Package 'rip.opencv'

May 9, 2026

Version 0.3-1

Date 2026-04-11

Type Package

Title Interface to 'OpenCV' Image Processing Routines

Description R interface for calling 'OpenCV' routines that works by translating R objects to 'OpenCV' classes and back. Low-level wrappers for several 'OpenCV' routines are provided as 'Rcpp' modules. In addition, high level interfaces are provided for a limited selection of common operations.

License GPL-2 | GPL-3

SystemRequirements OpenCV 4 or newer

Imports grDevices, graphics, Rcpp

Depends R (>= 4.5.0)

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/deepayan/rip

BugReports https://github.com/deepayan/rip/issues

Author Deepayan Sarkar [aut, cre] (ORCID: <https://orcid.org/0000-0003-4107-1553>), Kaustav Nandy [aut]

Maintainer Deepayan Sarkar <deepayan.sarkar@r-project.org>

NeedsCompilation yes

Repository CRAN

Date/Publication 2026-04-15 14:30:02 UTC

Contents

`as.rip` *Convert to and from rip objects*

Description

Convert compatible data to and from `"rip"` objects.

Usage

```
as.rip(x, ...)
## S3 method for class 'rip'
as.rip(x, ...)
## Default S3 method:
as.rip(x, channel = 1, ...)
## S3 method for class 'matrix'
as.rip(x, channel = 1, ...)
## S3 method for class 'array'
as.rip(x, reverse.rgb = TRUE, ...)
## S3 method for class 'raster'
as.rip(x, ...)
## S3 method for class 'nativeRaster'
as.rip(x, ...)
## S3 method for class 'rip'
as.raster(x, ..., rescale = TRUE, restrict = TRUE)
## S3 method for class 'rip'
as.array(x, reverse.rgb = TRUE, ...)
```

Arguments

`x`	Matrix containing data of rip object. The default method handles matrix-like input, where data for multiple channels should be in successive columns.
`channel`	The number of channels in the rip object to be created.

`reverse.rgb`	Logical flag indicating whether the order of the channels is to be reversed, only in the case where the number of channels is 3. The default is chosen for the common case of 3-channel RGB images, where the RGB ordering is often more common elsewhere but the OpenCV default is BGR.
`rescale`	Logical flag indicating whether data should be rescaled to lie between 0 and 255. If `TRUE`, `min(x)` becomes 0 and `max(x)` becomes 255.
`restrict`	Logical flag indicating whether values outside of the range [0, 255] should be truncated to their nearest endpoints.
`...`	Additional arguments for compatibility with generic; ignored.

Details

The various `as.rip` methods provide utilities to convert image-like objects to `"rip"` objects. This includes `raster` objects as well as `"nativeRaster"` objects which allow efficient representation of 8-bit RGBA images (e.g., see `readPNG`).

Conversely, the `as.raster` and `as.array` methods allow conversion from `"rip"` objects to representations more amenable to processing in R. For example, arrays support more intuitive row-column indexing for multi-channel images, and raster objects can be handled by R graphics.

Value

An object of the appropriate class.

Author(s)

Kaustav Nandy and Deepayan Sarkar

Examples

```
f <- system.file("sample/color.jpg", package = "rip.opencv", mustWork = TRUE)
x <- rip.import(f, type = "original")
x
## index via conversion to array
plot(as.array(x)[1:400, 1:400, ] |> as.rip())
```

`rip` *The rip class and methods*

Description

The `print` method for `"rip"` objects displays the dimension and number of channels.

The `plot` method essentially calls `plot(as.raster(x))` with an option to rescale the image first.

The `image` method provides more flexibility, and finally calls `rasterImage`.

Usage

```
## S3 method for class 'rip'
print(x, ...)
## S3 method for class 'rip'
plot(x, rescale = TRUE, ...)
## S3 method for class 'rip'
plot(x, rescale = TRUE, ...)
## S3 method for class 'rip'
image(x, ..., rescale = TRUE, restrict = !rescale,
    shift = FALSE, xlab = "", ylab = "", asp = 1,
    interpolate = FALSE)
nchannel(x)
```

Arguments

x	An object of class "rip".
rescale	Logical flag indicating whether data should be rescaled to lie between 0 and 255. If TRUE, min(x) becomes 0 and max(x) becomes 255.
restrict	Logical flag indicating whether values outside of the range [0, 255] should be truncated to their nearest endpoints.
shift	Logical flag indicating whether quadrants should be shifted by half of the range. This currently only works for single channel images and is useful primarily to visualize results of DFT, where shift controls whether the axis range is c(-pi, pi) or c(0, 2 * pi).
xlab, ylab	character string or expression specifying x-axis and y-axis labels.
asp	Aspect ratio for the plot.
interpolate	Logical flag indicating whether the image should be interpolated.
...	Additional arguments. The plot method passes extra arguments to plot.raster.

Details

The basic data type in the OpenCV library is the cv::Mat class, representing an n-dimensional dense numerical single-channel or multi-channel array. This class can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors.

The "rip" class mirrors this class, which is essentially a matrix with an additional "cvdim" attribute. For example, a 600 x 400 color image with three (RGB) channels will be represented as a 600 x 1200 matrix with attribute cvdim = c(600, 400, 3). In the underlying matrix, the first three columns will store the RBG channels for the first 'column' of the image, and so on. Similarly, for a complex matrix of size 600 x 400, the number of channels will be 2 (real and imaginary) and the underlying matrix will have dimension 600 x 800.

The conventional approach in R to represent such objects is a three-dimensional array. Coercion to and from arrays can be performed using as.array and as.rip methods.

A "rip" object by itself does not contain any information on the interpretation of the data contained therein. Its primary purpose is to serve as an R analog of cv::Mat objects; R functions

that wrap OpenCV operations should coerce between "rip" and cv::Mat objects (in C++ code) as appropriate.

Value

nchannel returns the number of channels in x. The other functions have no useful return value, and are called for their side effects.

Author(s)

Deepayan Sarkar

Examples

```
v <- as.rip(t(volcano), channel = 1)
v
class(v)
image(v)
plot(v)
```

---

rip.blur *Blur a "rip" object*

Description

Wrapper for blurring functions in OpenCV.

Usage

```
rip.blur(x,
         method = c("mean", "median", "gaussian"),
         ksize = c(1L, 1L),
         sigma.x = 1,
         sigma.y = 1,
         anchor = c(-1L, -1L),
       borderType = c("reflect_101", "constant", "replicate", "reflect", "wrap"))
```

Arguments

x	rip object (or an object that can be coerced to one).
method	Blurring method to use.
ksize	Blur kernel size.
sigma.x, sigma.y	Standard deviation of Gaussian blur kernel. This combined with ksize determines the blur kernel.
anchor	Determines the position of the kernel with reference to location in the output image. The default is to center the kernel.

borderType — Character string determining how the value of 'missing' pixels outside the image boundary are computed. The names are suggestive; see OpenCV documentation for details.

Details

Blurs an image using one of three algorithms in OpenCV, `blur` (local mean), `medianBlur` (local median), or `gaussianBlur` (local filtering with a Gaussian kernel). There are limitations on kernel size for certain methods (see OpenCV documentation for details).

Value

A blurred `"rip"` object.

Author(s)

Deepayan Sarkar

---

rip.conv *2-D convolution and filtering*

Description

Apply a convolution filter to an image.

Usage

```
rip.conv(x, k, type = c("full", "valid", "same"), flip = TRUE)
rip.filter(x, k, flip = FALSE, anchor = c(-1L, -1L), delta = 0,
       borderType = c("replicate", "constant", "reflect", "wrap", "reflect_101"))
```

Arguments

x — A `"rip"` object (or an object that can be coerced to one), giving the image to be convolved.

k — A `"rip"` object (or an object that can be coerced to one), giving the colvolution kernel.

type — Character string that determines the part of the 2-dimensional convolution to be returned. If `"full"`, the full convolution is returned. For `"same"`, a 'rip' object of same size as 'x' is returned. For `"valid"`, only the part of the result that did not use and pixels outside the boundary of `x` are returned.

flip — Logical flag indicating whether the kernel is to be flipped. This is the difference between convolution and filtering, that is, whether the index along the rows and columns of the kernel combine with the index along the rows and columns of `x` in increasing or decreasing order.

anchor — Determines the position of the kernel with reference to location in the output image. The default is to center the kernel.

`delta`	Value added to the return value after filtering.
`borderType`	Character string determining how the value of 'missing' pixels outside the image boundary are computed. The names are suggestive; see OpenCV documentation for details.

Details

These functions are wrappers for the OpenCV function `filter2D`. `rip.filter` is a minimal wrapper. `rip.conv` is a more user-friendly interface that allows finer control over the size of the resulting image.

Value

Convolution of 'x' and 'k'

Author(s)

Kaustav Nandy

`rip.desaturate` *Convert color image to grayscale*

Description

Converts a color image to a grayscale image using one of three algorithms.

Usage

```
rip.desaturate(x, method = c("simple", "decolor", "convert"))
```

Arguments

`x`	A `"rip"` object (or an object that can be coerced to one), giving the color image to be converted.
`method`	Character string giving the method used.

Details

Three methods are supported. The `"decolor"` method uses the OpenCV function `decolor`, `"convert"` uses the OpenCV function `cvtColor`, and `"simple"` uses a simple linear combination in R.

Author(s)

Deepayan Sarkar

`rip.dft` *DFT and DCT using OpenCV*

Description

Apply the Dicrete Fourier Transform (DFT) or the Discrete Cosine Transform (DCT) to a matrix.

Usage

```
rip.dft(x, pad = NULL,
        inverse = FALSE,
        scale = inverse,
        rowwise = FALSE,
        complex = TRUE,
        nonzerorows = 0)
rip.dct(x, pad = NULL, inverse = FALSE, rowwise = FALSE)
rip.ndft(x, pad = NULL, inverse = FALSE)
```

Arguments

`x`	`"rip"` object to transform
`pad`	Passed to `rip.pad` to pad the image with zeros. This may be useful to improve the performance of DCT, which is more efficient when the image sizes are highly composite numbers.
`inverse`	Logical flag. If `TRUE`, inverse DFT is computed, assuming real output.
`scale`	Logical flag. If `TRUE`, divides transformed values by number of elements.
`rowwise`	Logical flag. If `TRUE`, computes DFT or DCT rowwise.
`complex`	Logical flag. If `TRUE`, stores output as complex numbers, otherwise imaginary part is stored in subdiagonal exploiting symmetry. Ignored when `inverse=TRUE`, in which case it is inferred from the input.
`nonzerorows`	Integer. Assumes that only first nonzerorows have non-zero data. When 'inverse=TRUE', only first nonzerorows rows in the output will have non-zero data.

Details

These functions are user-friendly interfaces to the OpenCV `cv::dft` and `cv::dct` functions via the `rip.cv$transforms` module.

Value

A `"rip"` object containing the transformed data. In the case of DFT, complex values are stored using the R-native complex type, and converted to the native 2-channel OpenCV representation as needed.

`rip.ndft` is similar to `rip.dft`, except that it 'normalizes' the output to adjust for image size.

Examples

```
n <- 100
x <- matrix(rnorm(n*n), n, n)
(F <- rip.dft(x))
(C <- rip.dct(x))
mean(F)
mean(Mod(F)^2)
mean(C^2)
mean(Mod(rip.ndft(x))^2)
```

`rip.import` *Import from or export to an image file*

Description

Import image data from an external file or write image data to an external file.

Usage

```
rip.import(file, type = c("grayscale", "color", "original"))
rip.export(x, file)
```

Arguments

`file`	Path of the image file.
`x`	An object of class `"rip"` containing data that can be interpreted as an image by OpenCV (usually with 1, 3, or 4 channels with values between 0 and 255).
`type`	Character string that determines whether the imported data will be stored as grayscale (default) or color (including possibly an alpha channel). If `"original"`, the choice is taken from the image.

Details

These functions are user-friendly interfaces to the OpenCV functions `imread` and `imwrite` via the `rip.cv$IO` module. The file format is automatically determined by OpenCV.

Value

If successful, `rip.import` returns the image data in `file` as an object of `class` `"rip"`. Such objects store data as a (typically numeric) matrix, regardless of the number of channels, in the same layout that is used natively by the `cv::Mat` type in OpenCV. The `"cvdim"` attribute gives the number or rows, columns, and channels as an integer vector of length 3 with names `"nrow"`, `"ncol"`, and `"nchannel"`.

Essentially, `"rip"` objects store data column-wise, but all channels for a given column in the image are stored in contiguous columns. For the purposes of importing and exporting color images, the first three channels are interpreted as Blue, Green, and Red (BGR) respectively (which is not the more typical Red, Green, and Blue, RGB ordering).

Author(s)

Kaustav Nandy

See Also

The `rip` class is a convenient format for manipulating data using OpenCV functions as it closely reflects the native OpenCV representation. It is also convenient for manipulation as a matrix using R functions as long it represents a single channel image. For multi-channel images, it may be more convenient to convert it either into a three-dimensional array or a `raster` object using convenient `conversion functions`.

`rip.opencv` *R Interface to OpenCV*

Description

The **rip.opencv** add-on package provides an R interface for tools in the popular OpenCV library. It is not an exhaustive wrapper; instead it exposes selected OpenCV functions, using standard R objects to represent corresponding OpenCV objects, and explicitly converting between the two forms as necessary. The idea behind this design is to facilitate a primarily R-based workflow, with OpenCV providing access an additional suite of operations.

Details

The package defines a `rip` class in R that parallels the `cv::Mat` class in OpenCV, which is essentially an analogue of three-dimensional arrays in R. Conversion functions written in C++ map `"rip"` objects to and from `cv::Mat` objects as necessary.

Wrappers to OpenCV functions are exposed as Rcpp modules, collected together in the environment object `rip.cv`. See the vignette and examples. More functions may be exposed in future.

In addition to these modules, user-friendly high-level functions are provided for certain common operations. See the other help pages in this package for details.

Author(s)

Deepayan Sarkar <Deepayan.Sarkar@R-project.org>

Examples

```
ls(rip.cv)
(imreadModes <- rip.cv$enums$ImreadModes)

f <- system.file("sample/color.jpg", package = "rip.opencv", mustWork = TRUE)
(x <- rip.cv$IO$imread(f, imreadModes["IMREAD_COLOR"]))
(gx <- rip.cv$photo$decolor(x))
opar <- par(mfrow = c(1, 2))
image(x, rescale = FALSE, main = "original")
image(gx, rescale = FALSE, main = "decolored")
par(opar)
```

`rip.pad` *Add a border to a* `"rip"` *object*

Description

Wrapper for OpenCV function `copyMakeBorder`.

Usage

```
rip.pad(x, pad,
        offset = round((pad - dim(x))/2),
        value = 0,
        top = offset[1],
        left = offset[2],
        bottom = pad[1] - nrow(x) - top,
        right = pad[2] - ncol(x) - left,
       borderType = c("constant", "replicate", "reflect", "wrap", "reflect_101")
)
```

Arguments

`x`	rip object (or an object that can be coerced to one).
`pad`	Desired size of the output image. Must not be smaller than the size of `x`.
`offset`	Starting position of the top-left corner of `x` in the new image. By default, the padding is added symmetrically on all sides.
`value`	A constant to be used for `borderType = "constant"`.
`top`	border size at top direction
`left`	border size at left direction
`bottom`	border size at bottom direction
`right`	border size at right direction
`borderType`	border type

Details

Adds a border (padding) around an image. Various kinds of border types are supported; the names are suggestive, but see OpenCV documentation for precise details. The `pad` and `offset` arguments are provided for convenience, and overridden by `top`, `left`, etc.

Value

A padded `"rip"` object.

Author(s)

Kaustav Nandy

rip.resize *Resize a "rip" object via interpolation*

Description

Wrapper for the OpenCV function `resize`

Usage

```
rip.resize(x, d = c(0, 0),
           fx = 1,
           fy = fx,
           method = c("nearest", "linear", "area", "cubic", "lanczos4"))
```

Arguments

`x`	rip object (or an object that can be coerced to one).
`d`	Size of the resized image. If specified, `fx` and `fy` are ignored.
`fx`, `fy`	Scaling factors for the resized image if `d` is not specified.
`method`	Type of interpolation.

Value

A resized "`rip`" object.

Author(s)

Deepayan Sarkar

utilities *Helper utilities*

Description

`rip.flip` flips a convolution kernel by reversing row and column order.

`rip.shift` shift-wraps an image. This is typically used for the result of `rip.dft`, where it has the effect of shifting the frequency range.

Usage

```
rip.flip(k, cv)
rip.shift(x, inverse = FALSE,
          orow = 1 + ceiling(nrow(x)/2),
          ocol = 1 + ceiling(ncol(x)/2))
```

Arguments

k, x	A "rip" object (or an object that can be coerced to one). Typically a convolution kernel for rip.flip.
cv	Logical flag indicating whether to use the OpenCV funtion flip instead of an R implementation. The R implementation only supports single channel image matrices.
inverse	Whether to (conceptually) invert a previous shift operation, accounting for rounding differences.
orow, ocol	Integer giving the index of the 'origin' row / column. The image is shifted (and wrapped) to that these become the first row and column of the resulting image.

Index