The biOps Package

August 14, 2007

Type Package

Title Basic image operations and image processing

Version 0.1-1

Date 2007-08-02

Author Matias Bordese, Walter Alini

Maintainer Matias Bordese <mbordese@gmail.com>

Encoding UTF-8

Description This package includes arithmetic, logic, look up table and geometric operations. Some image processing functions, for edge detection (several algorithms including roberts, sobel, kirsch, marr-hildreth, canny) and operations by convolution masks (with predefined as well as user defined masks) are provided. Supported file formats are jpeg and tiff (it requires libtiff and libjpeg libraries installed).

SystemRequirements libtiff, libjpeg

License GPL

R topics documented:

`biOps-package` *Basic image operations*

Description

Basic image operations. It includes: arithmetic, logic, look up table and geometric operations. The supported file formats are jpeg and tiff.

Details

Package:	biOps
Type:	Package
Version:	0.1
Date:	2007-06-18
License:	GPL
Built:	R 2.2.1; i486-pc-linux-gnu; 2007-06-27 18:02:45; unix

Index:

```
biOps-package           Basic image operations
```

imageType	Get information on color type of imagedata
imagedata	Generate an imagedata
imgAND	And two images
imgAdd	Add two images
imgAverage	Average images
imgAverageShrink	Shrink an image
imgBilinearRotate	Rotate an image
imgBilinearScale	Scale an image
imgBlueBand	Return the image blue band
imgCubicRotate	Rotate an image
imgCubicScale	Scale an image
imgDecreaseContrast	Decrease contrast
imgDecreaseIntensity	Decrease intensity
imgDiffer	Substract two images
imgDivide	Divide two images
imgGamma	Gamma correct an image
imgGreenBand	Return the image green band
imgHorizontalMirroring	Horizontal mirror an image
imgIncreaseContrast	Increase contrast
imgIncreaseIntensity	Increase intensity
imgMedianShrink	Shrink an image
imgMultiply	Multiply two images
imgNearestNeighborRotate	Rotate an image
imgNearestNeighborScale	Scale an image
imgNegative	Negate an image
imgNormalize	Normalization for vector and matrix
imgOR	Or two images
imgRGB2Grey	Convert color imagedata to grey imagedata
imgRedBand	Return the image red band
imgRotate	Rotate an image
imgRotate90Clockwise	Rotate an image
imgRotate90CounterClockwise	Rotate an image
imgScale	Scale an image
imgSplineRotate	Rotate an image
imgSplineScale	Scale an image
imgThreshold	Threshold an image
imgTranslate	Translate an image block
imgVerticalMirroring	Vertical mirror an image
imgXOR	Xor two images
logo	R logo imagedata
plot.imagedata	Plotting an imagedata object
print.imagedata	Print information on a given imagedata object
r_dec_contrast	Decrease contrast
r_dec_intensity	Decrease intensity

`r_gamma_img`	Gamma correct an image
`r_imgAdd`	Add two images
`r_imgAverage`	Average images
`r_imgDiffer`	Substract two images
`r_inc_contrast`	Increase contrast
`r_inc_intensity`	Increase intensity
`r_negative`	Negate an image
`r_threshold`	Threshold an image
`readJpeg`	Read jpeg file
`readTiff`	Read tiff file
`violet.picture`	JPEG picture of a violet flower
`writeJpeg`	Write jpeg file
`writeTiff`	Write tiff file

Author(s)

MatÃ■as Bordese, Walter Alini

Maintainer: MatÃ■as Bordese <mbordese@yahoo.com>

`imageType` *Get information on color type of imagedata*

Description

This function returns color type ("rgb" or "grey") of a given imagedata.

Usage

```
imageType(x)
```

Arguments

`x`	The image

Value

"rgb" or "grey"

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                cat("Image Type", imageType(x))

## End(Not run)
```

imagedata *Generate an imagedata*

Description

This function makes an imagedata object from a matrix. This data structure is primary data structure to represent image in biOps package.

Usage

```
imagedata(mat, type=NULL, ncol=dim(mat)[1], nrow=dim(mat)[2])
```

Arguments

`mat`	array, matrix or vector
`type`	"rgb" or "grey"
`ncol`	width of image
`nrow`	height of image

Details

For grey scale image, matrix should be given in the form of 2 dimensional matrix. First dimension is row, and second dimension is column.

For rgb image, matrix should be given in the form of 3 dimensional array (row, column, channel). mat[,,1], mat[,,2], mat[,,3] are red plane, green plane and blue plane, respectively.

You can omit 'type' specification if you give a proper array or matrix.

Value

return an imagedata object

See Also

`plot.imagedata` `print.imagedata`

Examples

```
        p <- q <- seq(-1, 1, length=20)
        r <- 1 - outer(p^2, q^2, "+") / 2
        plot(imagedata(r))
```

imgAND *And two images*

Description

This function does a logic AND between two images and returns a new image.

Usage

```
imgAND(imgdata1, imgdata2)
```

Arguments

imgdata1	The first image
imgdata2	The second image

Value

return an imagedata object

See Also

imgOR imgXOR

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgAND(x, x)

## End(Not run)
```

imgAdd *Add two images*

Description

This function adds two images and returns a new image.

Usage

```
imgAdd(imgdata1, imgdata2)
```

Arguments

imgdata1	The first image
imgdata2	The second image

Value

return an imagedata object

Note

To add a constant c to an image you can just do: »> imgdata + c.

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgAdd(x, x)

## End(Not run)
```

imgAverage *Average images*

Description

This function calculates the average of the given images and returns a new image.

Usage

```
imgAverage(imgdata_list)
```

Arguments

`imgdata_list` An image list

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgAverage(list(x, x))

## End(Not run)
```

imgAverageShrink *Shrink an image*

Description

This function shrinks an image using the average and returns a new image.

Usage

```
imgAverageShrink(imgdata, x_scale, y_scale)
```

Arguments

`imgdata`	The image
`x_scale`	The horizontal scale factor
`y_scale`	The vertical scale factor

Value

return an imagedata object

Note

The scale factors are expected to be less than 1.

See Also

`imgMedianShrink` `imgNearestNeighborScale` `imgBilinearScale` `imgCubicScale`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgAverageShrink(x, 0.5, 0.5)

## End(Not run)
```

imgBilinearRotate *Rotate an image*

Description

This function rotates an image using bilinear interpolation and returns a new image.

Usage

```
imgBilinearRotate(imgdata, angle)
```

Arguments

imgdata	The image
angle	The clockwise deg angle to rotate

Value

return an imagedata object

See Also

imgRotate imgNearestNeighborRotate imgCubicRotate imgSplineRotate imgRotate90Clockwise
imgRotate90CounterClockwise

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgBilinearRotate(x, 45)

## End(Not run)
```

imgBilinearScale *Scale an image*

Description

This function scales an image using bilinear interpolation and returns a new image.

Usage

```
imgBilinearScale(imgdata, x_scale, y_scale)
```

Arguments

imgdata	The image
x_scale	The horizontal scale factor
y_scale	The vertical scale factor

Value

return an imagedata object

Note

The scale factors are expected to be greater than 1. To reduce an image use the minification functions instead.

See Also

imgScale imgNearestNeighborScale imgCubicScale imgSplineScale imgMedianShrink imgAverageShrink

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgBilinearScale(x, 1.5, 1.5)

## End(Not run)
```

---

imgBlueBand *Return the image blue band*

Description

This function returns the blue band of the imagedata.

Usage

```
imgBlueBand(x)
```

Arguments

x	The image

Value

grey imagedata

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                plot(imgBlueBand(x))

## End(Not run)
```

imgBlur *Blurs an image*

Description

This function blurs an image by convoluting with the following matrix:

$$\begin{array}{ccc} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{array}$$

Usage

```
imgBlur(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

See Also

`imgStdBlur`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgStdBlur(x)

## End(Not run)
```

`imgBoost` *High Boosts an image*

Description

This function high boosts an image by convoluting with the following matrix:

-1/9	-1/9	-1/9
-1/9	(9p-1)/9	-1/9
-1/9	-1/9	-1/9

It increases intensity by a given proportion (p) and substracting a lowpass filter

Usage

```
imgBoost(imgdata, proportion)
```

Arguments

`imgdata`	The image
`proportion`	Proportion of intensity to be increased (optional: default = 1 -HighPassFilter-)

Value

return an imagedata object

Note

When proportion=1, it's the same as `imgHighPassFilter`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgBoost(x, 1.2)

## End(Not run)
```

`imgCanny` *Canny Edge Detection Method*

Description

This function does edge detection using the Canny algorithm.

Usage

```
imgCanny(imgdata, sigma, low=0, high=-1)
```

Arguments

`imgdata`	The image
`sigma`	The standard deviation used for the gaussian smoothing convolution
`low`	The lower threshold for hysteresis
`high`	The higher threshold for hysteresis

Value

return an imagedata object

Note

If not specified, the low and high parameters are estimated based in a histogram of the image.

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgCanny(x, 0.7)

## End(Not run)
```

`imgConvolve` *Performs an image convolution*

Description

This function performs an image convolution with given mask

Usage

```
imgConvolve(imgdata, mask, bias)
```

Arguments

`imgdata`	The image
`mask`	Kernel's convolution matrix
`bias`	Value to be added to each pixel after method is applied (used to correct some expected behaviour). This argument is optional (default = 32)

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                m <- matrix(c(1,2,1,2,4,2,1,2,1)/16, 3, 3, byrow = TRUE)
                y <- imgConvolve(x, m, 64)

## End(Not run)
```

imgCrop *Crops an image*

Description

This function crops image.

Usage

```
imgCrop(imgdata, x_start, y_start, c_width, c_height)
```

Arguments

imgdata	The image
x_start	Upper left x coordinate of source block
y_start	Upper left y coordinate of source block
c_width	Width of the block to crop
c_height	Height of the block to crop

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgCrop(x, 100, 50, 100, 50)

## End(Not run)
```

imgCubicRotate *Rotate an image*

Description

This function rotates an image using cubic interpolation and returns a new image.

Usage

```
imgCubicRotate(imgdata, angle)
```

Arguments

imgdata	The image
angle	The clockwise deg angle to rotate

Value

return an imagedata object

See Also

imgRotate imgNearestNeighborRotate imgBilinearRotate imgSplineRotate imgRotate90Clockwise imgRotate90CounterClockwise

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgCubicRotate(x, 45)

## End(Not run)
```

imgCubicScale *Scale an image*

Description

This function scales an image using cubic interpolation and returns a new image.

Usage

```
imgCubicScale(imgdata, x_scale, y_scale)
```

Arguments

imgdata	The image
x_scale	The horizontal scale factor
y_scale	The vertical scale factor

Value

return an imagedata object

Note

The scale factors are expected to be greater than 1. To reduce an image use the minification functions instead.

See Also

imgScale imgNearestNeighborScale imgBilinearScale imgSplineScale imgMedianShrink imgAverageShrink

Examples

```
          ## Not run:
                  x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                  y <- imgCubicScale(x, 1.5, 1.5)

## End(Not run)
```

imgDecreaseContrast

Decrease contrast

Description

This function decreases an image contrast, leaving each pixel value between given values.

Usage

```
imgDecreaseContrast(imgdata, min_desired, max_desired)
```

Arguments

`imgdata`	The image
`min_desired`	The min value
`max_desired`	The max value

Value

return an imagedata object

See Also

`imgIncreaseContrast` `r_dec_contrast` `r_inc_contrast`

Examples

```
          ## Not run:
                  x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                  y <- imgDecreaseContrast(x, 60, 200)

## End(Not run)
```

imgDecreaseIntensity *Decrease intensity*

Description

This function decreases an image intensity by a given factor.

Usage

```
imgDecreaseIntensity(imgdata, percentage)
```

Arguments

`imgdata`	The image
`percentage`	A non negative value representing the intensity percentage to be decreased. 1 stands for 100% (eg. 0.5 = 50%).

Value

return an imagedata object

See Also

imgIncreaseIntensity r_dec_intensity r_inc_intensity

Examples

```
        ## Not run:
               x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
               y <- imgDecreaseIntensity(x, 0.3)

## End(Not run)
```

imgDiffer *Substract two images*

Description

This function substracts two images and returns a new image, imgdata1 - imgdata2.

Usage

```
imgDiffer(imgdata1, imgdata2)
```

Arguments

`imgdata1`	The first image
`imgdata2`	The second image

Value

return an imagedata object

Note

To substract a constant c to an image you can just do: »> imgdata - c.

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgDiffer(x, x)

## End(Not run)
```

`imgDifferenceEdgeDetection` *Enhaces image edges*

Description

This function enhaces image's edge by the difference method. It uses a 3x3 matrix to determine the current pixel value (by getting the maximum value between the distances of matrix's opposite neighbors

Usage

```
imgDifferenceEdgeDetection(imgdata, bias)
```

Arguments

`imgdata`	The image
`bias`	Value to be added to each pixel after method is applied (used to correct some expected behaviour). This argument is optional (default = 32)

Value

return an imagedata object

See Also

`imgHomogeneityEdgeDetection`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgDifferenceEdgeDetection(x, bias=64)

## End(Not run)
```

imgDivide *Divide two images*

Description

This function divides two images and returns a new image.

Usage

```
imgDivide(imgdata1, imgdata2)
```

Arguments

imgdata1	The first image
imgdata2	The second image

Value

return an imagedata object

Note

To divide an image by a constant c you can just do: »> imgdata / c.

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgDivide(x, x)

## End(Not run)
```

imgFreiChen *Frei-Chen Edge Detection Method*

Description

This function enhaces image's edges by convoluting with the Frei-Chen method matrices:

	H_r				**H_c**	
1	0	-1	||	-1	$-\sqrt{2}$	-1
$\sqrt{2}$	0	$-\sqrt{2}$	||	0	0	0
1	0	-1	||	1	$\sqrt{2}$	1

Usage

```
imgFreiChen(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgFreiChen(x)

## End(Not run)
```

`imgGamma` *Gamma correct an image*

Description

This function applies gamma operation to a given image. Each pixel value is taken to the inverse of gamma_value-th exponent.

Usage

```
imgGamma(imgdata, gamma_value)
```

Arguments

`imgdata`	The image
`gamma_value`	A non negative value representing operation gamma value

Value

return an imagedata object

See Also

`r_gamma`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgGamma(x, 1.3)

## End(Not run)
```

`imgGetRGBFromBands` *Return an RGB image*

Description

This function returns the RGB image compositing the given bands.

Usage

```
imgGetRGBFromBands(R, G, B)
```

Arguments

`R`	A one-band image for the Red band
`G`	A one-band image for the Green band
`B`	A one-band image for the Blue band

Value

RGB imagedata

Examples

```
        ## Not run:
x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                r <- imgRedBand(x)
                g <- imgGreenBand(x)
                b <- imgBlueBand(x)
                rgb <- imgGetRGBFromBands(r, g, b)

## End(Not run)
```

imgGreenBand *Return the image green band*

Description

This function returns the green band of the imagedata.

Usage

```
imgGreenBand(x)
```

Arguments

x	The image

Value

grey imagedata

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                plot(imgGreenBand(x))

## End(Not run)
```

imgHighPassFilter *Sharpens an image*

Description

This function sharpens an image by convoluting with the following matrix:

$$\begin{array}{ccc} -1/9 & -1/9 & -1/9 \\ -1/9 & 8/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{array}$$

Usage

```
imgHighPassFilter (imgdata)
```

Arguments

imgdata	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgHighPassFilter(x)

## End(Not run)
```

---

imgHistogram *Return the image histogram*

Description

This function returns the image pixel values histogram.

Usage

```
imgHistogram(x, main='Image Histogram', col='Midnight Blue', ...)
```

Arguments

`x`	The image
`main`	The histogram title
`col`	The histogram bars color
`...`	Same options of hist function

Value

histogram object

See Also

`hist`

Examples

```
        x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
        h <- imgHistogram(x)
```

imgHomogeneityEdgeDetection

Enhaces image edges

Description

This funtions enhaces image's edge by the homogeneity method. It uses a 3x3 matrix to determine the current pixel value (by getting the maximum value between the distances of the pixel and its neighbors)

Usage

```
imgHomogeneityEdgeDetection(imgdata, bias)
```

Arguments

imgdata	The image
bias	Value to be added to each pixel after method is applied (used to correct some expected behaviour). This argument is optional (default = 32)

Value

return an imagedata object

See Also

imgHomogeneityEdgeDetection

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgHomogeneityEdgeDetection(x, bias=64)

## End(Not run)
```

imgHorizontalMirroring

Horizontal mirror an image

Description

This function flips an image about the y axis.

Usage

```
imgHorizontalMirroring(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

See Also

`imgVerticalMirroring`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgHorizontalMirroring(x)

## End(Not run)
```

`imgIncreaseContrast` *Increase contrast*

Description

This function increases an image contrast, augmenting pixel values differences between given limits (in a linear fashion).

Usage

```
imgIncreaseContrast(imgdata, min_limit, max_limit)
```

Arguments

`imgdata`	The image
`min_limit`	The minimum limit to apply lineal modification
`max_limit`	The maximum limit to apply lineal modification

Value

return an imagedata object

See Also

`imgDecreaseContrast` `r_inc_contrast` `r_dec_contrast`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgIncreaseContrast(x, 60, 200)

## End(Not run)
```

imgIncreaseIntensity *Increase intensity*

Description

This function increases an image intensity by a given factor.

Usage

```
imgIncreaseIntensity(imgdata, percentage)
```

Arguments

`imgdata`	The image
`percentage`	A non negative value representing the intensity percentage to be increased. 1 stands for 100% (eg. 0.5 = 50%)

Value

return an imagedata object

See Also

`imgDecreaseIntensity` `r_inc_intensity` `r_dec_intensity`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgIncreaseIntensity(x, 0.3)

## End(Not run)
```

imgKirsch *Kirsch Edge Detection Method*

Description

This function enhaces image's edges by convoluting with the Kirsch method. Base matrix is:

5	-3	-3
5	0	-3
5	-3	-3

Usage

```
imgKirsch(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgKirsch(x)

## End(Not run)
```

`imgMarrHildreth` *Marr-Hildreth Edge Detection Method*

Description

This function does edge detection using the Marr-Hildreth algorithm.

Usage

```
imgMarrHildreth(imgdata, sigma)
```

Arguments

`imgdata`	The image
`sigma`	The standard deviation of Gaussian for convolution

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgMarrHildreth(x, 2)

## End(Not run)
```

imgMaximum *Calculates image maximum*

Description

This function calculates the maximum of the given images and returns a new image.

Usage

```
imgMaximum(imgdata_list)
```

Arguments

imgdata_list	An image list

Value

return an imagedata object

Examples

```
        ## Not run: 
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgMaximum(list(x, x))

## End(Not run)
```

imgMedianShrink *Shrink an image*

Description

This function shrinks an image using the median and returns a new image.

Usage

```
imgMedianShrink(imgdata, x_scale, y_scale)
```

Arguments

imgdata	The image
x_scale	The horizontal scale factor
y_scale	The vertical scale factor

Value

return an imagedata object

Note

The scale factors are expected to be less than 1.

See Also

imgAverageShrink imgNearestNeighborScale imgBilinearScale imgCubicScale

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgMedianShrink(x, 0.5, 0.5)

## End(Not run)
```

---

imgMultiply *Multiply two images*

Description

This function multiplies two images and returns a new image.

Usage

```
imgMultiply(imgdata1, imgdata2)
```

Arguments

imgdata1	The first image
imgdata2	The second image

Value

return an imagedata object

Note

To multiply an image by a constant c you can just do: »> imgdata * c.

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgMultiply(x, x)

## End(Not run)
```

imgNearestNeighborRotate

Rotate an image

Description

This function rotates an image using nearest neighbor interpolation and returns a new image.

Usage

```
imgNearestNeighborRotate(imgdata, angle)
```

Arguments

imgdata	The image
angle	The clockwise deg angle to rotate

Value

return an imagedata object

See Also

imgRotate imgBilinearRotate imgCubicRotate imgSplineRotate imgRotate90Clockwise imgRotate90CounterClockwise

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgNearestNeighborRotate(x, 45)

## End(Not run)
```

imgNearestNeighborScale

Scale an image

Description

This function scales an image using nearest neighbor interpolation and returns a new image.

Usage

```
imgNearestNeighborScale(imgdata, x_scale, y_scale)
```

Arguments

`imgdata`	The image
`x_scale`	The horizontal scale factor
`y_scale`	The vertical scale factor

Value

return an imagedata object

Note

The scale factors are expected to be greater than 1. To reduce an image use the minification functions instead.

See Also

`imgScale` `imgBilinearScale` `imgCubicScale` `imgSplineScale` `imgMedianShrink` `imgAverageShrink`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgNearestNeighborScale(x, 1.5, 1.5)

## End(Not run)
```

`imgNegative` *Negate an image*

Description

This function negates an image.

Usage

```
imgNegative(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

See Also

r_negative r_negative_lut

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgNegative(x)

## End(Not run)
```

---

imgNormalize *Normalization for vector and matrix*

Description

This function normalizes image so that the minimum value is 0 and the maximum value is 1.

Usage

```
imgNormalize(x)
```

Arguments

x	The image

Value

Data of the same type as 'x', in which minimum value is 0 and maximum value is 255.

Examples

```
        ## Not run:
                data(logo)
                plot(imgNormalize(logo))

## End(Not run)
```

`imgOR` *Or two images*

Description

This function does a logic OR between two images and returns a new image.

Usage

```
imgOR(imgdata1, imgdata2)
```

Arguments

`imgdata1`	The first image
`imgdata2`	The second image

Value

return an imagedata object

See Also

`imgAND` `imgXOR`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgOR(x, x)

## End(Not run)
```

`imgPrewitt` *Prewitt Edge Detection Method*

Description

This function enhaces image's edges by convoluting with the Prewitt method matrices:

	H_r				**H_c**	
1	0	-1	||	-1	-1	-1
1	0	-1	||	0	0	0
1	0	-1	||	1	1	1

Usage

```
imgPrewitt(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgPrewitt(x)

## End(Not run)
```

`imgPrewittCompassGradient`
Prewitt Compass Gradient Edge Detection Method

Description

This function enhaces image's edges by convoluting with the Prewitt method. Base matrix is:

$$\begin{array}{ccc} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{array}$$

Usage

```
imgPrewittCompassGradient(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgPrewittCompassGradient(x)
```

```
## End(Not run)
```

imgRGB2Grey *Convert color imagedata to grey imagedata*

Description

This function convert color imagedata to grey imagedata.

Usage

```
imgRGB2Grey(x, coefs=c(0.30, 0.59, 0.11))
```

Arguments

x	The image
coefs	The coefficients for red, green and blue bands

Value

grey imagedata

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                plot(imgRGB2Grey(x))

## End(Not run)
```

imgRedBand *Return the image red band*

Description

This function returns the red band of the imagedata.

Usage

```
imgRedBand(x)
```

Arguments

x	The image

Value

grey imagedata

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                plot(imgRedBand(x))

## End(Not run)
```

`imgRoberts` *Roberts Edge Detection Method*

Description

This function enhaces image's edges by convoluting with the Roberts method matrices:

	H_r				**H_c**	
0	0	-1	||	-1	0	0
0	1	0	||	0	1	0
0	0	0	||	0	0	0

Usage

```
imgRoberts(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgRoberts(x)

## End(Not run)
```

`imgRobinson3Level` *Robinson 3-level Edge Detection Method*

Description

This function enhaces image's edges by convoluting with the Robinson 3-level method. Base matrix is:

$$\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

Usage

```
imgRobinson3Level(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgRobinson3Level(x)

## End(Not run)
```

`imgRobinson5Level` *Robinson 5-level Edge Detection Method*

Description

This function enhaces image's edges by convoluting with the Robinson 5-level method. Base matrix is:

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Usage

```
imgRobinson5Level(imgdata)
```

Arguments

imgdata	The image

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgRobinson5Level(x)

## End(Not run)
```

imgRotate *Rotate an image*

Description

This function rotates an image using the given interpolation and returns a new image.

Usage

```
imgRotate(imgdata, angle, interpolation)
```

Arguments

imgdata	The image
angle	The clockwise deg angle to rotate
interpolation	The interpolation method: nearestneighbor | bilinear | cubic | spline

Value

return an imagedata object

See Also

imgNearestNeighborRotate imgBilinearRotate imgCubicRotate imgSplineRotate imgRotate90Clockwise imgRotate90CounterClockwise

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgRotate(x, 45, 'spline')

## End(Not run)
```

imgRotate90Clockwise

Rotate an image

Description

This function rotates the image 90 degrees clockwise.

Usage

```
imgRotate90Clockwise(imgdata)
```

Arguments

`imgdata` The image

Value

return an imagedata object

See Also

`imgRotate90CounterClockwise`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgRotate90Clockwise(x)

## End(Not run)
```

imgRotate90CounterClockwise *Rotate an image*

Description

This function rotates the image 90 degrees counter-clockwise.

Usage

```
imgRotate90CounterClockwise(imgdata)
```

Arguments

imgdata	The image

Value

return an imagedata object

See Also

imgRotate90Clockwise

Examples

```
        ## Not run: 
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgRotate90CounterClockwise(x)

## End(Not run)
```

imgScale *Scale an image*

Description

This function scales an image using the given interpolation and returns a new image.

Usage

```
imgScale(imgdata, x_scale, y_scale, interpolation)
```

Arguments

`imgdata` The image

`x_scale` The horizontal scale factor

`y_scale` The vertical scale factor

`interpolation`
The interpolation method: nearestneighbor | bilinear | cubic | spline

Value

return an imagedata object

Note

The scale factors are expected to be greater than 1. To reduce an image use the minification functions instead.

See Also

`imgNearestNeighborScale` `imgBilinearScale` `imgCubicScale` `imgSplineScale` `imgMedianShrink` `imgAverageShrink`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgScale(x, 1.5, 1.5, 'bilinear')

## End(Not run)
```

`imgSharpen` *Sharpens an image with selected mask*

Description

This function sharpens an image by convoluting with one of the following matrices:

	1				**2**				**3**	1
0	-1	0	||	-1	-1	-1	||	1	-2	1
-1	5	-1	||	-1	9	-1	||	-2	5	-2
0	-1	0	||	-1	-1	-1	||	1	-2	1

Usage

```
imgSharpen (imgdata, mask)
```

Arguments

`imgdata` The image

`mask` The matrix to be used in the convolution. Must be one of 1, 2, 3 (default=1)

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgSharpen(x, 2)

## End(Not run)
```

`imgShenCastan` *Shen-Castan Edge Detection Method*

Description

This function does edge detection using the Shen-Castan algorithm.

Usage

```
imgShenCastan(imgdata, smooth_factor=0.9, thin_factor=2, adapt_window=7, thresh_rat
```

Arguments

`imgdata` The image

`smooth_factor` The smooth factor

`thin_factor` The thinning factor

`adapt_window` The size of the window for adaptive gradient

`thresh_ratio` The percentage of pixels to be above high threshold

`do_hysteresis` If true, do hysteresis

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgShenCastan(x)

## End(Not run)
```

`imgSobel` *Sobel Edge Detection Method*

Description

This function enhaces image's edges by convoluting with the Sobel method matrices:

	H_r				**H_c**	
1	0	-1	||	-1	-2	-1
2	0	-2	||	0	0	0
1	0	-1	||	1	2	1

Usage

```
imgSobel(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

Examples

```
        x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
        y <- imgSobel(x)
```

`imgSplineRotate` *Rotate an image*

Description

This function rotates an image using b-spline interpolation and returns a new image.

Usage

```
imgSplineRotate(imgdata, angle)
```

Arguments

`imgdata`	The image
`angle`	The clockwise deg angle to rotate

Value

return an imagedata object

See Also

imgRotate imgNearestNeighborRotate imgBilinearRotate imgCubicRotate imgRotate90Clockwise imgRotate90CounterClockwise

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgSplineRotate(x, 45)

## End(Not run)
```

---

imgSplineScale *Scale an image*

Description

This function scales an image using b-spline interpolation and returns a new image.

Usage

```
imgSplineScale(imgdata, x_scale, y_scale)
```

Arguments

imgdata	The image
x_scale	The horizontal scale factor
y_scale	The vertical scale factor

Value

return an imagedata object

Note

The scale factors are expected to be greater than 1. To reduce an image use the minification functions instead.

See Also

imgScale imgNearestNeighborScale imgBilinearScale imgCubicScale imgMedianShrink imgAverageShrink

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgSplineScale(x, 1.5, 1.5)

## End(Not run)
```

`imgStdBlur` *Blurs an image*

Description

This function blurs an image by convoluting with a average square matrix

Usage

```
imgStdBlur(imgdata, dim)
```

Arguments

`imgdata`	The image
`dim`	Square matrix dimension (optional, default = 5)

Value

return an imagedata object

See Also

`imgBlur`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgStdBlur(x, 3)

## End(Not run)
```

imgThreshold *Threshold an image*

Description

This function thresholds an image using a given filter.

Usage

```
imgThreshold(imgdata, thr_value)
```

Arguments

imgdata	The image
thr_value	Filter value for thresholding

Value

return an imagedata object

See Also

r_threshold

Examples

```
        x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
        y <- imgThreshold(x, 80)
```

imgTranslate *Translate an image block*

Description

This function translates an image block and returns a new image.

Usage

```
imgTranslate(imgdata, x_start, y_start, x_end, y_end, t_width, t_height)
```

Arguments

`imgdata`	The image
`x_start`	Upper left x coordinate of source block
`y_start`	Upper left y coordinate of source block
`x_end`	Upper left x coordinate of destination block
`y_end`	Upper left y coordinate of destination block
`t_width`	Width of the block to move
`t_height`	Height of the block to move

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgTranslate(x, 100, 100, 200, 200, 50, 50)

## End(Not run)
```

`imgUnsharpen` *Unsharpens an image with selected mask*

Description

This function unsharpens an image by convoluting with one of the following matrices:

	1		||		**2**		||		**3**	1
0	-1	0	||	-1	-1	-1	||	1	-2	1
-1	5	-1	||	-1	9	-1	||	-2	5	-2
0	-1	0	||	-1	-1	-1	||	1	-2	1

Performs a difference between original image and sharpen convolved image with the specified mask

Usage

```
imgUnsharpen (imgdata, mask)
```

Arguments

`imgdata`	The image
`mask`	The matrix to be used in the convolution. Must be one of 1, 2, 3 (default=1)

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgUnsharpen(x, 2)

## End(Not run)
```

imgVerticalMirroring

Vertical mirror an image

Description

This function flips an image about the x axis.

Usage

```
imgVerticalMirroring(imgdata)
```

Arguments

`imgdata`	The image

Value

return an imagedata object

See Also

`imgHorizontalMirroring`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgVerticalMirroring(x)

## End(Not run)
```

imgXOR *Xor two images*

Description

This function does a logic XOR between two images and returns a new image.

Usage

```
imgXOR(imgdata1, imgdata2)
```

Arguments

`imgdata1`	The first image
`imgdata2`	The second image

Value

return an imagedata object

See Also

`imgOR` `imgAND`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- imgXOR(x, x)

## End(Not run)
```

logo *R logo imagedata*

Description

The imagedata object of R logo of the size 101x77.

Usage

```
data(logo)
```

Format

imagedata

Examples

```
        ## Not run:
                data(logo)
                plot(logo)

## End(Not run)
```

plot.imagedata *Plotting an imagedata object*

Description

This function outputs an imagedata object as an image.

Usage

```
plot.imagedata(x, ...)
```

Arguments

x	The image
...	Plotting options

See Also

imagedata

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                plot(x)

## End(Not run)
```

print.imagedata *Print information on a given imagedata object*

Description

This function outputs information on a given imagedata object.

Usage

```
print.imagedata(x, ...)
```

Arguments

`x`	The image
`...`	Ignored

See Also

`imagedata`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                print(x)

## End(Not run)
```

`r_dec_contrast` *Decrease contrast*

Description

This function decreases an image contrast, leaving each pixel value between given values.

Usage

```
r_dec_contrast(imgdata, min_desired, max_desired)
```

Arguments

`imgdata`	The image
`min_desired`	The min value
`max_desired`	The max value

Value

return an imagedata object

Note

This is the R implementation of imgDecreaseContrast.

See Also

`imgDecreaseContrast` `imgIncreaseContrast` `r_inc_contrast`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_dec_contrast(x, 60, 200)

## End(Not run)
```

`r_dec_intensity` *Decrease intensity*

Description

This function decreases an image intensity by a given factor.

Usage

```
r_dec_intensity(imgdata, percentage)
```

Arguments

`imgdata`	The image
`percentage`	A non negative value representing the intensity percentage to be decreased. 1 stands for 100% (eg. 0.5 = 50%).

Value

return an imagedata object

Note

This is the R implementation of imgDecreaseIntensity.

See Also

`imgDecreaseIntensity` `imgIncreaseIntensity` `r_inc_intensity`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_dec_intensity(x, 0.3)

## End(Not run)
```

r_gamma *Gamma correct an image*

Description

This function applies gamma operation to a given image. Each pixel value is taken to the inverse of gamma_value-th exponent

Usage

```
r_gamma(imgdata, gamma_value)
```

Arguments

imgdata	The image
gamma_value	A non negative value representing operation gamma value

Value

return an imagedata object

Note

This is the R implementation of imgGamma.

See Also

imgGamma

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_gamma(x, 1.3)

## End(Not run)
```

r_imgAdd *Add two images*

Description

This function adds two images and returns a new image.

Usage

```
r_imgAdd(imgdata1, imgdata2)
```

Arguments

imgdata1	The first image
imgdata2	The second image

Value

return an imagedata object

Note

This is the R implementation of imgAdd.

See Also

imgAdd

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_imgAdd(x, x)

## End(Not run)
```

r_imgAverage *Average images*

Description

This function calculates the average of the given images and returns a new image.

Usage

```
r_imgAverage(imgdata_list)
```

Arguments

`imgdata_list` An image list

Value

return an imagedata object

Note

This is the R implementation of imgAverage.

See Also

`imgAverage`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_imgAverage(list(x, x))

## End(Not run)
```

`r_imgDiffer` *Substract two images*

Description

This function substracts two images and returns a new image, imgdata1 - imgdata2.

Usage

```
r_imgDiffer(imgdata1, imgdata2)
```

Arguments

`imgdata1` The first image

`imgdata2` The second image

Value

return an imagedata object

Note

This is the R implementation of imgDiffer.

See Also

imgDiffer

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_imgDiffer(x, x)

## End(Not run)
```

r_imgMaximum *Images maximum*

Description

This function calculates the maximum of the given images and returns a new image.

Usage

```
r_imgMaximum(imgdata_list)
```

Arguments

imgdata_list	An image list

Value

return an imagedata object

Note

This is the R implementation of imgAverage.

See Also

imgMaximum

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_imgMaximum(list(x, x))

## End(Not run)
```

r_inc_contrast *Increase contrast*

Description

This function increases an image contrast, augmenting pixel values differences between given limits (in a linear fashion).

Usage

```
r_inc_contrast(imgdata, min_limit, max_limit)
```

Arguments

imgdata	The image
min_limit	The minimum limit to apply lineal modification
max_limit	The maximum limit to apply lineal modification

Value

return an imagedata object

Note

This is the R implementation of imgIncreaseContrast.

See Also

imgIncreaseContrast imgDecreaseContrast r_dec_contrast

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_inc_contrast(x, 60, 200)

## End(Not run)
```

r_inc_intensity *Increase intensity*

Description

This function increases an image intensity by a given factor.

Usage

```
r_inc_intensity(imgdata, percentage)
```

Arguments

imgdata	The image
percentage	A non negative value representing the intensity percentage to be increased. 1 stands for 100% (eg. 0.5 = 50%).

Value

return an imagedata object

Note

This is the R implementation of imgIncreaseIntensity.

See Also

imgIncreaseIntensity imgDecreaseIntensity r_dec_intensity

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_inc_intensity(x, 0.3)

## End(Not run)
```

r_look_up_table *Transforms an image by a given look-up table*

Description

This function applies a transformation to an image using a given look-up table.

Usage

```
r_look_up_table(imgdata, table)
```

Arguments

imgdata	The image
table	Look up table which determines the image operation to be applied

Value

return an imagedata object

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                lut <- seq(255, 0, by=-1)
                y <- r_threshold(x, lut)

## End(Not run)
```

r_negative *Negate an image*

Description

This function negates an image.

Usage

```
r_negative(imgdata)
```

Arguments

imgdata	The image

Value

return an imagedata object

Note

This is the R implementation of imgNegative.

See Also

imgNegative r_negative_lut

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_negative(x)

## End(Not run)
```

---

r_negative_lut *Negate an image*

Description

This function negates an image.

Usage

```
r_negative_lut(imgdata)
```

Arguments

imgdata	The image

Value

return an imagedata object

Note

This is the R implementation of imgNegative using look up tables.

See Also

imgNegative r_negative

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_negative_lut(x)

## End(Not run)
```

`r_threshold` *Threshold an image*

Description

This function thresholds an image using a given filter.

Usage

```
r_threshold(imgdata, thr_value)
```

Arguments

`imgdata`	The image
`thr_value`	Filter value for thresholding

Value

return an imagedata object

Note

This is the R implementation of imgThreshold.

See Also

`imgThreshold`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                y <- r_threshold(x, 80)

## End(Not run)
```

`readJpeg` *Read jpeg file*

Description

This function reads a jpeg image file and return an imagedata object.

Usage

```
readJpeg(filename)
```

Arguments

`filename`	filename of JPEG image

Value

return an imagedata object

See Also

`imagedata`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                plot(x)

## End(Not run)
```

`readTiff` *Read tiff file*

Description

This function reads a tiff image file and return an imagedata object.

Usage

```
readTiff(filename)
```

Arguments

`filename`	filename of TIFF image

Value

return an imagedata object

See Also

`imagedata`

Examples

```
        ## Not run:
                x <- readTiff(system.file("data", "violet.tif", package="biOps"))
                plot(x)

## End(Not run)
```

writeJpeg *Write jpeg file*

Description

This function writes an imagedata object into a jpeg image file.

Usage

```
writeJpeg(filename, imgdata)
```

Arguments

`filename`	filename of JPEG image
`imgdata`	imagedata to write

See Also

`readJpeg`

Examples

```
        ## Not run:
                x <- readJpeg(system.file("data", "violet.jpg", package="biOps"))
                writeJpeg("new_image.jpg", x)

## End(Not run)
```

writeTiff *Write tiff file*

Description

This function writes an imagedata object into a tiff image file.

Usage

```
writeTiff(filename, imgdata)
```

Arguments

`filename`	filename of TIFF image
`imgdata`	imagedata to write

See Also

`readTiff`

Examples

```
          ## Not run:
                 x <- readTiff(system.file("data", "violet.tif", package="biOps"))
                 writeTiff("new_image.tif", x)

   ## End(Not run)
```

Index