementMetadata entry.

`getImages(x, i)` Returns image(s) indicated by `i` of the CytoImageList object `x`

`setImages(x, i) <- value` Replaces the image(s) indicated by `i` of the CytoImageList object `x` with `value`. For this, `value` needs to have the same length as `i`

These setter and getter functions are the recommended way of extracting and replacing images in a CytoImageList object. Alternatively, the standard operations via `` `[` ``, `` `[[` ``, `` `[<-` `` and `` `[[<-` `` can be performed (see `?List` for S4Vectors subsetting functionality). However, these operations do not change element names during replacment calls. The `setImages()` function makes sure that element names are replaced if `value` is named or if `i` is a character or vector of characters.

Getting and setting channels

Functions to extract and replace channels of a CytoImageList object. Here, `x` is a CytoImageList object. The parameter `i` indicates the channels of `x` that should be returned or replaced. Replacement is done by `value`, which takes a CytoImageList object. The CytoImageList object `value` needs to have the same length as `x`. Furthermore, the number of channels in `value` should be identical to `length(i)`.

`getChannels(x, i)` Returns channel(s) indicated by `i` of the CytoImageList object `x`

`setChannels(x, i) <- value` Replaces the channel(s) indicated by `i` of the CytoImageList object `x` with `value`. For this, `value` needs to have the same length as `i` and the same number of channels as `length(i)`.

The `setChannels()` setter function does not allow adding new channels to the CytoImageList object. For this operation, the `mergeChannels` function was implemented (see below).

Merging images

Merging images is possible by merging two or more CytoImageList objects via:

`c(x, y)` Returns an composite CytoImageList object with elements of both CytoImageList objects `x` and `y`. More than two CytoImageList objects can be merged in that way.

Merging channels

Merging channels is possible via:

`mergeChannels(x, y, h5FilesPath = NULL):` Returns a CytoImageList in which the channels of the CytoImageList object `y` have been appended to the channels of the CytoImageList object `x`. Only channels of two CytoImageList objects can be merged in that way. The `h5FilesPath` argument can be ignored unless images are stored on disk. To avoid overriding the .h5 files, one needs to specify a new location where the merged images are stored on disk.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

Examples

```
data("pancreasImages")

# Get images
getImages(pancreasImages, 1)
getImages(pancreasImages, "E34_imc")
getImages(pancreasImages, 1:2)
getImages(pancreasImages, c("E34_imc", "G01_imc"))
getImages(pancreasImages, grepl("E34_imc", names(pancreasImages)))

# Set images
setImages(pancreasImages, 1) <- pancreasImages[1]
setImages(pancreasImages, "J02_imc") <- pancreasImages[1]
setImages(pancreasImages, "J02_imc") <- NULL

# Get channels
getChannels(pancreasImages, 1)
getChannels(pancreasImages, "CD99")
getChannels(pancreasImages, c("CD99", "PIN"))

# Set channels
channel1 <- getChannels(pancreasImages, 1)
setChannels(pancreasImages, 1) <- channel1
channelPIN <- getChannels(pancreasImages, "PIN")
setChannels(pancreasImages, "CD8a") <- channelPIN
setChannels(pancreasImages, "CD8a") <- NULL

# Merge images
data("pancreasImages")
c(pancreasImages[c(1,3)], pancreasImages[2])

# Merge channels
channel12 <- getChannels(pancreasImages, c(1,2))
channel34 <- getChannels(pancreasImages, c(3,4))
mergeChannels(channel12, channel34)

# Merge channels on disk
cur_images <- CytoImageList(pancreasImages,
                    on_disk = TRUE,
                    h5FilesPath = HDF5Array::getHDF5DumpDir())
channel12 <- getChannels(cur_images, c(1,2))
channel34 <- getChannels(cur_images, c(3,4))

# This will overwrite the initial .h5 files
mergeChannels(channel12, channel34,
                h5FilesPath = HDF5Array::getHDF5DumpDir())
```

---

cytomapperShiny *Shiny application to visualise gated cells on images*

Description

This shiny application allows users to gate cells based on their raw or transformed expression values and visualises gated cells on their corresponding images.

Usage

```
cytomapperShiny(
  object,
  mask = NULL,
  image = NULL,
  cell_id = NULL,
  img_id = NULL,
  ...
)
```

Arguments

object	a SingleCellExperiment object.
mask	(optional) a CytoImageList containing single-channel Image objects.
image	(optional) a CytoImageList object containing single or multi-channel Image objects.
cell_id	character specifying the colData(object) entry, in which the integer cell IDs are stored. These IDs should match the integer pixel values in the segmentation mask object (mask).
img_id	character specifying the colData(object) and mcols(mask) and/or mcols(image) entry, in which the image IDs are stored.
...	parameters passed to the plotCells or plotPixels function.

Value

A Shiny app object for hierarchical gating of cells

User inputs

This function requires at least a SingleCellExperiment input object. Gating is performed on cell-specific marker counts stored in the assay slots of the object. These can either be raw counts (usually stored in the counts slot) or transformed/scaled counts stored in other assay slots. Gating can only be performed sample-wise; therefore, even if mask or image are not specified, img_id needs to point to the colData entry storing unique sample IDs. Furthermore, the cell_id entry is required to identify cells during hierarchical gating.

If mask is specified, marker expression values and selected cells will be visualised by using the plotCells function. To visualise pixel-level information with plotPixels, the user has to further provide multi-channel images stored in a CytoImageList.

The user interface (UI)

The UI's body is composed of two tabs (*Scatter Plots* and *Images*). The side bar contains the *General Control* and *Plots* panels. Both panels can be collapsed; however only one can be expanded again.. Using the control panel, the user can set the number of plots, which sample to display and which `assay` slot to use for plotting expression. The *Plots* panel can be used to select up to two markers per plot. A one-marker selection is displayed as violin and jittered points, two markers are shown as scatter plot. To define a subset of cells, the user can draw gates on the plots. When using multiple plots, the gate applied in a plot will define the subset of cells displayed in the next plot. The number of plots, the marker selection and the applied gates will be transferred when switching between samples. When switching markers or assay slots, the gates will be removed.

By switching to the `Images` tab, the user can inspect marker expression levels and the result of the gating. This requires either an entry to `mask` and (optionally) `image`. Depending on the input, the left image visualises the expression of the selected markers by using `plotCells` (if `mask` is supplied) or `plotPixels` (if `image` is supplied). Displayed markers can be changed using the dropdown menu. If `images` are provided, the user can change the contrast of the selected markers. The right image colors (`masks`) or outlines (`images`) gated cells. Both the left and the right image come with a zoom-in functionality.

The header section of the shiny allows to hide the side bar, to download selected cells (see below), or to display the `sessionInfo` output and the *Help* section (see below).

Download of gated cells

The user can download the selected cells in form of a `SingleCellExperiment` object. The output is a subset of the input `SingleCellExperiment` object. The downloaded object (in form of a .rds file) contains a new `colData` entry containing the label of the subset defined by the user (using the *Cell label* argument). The cell label is stored in `colData(object)$cytomapper_CellLabel`

The `metadata` slot of the `SingleCellExperiment` object will be converted to a list. It contains the original metadata (`metadata(object)$metadata`), gating parameters (e.g. `metadata(object)$cytomapper_gate_1`), `sessionInfo()` output (`metadata(object)$cytomapper_SessionInfo`), and the date (`metadata(object)$cytomapper_C`

Passing further parameters

To customise the visual output of `cytomapperShiny`, refer to `plotting-param`. Further arguments can be passed to the `plotCells` and `plotPixels` function. As an example: To avoid interpolation of output images, set `interpolate = FALSE`. Passing the parameter `interpolate = FALSE` will make images less blurry, which might be useful while using the zoom-in functionality of `cytomapperShiny` .

Getting further help

Please refer to the *Help* section found when clicking the '?' button in the upper right corner. The *Help* section contains a recommended workflow on how to use the app.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

Tobias Hoch (<tobias.hoch@dqbm.uzh.ch>)

See Also

plotCells and plotPixels for the main plotting functions

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
   # Load example data sets
   data("pancreasSCE")
   data("pancreasImages")
   data("pancreasMasks")

   # Use shiny with SCE object, images and masks
   cytomapperShiny(object = pancreasSCE, mask = pancreasMasks,
                   image = pancreasImages, cell_id = "CellNb",
                   img_id = "ImageNb")

   # Use shiny with SCE object and masks
   cytomapperShiny(object = pancreasSCE, mask = pancreasMasks,
                   cell_id = "CellNb", img_id = "ImageNb")

   # Use shiny with SCE object only
   cytomapperShiny(object = pancreasSCE,
                   cell_id = "CellNb", img_id = "ImageNb")
}
```

---

loadImages *Read images into CytoImageList object*

Description

Function to read in single- or multi-channel images from a specified path or file. The function returns a CytoImageList object containing one image per slot. Supported file extensions are: '.tiff', '.tif', '.png', '.jpeg', '.jpg', ".h5".

Usage

```
loadImages(
  x,
  pattern = NULL,
  on_disk = FALSE,
  h5FilesPath = NULL,
  name = NULL,
  BPPARAM = SerialParam(),
  ...
)
```

Arguments

x	The function takes a variety of possible character inputs: **A single file** Full path and file name of an individual image file. **A path** A path to where image files are located. **A vector of files** A character vector where each entry represents an individual file.
pattern	Character inputs of the following form: **A single character** A pattern to search for in the specified path (regular expressions are supported). **A character vector** Unique entries are matched against file names in the specified path.
on_disk	Logical indicating if images in form of HDF5Array objects (as .h5 files) should be stored on disk rather than in memory.
h5FilesPath	path to where the .h5 files for on disk representation are stored. This path needs to be defined when on_disk = TRUE. When files should only temporarily be stored on disk, please set h5FilesPath = getHDF5DumpDir()
name	(if reading in .h5 files) a single character, a character vector of length equal to the length of x or NULL. See details for how to set name.
BPPARAM	parameters for parallelised reading in of images. This is only recommended for very large images. See MulticoreParam for information on how to use multiple cores for parallelised processing.
...	arguments passed to the readImage function.

Value

A CytoImageList object

Reading in 16-bit integer images

To correctly read in the original integer values of 16-bit, the as.is = TRUE parameter needs to be added to the function call. This will prevent the readTIFF function to re-scale integer values.

Loading specific images

This function loads images via the readImage function and stores them in a CytoImageList object. In the simplest case, x is an image file name. If x is a path, the pattern argument can be used to select image names with certain patterns. For convenience, pattern also takes a vector of characters (e.g. a colData entry in a SingleCellExperiment object) to select by unique image names. Furthermore, a vector of image names can be provided to read in multiple images.

Reading in .h5 files

When reading in .h5 files by default the loadImages function will try to read in the dataset with the same name as the .h5 file from within the file. If datasets are stored with different names, the name argument must be specified. This can either be a single character if datasets across all files are named the same or a character vector of the same length as x indicating the dataset name within each .h5 file. By default, the images/datasets are not read into memory when stored in .h5 files.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>),

Nicolas Damond (<nicolas.damond@dqbm.uzh.ch>)

See Also

readImage, for reading in individual images.

Examples

```
# Providing a single file
single.image <- system.file("extdata/E34_mask.tiff", package = "cytomapper")
single.image <- loadImages(single.image)

# Providing a path and pattern
path.to.images <- system.file("extdata", package = "cytomapper")
image.list <- loadImages(path.to.images, pattern = "mask.tiff")

# Providing multiple patterns
data(pancreasSCE)
path.to.images <- system.file("extdata", package = "cytomapper")
image.list <- loadImages(path.to.images, pattern = pancreasSCE$MaskName)

# Providing multiple files
list.images <- list.files(system.file("extdata", package = "cytomapper"),
                          pattern = "_mask.tiff", full.names = TRUE)
image.list <- loadImages(list.images)

# On disk representation
path.to.images <- system.file("extdata", package = "cytomapper")
image.list <- loadImages(path.to.images, pattern = "mask.tiff",
                          on_disk = TRUE,
                          h5FilesPath = HDF5Array::getHDF5DumpDir())

# Parallel processing
path.to.images <- system.file("extdata", package = "cytomapper")
image.list <- loadImages(path.to.images, pattern = "mask.tiff",
                          BPPARAM = BiocParallel::MulticoreParam())
```

---

measureObjects *Compute morphological and intensity features from objects on images.*

Description

For each object (e.g. cell) identified by segmentation, the measureObjects function computes intensity features (also referred to as basic features; e.g. mean intensity), shape features (e.g. area), moment features (e.g. position) and haralick features. These features are returned in form of a SingleCellExperiment object.

Usage

```
measureObjects(
  mask,
  image,
  img_id,
  feature_types = c("basic", "shape", "moment"),
  basic_feature = "mean",
  basic_quantiles = NULL,
  shape_feature = c("area", "radius.mean"),
  moment_feature = c("cx", "cy", "majoraxis", "eccentricity"),
  haralick_feature = NULL,
  haralick_nbins = 32,
  haralick_scales = c(1, 2),
  BPPARAM = SerialParam()
)
```

Arguments

mask	a CytoImageList object containing single-channel Image or HDF5Array objects. Segmentation masks must contain integer pixel values where groups of pixels correspond to objects.
image	a CytoImageList object containing single or multi-channel Image or HDF5Array objects, where each channel indicates the measured pixel intensities.
img_id	character specifying the mcols(image) and mcols(mask) entry, in which the image IDs are stored.
feature_types	character vector or string indicating which features to compute. Needs to contain "basic". Optionally, "shape", "moment" and "haralick" are allowed. Default "basic", "shape" and "moment".
basic_feature	string indicating which intensity measurement per object and channel should be used to populate the counts(x) slot; where x is the returned SingleCellExperiment object. Default "mean" but "sd", "mad" and "q*" allowed. Here, * indicates the computed quantile (see basic_quantiles).
basic_quantiles	numeric vector or single number indicating which quantiles to compute. Default none.
shape_feature	string or character vector specifying which shape features to compute. Default "area" and "radius.mean". Allowed entries are: "area", "perimeter", "radius.mean", "radius.sd", "radius.max", "radius.min".
moment_feature	string or character vector indicating which moment features to compute. Default "cx", "cy", "majoraxis", and "eccentricity". Other allowed features are "theta". Here moment features are only computed on the segmentation mask without incorporating pixel intensities. Therefore, "cx" and "cy" are the x and y coordinates of the cell centroids.
haralick_feature	string or character vector indicating which haralick features to compute. Default none. Allowed are the 13 haralick features: "asm", "con", "cor", "var", "idm", "sav", "sva", "sen", "ent", "dva", "den", "f12", "f13"

`haralick_nbins` an integer indicating the number of bins used to compute the haralick matrix. Pixel intensities are binned in `haralick_nbins` discrete gray levels before computing the haralick matrix.

`haralick_scales` an integer vector indicating the number of scales (distance at which to consider neighbouring pixels) to use to compute the haralick features.

`BPPARAM` parameters for parallelised processing of images. See `MulticoreParam` for information on how to use multiple cores for parallelised processing.

Value

A SingleCellExperiment object (see details)

The returned SingleCellExperiment objects

The returned SingleCellExperiment object `sce` contains a single assay. This assay contains individual objects in columns and channels in rows. Each entry summarises the intensities per object and channel. This summary statistic is typically the mean intensity per object and channel. However, other summary statistics can be computed. When the mean intensity per object and channel is computed (default), the assay is accessible via `counts(sce)`. Otherwise, the assay needs to be accessed via `assay(sce, "counts_*")`, where `*` indicates the argument to `basic_feature`.

The `colData(x)` entry is populated by the computed shape, moment and haralick features per object. The prefix of the feature names indicate whether these features correspond to shape (`s.`), moment (`m.`) or haralick (`h.`) features. Default features are the following:

- s.areaobject size in pixels
- s.radius.meanmean object radius in pixels
- m.cxx centroid position of object
- m.cyy centroid position of object
- m.majoraxismajor axis length in pixels of elliptical fit
- m.eccentricityelliptical eccentricity. 1 meaning straight line and 0 meaning circle.

Computing quantiles

Sometimes it can be useful to describe the summarised pixel intensity per object and channel not in terms of the mean but some quantile of the pixel distribution. For example, to compute the median pixel intensity per object and channel, set `basic_feature = "q05"` and `basic_quantiles = 0.5`.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>),

See Also

`computeFeatures`, for detailed explanation for the computed features. https://earlglynn.github.io/RNotes/package/EBImage/Haralick-Textural-Features.html for more discussion on the haralick features

Examples

```
# Standard example
data(pancreasImages)
data(pancreasMasks)

sce <- measureObjects(pancreasMasks, pancreasImages, img_id = "ImageNb")
sce

# Compute only intensity feature
sce <- measureObjects(pancreasMasks, pancreasImages, img_id = "ImageNb",
                        feature_types = "basic")
colData(sce)

# Visualize on segmentation masks
plotCells(pancreasMasks, object = sce, img_id = "ImageNb",
            cell_id = "object_id", colour_by = "PIN")
```

---

pancreasImages *Example CytoImageList object of image files*

Description

This CytoImageList object contains multi-channel stacks of three images acquired by imaging mass cytometry. Each channel represents the pixel-intensities of each of the 5 measured proteins. The data is part of a imaging mass cytometry study on the progression of Type 1 diabetes and contains pancreas cells.

Usage

```
pancreasImages
```

Format

A CytoImageList object containing 3 `Image` objects with 5 channels each. Channel names can be accessed via ?`channelNames`.

References

Damond, N. et al., A Map of Human Type 1 Diabetes Progression by Imaging Mass Cytometry, Cell Metabolism 29:3, 2019

pancreasMasks *Example CytoImageList object of segmentation masks*

Description

This CytoImageList object contains single-channel images representing the segmentation masks after preprocessing of imaging mass cytometry data. The data is part of a imaging mass cytometry study on the progression of Type 1 diabetes and contains pancreas cells.

Usage

```
pancreasMasks
```

Format

A CytoImageList object containing 3 `Image` objects with 1 channel each. These images are the result to segmentation and associated to the images stored in the `pancreasImages` object. Pixel values indicate the numeric cell identifier while a value of 0 represents the image background.

References

Damond, N. et al., A Map of Human Type 1 Diabetes Progression by Imaging Mass Cytometry, Cell Metabolism 29:3, 2019

pancreasSCE *Example SingleCellExperiment object*

Description

This SingleCellExperiment object contains the expression values of 5 proteins (rows) from 362 cells (columns) across 3 images. The data is part of a imaging mass cytometry study on the progression of Type 1 diabetes and therefore contains pancreas cells.

Usage

```
pancreasSCE
```

Format

A SingleCellExperiment object containing the raw and arcsinh-transformed mean pixel counts per cell as well as associated cell- and protein-specific metadata. Row names represent the names of the target proteins and column names represent the image name and cell id of each cell.

colData Cell-specific metadata where rownames represent the image name and cell id. It containins the

1. image number (ImageNb),

2. cell number/identifier (CellNb),
3. spatial position on the image (Pos_X, Pos_Y),
4. the associated image name (ImageName, see ?"pancreasImages"),
5. the associated mask name (MaskName, see ?"pancreasMasks"),
6. a arbitrary cell-type label (CellType)
7. a logical (Pattern) indicating exocrine cells

rowData Protein-specific metadata where rownames represent the names of the target proteins. It contains the

1. channel number (frame),
2. metal tag of the antibody (MetalTag)
3. Target (the expanded name of the targeted protein)
4. clean_Target (the abbreviated name of the targeted protein)

assays List of protein expression counts containing:

1. the raw expression counts (counts): mean pixel value per cell and protein
2. arcsinh-transformed raw expression counts using a co-factor of 1 (exprs)

References

Damond, N. et al., A Map of Human Type 1 Diabetes Progression by Imaging Mass Cytometry, Cell Metabolism 29:3, 2019

`plotCells` *Function to visualize cell-level information on segmentation masks*

Description

This function takes a `SingleCellExperiment` and `CytoImageList` object containing segmentation masks to colour cells by marker expression or metadata.

Usage

```
plotCells(
  mask,
  object = NULL,
  cell_id = NULL,
  img_id = NULL,
  colour_by = NULL,
  outline_by = NULL,
  exprs_values = "counts",
  colour = NULL,
  ...
)
```

Arguments

mask	a CytoImageList containing single-channel Image objects (see section 'Segmentation mask object' below).
object	a SingleCellExperiment object.
cell_id	character specifying the colData(object) entry, in which the integer cell IDs are stored. These IDs should match the integer pixel values in the segmentation mask object.
img_id	character specifying the colData(object) and mcols(mask) entry, in which the image IDs are stored (see section 'Linking the SingleCellExperiment and CytoImageList object' below)
colour_by	character or character vector specifying the features (rownames(object)) or metadata (colData(object) entry) used to colour individual cells. Cells can be coloured by single colData(object) entries or by up to six features.
outline_by	single character indicating the colData(object) entry by which to outline individual cells.
exprs_values	single character indicating which assay(object) entry to use when visualizing feature counts.
colour	a list with names matching the entries to colour_by and/or outline_by. When setting the colour for continous features, at least two colours need to be provided indicating the colours for minimum and maximum values. When colouring discrete vectors, a colour for each unique entry needs to be provided (see section 'Setting the colours' and examples)
...	Further arguments passed to ?"plotting-param"

Value

a list if return_images and/or return_plot is TRUE (see ?"plotting-param").

- plot: a single plot object (display = "all") or a list of plot objects (display = "single")
- images: a SimpleList object containing three-colour Image objects.

Segmentation mask object

In the plotCells function, mask refers to a CytoImageList object that contains a single or multiple segmentation masks in form of individual Image objects. The function assumes that each object in the segmentation mask is a cell. The key features of such masks include:

- each Image object contains only one channel
- pixel values are integers indicating the cells' IDs or 0 (background)

Linking SingleCellExperiment and CytoImageList objects

To colour individual cells contained in the segmentation masks based on features and metadata stored in the SingleCellExperiment object, an img_id and cell_id entry needs to be provided. Image IDs are matched between the SingleCellExperiment and CytoImageList object via entries to the colData(object)[,img_id] and the mcols(mask)[,img_id] slots. Cell IDs are matched between the SingleCellExperiment and CytoImageList object via entries to colData(object)[,cell_id] and the integer values of the segmentation masks.

Setting the colours

By default, features and metadata are coloured based on internally-set colours. To set new colours, a `list` object must be provided. The names of the object must correspond to the entries to `colour_by` and/or `outline_by`. When setting the colours for continous expression values or continous metadata entries, a vector of at least two colours need to be specified. These colours will be passed onto `colorRampPalette` for interpolation. Discrete metadata entries can be coloured by specifying a named vector in which each entry corresponds to a unique entry to the metadata vector.

Subsetting the `CytoImageList` object

The `CytoImageList` object can be subsetted before calling the `plotCells` function. In that case, only the selected images are displayed.

Subsetting the `SingleCellExperiment` object

The `SingleCellExperiment` object can be subsetted before calling the `plotCells` function. In that case, only cells contained in the `SingleCellExperiment` object are coloured/outlined.

Colour scaling

When colouring features using the plotCells function, colours are scaled between the minimum and maximum per feature across the full assay contained in the SingleCellExperiment object. When subsetting images, cell-level expression is not scaled across the subsetted images but the whole SingleCellExperiment object. To avoid this, the SingleCellExperiment object can be subsetted to only contain the cells that should be displayed before plotting.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

Nicolas Damond (<nicolas.damond@dqbm.uzh.ch>)

See Also

For further plotting parameters see ?"plotting-param"

Examples

```
data(pancreasMasks)
data(pancreasSCE)

# Visualize the masks
plotCells(pancreasMasks)

# Colour the masks based on averaged expression
plotCells(pancreasMasks, object = pancreasSCE, img_id = "ImageNb",
            cell_id = "CellNb", colour_by = c("CD99", "CDH"))

# Colour the masks based on metadata
plotCells(pancreasMasks, object = pancreasSCE, img_id = "ImageNb",
            cell_id = "CellNb", colour_by = "CellType")
```

```
# Outline the masks based on metadata
plotCells(pancreasMasks, object = pancreasSCE, img_id = "ImageNb",
            cell_id = "CellNb", colour_by = "CD99",
            outline_by = "CellType")

# Colour the masks based on arcsinh-transformed expression
plotCells(pancreasMasks, object = pancreasSCE, img_id = "ImageNb",
            cell_id = "CellNb", colour_by = "CD99",
            exprs_values = "exprs")

# Subset the images
cur_images <- getImages(pancreasMasks, 1:2)
plotCells(cur_images, object = pancreasSCE, img_id = "ImageNb",
            cell_id = "CellNb", colour_by = "CD99")

# Set colour
plotCells(pancreasMasks, object = pancreasSCE, img_id = "ImageNb",
            cell_id = "CellNb", colour_by = "CD99", outline_by = "CellType",
            colour = list(CD99 = c("black", "red"),
                          CellType = c(celltype_A = "blue",
                                       celltype_B = "green",
                                       celltype_C = "red")))
```

`plotPixels` *Function to visualize pixel-level information of multi-channel images*

Description

This function takes a `CytoImageList` object to colour pixels by marker expression. Additionally, a `SingleCellExperiment` object and `CytoImageList` object containing segmentation masks can be provided to outline cells based on metadata.

Usage

```
plotPixels(
  image,
  object = NULL,
  mask = NULL,
  cell_id = NULL,
  img_id = NULL,
  colour_by = NULL,
  outline_by = NULL,
  bcg = NULL,
  colour = NULL,
  ...
)
```

Arguments

image	a CytoImageList object containing single or multi-channel Image objects (see details below).
object	an optional SingleCellExperiment object.
mask	an optional CytoImageList object containing segmentaion masks in form of single-channel Image objects (see details below)
cell_id	character specifying the colData(object) entry, in which the integer cell IDs are stored. These IDs should match the integer pixel values in the segmentation mask object.
img_id	character specifying the colData(object), mcols(image) and mcols(mask) entry, in which the image IDs are stored (see section 'Linking the SingleCellExperiment and CytoImageList objects' below)
colour_by	character or character vector specifying the features (contained in channelNames(image)) used to colour individual cells. Pixels can be coloured by up to six features.
outline_by	single character indicating the colData(object) entry by which to outline individual cells
bcg	a list with names matching the entries to colour_by. Each entry contains a numeric vector of three entries: 1. brightness value added to the specified channel 2. contrast value multiplied with the specified channel 3. gamma value (channel is exponentiated by this value) Default is c(0,1,1).
colour	a list with names matching the entries to colour_by and/or outline_by. When setting the colour for continous features, at least two colours need to be provided indicating the colours for minimum and maximum values. When outlining by discrete values, a colour for each unique entry needs to be provided (see section 'Setting the colours' and examples)
...	Further arguments passed to ?"plotting-param"

Value

a list if return_images and/or return_plot is TRUE (see ?"plotting-param").

- plot: a single plot object (display = "all") or a list of plot objects (display = "single")
- images: a SimpleList object containing three-colour Image objects.

Multi-channel image and segmentation mask objects

In the plotPixels function, image refers to a CytoImageList object that contains a single or multiple single- or multi-channel Image objects. Up to six channels can be overlayed to generate a composite image. When outlining cells, a SingleCellExperiment object and CytoImageList object containing segmentation masks must be provided. The function assumes that each object in the segmentation mask is a cell. The key features of such segmentation masks include:

- each Image object contains only one channel
- pixel values are integers indicating the cells' IDs

Linking SingleCellExperiment and CytoImageList objects

To outline individual cells contained in the segmentation masks based on metadata stored in the SingleCellExperiment object, an `img_id` and `cell_id` entry needs to be provided. Image IDs are matched between the `SingleCellExperiment` and `CytoImageList` objects via entries to the `colData(object)[,img_id]`, `mcols(image)[,img_id]` and the `mcols(image)[,img_id]` slots. Cell IDs are matched between the `SingleCellExperiment` and `CytoImageList` object via entries to `colData(object)[,cell_id]` and the integer values of the segmentation masks.

Setting the colours

By default, features and metadata are coloured based on internally-set colours. To set new colours, a `list` object must be provided. The names of the object must correspond to the entries to `colour_by` and/or `outline_by`. When setting the colours for continous expression values or continous metadata entries, a vector of at least two colours need to be specified. These colours will be passed onto `colorRampPalette` for interpolation. Cells can be outlined by discrete metadata entries when specifying a named vector in which each entry corresponds to a unique entry to the metadata vector.

Subsetting the `CytoImageList` objects

The `CytoImageList` object(s) can be subsetted before calling the `plotPixels` function. In that case, only the selected images are displayed.

Subsetting the `SingleCellExperiment` object

The `SingleCellExperiment` object can be subsetted before calling the `plotPixels` function. In that case, only cells contained in the `SingleCellExperiment` object are outlined.

Colour scaling

When plotting pixel intensities, colours are scaled to the minimum and maximum per channel across all images that are being displayed. Therefore, when subsetting images, displayed intensities might change. However, the colour legend will display the correct numeric minimum and maximum pixel intensity across all displayed images per channel.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

Nicolas Damond (<nicolas.damond@dqbm.uzh.ch>)

See Also

For further plotting parameters see `?"plotting-param"`. For instructions on how to normalize images see `normalize`.

Examples

```
data(pancreasMasks)
data(pancreasImages)
data(pancreasSCE)
```

```
# Visualize the images - by default the first channel
plotPixels(pancreasImages)

# Colour the channels
plotPixels(pancreasImages, colour_by = c("CD99", "CDH"))

# Outline the cells based on metadata
plotPixels(pancreasImages, object = pancreasSCE, mask = pancreasMasks,
            img_id = "ImageNb", cell_id = "CellNb",
            colour_by = c("CD99", "CDH"), outline_by = "CellType")

# Enhance individual channels
plotPixels(pancreasImages, colour_by = c("CD99", "CDH"),
            bcg = list(CD99 = c(0, 2, 1)))

# Subset the images
cur_images <- getImages(pancreasImages, 1:2)
plotPixels(cur_images, colour_by = c("CD99", "CDH"))

# Set colour
plotPixels(pancreasImages, colour_by = c("CD99", "CDH"),
            colour = list(CD99 = c("black", "green"),
                          CDH = c("black", "blue")))
```

`plotting-param` *Further plotting parameters for the cytomapper package*

Description

The `plotCells` and `plotPixels` functions share a number of parameter that can be set to change the visual representation of plotted images.

Arguments

`missing_colour`	a single character specifying a valid colour. Cells that are not contained in the SingleCellExperiment object will be coloured based on `missing_colour`. In the `plotPixels` function, `missing_colour` defines the outline of the cells if `outline_by` is not set.
`background_colour`	(only `plotCells`) a single character specifying a valid colour that is set as the background of the image.
`scale_bar`	a list specifying features of the scale bar. One or multiple of the following entries are supported: • `length`: numeric length in pixels (default 20% of the largest image width). • `label`: single character specifying the scale bar label. • `cex`: numeric value indicating the size of the scale bar label.

- `lwidth`: numeric value indicating the line width of the scale bar in pixels. By default, the line width is adjusted relative to the maximum height of the images.
- `colour`: single character specifying the colour of scale bar and label (default "white").
- `position`: position of scale bar. Supported values: "topleft", "topright", "bottomleft", "bottomright" (default "bottomright").
- `margin`: vector of two numeric entries specifying the x and y margin between image boundary and the scale bar (default c(10,10)).
- `frame`: either "all" to display scale bar on all images or a single number specifying the image for which the scale bar should be displayed (default "all").

Plotting of the scale bar is suppressed if set to `NULL`.

`image_title` a list specifying features of the image titles. One or multiple of the following entries are supported:

- `text`: character vector of image titles. Same length as the `CytoImageList` object.
- `position`: single character specifying the position of the title. Supported entries: "top", "bottom", "topleft", "bottomleft", "topright", "bottomright" (default "top").
- `colour`: single character specifying the colour of image title (default "white").
- `margin`: vector of two numeric entries specifying the x and y margin between image boundary and the image title (default c(10,10)).
- `font`: numeric entry specifying the font of the image title (default 1, see `par` for details)
- `cex`: numeric value indicating the size of the image title.

Plotting of the image title is suppressed if set to `NULL`.

`save_plot` a list specifying how to save the plot. One or multiple of the following entries are supported:

- `filename`: single character specifying a valid file name. The file extension specifies the format in which the file is saved. Supported formats are: jpeg, tiff and png. If `display = "single"`, each image will be written in an individual file. The file names obtain a "_x" ending where x indicates the position of the image in the `CytoImageList` object or "legend".
- `scale`: by default the height and width of the saved image is defined by the maximum image size times the number of rows and number of columns. This resolution is often not sufficient to clearly display the text. The `scale` parameter can be set to increase the resolution of the image while keeping the text size constant (default 1 but can be increased for optimal results).

`return_plot` logical indicating whether to return the plot (see `recordPlot` for more infos).

`return_images` logical indicating whether to return the coloured images in form of a `SimpleList` object. Each entry to this list is a three-colour `Image` object. However, the image title and scale bar are not retained.

`legend` a list specifying features of the legend. One or multiple of the following entries are supported:

- `colour_by.title.font`: numeric entry specifying the font of the legend title for features specified by `colour_by`.
- `colour_by.title.cex`: numeric entry specifying the size of the legend title for features specified by `colour_by`.
- `colour_by.labels.cex`: numeric entry specifying the size of the legend labels for features specified by `colour_by`.
- `colour_by.legend.cex`: (only discrete features) numeric entry specifying the size of the legend for features specified by `colour_by`.
- `outline_by.title.font`: numeric entry specifying the font of the legend title for features specified by `outline_by`.
- `outline_by.title.cex`: numeric entry specifying the size of the legend title for features specified by `outline_by`.
- `outline_by.labels.cex`: numeric entry specifying the size of the legend labels for features specified by `outline_by`.
- `outline_by.legend.cex`: (only discrete features) numeric entry specifying the size of the legend for features specified by `outline_by`.
- `margin`: numeric value indicating the margin (in pixels) between the legends and the outer boundary (default 2)

Plotting of the legend is suppressed if set to `NULL`.

`margin` numeric value indicating the gap (in pixels) between individual images (default 0).

`thick` logical indicating whether cell borders should be drawn as thick lines (default `FALSE`).

`display` one of two possible values: "all" or "single". When set to "all", all images are displayed at once in a grid-like fashion. When set to "single", individual images are plotted in single graphics devices. The second option is useful when saving individual images in pdf format or when displaying in Rmarkdown files.

`scale` logical indicating whether to scale each feature individually to its minimum/maximum across the SingleCellExperiment object (see `plotCells`) or across all displayed images (see `plotCells`). If set to `FALSE` each value is displayed relative to the maximum of all selected features.

`interpolate` a logical indicating whether to apply linear interpolation to the image when drawing (see `rasterImage`) (default TRUE).

Value

a list if `return_images` and/or `return_plot` is TRUE.

- `plot`: a single plot object (`display = "all"`) or a list of plot objects (`display = "single"`)
- `images`: a `SimpleList` object containing three-colour `Image` objects.

Author(s)

Nils Eling (<nils.eling@dqbm.uzh.ch>)

Nicolas Damond (<nicolas.damond@dqbm.uzh.ch>)

See Also

plotCells and plotPixels for the main plotting functions

Examples

```
data("pancreasImages")
data("pancreasMasks")
data("pancreasSCE")

# Setting missing colour
plotCells(pancreasMasks, missing_colour = "blue")

# Setting background colour
plotCells(pancreasMasks, background_colour = "blue")

# Setting the scale bar
plotCells(pancreasMasks, scale_bar = list(length = 10,
                                          cex = 2,
                                          lwidth = 10,
                                          colour = "red",
                                          position = "bottomleft",
                                          margin = c(5,5),
                                          frame = 3))

# Setting the image title
plotCells(pancreasMasks,
            image_title = list(text = c("image1", "image2", "image3"),
                               position = "topleft",
                               colour = "blue",
                               margin = c(0,5),
                               font = 2,
                               cex = 2))

# Return plot
cur_out <- plotPixels(pancreasImages, return_plot = TRUE)
cur_out$plot

# Return images
cur_out <- plotPixels(pancreasImages, return_images = TRUE)
cur_out$images

# Setting the legend
plotCells(pancreasMasks, object = pancreasSCE,
            img_id = "ImageNb", cell_id = "CellNb",
            colour_by = c("CD99", "CDH"),
            outline_by = "CellType",
            legend = list(colour_by.title.font = 0.5,
                        colour_by.title.cex = 0.5,
                        colour_by.labels.cex = 0.5,
                        outline_by.legend.cex = 0.5,
                        margin = 0))
```

```
# Setting the margin between images
plotPixels(pancreasImages, margin = 3)

# Thick outlines
plotCells(pancreasMasks, object = pancreasSCE,
            img_id = "ImageNb", cell_id = "CellNb",
            colour_by = "CD99",
            outline_by = "CellType",
            thick = TRUE)

# Displaying individual images
plotPixels(pancreasImages, display = "single")

# Supress scaling
plotPixels(pancreasImages, colour_by = c("CD99", "PIN"),
            scale = TRUE)
plotPixels(pancreasImages, colour_by = c("CD99", "PIN"),
            scale = FALSE)

# Suppress interpolation
plotPixels(pancreasImages, colour_by = c("CD99", "PIN"),
            interpolate = TRUE)
plotPixels(pancreasImages, colour_by = c("CD99", "PIN"),
            interpolate = FALSE)
```

Index