

Package: aznyan (via r-universe)

September 28, 2026

Type Package

Title Image Filters with 'OpenCV'

Version 26.08.29

Maintainer Akiru Kato <paithiov909@gmail.com>

Description Offers image filters wrapping 'OpenCV'
<<https://opencv.org/>>, ported from
<https://github.com/5PB-3-4/AviUtl_OpenCV_Scripts>.

License MIT + file LICENSE

URL <https://github.com/paithiov909/aznyan>

BugReports <https://github.com/paithiov909/aznyan/issues>

Depends R (>= 4.2)

Imports cli, colorfast, grDevices, grid, rlang

Suggests dplyr, testthat (>= 3.0.0), vdiffr

LinkingTo cpp11

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

SystemRequirements OpenCV 4 or newer: libopencv-dev (Debian, Ubuntu)
or opencv-devel (Fedora)

Config/pak/sysreqs libopencv-dev

Repository <https://paithiov909.r-universe.dev>

Date/Publication 2026-08-29 04:26:25 UTC

RemoteUrl <https://github.com/paithiov909/aznyan>

RemoteRef HEAD

RemoteSha 94389b9cd3cc074fea26344a721c61d5ced524fc

Contents

bilateral_filter	3
blend-mode	4
blue_noise_64x64	5
blur	5
blurhash	7
canny_filter	7
color-manip	9
color_filter	10
color_map	11
convolve	13
detail_enhance	14
diffusion_filter	14
fill_with	15
hist_eq	16
image-io	16
kuwahara_filter	18
laplacian_filter	19
lineweave	20
luma_gate	21
make-kernels	22
mean_shift	23
median_cut	24
morphology	25
oilpaint	27
pack-unpack	27
pencil_sketch	28
pixel_positions	28
preserve_edge	30
resize	30
screen_tone	31
sobel_filter	32
stylize	34
swap_channels	34
thres	35
tile_matrix	36
warp_perspective	37

Index

38

bilateral_filter	<i>Bilateral filter</i>
------------------	-------------------------

Description

Applies a bilateral filter to a nativeRaster image.

This edge-preserving smoothing technique reduces noise while retaining sharp boundaries by considering both spatial distance and color similarity.

Usage

```
bilateral_filter(
  nr,
  d = 5,
  sigmacolor = 1,
  sigmaspace = 1,
  border = c(3, 4, 0, 1, 2),
  alphasync = FALSE
)
```

Arguments

nr	A nativeRaster object.
d	An integer scalar specifying the diameter of the pixel neighborhood used for filtering. If set to a non-positive value, OpenCV computes the diameter from the sigmas.
sigmacolor	A numeric scalar giving the filter sigma in color space. Larger values allow blending of pixels with greater color differences.
sigmaspace	A numeric scalar giving the filter sigma in coordinate space. Larger values increase the spatial extent of smoothing.
border	An integer scalar specifying the border-handling mode. One of 0, 1, 2, 3, 4, corresponding to OpenCV's bilateral filter border modes.
alphasync	A logical scalar. If TRUE, the same bilateral filter is applied to the alpha channel; if FALSE, the alpha channel is preserved.

Details

border corresponds to the OpenCV extrapolation types:

1. cv::BORDER_CONSTANT
2. cv::BORDER_REPLICATE
3. cv::BORDER_REFLECT
4. cv::BORDER_WRAP
5. cv::BORDER_REFLECT_101

Value

A nativeRaster object.

blend-mode

Blend modes

Description

Blends two nativeRaster objects with the specified blend mode.

Usage

blend_alpha(src, dst)

blend_darken(src, dst)

blend_multiply(src, dst)

blend_colorburn(src, dst)

blend_lighten(src, dst)

blend_screen(src, dst)

blend_add(src, dst)

blend_colordodge(src, dst)

blend_hardlight(src, dst)

blend_softlight(src, dst)

blend_overlay(src, dst)

blend_hardmix(src, dst)

blend_linearlight(src, dst)

blend_vividlight(src, dst)

blend_pinlight(src, dst)

blend_average(src, dst)

blend_exclusion(src, dst)

blend_difference(src, dst)

```
blend_divide(src, dst)
blend_subtract(src, dst)
blend_luminosity(src, dst)
blend_ghosting(src, dst)
```

Arguments

src, dst A nativeRaster object.

Value

A nativeRaster object.

blue_noise_64x64	<i>Blue noise pattern</i>
------------------	---------------------------

Description

Blue noise pattern of size 64x64 generated with `ambient::noise_blue(c(64, 64))`.

Range: `[0, 1]`

Usage

```
blue_noise_64x64
```

Format

An object of class `matrix` (inherits from `array`) with 64 rows and 64 columns.

blur	<i>Blur filters</i>
------	---------------------

Description

Blur filters

Usage

```

median_blur(nr, ksize = 1)

box_blur(
    nr,
    box_w = 1,
    box_h = box_w,
    normalize = TRUE,
    border = c(3, 4, 0, 1, 2)
)

gaussian_blur(
    nr,
    box_w = 1,
    box_h = box_w,
    sigma_x = 0,
    sigma_y = sigma_x,
    border = c(3, 4, 0, 1, 2)
)

```

Arguments

<code>nr</code>	A nativeRaster object.
<code>ksize</code>	An integer scalar specifying the kernel size. The actual kernel size becomes $2 * ksize + 1$.
<code>box_w</code>	An integer scalar controlling the half-size of the kernel in the horizontal direction. The actual kernel width becomes $2 * box_w - 1$.
<code>box_h</code>	An integer scalar controlling the half-size of the kernel in the vertical direction. Defaults to <code>box_w</code> . The actual kernel height becomes $2 * box_h - 1$.
<code>normalize</code>	A logical scalar specifying whether the kernel is normalized by its area or not. Defaults to <code>TRUE</code> .
<code>border</code>	An integer scalar specifying the border-handling mode. One of 0, 1, 2, 3, 4, corresponding to OpenCV's border modes.
<code>sigma_x, sigma_y</code>	A numeric scalar giving the standard deviation of the Gaussian kernel along the x-axis. A value of 0 lets OpenCV compute it automatically from the kernel size.

Details

`border` corresponds to the OpenCV extrapolation types:

1. `cv::BORDER_CONSTANT`
2. `cv::BORDER_REPLICATE`
3. `cv::BORDER_REFLECT`
4. `cv::BORDER_REFLECT_101`
5. `cv::BORDER_ISOLATED`

Value

A nativeRaster object.

blurhash	<i>BlurHash-style DCT reconstruction</i>
----------	--

Description

Reconstructs a low-frequency approximation of an image using a BlurHash-like discrete cosine transform (DCT) basis. The image is first converted to linear RGB, projected onto a grid of cosine components, and then reconstructed back into an sRGB image using only the specified number of horizontal and vertical components. This produces a smooth, compressed representation capturing the coarse structure and color of the input.

Usage

```
blurhash(nr, x_comps = 6, y_comps = 6)
```

Arguments

nr	A nativeRaster object.
x_comps	An integer scalar specifying the number of horizontal DCT components to use. Must be greater than 0. Larger values capture more detail.
y_comps	An integer scalar specifying the number of vertical DCT components to use. Must be greater than 0. Larger values capture more detail.

Value

A nativeRaster object.

canny_filter	<i>Canny edge detection (grayscale or RGB)</i>
--------------	--

Description

Applies the Canny edge detector to a nativeRaster image.

The filter can operate either on a grayscale conversion or by aggregating edges detected independently from each RGB channel.

The result is returned as a 3-channel binary edge map, with optional alpha masking.

Usage

```
canny_filter(  
    nr,  
    asize = 2,  
    balp = TRUE,  
    use_rgb = TRUE,  
    thres1 = 100,  
    thres2 = 200,  
    grad = TRUE  
)
```

Arguments

nr	A nativeRaster object.
asize	An integer scalar giving the aperture size parameter; the actual Sobel kernel size used internally by Canny is $2 * asize - 1$.
balp	A logical scalar. If TRUE, derive the alpha channel from the detected edges by thresholding; if FALSE, preserve the input alpha.
use_rgb	A logical scalar. If TRUE, compute edges per RGB channel and aggregate them; if FALSE, operate on grayscale only.
thres1	A numeric scalar giving the lower hysteresis threshold.
thres2	A numeric scalar giving the upper hysteresis threshold.
grad	A logical scalar indicating whether to use a more accurate gradient calculation (L2gradient = TRUE) or the default (FALSE).

Details**Grayscale mode** (use_rgb = FALSE):

The input is converted to grayscale and passed to the Canny operator. The aperture size of the Sobel derivative is $2 * asize - 1$. When balp = TRUE, the alpha channel is thresholded from the detected edges.

RGB mode (use_rgb = TRUE):

Canny is applied independently to each RGB channel, and the resulting edge maps are summed to form a combined edge representation. The alpha channel is either thresholded (balp = TRUE) or preserved from the input.

Options:

border corresponds to:

1. cv::BORDER_CONSTANT
2. cv::BORDER_REPLICATE
3. cv::BORDER_REFLECT
4. cv::BORDER_REFLECT_101
5. cv::BORDER_ISOLATED

Value

A nativeRaster object.

color-manip

Color manipulation

Description

Color manipulation

Usage

`apply_lut1d(nr, lut)`

`apply_lut3d(nr, cubefile)`

`brighten(nr, intensity)`

`contrast(nr, intensity)`

`duotone(nr, color_a = "yellow", color_b = "navy", gamma = 2.2)`

`grayscale(nr)`

`hue_rotate(nr, rad)`

`invert(nr)`

`linocut(nr, ink = "navy", paper = "snow", threshold = 0.4)`

`posterize(nr, shades = 4)`

`reset_alpha(nr, alpha = 1)`

`saturate(nr, intensity)`

`sepia(nr, intensity, depth = 20)`

`set_matte(nr, color = "green")`

`solarize(nr, threshold = 0.5)`

`unpremul(nr, max = 255L)`

Arguments

nr	A nativeRaster object.
lut	A 256x3 double matrix in range [0, 255].
cubefile	A character string specifying the path to the .cube file.
intensity	A numeric scalar.
gamma	A numeric scalar. The gamma exponent.
rad	A numeric scalar. The rotation angle in radians.
alpha, threshold	A numeric scalar in range [0, 1].
depth, shades	A positive integer scalar.
color, color_a, color_b, ink, paper	A character string; color name or hex code.
max	An integer scalar. The maximum value of the color code.

Value

A nativeRaster object.

color_filter	<i>Apply predefined color filter</i>
--------------	--------------------------------------

Description

Applies one of the predefined color filters that are ported from **Rustagram**.

Usage

```
color_filter(
  nr,
  filter = c("1977", "aden", "brannan", "brooklyn", "clarendon", "earlybird", "gingham",
    "hudson", "inkwell", "kelvin", "lark", "lofi", "maven", "mayfair", "moon",
    "nashville", "reyes", "rise", "slumber", "stinson", "toaster", "valencia", "walden")
)
```

Arguments

nr	A nativeRaster object.
filter	The name of the filter.

Details**Options:**

The following filters are available:

- 1977
- aden
- brannan
- brooklyn
- clarendon
- earlybird
- gingham
- hudson
- inkwell
- kelvin
- lark
- lofi
- maven
- mayfair
- moon
- nashville
- reyes
- rise
- slumber
- stinson
- toaster
- valencia
- walden

Value

A nativeRaster object.

color_map

Apply a color map

Description

Maps grayscale or hue values of a nativeRaster image to a predefined color lookup table (OpenCV colormaps). The input can be derived either from the grayscale intensity or from the hue channel in HSV space. The mapped values replace the image's color channels while preserving the alpha channel.

Usage

```
color_map(nr, map = 0, use_hsv = FALSE, invert = FALSE)
```

Arguments

nr	A nativeRaster object.
map	An integer scalar specifying the colormap ID. Must be one of 0–21, corresponding to OpenCV’s built-in color maps.
use_hsv	A logical scalar. If TRUE, the hue channel (HSV_FULL) is used as the input to the colormap; if FALSE, a grayscale conversion is used instead.
invert	A logical scalar. If TRUE, the input channel is inverted (255 - value) before applying the colormap.

Details**Options:**

map corresponds to:

1. cv::COLORMAP_AUTUMN
2. cv::COLORMAP_BONE
3. cv::COLORMAP_JET
4. cv::COLORMAP_WINTER
5. cv::COLORMAP_RAINBOW
6. cv::COLORMAP_OCEAN
7. cv::COLORMAP_SUMMER
8. cv::COLORMAP_SPRING
9. cv::COLORMAP_COOL
10. cv::COLORMAP_HSV
11. cv::COLORMAP_PINK
12. cv::COLORMAP_HOT
13. cv::COLORMAP_PARULA
14. cv::COLORMAP_MAGMA
15. cv::COLORMAP_INFERNO
16. cv::COLORMAP_PLASMA
17. cv::COLORMAP_VIRIDIS
18. cv::COLORMAP_CIVIDIS
19. cv::COLORMAP_TWILIGHT
20. cv::COLORMAP_TWILIGHT_SHIFTED
21. cv::COLORMAP_TURBO
22. cv::COLORMAP_DEEPPGREEN

Value

A nativeRaster object.

convolve	<i>Convolution with a custom kernel</i>
----------	---

Description

Applies a custom convolution kernel to a `nativeRaster` image.

This function wraps OpenCV's `filter2D`, allowing arbitrary linear filters such as sharpening, blurring, or edge detection. Optionally, the same convolution can be applied to the alpha channel.

Usage

```
convolve(
  nr,
  kernel = matrix(c(0, -1, 0, -1, 5, -1, 0, -1, 0), 3, 3),
  border = c(3, 4, 0, 1, 2),
  alphasync = FALSE
)
```

Arguments

<code>nr</code>	A <code>nativeRaster</code> object.
<code>kernel</code>	A numeric matrix giving the convolution kernel. Larger or weighted matrices can be used to define custom filters. The matrix is passed directly to OpenCV, which normalizes nothing automatically.
<code>border</code>	An integer scalar specifying the border-handling mode. One of 0, 1, 2, 3, 4, corresponding to OpenCV's border modes. See package documentation for details.
<code>alphasync</code>	A logical scalar. If <code>TRUE</code> , the convolution is also applied to the alpha channel; if <code>FALSE</code> , the alpha channel is preserved as-is.

Details

`border` corresponds to the OpenCV extrapolation types:

1. `cv::BORDER_CONSTANT`
2. `cv::BORDER_REPLICATE`
3. `cv::BORDER_REFLECT`
4. `cv::BORDER_REFLECT_101`
5. `cv::BORDER_ISOLATED`

Value

A `nativeRaster` object.

detail_enhance	<i>Detail enhancement filter</i>
----------------	----------------------------------

Description

Applies OpenCV's detailEnhance to a nativeRaster image. This filter enhances fine structures such as edges and textures while preserving the overall appearance of the image.

Usage

```
detail_enhance(nr, sgms = 10, sgmr = 0.15)
```

Arguments

nr	A nativeRaster object.
sgms	A numeric scalar giving the spatial sigma used for edge-preserving filtering. Larger values increase the spatial smoothing radius. Valid range: 0–200.
sgmr	A numeric scalar giving the range sigma controlling how strongly edges are preserved based on color differences. Valid range: 0–1.

Value

A nativeRaster object.

diffusion_filter	<i>Diffusion-based smoothing and enhancement</i>
------------------	--

Description

Applies an iterative diffusion-style filter to a nativeRaster image. This custom process repeatedly blurs a gamma-transformed version of the image and accumulates the diffused values with a decaying gain factor. The result is then inverse-gamma-corrected, producing a soft, glowy, detail-enhancing effect reminiscent of multi-scale diffusion or photographic bloom.

Usage

```
diffusion_filter(nr, factor = 5, offset = 0.1, gamma = 1.3, sigma = 2)
```

Arguments

nr	A nativeRaster object.
factor	A numeric scalar controlling the decay factor used at each iteration. Larger values reduce the contribution of later diffusion steps.
offset	A numeric scalar added to the iteration index when computing the decaying gain. Helps adjust the early-iteration weighting.
gamma	A numeric scalar specifying the gamma exponent applied before diffusion (and inverted afterward). Values greater than 1 emphasize bright regions.
sigma	An integer scalar giving the initial Gaussian blur radius (converted to a standard deviation internally). The value is squared on each iteration, producing progressively wider diffusion.

Value

A nativeRaster object.

fill_with	<i>Create a native raster filled with a color</i>
-----------	---

Description

Create a native raster filled with a color

Usage

```
fill_with(color, width, height)
```

Arguments

color	Color name or hex code.
width, height	A positive integer scalar.

Value

A nativeRaster object.

hist_eq	<i>Histogram equalization</i>
---------	-------------------------------

Description

Performs histogram equalization on a nativeRaster image. This function supports both standard histogram equalization and adaptive histogram equalization (CLAHE). Optionally, the equalized luminance can be merged back into the original color channels to produce a color-preserving enhancement.

Usage

```
hist_eq(nr, limit = 40, grid = c(8, 8), adp = FALSE, color = TRUE)
```

Arguments

nr	A nativeRaster object.
limit	A numeric scalar giving the clip limit used for adaptive histogram equalization (CLAHE). Ignored when adp = FALSE. Typical range: values around 40 are common.
grid	An integer vector of length 2 specifying the grid size (width, height) for dividing the image when using CLAHE.
adp	A logical scalar. If TRUE, uses adaptive histogram equalization (CLAHE). If FALSE, applies standard histogram equalization.
color	A logical scalar. If TRUE, apply the equalized luminance back to the original BGR channels; if FALSE, return a grayscale-based result.

Value

A nativeRaster object.

image-io	<i>Read and write images</i>
----------	------------------------------

Description

These functions provide convenient helpers for reading still images, writing still images, and creating animated image files.

Since the underlying image writers always treat the input as 4 channels images, writing with some formats would fail even if the linked 'OpenCV' library supports it.

Usage

```

read_still(filename)

read_data(x)

write_still(nr, filename = "azny-still.png")

write_data(nr, ext = ".jpeg", quality = 80)

write_animation(
  frames,
  filename = "azny-anime.webp",
  delay = 1/12,
  quality = 80,
  loop_count = 0
)

```

Arguments

filename	A file name. For <code>read_still()</code> , the path to the input image file. For <code>write_still()</code> and <code>write_animation()</code> , the output file name.
x	A raw vector containing image data.
nr	A <code>nativeRaster</code> object.
ext	File extension (e.g., ".jpg", ".png") to specify the output format.
quality	Image quality. For <code>write_animation()</code> , when the output format is not WebP, this parameter is ignored.
frames	A character vector of file names representing animation frames.
delay	Frame delay in seconds. Internally converted to milliseconds, with a minimum of 10ms.
loop_count	Number of animation loops. A value of 0 means infinite looping.

Details

- `read_still()`: Reads a still image file and converts it into a `nativeRaster`.
- `read_data()`: Reads image data from a raw vector and converts it into a `nativeRaster`.
- `write_still()`: Writes a still image file from a `nativeRaster`.
- `write_data()`: Writes and returns the image data as a raw vector.
- `write_animation()`: Writes an animated image file from a sequence of image files.

Value

- `read_still()` and `read_data()` return a `nativeRaster` object.
- `write_still()` and `write_animation()` invisibly return `filename`.
- `write_data()` returns a raw vector containing the image data.

kuwahara_filter	<i>Softmax-guided edge-aware smoothing (Kuwahara-like)</i>
-----------------	--

Description

This filter produces a painting-like, edge-aware smoothing effect inspired by the Kuwahara filter, but it does *not* implement the classical four-region Kuwahara algorithm.

Instead, it estimates local variance using kernel1 and applies a softmax-like weighting ($\exp(-\text{beta} * \text{variance})$) to emphasize smoother directions, then computes a weighted average using kernel2.

The method originates from the idea described in <https://qiita.com/Carletelet/items/7773cd56c7ce016476d9>.

Usage

```
kuwahara_filter(  
    nr,  
    kernel1 = kernel_cone(7),  
    kernel2 = kernel_cone(5),  
    beta = 30,  
    border = c(3, 4, 0, 1, 2)  
)
```

Arguments

nr	A nativeRaster object.
kernel1	A numeric matrix used to compute local mean and variance.
kernel2	A numeric matrix used to compute the final weighted average.
beta	A double scalar controlling the sharpness of softmax weighting. Higher values make the filter more strongly directional.
border	An integer scalar. The type of pixel extrapolation method.

Details

border corresponds to the OpenCV extrapolation types:

1. cv::BORDER_CONSTANT
2. cv::BORDER_REPLICATE
3. cv::BORDER_REFLECT
4. cv::BORDER_REFLECT_101
5. cv::BORDER_ISOLATED

Value

A nativeRaster object.

laplacian_filter	<i>Laplacian edge detection (grayscale or RGB)</i>
------------------	--

Description

Applies a Laplacian filter to detect edges in a nativeRaster image.

The Laplacian operator computes second-order derivatives, highlighting regions of rapid intensity change.

The result is returned as a 3-channel grayscale image, with optional binary alpha masking.

Usage

```
laplacian_filter(
    nr,
    ksize = 3,
    balp = TRUE,
    use_rgb = TRUE,
    border = c(3, 4, 0, 1, 2),
    scale = 1,
    delta = 0
)
```

Arguments

nr	A nativeRaster object.
ksize	An integer scalar specifying the kernel radius. The effective Laplacian kernel size becomes $2 * ksize - 1$. Must be non-negative.
balp	A logical scalar. If TRUE, generate alpha from edge intensity by thresholding; if FALSE, preserve the original alpha.
use_rgb	A logical scalar. If TRUE, compute Laplacian derivatives for each RGB channel separately; if FALSE, operate on grayscale only.
border	An integer scalar selecting the border-handling mode. One of 0–4, corresponding to OpenCV border modes.
scale	A numeric scalar that scales the computed Laplacian derivative.
delta	A numeric scalar added to the filtered result before scaling.

Details

Grayscale mode (use_rgb = FALSE):

The input image is converted to grayscale, normalized to $[0, 1]$, and the Laplacian operator is applied using a kernel of size $2 * ksize - 1$. The output is rescaled to $[0, 255]$. When balp = TRUE, the alpha channel is thresholded based on edge intensity.

RGB mode (use_rgb = TRUE):

The Laplacian operator is applied independently to each RGB channel, and the channelwise derivatives are summed to produce a combined edge map. The alpha channel is determined either by thresholding (balp = TRUE) or by preserving the original.

Options:

border corresponds to:

1. cv::BORDER_CONSTANT
2. cv::BORDER_REPLICATE
3. cv::BORDER_REFLECT
4. cv::BORDER_REFLECT_101
5. cv::BORDER_ISOLATED

Value

A nativeRaster object.

lineweave

Modulation filter

Description

Weaves short line segments across an image by scanning pixels and triggering strokes based on accumulated luminance.

This effect walks through the image in the selected direction. At each pixel, luminance is accumulated and compared with phase. When the threshold is reached, a stroke event may start depending on init and interval, and step pixels are drawn from fg; otherwise pixels are taken from bg.

Usage

```
lineweave(
  nr,
  fg = nr,
  bg = blurhash(nr, 3, 3),
  omega = 10,
  phase = 5,
  init = 1,
  interval = 1,
  step = 2,
  invert = FALSE,
  direction = c(3, 0, 1, 2)
)
```

Arguments

nr	A nativeRaster object.
fg	A nativeRaster object used as the foreground source for stroke pixels. It must have the same dimensions as nr.
bg	A nativeRaster object used as the background source for non-stroke pixels. It must have the same dimensions as nr.
omega	A positive numeric scalar scaling how fast luminance accumulates per pixel.
phase	A positive numeric scalar giving the luminance threshold for triggering events.
init	A positive integer scalar giving the number of stroke events to emit in one cycle.
interval	A non-negative integer scalar giving the number of trigger events to skip between cycles.
step	A non-negative integer scalar giving the stroke length in pixels for each event.
invert	A logical scalar indicating whether bright pixels (instead of dark pixels) contribute more strongly to triggering.
direction	An integer giving the scan direction • 3 = left-to-right • 0 = right-to-left • 1 = top-to-bottom • 2 = bottom-to-top

Value

A nativeRaster object.

luma_gate

Luminance-gated additive blending

Description

Adds fg to bg through mask, with the contribution modulated by the luminance of bg. Brighter regions of bg receive a stronger contribution from fg, while darker regions receive less.

The blend strength is controlled by gamma: values greater than 1 concentrate the effect in brighter regions, while values between 0 and 1 spread the effect into darker regions.

Usage

```
luma_gate(fg, bg = hist_eq(fg), mask = blurhash(fg, 3, 3), gamma = 1)
```

Arguments

fg	A nativeRaster image to add to bg.
bg	A nativeRaster image used as the destination and as the luminance gate. By default, <code>hist_eq(fg)</code> is used.
mask	A nativeRaster image controlling the per-channel amount of fg added to bg. By default, <code>blurhash(fg, 3, 3)</code> is used.
gamma	A finite numeric value controlling the luminance response. The default is 1.

Details

For each color channel, the output is approximately computed as

$$\text{out} = \text{bg} + \text{fg} \times \text{mask} \times \text{luma}(\text{bg})^\gamma$$

with the resulting values clamped to the valid image range.

The alpha channel of mask is used as the alpha channel of the returned image.

Value

A nativeRaster object.

make-kernels

Generate common convolution kernels

Description

A collection of helper functions that generate numeric matrices usable as convolution kernels. These kernels are intended for use with `convolve()` and other filtering operations in this package.

Usage

```
kernel_bayer(n, normalize = TRUE)

kernel_cone(size)

kernel_disc(size)

kernel_emboss(theta, strength = 1)

kernel_motion(size, theta)

kernel_ring(size, thickness = 1)

kernel_stripe(n, mod, step = 1)
```

Arguments

n	An integer specifying the number of rows and columns.
normalize	A logical scalar specifying whether the kernel should be divided by its sum. Set to FALSE in case you use the kernel as a mask.
size	An odd integer specifying the kernel size.
theta	A numeric value (in radians) specifying the orientation.
strength	A numeric value controlling emboss intensity. Defaults to 1.
thickness	A numeric value controlling ring thickness. Defaults to 1.
mod	An integer specifying the pattern modulus.
step	An integer specifying the pattern step size.

Details

The following kernel constructors are available:

- `kernel_bayer`: A Bayer pattern kernel.
- `kernel_cone`: A cone-shaped, radially weighted kernel.
- `kernel_disc`: A uniformly weighted disc kernel.
- `kernel_emboss`: A direction-aware emboss kernel.
- `kernel_motion`: A linear motion kernel oriented by theta.
- `kernel_ring`: A ring-shaped (donut-like) kernel.
- `kernel_stripe`: A stripe pattern kernel.

Value

A numeric matrix.

mean_shift	<i>Mean shift filtering</i>
------------	-----------------------------

Description

Applies mean shift filtering to a `nativeRaster` image. This edge-preserving smoothing technique groups pixels in joint spatial-color space, producing a stylized effect with flattened regions while retaining clear boundaries.

Usage

```
mean_shift(nr, sp = 10, sr = 30, max_level = 1)
```

Arguments

nr	A nativeRaster object.
sp	A numeric scalar giving the spatial window radius. Larger values increase the smoothing effect across neighboring pixels.
sr	A numeric scalar giving the color (range) window radius. Larger values allow grouping across larger color differences.
max_level	An integer scalar specifying the number of pyramid levels to process. Must be non-negative.

Value

A nativeRaster object.

median_cut	<i>Reduce the number of colors with median cut quantization</i>
------------	---

Description

Applies median cut color quantization to a nativeRaster image and returns a new nativeRaster with a reduced palette.

This function groups similar colors into at most `n_colors` representative colors. The representative color of each group is computed from the weighted average of the original colors in that group.

If the number of unique colors in `nr` is already less than or equal to `n_colors`, the input is returned unchanged.

Usage

```
median_cut(nr, n_colors = 16)
```

Arguments

nr	A nativeRaster object.
n_colors	Maximum number of colors in the output image. Must be a positive integer.

Value

A nativeRaster object.

Description

Applies morphological image-processing operations to a nativeRaster image. Depending on `use_rgb`, operations are applied either to the grayscale image masked by the alpha channel or independently to each RGB channel with per-channel structuring elements. Supported operations include erosion, dilation, opening, closing, gradient, and related transforms.

Usage

```

morphology(
  nr,
  ksize = c(2, 2, 2),
  ktype = c(0, 1, 2),
  mode = c(0, 1, 2, 3, 4, 5, 6, 7),
  border = c(3, 4, 0, 1, 2),
  iterations = 1,
  alphasync = FALSE,
  use_rgb = TRUE,
  anchor = c(-1, -1)
)

```

Arguments

<code>nr</code>	A nativeRaster object.
<code>ksize</code>	An integer vector specifying kernel size(s). - Length 1 (grayscale mode): kernel radius. - Length 3 (RGB mode): per-channel kernel radii. Must be non-negative.
<code>ktype</code>	An integer scalar selecting the structuring element shape. One of 0, 1, 2, corresponding to OpenCV kernel types.
<code>mode</code>	An integer scalar selecting the morphological operation. One of 0–7, mapped to OpenCV operation modes (morphologyEx operations).
<code>border</code>	An integer scalar selecting the border-handling mode. One of 0–4, corresponding to OpenCV's border modes.
<code>iterations</code>	An integer scalar giving the number of repetitions of the operation. Must be greater than 0.
<code>alphasync</code>	A logical scalar. If TRUE, apply the morphological operation to the alpha channel as well.
<code>use_rgb</code>	A logical scalar. If TRUE, apply the operation independently to each RGB channel; if FALSE, apply it to a grayscale image derived from masked RGB.
<code>anchor</code>	An integer vector of length 2 specifying the anchor point (kernel origin). Use <code>c(-1, -1)</code> for the default centered anchor.

Details**Kernel specification:**

The shape of the structuring element is controlled by `ktype`, and its size is determined by `ksize`.

- When `use_rgb = FALSE`, `ksize` must be a single non-negative integer.
- When `use_rgb = TRUE`, `ksize` must be a length-3 integer vector, allowing separate kernel sizes for the R, G, and B channels.

The location of the anchor (kernel origin) can be specified via `anchor`. Use `c(-1, -1)` for the default centered anchor.

Alpha handling:

If `alphasync = TRUE`, the same morphological operation is applied to the alpha channel. Otherwise, the alpha channel is preserved.

Options:

`ktype` corresponds to:

1. `cv::MORPH_RECT`
2. `cv::MORPH_CROSS`
3. `cv::MORPH_ELLIPSE`

`mode` is an integer scalar in range `[0, 7]` corresponding to:

1. `cv::MORPH_ERODE`
2. `cv::MORPH_DILATE`
3. `cv::MORPH_OPEN`
4. `cv::MORPH_CLOSE`
5. `cv::MORPH_GRADIENT`
6. `cv::MORPH_TOPHAT`
7. `cv::MORPH_BLACKHAT`
8. `cv::MORPH_HITMISS`

`border` corresponds to:

1. `cv::BORDER_CONSTANT`
2. `cv::BORDER_REPLICATE`
3. `cv::BORDER_REFLECT`
4. `cv::BORDER_REFLECT_101`
5. `cv::BORDER_ISOLATED`

Value

A `nativeRaster` object.

oilpaint	<i>Oil painting effect</i>
----------	----------------------------

Description

Applies an oil-paint-style effect to a `nativeRaster` image. This filter produces a painterly appearance by quantizing and aggregating colors within a local neighborhood.

Usage

```
oilpaint(nr, size = 10, ratio = 1)
```

Arguments

<code>nr</code>	A <code>nativeRaster</code> object.
<code>size</code>	An integer scalar specifying the radius of the neighborhood used for computing the oil painting effect. Must be at least 2.
<code>ratio</code>	An integer scalar controlling the number of intensity bins used in the stylization process. Larger values preserve finer variations in tone.

Value

A `nativeRaster` object.

pack-unpack	<i>Pack and unpack RGBA values</i>
-------------	------------------------------------

Description

Pack and unpack RGBA values

Usage

```
pack_color(r, g, b, a)
```

```
unpack_color(x)
```

Arguments

<code>r, g, b, a</code>	Numeric vectors of equal length in range $[0, 255]$.
<code>x</code>	An integer vector that contains native packed integers.

Value

Integers.

pencil_sketch	<i>Pencil sketch effect</i>
---------------	-----------------------------

Description

Generates a pencil-sketch-style rendering of a nativeRaster image. This filter produces grayscale or color pencil-like strokes by combining edge extraction with tone mapping.

Usage

```
pencil_sketch(nr, sgms = 60, sgmr = 0.07, shade = 0.02, color = TRUE)
```

Arguments

nr	A nativeRaster object.
sgms	A numeric scalar giving the spatial sigma controlling the smoothness of the underlying edge-preserving filter. Valid range: 0–200.
sgmr	A numeric scalar giving the range sigma controlling sensitivity to color differences when building the sketch effect. Valid range: 0–1.
shade	A numeric scalar specifying the shading factor determining how much texture is included in the sketch. Valid range: 0–1.
color	A logical scalar. If TRUE, return the stylized color sketch; if FALSE, return the grayscale pencil sketch.

Value

A nativeRaster object.

pixel_positions	<i>Select pixel positions from a native raster object</i>
-----------------	---

Description

This function scans a nativeRaster image and returns the positions (row, column, and linear index) of pixels whose values fall within a given range.

Optionally, pixels can be grouped into contiguous runs along rows or columns, and only runs of a minimum length can be retained. This makes it suitable as a building block for pixel sorting or other effects that operate on contiguous pixel segments.

Usage

```
pixel_positions(
  nr,
  range = c(0, 0.5),
  by = c("luma", "blue", "green", "red", "hue", "luminance", "saturation"),
  direction = c("row", "col"),
  min_length = 1
)
```

Arguments

nr	A nativeRaster object.
range	A numeric vector of length 2 specifying the lower and upper bounds (in $[0, 1]$) used to select pixels.
by	A string specifying which pixel value to use for selection.
direction	Direction used to define contiguous runs when <code>min_length > 1</code> . Either "row" (horizontal runs) or "col" (vertical runs).
min_length	Minimum length of contiguous pixel runs to retain. If <code>min_length <= 1</code> , all matching pixels are returned without run grouping.

Details

Pixel values are normalized to the range $[0, 1]$ before comparison. For "hue", "luminance", and "saturation", the image is internally converted to HLS color space using OpenCV.

Value

If `min_length <= 1`, a tibble with columns:

- row: Row index (1-based).
- col: Column index (1-based).
- index: Linear index in the nativeRaster vector (1-based).

If `min_length > 1`, a tibble with columns:

- group: Row or column index defining the scan direction.
- pos: Position within each group.
- run: Contiguous run identifier.
- index: Linear index in the nativeRaster vector (1-based).

preserve_edge	<i>Edge-preserving smoothing filter</i>
---------------	---

Description

Applies an edge-preserving smoothing filter to a nativeRaster image. This filter attenuates noise and texture while retaining sharp boundaries, producing a clean, stylized appearance.

Usage

```
preserve_edge(nr, sgmS = 60, sgmR = 0.44, recursive = TRUE)
```

Arguments

nr	A nativeRaster object.
sgmS	A numeric scalar giving the spatial sigma, which controls the effective smoothing radius. Valid range: 0–200.
sgmR	A numeric scalar giving the range sigma, which determines how strongly edges are preserved based on color differences. Valid range: 0–1.
recursive	A logical scalar. If TRUE, use the recursive filtering mode (RECURS_FILTER); if FALSE, use normalized convolution (NORMCONV_FILTER).

Value

A nativeRaster object.

resize	<i>Resize or resample a native raster image</i>
--------	---

Description

These functions provide flexible image scaling based on OpenCV's `resize`, supporting both direct resizing and two-stage downsample–upsample resampling. The operations apply to all BGRA channels.

Usage

```
resize(
  nr,
  wh = c(1, 1),
  resize_mode = c(1, 2, 3, 4, 5, 6, 0),
  set_size = FALSE
)

resample(
```

```

    nr,
    wh = c(0.2, 0.2),
    resize_mode1 = c(1, 2, 3, 4, 5, 6, 0),
    resize_mode2 = c(1, 2, 3, 4, 5, 6, 0)
)

```

Arguments

nr	A nativeRaster object.
wh	A numeric vector of length 2. - For <code>resize()</code>: scaling factors when <code>set_size = FALSE</code>, or target dimensions (width, height) when <code>set_size = TRUE</code>. - For <code>resample()</code>: downsampling factors in each direction.
resize_mode	An integer scalar selecting the interpolation mode for <code>resize()</code> . Must be one of 0–6, corresponding to OpenCV's interpolation flags.
set_size	A logical scalar. If <code>TRUE</code> , <code>wh</code> is interpreted as absolute output dimensions; if <code>FALSE</code> , <code>wh</code> is used as scaling factors.
resize_mode1	An integer scalar giving the interpolation mode for the downsampling step in <code>resample()</code> . Must be one of 0–6.
resize_mode2	An integer scalar giving the interpolation mode for the upsampling step in <code>resample()</code> . Must be one of 0–6.

Details

`resize()`:

Resizes an image either by specifying a scaling factor (`wh`) or, when `set_size = TRUE`, by specifying a target width and height directly. Interpolation is controlled through `resize_mode`.

`resample()`:

Downsamples and then upsamples an image, potentially using different interpolation modes for reduction and expansion. This can be used to create stylized pixelation or texture effects, depending on the scaling factors and interpolation methods.

Value

A nativeRaster object.

screen_tone

Apply a screen-tone texture using a threshold map

Description

Applies a tiled tone/texture to an image using a grayscale threshold map (`pattern`). For each pixel, the input intensity is compared with the corresponding value in `pattern` (plus cutoff), and then either `lift` or `bias` is added to the pixel value.

This is useful for creating retro print-like textures, ordered-tone patterns, or "screen tone" overlays when `pattern` is a tiled image.

Usage

```
screen_tone(nr, pattern, bias = 0L, lift = 60L, cutoff = 8L)
```

Arguments

<code>nr</code>	A <code>nativeRaster</code> object.
<code>pattern</code>	A <code>nativeRaster</code> image used as a threshold map. Typically this is created by tiling a small matrix (e.g. Bayer matrix) to match <code>nr</code> .
<code>bias</code>	Integer. Value to add to pixels that do <i>not</i> pass the threshold. Larger values brighten the corresponding regions (clamped to <code>[0, 255]</code>).
<code>lift</code>	Integer. Value to add to pixels that <i>pass</i> the threshold. Larger values brighten the corresponding regions (clamped to <code>[0, 255]</code>).
<code>cutoff</code>	Integer. Offset added to <code>pattern</code> before thresholding. Increase this to make it harder for pixels to pass the threshold.

Value

A `nativeRaster` object.

<code>sobel_filter</code>	<i>Sobel edge detection (grayscale or RGB)</i>
---------------------------	--

Description

Applies a Sobel filter to extract edges from a `nativeRaster` image. The operation can be performed on a grayscale conversion or by aggregating per-channel gradients from the RGB channels. The resulting edge intensity is returned as a 3-channel grayscale image, with optional binary alpha masking.

Usage

```
sobel_filter(
  nr,
  ksize = 3,
  balp = TRUE,
  use_rgb = TRUE,
  border = c(3, 4, 0, 1, 2),
  dx = 1,
  dy = dx,
  scale = 1,
  delta = 0
)
```

Arguments

nr	A nativeRaster object.
ksize	An integer scalar giving the filter radius. The actual Sobel kernel size becomes $2 * ksize - 1$. Must be non-negative.
balp	A logical scalar. If TRUE, the alpha channel is thresholded from the edge map; if FALSE, the alpha channel is preserved.
use_rgb	A logical scalar. If TRUE, compute Sobel edges from each RGB channel and aggregate; if FALSE, compute on grayscale only.
border	An integer scalar selecting the border-handling mode. One of 0–4, corresponding to OpenCV border types.
dx	An integer scalar specifying the order of the derivative in x.
dy	An integer scalar specifying the order of the derivative in y.
scale	A numeric scalar scaling the computed Sobel derivative.
delta	A numeric scalar added to the results prior to scaling.

Details**Grayscale mode** (use_rgb = FALSE):

The image is converted to grayscale, normalized to $[0, 1]$, and the Sobel derivative is computed using the specified derivative orders dx, dy, and kernel size ksize. Edges are mapped to $[0, 255]$. When balp = TRUE, the alpha channel is binarized based on the detected edges.

RGB mode (use_rgb = TRUE):

Sobel derivatives are computed independently for each RGB channel and summed to produce a combined edge response. The alpha channel is either binarized from the edge map (balp = TRUE) or preserved from the input.

Options:

border corresponds to:

1. cv::BORDER_CONSTANT
2. cv::BORDER_REPLICATE
3. cv::BORDER_REFLECT
4. cv::BORDER_REFLECT_101
5. cv::BORDER_ISOLATED

Value

A nativeRaster object.

stylize	<i>Stylization filter</i>
---------	---------------------------

Description

Applies a stylization effect to a nativeRaster image. This filter combines edge-aware smoothing and abstraction to produce a painterly, cartoon-like appearance while preserving prominent structures.

Usage

```
stylize(nr, sgmS = 60, sgmR = 0.44)
```

Arguments

nr	A nativeRaster object.
sgmS	A numeric scalar giving the spatial sigma controlling the smoothness of the stylization process. Valid range: 0–200.
sgmR	A numeric scalar giving the range sigma determining sensitivity to color differences during abstraction. Valid range: 0–1.

Value

A nativeRaster object.

swap_channels	<i>Swap or remap image channels</i>
---------------	-------------------------------------

Description

Remaps color or alpha channels of a nativeRaster image using arbitrary pairwise assignments. This function wraps OpenCV's mixChannels, allowing reordering or selective copying of the BGRA channels.

Usage

```
swap_channels(nr, from = c(0, 1, 2, 3), to = c(1, 2, 0, 3))
```

Arguments

nr	A nativeRaster object.
from	An integer vector of length 4 giving the source channel indices (0 = B, 1 = G, 2 = R, 3 = A). Each element specifies which channel to read from.
to	An integer vector of length 4 giving the destination channel indices (0 = B, 1 = G, 2 = R, 3 = A). Each element specifies where the corresponding from channel is written.

Value

A nativeRaster object.

thres	<i>Thresholding and adaptive thresholding</i>
-------	---

Description

Performs global (fixed) thresholding or adaptive thresholding on a nativeRaster image. Both functions operate on a grayscale conversion of the input and return a 3-channel binary (0/maxv) image with the original alpha preserved.

Usage

```
thres(nr, threshold = 100, maxv = 255, mode = c(0, 1, 2, 3, 4, 5, 6))
```

```
adpthres(nr, maxv = 255, bsize = 1, C = 5, mode = c(0, 1), invert = FALSE)
```

Arguments

nr	A nativeRaster object.
threshold	A numeric scalar in $[0, 255]$ specifying the fixed threshold value.
maxv	A numeric scalar in $[0, 255]$. • For <code>thres()</code>: A numeric scalar in $[0, 255]$ giving the value assigned to "foreground" pixels. • For <code>adpthres()</code>: A numeric scalar in $[0, 255]$ specifying the output value for "foreground" pixels.
mode	An integer scalar specifying the thresholding mode. • For <code>thres()</code>: An integer scalar selecting the thresholding mode. Must be one of 0–6, corresponding to OpenCV's threshold flags. • For <code>adpthres()</code>: An integer scalar selecting the binary mode. 0 (THRESH_BINARY_INV) or 1 (THRESH_BINARY).
bsize	An integer scalar in $[1, 50]$ giving the neighborhood radius. The actual window size is computed as $2 * \text{bsize} + 1$.
C	A numeric scalar subtracted from the local threshold (used by OpenCV). Controls how strong the threshold offset is.
invert	A logical scalar. If TRUE, invert the local thresholding result.

Details**thres():**

Applies a fixed threshold to the grayscale image. Pixels with values greater than threshold (depending on the thresholding mode) are set to maxv; others are set to 0. The behavior is controlled by the thresholding mode, which corresponds to OpenCV's threshold types.

adpthres():

Applies adaptive thresholding, where the threshold value is computed locally for each pixel based on a neighborhood mean or Gaussian-weighted sum. The block size and constant C adjust how the local threshold is computed.

Options:

For thres, mode is an integer scalar in range [0, 6] corresponding to:

1. cv::THRESH_BINARY
2. cv::THRESH_BINARY_INV
3. cv::THRESH_TRUNC
4. cv::THRESH_TOZERO
5. cv::THRESH_TOZERO_INV
6. cv::THRESH_OTSU
7. cv::THRESH_TRIANGLE

Value

A nativeRaster object.

tile_matrix

Tile a numeric matrix into a pattern

Description

Repeats a numeric matrix x to fill an image of size width by height, then returns it as a grayscale nativeRaster (RGB channels are identical; alpha is set to 255).

This is mainly intended to build threshold/texture maps for effects such as [screen_tone\(\)](#), where small matrices (e.g. Bayer matrices or custom kernels) are tiled across an image.

Values are clamped to [0, 255] before being packed.

Usage

```
tile_matrix(x, width, height)
```

Arguments

x	A numeric matrix. Interpreted as a single-channel pattern.
width, height	Integers. Output image width and height in pixels.

Value

A nativeRaster of dimensions height * width.

Examples

```
# 4x4 Bayer matrix, scaled to 0..255
b4 <- kernel_bayer(4, normalize = FALSE) * 255
## Not run:
pat <- tile_matrix(b4, width = 320, height = 180)
grid::grid.newpage()
grid::grid.raster(pat, interpolate = FALSE)

## End(Not run)
```

warp_perspective

Warp an image using perspective transformation

Description

Warpes an image using a perspective transformation, keeping the same dimensions as the input image.

Usage

```
warp_perspective(nr, mat = diag(1, 3), border = c(3, 4, 0, 1, 2))
```

Arguments

nr	A nativeRaster object.
mat	A 3x3 matrix specifying the perspective transformation.
border	An integer scalar selecting pixel extrapolation method.

Details

border corresponds to the OpenCV extrapolation types:

1. cv::BORDER_CONSTANT
2. cv::BORDER_REPLICATE
3. cv::BORDER_REFLECT
4. cv::BORDER_WRAP
5. cv::BORDER_REFLECT_101

Value

A nativeRaster object.

Index

* datasets

- [blue_noise_64x64](#), [5](#)
- [adpthres](#) (thres), [35](#)
- [apply_lut1d](#) (color-manip), [9](#)
- [apply_lut3d](#) (color-manip), [9](#)
- [bilateral_filter](#), [3](#)
- [blend-mode](#), [4](#)
- [blend_add](#) (blend-mode), [4](#)
- [blend_alpha](#) (blend-mode), [4](#)
- [blend_average](#) (blend-mode), [4](#)
- [blend_colorburn](#) (blend-mode), [4](#)
- [blend_colordodge](#) (blend-mode), [4](#)
- [blend_darken](#) (blend-mode), [4](#)
- [blend_difference](#) (blend-mode), [4](#)
- [blend_divide](#) (blend-mode), [4](#)
- [blend_exclusion](#) (blend-mode), [4](#)
- [blend_ghosting](#) (blend-mode), [4](#)
- [blend_hardlight](#) (blend-mode), [4](#)
- [blend_hardmix](#) (blend-mode), [4](#)
- [blend_lighten](#) (blend-mode), [4](#)
- [blend_linearlight](#) (blend-mode), [4](#)
- [blend_luminosity](#) (blend-mode), [4](#)
- [blend_multiply](#) (blend-mode), [4](#)
- [blend_overlay](#) (blend-mode), [4](#)
- [blend_pinlight](#) (blend-mode), [4](#)
- [blend_screen](#) (blend-mode), [4](#)
- [blend_softlight](#) (blend-mode), [4](#)
- [blend_subtract](#) (blend-mode), [4](#)
- [blend_vividlight](#) (blend-mode), [4](#)
- [blue_noise_64x64](#), [5](#)
- [blur](#), [5](#)
- [blurhash](#), [7](#)
- [box_blur](#) (blur), [5](#)
- [brighten](#) (color-manip), [9](#)
- [canny_filter](#), [7](#)
- [color-manip](#), [9](#)
- [color_filter](#), [10](#)
- [color_map](#), [11](#)
- [contrast](#) (color-manip), [9](#)
- [convolve](#), [13](#)
- [detail_enhance](#), [14](#)
- [diffusion_filter](#), [14](#)
- [duotone](#) (color-manip), [9](#)
- [fill_with](#), [15](#)
- [gaussian_blur](#) (blur), [5](#)
- [grayscale](#) (color-manip), [9](#)
- [hist_eq](#), [16](#)
- [hue_rotate](#) (color-manip), [9](#)
- [image-io](#), [16](#)
- [invert](#) (color-manip), [9](#)
- [kernel_bayer](#) (make-kernels), [22](#)
- [kernel_cone](#) (make-kernels), [22](#)
- [kernel_disc](#) (make-kernels), [22](#)
- [kernel_emboss](#) (make-kernels), [22](#)
- [kernel_motion](#) (make-kernels), [22](#)
- [kernel_ring](#) (make-kernels), [22](#)
- [kernel_stripe](#) (make-kernels), [22](#)
- [kuwahara_filter](#), [18](#)
- [laplacian_filter](#), [19](#)
- [lineweave](#), [20](#)
- [linocut](#) (color-manip), [9](#)
- [luma_gate](#), [21](#)
- [make-kernels](#), [22](#)
- [mean_shift](#), [23](#)
- [median_blur](#) (blur), [5](#)
- [median_cut](#), [24](#)
- [morphology](#), [25](#)
- [oilpaint](#), [27](#)
- [pack-unpack](#), [27](#)

pack_color (pack-unpack), 27
pencil_sketch, 28
pixel_positions, 28
posterize (color-manip), 9
preserve_edge, 30

read_data (image-io), 16
read_still (image-io), 16
resample (resize), 30
reset_alpha (color-manip), 9
resize, 30

saturate (color-manip), 9
screen_tone, 31
screen_tone(), 36
sepia (color-manip), 9
set_matte (color-manip), 9
sobel_filter, 32
solarize (color-manip), 9
stylize, 34
swap_channels, 34

thres, 35
tile_matrix, 36

unpack_color (pack-unpack), 27
unpremul (color-manip), 9

warp_perspective, 37
write_animation (image-io), 16
write_data (image-io), 16
write_still (image-io), 16