Package 'OpenImageR'

June 18, 2020

Type Package

Title An Image Processing Toolkit

Version 1.1.7

Date 2020-06-18

Depends R(>= 3.2.3)

Imports Rcpp (>= 0.12.17), graphics, grid, shiny, jpeg, png, tiff, R6

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

BugReports https://github.com/mlampros/OpenImageR/issues

URL https://github.com/mlampros/OpenImageR

Description

Incorporates functions for image preprocessing, filtering and image recognition. The package takes advantage of 'RcppArmadillo' to speed up computationally intensive functions. The histogram of oriented gradients descriptor is a modification of the 'findHOGFeatures' function of the 'SimpleCV' computer vision platform, the average_hash(), dhash() and phash() functions are based on the 'ImageHash' python library. The Gabor Feature Extraction functions are based on 'Matlab' code of the paper, ``CloudID: Trustworthy cloud-based and cross-enterprise biometric identification'' by M. Haghighat, S. Zonouz, M. Abdel-Mottaleb, Expert Systems with Applications, vol. 42, no. 21, pp. 7905-
7916, 2015, <doi:10.1016/j.eswa.2015.06.025>. The 'SLIC' and 'SLICO' superpixel algorithms were explained in detail in (i) ``SLIC Superpixels Compared to State-of-the-art Superpixel Methods'', Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Suesstrunk, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, num. 11, p. 2274-
2282, May 2012, <doi:10.1109/TPAMI.2012.120> and (ii) ``SLIC Superpixels'', Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Suesstrunk, EPFL Technical Report no. 149300, June 2010.

License GPL-3

Copyright inst/COPYRIGHTS

SystemRequirements libarmadillo: apt-get install -y libarmadillo-dev
(deb), libblas: apt-get install -y libblas-dev (deb),
liblapack: apt-get install -y liblapack-dev (deb),
libarpack++2: apt-get install -y libarpack++2-dev (deb),

gfortran: apt-get install -y gfortran (deb), libjpeg-dev: apt-get install -y libjpeg-dev (deb), libpng-dev: apt-get install -y libpng-dev (deb), libfftw3-dev: apt-get install -y libfftw3-dev (deb), libtiff5-dev: apt-get install -y libtiff5-dev (deb)

LinkingTo Rcpp, RcppArmadillo (>= 0.8.0)

Suggests testthat, knitr, rmarkdown, covr

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation yes

Author Lampros Mouselimis [aut, cre],
Sight Machine [cph] (findHOGFeatures function of the SimpleCV computer vision platform),
Johannes Buchner [cph] (average_hash, dhash and phash functions of the ImageHash python library),
Mohammad Haghighat [cph] (Gabor Feature Extraction),
Radhakrishna Achanta [cph] (Author of the C++ code of the SLIC and SLICO algorithms (for commercial use please contact the author))

Repository CRAN

Date/Publication 2020-06-18 18:40:07 UTC

R topics documented:

`Augmentation` *image augmentations of a matrix, data frame, array or a list of 3-dimensional arrays*

Description

image augmentations of a matrix, data frame, array or a list of 3-dimensional arrays

Usage

```
Augmentation(
  image,
  flip_mode = NULL,
  crop_width = NULL,
  crop_height = NULL,
  resiz_width = 0,
  resiz_height = 0,
  resiz_method = "nearest",
  shift_rows = 0,
  shift_cols = 0,
  rotate_angle = 0,
  rotate_method = "nearest",
  zca_comps = 0,
  zca_epsilon = 0,
  image_thresh = 0,
  padded_value = 0,
  verbose = FALSE
)
```

Arguments

image	a matrix, data frame, array or list of 3-dimensional arrays
flip_mode	a character string ('horizontal', 'vertical')
crop_width	an integer specifying the new width of the image, after the image is cropped. Corresponds to the image-rows.
crop_height	an integer specifying the new height of the image, after the image is cropped. Corresponds to the image-columns.
resiz_width	an integer specifying the new width of the image, after the image is resized. Corresponds to the image-rows.
resiz_height	an integer specifying the new height of the image, after the image is resized. Corresponds to the image-columns.
resiz_method	a string specifying the interpolation method when resizing an image ('nearest', 'bilinear')
shift_rows	a positive or negative integer specifying the direction that the rows should be shifted
shift_cols	a positive or negative integer specifying the direction that the columns should be shifted
rotate_angle	an integer specifying the rotation angle of the image
rotate_method	a string specifying the interpolation method when rotating an image ('nearest', 'bilinear')
zca_comps	an integer specifying the number of components to keep by zca whitening, when svd is performed
zca_epsilon	a float specifying the regularization parameter by zca whitening
image_thresh	the threshold parameter, by image thresholding, should be between 0 and 1 if the data is normalized or between 0-255 otherwise
padded_value	either a numeric value or a numeric vector of length equal to N of an N-dimensional array. If it's not equal to 0 then the values of the shifted rows or columns will be filled with the user-defined padded_value. Applies only to the shift_rows and shift_cols parameters.
verbose	a boolean (TRUE, FALSE). If TRUE, then the total time of the preprocessing task will be printed.

Details

This function takes advantage of various methods to accomplish image augmentations. The order of the preprocessing steps, in case that all transformations are applied to an image, is : 1st flip image, 2nd crop image, 3rd resize image, 4th shift rows or columns, 5th rotate image, 6th zca-whitening and 7th image-thresholding.

Value

the output is of the same type with the input (in case of a data frame it returns a matrix)

Author(s)

Lampros Mouselimis

Examples

```
## Not run:

# a matrix
object = matrix(1, 10, 10)

res = Augmentation(object, resiz_width = 8, resiz_height = 8, rotate_angle = 40)


# an array
object = array(0, dim = c(10, 10, 3))

res = Augmentation(object, resiz_width = 8, resiz_height = 8, rotate_angle = 30)


# an array (multiple matrices)
object = array(0, dim = c(10, 10, 10))

res = Augmentation(object, resiz_width = 8, resiz_height = 8, rotate_angle = 20)


# a list of 3-dimensional arrays
object = list(array(0, dim = c(10, 10, 3)), array(0, dim = c(10, 10, 3)))

res = Augmentation(object, resiz_width = 8, resiz_height = 8, rotate_angle = 40)

## End(Not run)
```

---

average_hash *calculation of the 'average hash' of an image*

Description

This function calculates the average hash of an image

Usage

```
average_hash(gray_image, hash_size = 8, MODE = "hash", resize = "nearest")
```

Arguments

gray_image	a (2-dimensional) matrix or data frame
hash_size	an integer specifying the hash size (should be less than number of rows or columns of the gray_image)

MODE	one of 'hash' (returns the hash of the image), 'binary' (returns binary identifier of the image)
resize	corresponds to one of 'nearest', 'bilinear' (resizing method)

Details

The function is a modification of the 'average_hash' function of the imagehash package [please consult the COPYRIGHT file]. The average hash works in the following way : 1st convert to grayscale, 2nd, reduce the size of an image (for instance to an 8x8 image, to further simplify the number of computations), 3rd average the resulting colors (for an 8x8 image we average 64 colors), 4th compute the bits by comparing if each color value is above or below the mean, 5th construct the hash.

Value

either a hash-string or a binary vector

Examples

```
image = readImage(system.file("tmp_images", "1.png", package = "OpenImageR"))

image = rgb_2gray(image)

res_hash = average_hash(image, hash_size = 8, MODE = 'hash')

res_binary = average_hash(image, hash_size = 8, MODE = 'binary')
```

---

convolution *convolution*

Description

convolution

Usage

```
convolution(image, kernel, mode = "same")
```

Arguments

image	either a matrix, data frame or array
kernel	a kernel in form of a matrix
mode	the convolution mode (one of 'same', 'full')

Details

This function performs convolution using a kernel matrix. When mode 'same' the output object has the same dimensions with the input, whereas when mode 'full' the rows and columns of the output object equals : ROWS = nrow(image) + nrow(kernel) - 1 and COLUMNS = ncol(image) + ncol(kernel) - 1

Value

either a matrix or an array, depending on the input data

Author(s)

Lampros Mouselimis

Examples

```
# kernel
x = matrix(1, nrow = 4, ncol = 4) / 16   # uniform


# matrix
image_matrix = matrix(runif(100), 10, 10)

res = convolution(image_matrix, x, "same")


# array
image_array = array(runif(100), dim = c(10, 10, 3))

res = convolution(image_array, x, "same")
```

`cropImage` *crop an image*

Description

crop an image

Usage

```
cropImage(image, new_width, new_height, type = "equal_spaced")
```

Arguments

image	matrix or 3-dimensional array
new_width	Corresponds to the image-rows. If 'equal_spaced' then the new_width should be numeric of length 1. If 'user_defined' then the new_width should be a sequence of numeric values.
new_height	Corresponds to the image-columns. If 'equal_spaced' then the new_height should be numeric of length 1. If 'user_defined' then the new_height should be a sequence of numeric values.
type	a string specifying the type ('equal_spaced' or 'user_defined'). If 'equal_spaced' the image will be cropped towards the center (equal distances horizontaly and verticaly). If 'user_defined' the user specifies the cropped region.

Details

This function crops an image in two different ways.

Value

depending on the input, either a matrix or an array

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "2.jpg", package = "OpenImageR")

image = readImage(path)

# IF 'equal_spaced':
crop1 = cropImage(image, new_width = 20, new_height = 20, type = 'equal_spaced')

# IF 'user_defined':
crop2 = cropImage(image, new_width = 5:20, new_height = 5:20, type = 'user_defined')
```

---

delationErosion *Delation or Erosion of an image*

Description

this function performs delation or erosion to a 2- or 3- dimensional image

Usage

```
delationErosion(image, Filter, method = "delation", threads = 1)
```

Arguments

image	a matrix, data frame or 3-dimensional array
Filter	a vector specifying the dimensions of the kernel, which will be used to perform either delation or erosion, such as c(3,3)
method	one of 'delation', 'erosion'
threads	number of cores to run in parallel (> 1 should be used if image high dimensional)

Details

This function utilizes a kernel to perform delation or erosion. The first value of the vector indicates the number of rows of the kernel, whereas the second value indicates the number of columns.

Value

a matrix or 3-dimensional array

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

res_delate = delationErosion(image, Filter = c(3,3), method = 'delation')

res_erode = delationErosion(image, Filter = c(5,5), method = 'erosion')
```

---

dhash *calculation of the 'dhash' of an image*

Description

This function calculates the dhash of an image

Usage

```
dhash(gray_image, hash_size = 8, MODE = "hash", resize = "nearest")
```

Arguments

gray_image	a (2-dimensional) matrix or data frame
hash_size	an integer specifying the hash size (should be less than number of rows or columns of the gray_image)
MODE	one of 'hash' (returns the hash of the image), 'binary' (returns binary identifier of the image)
resize	corresponds to one of 'nearest', 'bilinear'

Details

The function is a modification of the 'dhash' function of the imagehash package [please consult the COPYRIGHT file]. In comparison to average_hash and phash, the dhash algorithm takes into consideration the difference between adjacent pixels.

Value

either a hash-string or a binary vector

Examples

```
image = readImage(system.file("tmp_images", "3.jpeg", package = "OpenImageR"))

image = rgb_2gray(image)

res_hash = dhash(image, hash_size = 8, MODE = 'hash')

res_binary = dhash(image, hash_size = 8, MODE = 'binary')
```

---

down_sample_image *downsampling an image (by a factor) using gaussian blur*

Description

downsampling an image (by a factor) using gaussian blur

Usage

```
down_sample_image(
  image,
  factor,
  gaussian_blur = FALSE,
  gauss_sigma = 1,
  range_gauss = 2
)
```

Arguments

image	matrix or 3-dimensional array
factor	a positive number greater or equal to 1.0
gaussian_blur	a boolean (TRUE,FALSE) specifying if gaussian blur should be applied when downsampling
gauss_sigma	float parameter sigma for the gaussian filter
range_gauss	float number specifying the range of values for the gaussian filter

Details

This function downsamples an image with the option to use gaussian blur for optimal output.

Value

depending on the input, either a matrix or an array

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "2.jpg", package = "OpenImageR")

image = readImage(path)

dsamp = down_sample_image(image, factor = 2.0, gaussian_blur = TRUE)
```

---

edge_detection *edge detection (Frei_chen, LoG, Prewitt, Roberts_cross, Scharr, Sobel)*

Description

edge detection (Frei_chen, LoG, Prewitt, Roberts_cross, Scharr, Sobel)

Usage

```
edge_detection(
  image,
  method = NULL,
  conv_mode = "same",
  approx = F,
  gaussian_dims = 5,
  sigma = 1,
  range_gauss = 2,
  laplacian_type = 1
)
```

Arguments

image	matrix or 3-dimensional array
method	the method should be one of 'Frei_chen', 'LoG' (Laplacian of Gaussian), 'Prewitt', 'Roberts_cross', 'Scharr', 'Sobel'

conv_mode	the convolution mode should be one of 'same', 'full'
approx	if TRUE, approximate calculation of gradient (applies to all filters except for 'LoG')
gaussian_dims	integer specifying the horizontal and vertical dimensions of the gaussian filter
sigma	float parameter sigma for the gaussian filter
range_gauss	float number specifying the range of values for the gaussian filter
laplacian_type	integer value specifying the type for the laplacian kernel (one of 1, 2, 3, 4)

Details

This function takes either a matrix or a 3-dimensional array and it performs edge detection using one of the following filters : 'Frei_chen', 'LoG' (Laplacian of Gaussian), 'Prewitt', 'Roberts_cross', 'Scharr', 'Sobel'

Value

depending on the input, either a matrix or an array

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

res = edge_detection(image, method = 'Frei_chen', conv_mode = 'same')
```

---

flipImage *flip image horizontally or vertically*

Description

flip an image row-wise (horizontally) or column-wise (vertically)

Usage

```
flipImage(image, mode = "horizontal")
```

Arguments

image	a matrix, data frame or 3-dimensional array
mode	one of 'horizontal', 'vertical'

Details

This function flips an image row-wise or column-wise

Value

a matrix or 3-dimensional array

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

im = readImage(path)

flp = flipImage(im, mode = 'vertical')
```

GaborFeatureExtract *Gabor Feature Extraction*

Description

Gabor Feature Extraction

Gabor Feature Extraction

Usage

```
# init <- GaborFeatureExtract$new()
```

Details

In case of an RGB image (3-dimensional) one can use the *rgb_2gray()* to convert the image to a 2-dimensional one

I added the option *downsample_gabor* to the original matlab code based on the following question on stackoverflow : *https://stackoverflow.com/questions/49119991/feature-extraction-with-gabor-filters*

Methods

```
GaborFeatureExtract$new()
--------------
gabor_filter_bank(scales, orientations, gabor_rows, gabor_columns, plot_data = FALSE)

--------------
gabor_feature_extraction(image, scales, orientations, gabor_rows, gabor_columns, downsample_gabor = FAL

--------------
```

```
gabor_feature_engine(img_data, img_nrow, img_ncol, scales, orientations, gabor_rows, gabor_columns, dow

--------------

plot_gabor(real_matrices, margin_btw_plots = 0.15, thresholding = FALSE)

--------------

plot_multi_images(list_images, par_ROWS, par_COLS)

--------------
```

Methods

Public methods:

- GaborFeatureExtract$new()
- GaborFeatureExtract$gabor_filter_bank()
- GaborFeatureExtract$gabor_feature_extraction()
- GaborFeatureExtract$gabor_feature_engine()
- GaborFeatureExtract$plot_gabor()
- GaborFeatureExtract$plot_multi_images()
- GaborFeatureExtract$clone()

Method `new()`:

Usage:

```
GaborFeatureExtract$new()
```

Method `gabor_filter_bank()`:

Usage:

```
GaborFeatureExtract$gabor_filter_bank(
  scales,
  orientations,
  gabor_rows,
  gabor_columns,
  plot_data = FALSE
)
```

Arguments:

`scales` a numeric value. Number of scales (usually set to 5) (gabor_filter_bank function)

`orientations` a numeric value. Number of orientations (usually set to 8) (gabor_filter_bank function)

`gabor_rows` a numeric value. Number of rows of the 2-D Gabor filter (an odd integer number, usually set to 39 depending on the image size) (gabor_filter_bank function)

`gabor_columns` a numeric value. Number of columns of the 2-D Gabor filter (an odd integer number, usually set to 39 depending on the image size) (gabor_filter_bank function)

`plot_data` either TRUE or FALSE. If TRUE then data needed for plotting will be returned (gabor_filter_bank, gabor_feature_extraction functions)

Method `gabor_feature_extraction()`:

Usage:

```
GaborFeatureExtract$gabor_feature_extraction(
  image,
  scales,
  orientations,
  gabor_rows,
  gabor_columns,
  downsample_gabor = FALSE,
  plot_data = FALSE,
  downsample_rows = NULL,
  downsample_cols = NULL,
  normalize_features = FALSE,
  threads = 1,
  verbose = FALSE,
  vectorize_magnitude = TRUE
)
```

Arguments:

`image` a 2-dimensional image of type matrix (gabor_feature_extraction function)

`scales` a numeric value. Number of scales (usually set to 5) (gabor_filter_bank function)

`orientations` a numeric value. Number of orientations (usually set to 8) (gabor_filter_bank function)

`gabor_rows` a numeric value. Number of rows of the 2-D Gabor filter (an odd integer number, usually set to 39 depending on the image size) (gabor_filter_bank function)

`gabor_columns` a numeric value. Number of columns of the 2-D Gabor filter (an odd integer number, usually set to 39 depending on the image size) (gabor_filter_bank function)

`downsample_gabor` either TRUE or FALSE. If TRUE then downsampling of data will take place. The *downsample_rows* and *downsample_cols* should be adjusted accordingly. Downsampling does not affect the output plots but the output *gabor_features* (gabor_feature_extraction function)

`plot_data` either TRUE or FALSE. If TRUE then data needed for plotting will be returned (gabor_filter_bank, gabor_feature_extraction functions)

`downsample_rows` either NULL or a numeric value specifying the factor of downsampling along rows (gabor_feature_extraction function)

`downsample_cols` either NULL or a numeric value specifying the factor of downsampling along columns (gabor_feature_extraction function)

`normalize_features` either TRUE or FALSE. If TRUE then the output gabor-features will be normalized to zero mean and unit variance (gabor_feature_extraction function)

`threads` a numeric value specifying the number of threads to use (gabor_feature_extraction function)

`verbose` either TRUE or FALSE. If TRUE then information will be printed in the console (gabor_feature_extraction, gabor_feature_engine functions)

`vectorize_magnitude` either TRUE or FALSE. If TRUE the computed magnitude feature will be returned in the form of a vector, otherwise it will be returned as a list of matrices (gabor_feature_extraction function)

Method `gabor_feature_engine()`**:**

Usage:

```
GaborFeatureExtract$gabor_feature_engine(
  img_data,
  img_nrow,
  img_ncol,
  scales,
  orientations,
  gabor_rows,
  gabor_columns,
  downsample_gabor = FALSE,
  downsample_rows = NULL,
  downsample_cols = NULL,
  normalize_features = FALSE,
  threads = 1,
  verbose = FALSE
)
```

Arguments:

`img_data` a numeric matrix specifying the input data (gabor_feature_engine function)

`img_nrow` an integer specifying the number of rows of the input matrix (gabor_feature_engine function)

`img_ncol` an integer specifying the number of columns of the input matrix (gabor_feature_engine function)

`scales` a numeric value. Number of scales (usually set to 5) (gabor_filter_bank function)

`orientations` a numeric value. Number of orientations (usually set to 8) (gabor_filter_bank function)

`gabor_rows` a numeric value. Number of rows of the 2-D Gabor filter (an odd integer number, usually set to 39 depending on the image size) (gabor_filter_bank function)

`gabor_columns` a numeric value. Number of columns of the 2-D Gabor filter (an odd integer number, usually set to 39 depending on the image size) (gabor_filter_bank function)

`downsample_gabor` either TRUE or FALSE. If TRUE then downsampling of data will take place. The *downsample_rows* and *downsample_cols* should be adjusted accordingly. Downsampling does not affect the output plots but the output *gabor_features* (gabor_feature_extraction function)

`downsample_rows` either NULL or a numeric value specifying the factor of downsampling along rows (gabor_feature_extraction function)

`downsample_cols` either NULL or a numeric value specifying the factor of downsampling along columns (gabor_feature_extraction function)

`normalize_features` either TRUE or FALSE. If TRUE then the output gabor-features will be normalized to zero mean and unit variance (gabor_feature_extraction function)

`threads` a numeric value specifying the number of threads to use (gabor_feature_extraction function)

`verbose` either TRUE or FALSE. If TRUE then information will be printed in the console (gabor_feature_extraction, gabor_feature_engine functions)

Method `plot_gabor()`**:**

Usage:

```
GaborFeatureExtract$plot_gabor(
  real_matrices,
  margin_btw_plots = 0.65,
  thresholding = FALSE
)
```

Arguments:

`real_matrices` a list of 3-dimensional arrays. These arrays correspond to the *real part* of the complex output matrices (plot_gabor function)

`margin_btw_plots` a float between 0.0 and 1.0 specifying the margin between the multiple output plots (plot_gabor function)

`thresholding` either TRUE or FALSE. If TRUE then a threshold of 0.5 will be used to push values above 0.5 to 1.0 (similar to otsu-thresholding) (plot_gabor function)

Method `plot_multi_images()`**:**

Usage:

```
GaborFeatureExtract$plot_multi_images(list_images, par_ROWS, par_COLS)
```

Arguments:

`list_images` a list containing the images to plot (plot_multi_images function)

`par_ROWS` a numeric value specifying the number of rows of the plot-grid (plot_multi_images function)

`par_COLS` a numeric value specifying the number of columns of the plot-grid (plot_multi_images function)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GaborFeatureExtract$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://github.com/mhaghighat/gabor

https://stackoverflow.com/questions/20608458/gabor-feature-extraction

https://stackoverflow.com/questions/49119991/feature-extraction-with-gabor-filters

Examples

```
library(OpenImageR)

init_gb = GaborFeatureExtract$new()

# gabor-filter-bank
#------------------

gb_f = init_gb$gabor_filter_bank(scales = 5, orientations = 8, gabor_rows = 39,
```

```
                                gabor_columns = 39, plot_data = TRUE)


# plot gabor-filter-bank
#-----------------------

plt_f = init_gb$plot_gabor(real_matrices = gb_f$gabor_real, margin_btw_plots = 0.65,

                           thresholding = FALSE)


# read image
#-----------

pth_im = system.file("tmp_images", "car.png", package = "OpenImageR")

im = readImage(pth_im) * 255


# gabor-feature-extract
#----------------------

# gb_im = init_gb$gabor_feature_extraction(image = im, scales = 5, orientations = 8,

#                                          downsample_gabor = TRUE, downsample_rows = 3,

#                                 downsample_cols = 3, gabor_rows = 39, gabor_columns = 39,

#                                          plot_data = TRUE, normalize_features = FALSE,

#                                          threads = 6)


# plot real data of gabor-feature-extract
#----------------------------------------

# plt_im = init_gb$plot_gabor(real_matrices = gb_im$gabor_features_real, margin_btw_plots = 0.65,

#                             thresholding = FALSE)


# feature generation for a matrix of images (such as the mnist data set)
#------------------------------------------------------------------------

ROWS = 13; COLS = 13; SCAL = 3; ORIEN = 5; nrow_mt = 500; im_width = 12; im_height = 15

set.seed(1)
im_mt = matrix(sample(1:255, nrow_mt * im_width * im_height, replace = TRUE), nrow = nrow_mt,

                      ncol = im_width * im_height)

# gb_ex = init_gb$gabor_feature_engine(img_data = im_mt, img_nrow = im_width, img_ncol = im_height,
```

```
#                                  scales = SCAL, orientations = ORIEN, gabor_rows = ROWS,

#                                       gabor_columns = COLS, downsample_gabor = FALSE,

#                                       downsample_rows = NULL, downsample_cols = NULL,

#                                  normalize_features = TRUE, threads = 1, verbose = FALSE)


# plot of multiple image in same figure
#---------------------------------------

list_images = list(im, im, im)

plt_multi = init_gb$plot_multi_images(list_images, par_ROWS = 2, par_COLS = 2)
```

---

gamma_correction *Gamma correction*

Description

Gamma correction

Usage

```
gamma_correction(image, gamma)
```

Arguments

image	matrix or 3-dimensional array
gamma	a positive value

Details

This function applies gamma correction to a matrix or to a 3-dimensional array. The gamma correction controls the overall brightness of an image.

Value

depending on the input, either a matrix or an array

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "2.jpg", package = "OpenImageR")

image = readImage(path)

filt = gamma_correction(image, gamma = 0.5)
```

---

hash_apply *calculate the binary or the hexadecimal hash for a matrix, array or a folder of images for the average_hash, phash or dhash functions*

Description

This function takes either a matrix, array or a folder and returns either the binary hash features or the hashes (as a character vector)

Usage

```
hash_apply(
  object,
  rows = 28,
  columns = 28,
  hash_size = 8,
  highfreq_factor = 3,
  method = "phash",
  mode = "binary",
  threads = 1,
  resize = "nearest"
)
```

Arguments

object	a matrix, a data frame, a 3-dimensional array or a path to a folder of files (images)
rows	a number specifying the number of rows of the matrix
columns	a number specifying the number of columns of the matrix
hash_size	an integer specifying the hash size. IF method = 'phash' : the hash_size * highfreq_factor should be less than number of rows or columns of the gray_image. IF method = 'dhash' or 'average_hash' : the hash_size should be less than number of rows or columns of the gray_image
highfreq_factor	an integer specyfing the highfrequency factor (IF method = 'phash' : the hash_size * highfreq_factor should be less than number of rows or columns of the gray_image)
method	one of 'phash', 'average_hash', 'dhash'

mode	one of 'binary', 'hash'
threads	the number of cores to run in parallel
resize	corresponds to one of 'nearest', 'bilinear' (resizing method)

Details

This function calculates the binary hash or the hexadecimal hash for various types of objects.

Value

If the input is a matrix, data frame or array this function returns a matrix (if mode = 'binary') or a character vector (if mode = 'hex_hash'). If the input is a path to a folder the function returns a list of length 2, the 1st sublist is a vector with the names of the image files (the order of the files in the vector corresponds to the order of the rows of the output matrix), the 2nd sublist is a matrix (if mode = 'binary') or a character vector (if mode = 'hex_hash').

Examples

```
path = paste0(system.file("tmp_images", "same_type", package = "OpenImageR"), '/')

res_phash = hash_apply(path, method = 'phash', mode = 'binary')
```

HOG *calculate the HOG (Histogram of oriented gradients) for an image*

Description

The function is a modification of the 'findHOGFeatures' function of the SimpleCV package [please consult the COPYRIGHT file] The function takes either an RGB (it will be converted to gray) or a gray image and returns a vector of the HOG descriptors. The main purpose of the function is to create a vector of features, which can be used in classification tasks.

Usage

```
HOG(image, cells = 3, orientations = 6)
```

Arguments

image	matrix or 3-dimensional array
cells	the number of divisions (cells)
orientations	number of orientation bins

Details

This function takes either a matrix, a data frame or a 3-dimensional array and returns a vector with the HOG-descriptors (histogram of oriented gradients).

Value

a numeric vector

Examples

```
## Not run:

path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

res = HOG(image, cells = 3, orientations = 6)

## End(Not run)
```

HOG_apply *calculate the HOG (Histogram of oriented gradients) for a matrix, array or a folder of images*

Description

calculate the HOG (Histogram of oriented gradients) for a matrix, array or a folder of images

Usage

```
HOG_apply(
  object,
  cells = 3,
  orientations = 6,
  rows = NULL,
  columns = NULL,
  threads = 1
)
```

Arguments

object	a matrix, a data frame, a 3-dimensional array or a path to a folder of files (images)
cells	the number of divisions (cells)
orientations	number of orientation bins
rows	a value specifying the number of rows of each image-row of the matrix (required if object is a matrix)
columns	a value specifying the number of columns of each image-row of the matrix (required if object is a matrix)
threads	the number of parallel cores to use

Details

This function takes as input either a matrix, a data frame, a 3-dimensional array or a character path to a folder of files (images). It returns the HOG-descriptors (histogram of oriented gradients) for each row (if matrix or data frame), for each array-slice (if array) or for each file (if path to a folder of images).

Value

If the input is a matrix, data frame or array it returns a matrix of the hog descriptors. If the input is a path to a folder it returns a list of length 2, the 1st sublist is a vector with the names of the image files (the order of the files in the vector corresponds to the order of the rows of the output matrix), the 2nd sublist is the matrix of the hog descriptors.

Examples

```
## Not run:

MATR = matrix(runif(75), ncol = 25, nrow = 5)

res = HOG_apply(MATR, cells = 3, orientations = 5, rows = 5, columns = 5, threads = 1)


ARRAY = array(5, dim = c(10, 10, 3))

res = HOG_apply(ARRAY, cells = 3, orientations = 6, threads = 1)


FOLDER_path = paste0(system.file("tmp_images", "same_type", package = "OpenImageR"), '/')

res = HOG_apply(FOLDER_path, cells = 3, orientations = 6, threads = 1)

## End(Not run)
```

---

imageShow *display an image*

Description

This function displays an image

Usage

```
imageShow(file_path)
```

Arguments

file_path	if file_path is a character string, then a shiny application is utilized. If file_path is a matrix, data.frame OR a 3-dimensional array then the grid.raster function of the base grid package is used.

Details

This function displays an image using either a character path, a 2- or a 3-dimensional object.

Value

displays an image

Examples

```
# path = system.file("tmp_images", "1.png", package = "OpenImageR")

# imageShow(path)
```

`image_thresholding` *image thresholding*

Description

image thresholding

Usage

```
image_thresholding(image, thresh)
```

Arguments

`image`	matrix or 3-dimensional array
`thresh`	the threshold parameter should be between 0 and 1 if the data is normalized or between 0-255 otherwise

Details

This function applies thresholding to a matrix or to a 3-dimensional array.

Value

a matrix

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

filt = image_thresholding(image, thresh = 0.5)
```

`invariant_hash` *invariant hashing (caclulation of the hamming or the levenshtein distance when the image is flipped, rotated or cropped)*

Description

flip-rotate-crop an image and caclulate the hamming or the levenshtein distance for phash, average_hash, dhash

Usage

```
invariant_hash(
  image,
  new_image,
  method = "phash",
  mode = "binary",
  hash_size = 8,
  highfreq_factor = 4,
  resize = "nearest",
  flip = T,
  rotate = T,
  angle_bidirectional = 10,
  crop = T
)
```

Arguments

`image`	a 2-dimensional matrix or data frame (only gray-scale images are valid)
`new_image`	a new image to be compared with the previous input image
`method`	one of 'phash', 'average_hash', 'dhash'
`mode`	one of 'binary', 'hash'
`hash_size`	an integer specifying the hash size. IF method = 'phash' : the hash_size * highfreq_factor should be less than number of floor(rows * 0.8) or floor(columns * 0.8) of the gray_image IF method = 'dhash' or 'average_hash' : the hash_size should be less than number of floor(rows * 0.8) or floor(columns * 0.8) of the gray_image

highfreq_factor
: an integer specyfing the highfrequency factor (IF method = 'phash' : the hash_size * highfreq_factor should be less than number of floor(rows * 0.8) or floor(columns * 0.8) of the gray_image)

resize
: corresponds to one of 'nearest', 'bilinear' (resizing method)

flip
: if TRUE the new_image will be flipped both horizontal and vertical

rotate
: if TRUE the new_image will be rotated for a specified angle (see angle_bidirectional)

angle_bidirectional
: a float specifying the angle that the images should be rotated in both directions. For instance, if angle_bidirectional = 10 then the image will be rotated for 10 and 350 (360-10) degrees.

crop
: if TRUE the new_image will be cropped 10 or 20 percent (equally spaced horizontally and vertically)

Details

This function performs the following transformations : flips an image (no-flip, horizonal-flip, vertical-flip), rotates an image (no-angle, angle_bidirectional, 360-angle_bidirectional) and crops an image (no-crop, 10-percent-crop, 20-percent-crop). Depending on the type of mode ('binary', 'hash'), after each transformation the hamming or the levenshtein distance between the two images is calculated.

Value

If flip, rotate and crop are all FALSE then the function returns either the hamming distance (if mode = 'binary') or the levenshtein distance (if mode = 'hash') for the two images. If any of the flip, rotate, crop is TRUE then it returns the MIN, MAX of the hamming distance (if mode = 'binary') or the MIN,MAX of the levenshtein distance (if mode = 'hash').

Examples

```
## Not run:

path1 = system.file("tmp_images", "1.png", package = "OpenImageR")

path2 = system.file("tmp_images", "2.jpg", package = "OpenImageR")

image1 = rgb_2gray(readImage(path1))

image2 = rgb_2gray(readImage(path2))

res1 = invariant_hash(image1, image2, hash_size = 3, flip = TRUE, crop = FALSE)

res2 = invariant_hash(image1, image2, mode = 'hash', hash_size = 3, angle_bidirectional = 10)

## End(Not run)
```

List_2_Array *convert a list of matrices to an array of matrices*

Description

convert a list of matrices to an array of matrices

Usage

```
List_2_Array(data, verbose = FALSE)
```

Arguments

data	a list of matrices
verbose	if TRUE then the time taken to complete the task will be printed

Details

This is a helper function mainly for the HOG and hash functions. In case that matrices are stored in a list, this function converts the list to an array of 2-dimensional data.

Value

an array

Author(s)

Lampros Mouselimis

Examples

```
lst = list(matrix(0, 100, 100), matrix(1, 100, 100))

arr = List_2_Array(lst, verbose = FALSE)
```

load_binary *loads either 2- or 3-dimensional data from a binary file*

Description

loads either 2- or 3-dimensional data from a binary file

Usage

```
load_binary(path, type)
```

Arguments

path	a character string specifying a file path (where the binary data is saved)
type	a character string. Either '2d' or '3d' to indicate what kind of data data will be loaded from the specified *path*

Details

This function can be used to load either 2- or 3-dimensional data from a binary file. It is used in combination with the *superpixels* function in case that the *write_slic* parameter is not an empty string ("").

Examples

```
## Not run:

library(OpenImageR)

#-----------------------------------------
# assuming the saved data are 2-dimensional
#-----------------------------------------

path = "/my_dir/data.bin"

res = load_binary(path, type = '2d')


## End(Not run)
```

`MinMaxObject` *minimum and maximum values of vector, matrix, data frame or array*

Description

minimum and maximum values of vector, matrix, data frame or array

Usage

```
MinMaxObject(x)
```

Arguments

`x`	either a vector, matrix, data frame or array

Details

This helper function returns the minimum and maximum values of a vector, 2-dimensional or 3-dimensional objects. In case of a vector, matrix or data frame it returns a single value for the minimum and maximum of the object. In case of an array it returns the minimum and maximum values for each slice of the array.

Value

a list

Author(s)

Lampros Mouselimis

Examples

```
# vector
x = 1:10

res = MinMaxObject(x)


# matrix
x = matrix(runif(100), 10, 10)

res = MinMaxObject(x)


# data frame
x = data.frame(matrix(runif(100), 10, 10))

res = MinMaxObject(x)
```

```
# array
x = array(runif(100), dim = c(10, 10, 3))

res = MinMaxObject(x)
```

`NormalizeObject` *normalize a vector, matrix or array (in the range between 0 and 1)*

Description

normalize a vector, matrix or array (in the range between 0 and 1)

Usage

```
NormalizeObject(x)
```

Arguments

`x`	either a vector, matrix, data frame or array

Details

This is a helper function which normalizes all pixel values of the object to the range between 0 and 1. The function takes either a vector, matrix, data frame or array as input and returns a normalized object of the same type (in case of data frame it returns a matrix).

Value

either a normalized vector, matrix, or array

Author(s)

Lampros Mouselimis

Examples

```
# vector
x = 1:10

res = NormalizeObject(x)


# matrix
x = matrix(runif(100), 10, 10)
```

```
res = NormalizeObject(x)


# data frame
x = data.frame(matrix(runif(100), 10, 10))

res = NormalizeObject(x)


# array
x = array(runif(100), dim = c(10, 10, 3))

res = NormalizeObject(x)
```

`norm_matrix_range` *Normalize a matrix to specific range of values*

Description

Normalize a matrix to specific range of values

Usage

```
norm_matrix_range(data, min_value = -1, max_value = 1)
```

Arguments

`data`	a matrix
`min_value`	the new minimum value for the input *data*
`max_value`	the new maximum value for the input *data*

Value

a matrix

Examples

```
set.seed(1)
mt = matrix(1:48, 8, 6)

res = norm_matrix_range(mt, min_value = -1, max_value = 1)
```

padding *Padding of matrices or n-dimensional arrays with a user specified value*

Description

Padding of matrices or n-dimensional arrays with a user specified value

Usage

```
padding(input_data, new_rows, new_cols, fill_value = 0)
```

Arguments

input_data	either a matrix or a 3-dimensional array
new_rows	an integer specifying the new rows of the output matrix or array
new_cols	an integer specifying the new columns of the output matrix or array
fill_value	a numeric value to fill the extended rows / columns of the initial input data

Details

The *padding* function returns a list, where *data* is the padded / extended matrix or array and *padded_start*, *padded_end*, *padded_left* and *padded_right* are integer values specifying how many rows or columsn in up-, down-, left- or right-direction the input matrix or array was padded / extended with the specified fill-value.

Value

a list

Examples

```
library(OpenImageR)


#-------
# matrix
#-------

set.seed(1)
mt = matrix(runif(100), 10, 10)

res_mt = padding(mt, 15, 20, fill_value = -1)


#------
# array
```

```
#------

lst = list(matrix(1, 10, 10), matrix(2, 10, 10))

arr = List_2_Array(lst, verbose = FALSE)

res_arr = padding(arr, 15, 20, fill_value = mean(as.vector(mt)))
```

---

phash *calculation of the 'phash' of an image*

Description

This function calculates the phash of an image

Usage

```
phash(
  gray_image,
  hash_size = 8,
  highfreq_factor = 4,
  MODE = "hash",
  resize = "nearest"
)
```

Arguments

gray_image	a (2-dimensional) matrix or data frame
hash_size	an integer specifying the hash size (hash_size * highfreq_factor should be less than number of rows or columns of the gray_image)
highfreq_factor	an integer specyfing the highfrequency factor (hash_size * highfreq_factor should be less than number of rows or columns of the gray_image)
MODE	one of 'hash' (returns the hash of the image), 'binary' (returns binary identifier of the image)
resize	corresponds to one of 'nearest', 'bilinear' (resizing method)

Details

The function is a modification of the 'phash' function of the imagehash package [please consult the COPYRIGHT file]. The phash algorithm extends the average_hash by using the discrete cosine transform.

Value

either a hash-string or a binary vector

Examples

```
image = readImage(system.file("tmp_images", "2.jpg", package = "OpenImageR"))

image = rgb_2gray(image)

res_hash = phash(image, hash_size = 6, highfreq_factor = 3, MODE = 'hash')

res_binary = phash(image, hash_size = 6, highfreq_factor = 3, MODE = 'binary')
```

---

readImage *this function reads various types of images*

Description

Reads images of type .png, .jpeg, .jpg, .tiff

Usage

```
readImage(path, ...)
```

Arguments

path	a string specifying the path to the saved image
...	further arguments for the readPNG, readJPEG and readTIFF functions

Details

This function takes as input a string-path and returns the image in a matrix or array form. Supported types of images are .png, .jpeg, .jpg, .tiff. Extension types similar to .tiff such as .tif, .TIFF, .TIF are also supported

Value

the image in a matrix or array form

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)
```

`resizeImage` *resize an image using the 'nearest neighbors' or the 'bilinear' method*

Description

resize an image using the 'nearest neighbors' or the 'bilinear' method

Usage

```
resizeImage(image, width, height, method = "nearest")
```

Arguments

`image`	matrix or 3-dimensional array
`width`	a number specifying the new width of the image. Corresponds to the image-rows.
`height`	a number specifying the new height of the image. Corresponds to the image-columns.
`method`	one of 'nearest', 'bilinear'

Details

This function down- or upsamples an image using the 'nearest neighbors' or the 'bilinear' method

Value

depending on the input, either a matrix or an array

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "2.jpg", package = "OpenImageR")

image = readImage(path)

resiz = resizeImage(image, width = 32, height = 32, method = 'nearest')
```

rgb_2gray *convert an RGB image to Gray*

Description

convert an RGB image to Gray

Usage

```
rgb_2gray(RGB_image)
```

Arguments

RGB_image	a 3-dimensional array

Details

This function converts an RGB image to gray

Value

a matrix

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

gray = rgb_2gray(image)
```

RGB_to_HSV *Conversion of RGB to HSV colour type*

Description

Conversion of RGB to HSV colour type

Usage

```
RGB_to_HSV(input_data)
```

Arguments

input_data	a 3-dimensional array (RGB image)

Details

Meaning: RGB (Red-Green-Blue) to HSV (Hue, Saturation, Value) colour conversion

Examples

```
library(OpenImageR)

set.seed(1)
array_3d = array(sample(1:255, 675, replace = TRUE), c(15, 15, 3))

res = RGB_to_HSV(array_3d)
```

RGB_to_Lab *Conversion of RGB to Lab colour type*

Description

Conversion of RGB to Lab colour type

Usage

```
RGB_to_Lab(input_data)
```

Arguments

input_data	a 3-dimensional array (RGB image)

Details

Meaning: RGB (Red-Green-Blue) to LAB (Lightness, A-colour-dimension, B-colour-dimension) colour conversion

References

https://www.epfl.ch/labs/ivrl/research/snic-superpixels/

Examples

```
library(OpenImageR)

set.seed(1)
array_3d = array(sample(1:255, 675, replace = TRUE), c(15, 15, 3))

res = RGB_to_Lab(array_3d)
```

`rotateFixed` *Rotate an image by 90, 180, 270 degrees*

Description

Rotate an image by 90, 180, 270 degrees

Usage

```
rotateFixed(image, angle)
```

Arguments

`image`	matrix, data frame or 3-dimensional array
`angle`	one of 90, 180 and 270 degrees

Details

This function is faster than the rotateImage function as it rotates an image for specific angles (90, 180 or 270 degrees).

Value

depending on the input, either a matrix or an array

Examples

```
path = system.file("tmp_images", "3.jpeg", package = "OpenImageR")

image = readImage(path)

r = rotateFixed(image, 90)
```

rotateImage *Rotate an image using the 'nearest' or 'bilinear' method*

Description

Rotate an image by angle using the 'nearest' or 'bilinear' method

Usage

```
rotateImage(image, angle, method = "nearest", mode = "same", threads = 1)
```

Arguments

image	matrix, data frame or 3-dimensional array
angle	specifies the number of degrees
method	a string specifying the interpolation method when rotating an image ('nearest', 'bilinear')
mode	one of 'full', 'same' (same indicates that the ouput image will have the same dimensions with initial image)
threads	the number of cores to run in parallel

Details

This function rotates an image by a user-specified angle

Value

depending on the input, either a matrix or an array

Examples

```
path = system.file("tmp_images", "2.jpg", package = "OpenImageR")

image = readImage(path)

r = rotateImage(image, 75, threads = 1)
```

`superpixels` *SLIC and SLICO superpixel implementations*

Description

SLIC and SLICO superpixel implementations

Usage

```
superpixels(
  input_image,
  method = "slic",
  superpixel = 200,
  compactness = 20,
  return_slic_data = FALSE,
  return_lab_data = FALSE,
  return_labels = FALSE,
  write_slic = "",
  verbose = FALSE
)
```

Arguments

`input_image`	either a 2-dimensional or a 3-dimensional input image (the range of the pixel values should be preferably in the range 0 to 255)
`method`	a character string specifying the method to use. Either "slic" or "slico"
`superpixel`	a numeric value specifying the number of superpixels to use
`compactness`	a numeric value specifying the compactness parameter. The *compactness* parameter is needed only if *method* is "slic". The "slico" method adaptively chooses the compactness parameter for each superpixel differently.
`return_slic_data`	a boolean. If TRUE then the resulted slic or slico data will be returned
`return_lab_data`	a boolean. If TRUE then the Lab data will be returned (the Lab-colour format)
`return_labels`	a boolean. If TRUE then the labels will be returned
`write_slic`	a character string. If not an empty string ("") then it should be a path to the output file with extension .bin (for instance "/my_dir/output.bin"). The data will be saved in binary format.
`verbose`	a boolean. If TRUE then information will be printed in the R session

References

https://www.epfl.ch/labs/ivrl/research/slic-superpixels/

Examples

```
library(OpenImageR)

#-------------------
# 3-dimensional data
#-------------------

path = system.file("tmp_images", "slic_im.png", package = "OpenImageR")

im = readImage(path)

res = superpixels(input_image = im, method = "slic", superpixel = 200,

                  compactness = 20, return_slic_data = TRUE)


#-------------------
# 2-dimensional data
#-------------------

im_2d = im[,,1]

res_mt = superpixels(input_image = im_2d, method = "slic", superpixel = 200,

                     compactness = 20, return_slic_data = TRUE)
```

---

superpixel_bbox *Bounding box for the superpixel labels*

Description

Bounding box for the superpixel labels

Usage

```
superpixel_bbox(superpixel_labels, non_overlapping_superpixels = FALSE)
```

Arguments

`superpixel_labels`
: a matrix. The *superpixel_labels* parameter corresponds to the output *labels* of the *superpixels* function

`non_overlapping_superpixels`
: either TRUE or FALSE. If TRUE then besides the (x,y) coordinates of each superpixel-segment (matrix), the overlapping indices for each superpixel will be returned (list). See the details section for more information

Details

If the *non_overlapping_superpixels* parameter is set to *FALSE* then : the *superpixel_bbox* function returns the bounding box for the labels of the *superpixels* function. The output is a matrix which contains the min and max indices of the x-y-coordinates and the corresponding unique superpixel labels.

If the *non_overlapping_superpixels* parameter is set to *TRUE* then : the *superpixel_bbox* function returns besides the previously explained matrix also the overlapping indices for each superpixel. These indices can be used to overwrite pixels with a specific value (say 0.0), which might appear in two superpixels simultaneously. This feature might be useful in case a user intends to use an algorithm and the separability of superpixel-segments is of importance.

Therefore in both cases overlapping superpixels will be computed, however if the *non_overlapping_superpixels* parameter is set to *TRUE* then also a list of overlapping indices will be returned.

Examples

```
library(OpenImageR)


#-----------
# read image
#-----------

path = system.file("tmp_images", "slic_im.png", package = "OpenImageR")

im = readImage(path)

im = im[,, 1:3]


#--------------------
# compute superpixels
#--------------------

res = superpixels(input_image = im, method = "slic", superpixel = 200,

                  compactness = 20, return_labels = TRUE)


#-------------------------
# compute the bounding box
#-------------------------

bbox = superpixel_bbox(res$labels, non_overlapping_superpixels = FALSE)


#-----------------------------------------
# plot the bounding boxes of the superpixels ( for illustration purposes )
#-----------------------------------------
```

```
graphics::plot(1:ncol(im), type='n', xlim = c(ncol(im), 1), ylim = c(1, nrow(im)))

graphics::rasterImage( flipImage(im), 1, 1, ncol(im), nrow(im))


for (i in 1:nrow(bbox)) {

  # the order of the bounding box is c('xmin', 'ymin', 'xmax', 'ymax')
  graphics::rect(bbox[i,3], bbox[i,1], bbox[i,4], bbox[i,2], border = "red", lwd = 2)
}
```

`superpixel_bbox_subset` *Bounding box for a subset of superpixel labels*

Description

Bounding box for a subset of superpixel labels

Usage

```
superpixel_bbox_subset(superpixel_labels, superpixel_subset)
```

Arguments

`superpixel_labels`
: a matrix. The *superpixel_labels* parameter corresponds to the output *labels* of the *superpixels* function

`superpixel_subset`
: a numeric or integer vector specifying the subset of superpixel segments.

Details

This function should be utilized to return the bounding box for a subset of superpixel segments. To compute the bounding box for all superpixels use the *superpixel_bbox* function.

Examples

```
library(OpenImageR)


#-----------
# read image
#-----------

path = system.file("tmp_images", "slic_im.png", package = "OpenImageR")
```

```
im = readImage(path)

im = im[,, 1:3]


#--------------------
# compute superpixels
#--------------------

res = superpixels(input_image = im, method = "slic", superpixel = 200,

                  compactness = 20, return_labels = TRUE)


#-------------------------
# compute the bounding box   ( for subset of superpixels )
#-------------------------

bbox = superpixel_bbox_subset(res$labels, superpixel_subset = c(0, 10, 30))
```

`translation` *image translation*

Description

shift the position of an image by adding/subtracting a value to/from the X or Y coordinates

Usage

```
translation(image, shift_rows = 0, shift_cols = 0, padded_value = 0)
```

Arguments

`image`	a matrix, data frame or 3-dimensional array
`shift_rows`	a positive or negative integer specifying the direction that the rows should be shifted
`shift_cols`	a positive or negative integer specifying the direction that the columns should be shifted
`padded_value`	either a numeric value or a numeric vector of length 3 (corresponding to RGB). If it's not equal to 0 then the values of the shifted rows or columns will be filled with the user-defined padded_value

Details

If shift_rows is not zero then the image will be sifted row-wise (upsides or downsides depending on the sign). If shift_cols is not zero then the image will be sifted column-wise (right or left depending on the sign).

Value

a matrix or 3-dimensional array

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

res_tr = translation(image, shift_rows = 10, shift_cols = -10)
```

---

uniform_filter *uniform filter (convolution with uniform kernel)*

Description

uniform filter (convolution with uniform kernel)

Usage

```
uniform_filter(image, size, conv_mode = "same")
```

Arguments

image	matrix or 3-dimensional array
size	a 2-item vector specifying the horizontal and vertical dimensions of the uniform kernel, e.g. c(3,3)
conv_mode	the convolution mode should be one of 'same', 'full'

Details

This function applies a uniform filter to a matrix or to a 3-dimensional array

Value

depending on the input, either a matrix or an array

Author(s)

Lampros Mouselimis

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

filt = uniform_filter(image, c(4,4), conv_mode = "same")
```

---

writeImage *This function writes 2- or 3-dimensional image data to a file*

Description

This function writes 2- or 3-dimensional image data to a file. Supported types are .png, .jpeg, .jpg, .tiff (or .tif, .TIFF, .TIF)

Usage

```
writeImage(data, file_name, ...)
```

Arguments

data	a 2- or 3-dimensional object (matrix, data frame or array)
file_name	a string specifying the name of the new file
...	further arguments for the writePNG, writeJPEG and writeTIFF functions

Details

This function takes as input a matrix, data frame or array and saves the data in one of the supported image types (.png, .jpeg, .jpg, .tiff). Extension types similar to .tiff such as .tif, .TIFF, .TIF are also supported

Value

a saved image file

Examples

```
# path = system.file("tmp_images", "1.png", package = "OpenImageR")

# im = readImage(path)

# writeImage(im, 'new_image.jpeg')
```

ZCAwhiten *zca whiten of an image*

Description

this function performs zca-whitening to a 2- or 3- dimensional image

Usage

```
ZCAwhiten(image, k, epsilon)
```

Arguments

image	a matrix, data frame or 3-dimensional array
k	an integer specifying the number of components to keep when svd is performed (reduced dimension representation of the data)
epsilon	a float specifying the regularization parameter

Details

Whitening (or sphering) is the preprocessing needed for some algorithms. If we are training on images, the raw input is redundant, since adjacent pixel values are highly correlated. When using whitening the features become less correlated and all features have the same variance.

Value

a matrix or 3-dimensional array

References

http://ufldl.stanford.edu/wiki/index.php/Whitening

Examples

```
path = system.file("tmp_images", "1.png", package = "OpenImageR")

image = readImage(path)

res = ZCAwhiten(image, k = 20, epsilon = 0.1)
```

Index