Package 'magick'

September 2, 2017

Type Package

Title Advanced Graphics and Image-Processing in R

Version 1.3

Author Jeroen Ooms

Maintainer Jeroen Ooms <jeroen@berkeley.edu>

Description Bindings to 'ImageMagick': the most comprehensive open-source image processing library available. Supports many common formats (png, jpeg, tiff, pdf, etc) and manipulations (rotate, scale, crop, trim, flip, blur, etc). All operations are vectorized via the Magick++ STL meaning they operate either on a single frame or a series of frames for working with layers, collages, or animation. In RStudio images are automatically previewed when printed to the console, resulting in an interactive editing environment. The latest version of the package includes a native graphics device for creating in-memory graphics or drawing onto images using pixel coordinates.

License MIT + file LICENSE

URL https://github.com/ropensci/magick#readme

BugReports https://github.com/ropensci/magick/issues

SystemRequirements ImageMagick++: ImageMagick-c++-devel (rpm) or libmagick++-dev (deb)

VignetteBuilder knitr

Imports Rcpp (>= 0.12.12), magrittr, curl

LinkingTo Rcpp

Suggests knitr, rmarkdown, rsvg, webp, png, pdftools, ggplot2, raster, rgdal, gapminder, tesseract

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-09-02 15:22:58 UTC

R topics documented:

`analysis` *Image Analysis*

Description

Functions for image calculations and analysis. This part of the package needs more work.

Usage

```
image_compare(image, reference_image, metric = "")

image_fft(image)
```

Arguments

`image`	magick image object returned by `image_read()` or `image_graph()`
`reference_image`	another image to compare to
`metric`	string with metric type

Details

For details see Image++ documentation. Short descriptions:

- image_compare calculates a metric by comparing image with a reference image.
- image_fft returns Discrete Fourier Transform (DFT) of the image as a magnitude / phase image pair. I wish I knew what this means.

Other analysis functions will be added in future versions!

See Also

Other image: _index_, animation, attributes, color, composite, device, editing, effects, ocr, painting, transform

Examples

```
logo <- image_read("logo:")
logo2 <- image_blur(logo, 3)
logo3 <- image_oilpaint(logo, 3)
if(magick_config()$version >= "6.8.7"){
  image_compare(logo, logo2, metric = "phash")
  image_compare(logo, logo3, metric = "phash")
}
```

---

animation *Image Frames and Animation*

Description

Operations to manipulate or combine multiple frames of an image. Details below.

Usage

```
image_animate(image, fps = 10, loop = 0, dispose = c("background",
  "previous", "none"))

image_morph(image, frames = 8)

image_mosaic(image, operator = NULL)

image_montage(image)

image_flatten(image, operator = NULL)

image_average(image)

image_append(image, stack = FALSE)

image_apply(image, FUN, ...)
```

Arguments

image	magick image object returned by image_read() or image_graph()
fps	frames per second
loop	how many times to repeat the animation. Default is infinite.
dispose	frame disposal method

frames	number of frames to use in output animation
operator	string with a composite operator
stack	place images top-to-bottom (TRUE) or left-to-right (FALSE)
FUN	a function to be called on each frame in the image
...	additional parameters for FUN

Details

For details see Magick++ STL documentation. Short descriptions:

- image_animate coalesces frames by playing the sequence and converting to gif format.
- image_morph expands number of frames by interpolating intermediate frames to blend into each other when played as an animation.
- image_mosaic inlays images to form a single coherent picture.
- image_montage creates a composite image by combining frames.
- image_flatten merges frames as layers into a single frame using a given operator.
- image_average averages frames into single frame.
- image_append stack images left-to-right (default) or top-to-bottom.
- image_apply applies a function to each frame

The image_apply function calls an image function to each frame and joins results back into a single image. Because most operations are already vectorized this is often not needed. Note that FUN() should return an image. To apply other kinds of functions to image frames simply use lapply, vapply, etc.

See Also

Other image: _index_, analysis, attributes, color, composite, device, editing, effects, ocr, painting, transform

Examples

```
# Combine images
logo <- image_read("https://www.r-project.org/logo/Rlogo.png")
oldlogo <- image_read("https://developer.r-project.org/Logo/Rlogo-3.png")

# Create morphing animation
both <- image_scale(c(oldlogo, logo), "400")
image_average(image_crop(both))
image_animate(image_morph(both, 10))

# Create thumbnails from GIF
banana <- image_read("https://jeroen.github.io/images/banana.gif")
length(banana)
image_average(banana)
image_flatten(banana)
image_append(banana)
image_append(banana, stack = TRUE)
```

```
# Append images together
wizard <- image_read("wizard:")
image_append(image_scale(c(image_append(banana[c(1,3)], stack = TRUE), wizard)))

image_composite(banana, image_scale(logo, "300"))

# Break down and combine frames
front <- image_scale(banana, "300")
background <- image_background(image_scale(logo, "400"), 'white')
frames <- image_apply(front, function(x){image_composite(background, x, offset = "+70+30")})
image_animate(frames, fps = 10)
```

---

attributes *Image Attributes*

Description

Attributes are properties of the image that might be present on some images and might affect image manipualation methods.

Usage

```
image_comment(image, comment = NULL)

image_info(image)
```

Arguments

image	magick image object returned by image_read() or image_graph()
comment	string to set an image comment

Details

Each attribute can be get and set with the same function. The image_info() function returns a data frame with some commonly used attributes.

See Also

Other image: _index_, analysis, animation, color, composite, device, editing, effects, ocr, painting, transform

autoviewer *RStudio AutoViewer*

Description

This enables a addTaskCallback that automatically updates the viewer after the state of a magick graphics device has changed. This is enabled by default in RStudio.

Usage

```
autoviewer_enable()

autoviewer_disable()
```

Examples

```
# Only has effect in RStudio (or other GUI with a viewer):
autoviewer_enable()

img <- magick::image_graph()
plot(1)
abline(0, 1, col = "blue", lwd = 2, lty = "solid")
abline(0.1, 1, col = "red", lwd = 3, lty = "dotted")

autoviewer_disable()
abline(0.2, 1, col = "green", lwd = 4, lty = "twodash")
abline(0.3, 1, col = "black", lwd = 5, lty = "dotdash")

autoviewer_enable()
abline(0.4, 1, col = "purple", lwd = 6, lty = "dashed")
abline(0.5, 1, col = "yellow", lwd = 7, lty = "longdash")
```

coder_info *Magick Configuration*

Description

ImageMagick can be configured to support various additional tool and formats via external libraries. These funtions show which features ImageMagick supports on your system.

Usage

```
coder_info(format)

magick_config()
```

Arguments

format	image format such as png, tiff or pdf.

Details

Note that coder_info raises an error for unsupported formats.

References

https://www.imagemagick.org/Magick++/CoderInfo.html

Examples

```
coder_info("png")
coder_info("jpg")
coder_info("pdf")
coder_info("tiff")
coder_info("gif")
```

---

color *Image Color*

Description

Functions to adjust contrast, brightness, colors of the image. Details below.

Usage

```
image_modulate(image, brightness = 100, saturation = 100, hue = 100)

image_quantize(image, max = 256, colorspace = NULL, dither = NULL,
  treedepth = NULL)

image_map(image, map, dither = FALSE)

image_transparent(image, color, fuzz = 0)

image_background(image, color, flatten = TRUE)

image_colorize(image, opacity, color)

image_contrast(image, sharpen = 1)

image_normalize(image)

image_enhance(image)
```

```
image_equalize(image)

image_median(image, radius = 1)
```

Arguments

image	magick image object returned by image_read() or image_graph()
brightness	modulation of brightness as percentage of the current value (100 for no change)
saturation	modulation of saturation as percentage of the current value (100 for no change)
hue	modulation of hue is an absolute rotation of -180 degrees to +180 degrees from the current position corresponding to an argument range of 0 to 200 (100 for no change)
max	preferred number of colors in the image. The actual number of colors in the image may be less than your request, but never more.
colorspace	string with a magick ColorspaceType for example "gray", "rgb" or "cmyk"
dither	apply Floyd/Steinberg error diffusion to the image: averages intensities of ssveral neighboring pixels
treedepth	depth of the quantization color classification tree. Values of 0 or 1 allow selection of the optimal tree depth for the color reduction algorithm. Values between 2 and 8 may be used to manually adjust the tree depth.
map	reference image to map colors from
color	a valid color string such as "navyblue" or "#000080"
fuzz	Colors within this distance are considered equal. Use this option to match colors that are close to the target color in RGB space. I think max distance (from #000000 to #FFFFFF) is 256^2.
flatten	should image be flattened before writing? This also replaces transparency with background color.
opacity	percentage of opacity used for coloring
sharpen	enhance intensity differences in image
radius	replace each pixel with the median color in a circular neighborhood

Details

For details see Magick++ STL documentation. Short descriptions:

- image_modulate adjusts brighness, saturation and hue of image relative to current.
- image_quantize reduces number of unique colors in the image.
- image_map replaces colors of image with the closest color from a reference image.
- image_transparent sets pixels approximately matching given color to transparent.
- image_background sets background color. When image is flattened, transparent pixels get background color.
- image_colorize overlays a solid color frame using specified opacity.
- image_contrast enhances intensity differences in image

- image_normalize increases contrast by normalizing the pixel values to span the full range of colors
- image_enhance tries to minimize noise
- image_equalize equalizes using histogram equalization
- image_median replaces each pixel with the median color in a circular neighborhood

Note that colors are also determined by image properties imagetype and colorspace which can be modifie via `image_convert()`.

See Also

Other image: `_index_`, `analysis`, `animation`, `attributes`, `composite`, `device`, `editing`, `effects`, `ocr`, `painting`, `transform`

Examples

```
# manually adjust colors
logo <- image_read("logo:")
image_modulate(logo, brightness = 200)
image_modulate(logo, saturation = 150)
image_modulate(logo, hue = 200)

# Reduce image to 10 different colors using various spaces
image_quantize(logo, max = 10, colorspace = 'gray')
image_quantize(logo, max = 10, colorspace = 'rgb')
image_quantize(logo, max = 10, colorspace = 'cmyk')

# Change background color
translogo <- image_transparent(logo, 'white')
image_background(translogo, "pink", flatten = TRUE)

# Compare to flood-fill method:
image_fill(logo, "pink", fuzz = 10000)

# Other color tweaks
image_colorize(logo, 50, "red")
image_contrast(logo)
image_normalize(logo)
image_enhance(logo)
image_equalize(logo)
image_median(logo)

# Alternate way to convert into black-white
image_convert(logo, type = 'grayscale')
```

composite *Image Composite*

Description

Similar to the ImageMagick `composite` utility: compose an image on top of another one using a CompositeOperator.

Usage

```
image_composite(image, composite_image = image[1], operator = "atop",
  offset = "0x0", compose_args = "")

image_border(image, color = "lightgray", geometry = "10x10",
  operator = "copy")

image_frame(image, color = "lightgray", geometry = "25x25+6+6")
```

Arguments

`image`	magick image object returned by `image_read()` or `image_graph()`
`composite_image`	composition image
`operator`	string with a composite operator
`offset`	geometry string with offset
`compose_args`	additional arguments needed for some composite operations
`color`	a valid color string such as `"navyblue"` or `"#000080"`
`geometry`	a geometry string to set height and width of the border, e.g. `"10x8"`. In addition image_frame allows for adding shadow by setting an offset e.g. `"20x10+7+2"`.

Details

Basically image_border creates a slighly larger solid color frame and then composes the image frame on top. The image_frame function is similar but has an additional feature to create a shadow effect on the border (which is really ugly).

See Also

Other image: `_index_`, `analysis`, `animation`, `attributes`, `color`, `device`, `editing`, `effects`, `ocr`, `painting`, `transform`

Examples

```
# Compose images using one of many operators
imlogo <- image_scale(image_read("logo:"), "x275")
rlogo <- image_read("https://developer.r-project.org/Logo/Rlogo-3.png")

# Standard is atop
image_composite(imlogo, rlogo)

# Same as 'blend 50' in the command line
image_composite(imlogo, rlogo, operator = "blend", compose_args="50")

# Add a border frame around the image
image_border(imlogo, "red", "10x10")
image_frame(imlogo)
```

---

device *Magick Graphics Device*

Description

Graphics device that produces a Magick image. Can either be used like a regular device for making plots, or alternatively via `image_draw` to open a device which draws onto an existing image using pixel coordinates. The latter is vectorized, i.e. drawing operations are applied to each frame in the image.

Usage

```
image_graph(width = 800, height = 600, bg = "white", pointsize = 12,
  res = 72, clip = TRUE, antialias = TRUE)

image_draw(image, pointsize = 12, res = 72, antialias = TRUE, ...)

image_capture()
```

Arguments

`width`	in pixels
`height`	in pixels
`bg`	background color
`pointsize`	size of fonts
`res`	resolution in pixels
`clip`	enable clipping in the device. Because clippig can slow things down a lot, you can disable it if you don't need it.
`antialias`	TRUE/FALSE: enables anti-aliasing for text and strokes
`image`	an existing image on which to start drawing
`...`	additional device parameters passed to plot.window such as `xlim`, `ylim`, or `mar`.

Details

The device is a relatively recent feature of the package. It should support all operations but there might still be small inaccuracies. Also it is a bit slower than some of the other devices, in particular for rendering text and clipping. Hopefully this can be optimized in the next version.

By default `image_draw` sets all margins to 0 and uses graphics coordinates to match image size in pixels (width x height) where `(0,0)` is the top left corner. Note that this means the y axis increases from top to bottom which is the opposite of typical graphics coordinates. You can override all this by passing custom `xlim`, `ylim` or `mar` values to `image_draw`.

The `image_capture` function returns the current device as an image. This only works if the current device is a magick device or suppots dev.capture.

See Also

Other image: `_index_`, `analysis`, `animation`, `attributes`, `color`, `composite`, `editing`, `effects`, `ocr`, `painting`, `transform`

Examples

```
# Regular image
frink <- image_read("https://jeroen.github.io/images/frink.png")

# Produce image using graphics device
fig <- image_graph(res = 96)
ggplot2::qplot(mpg, wt, data = mtcars, colour = cyl)
dev.off()

# Combine
out <- image_composite(fig, frink, offset = "+70+30")
print(out)

# Or paint over an existing image
img <- image_draw(frink)
rect(20, 20, 200, 100, border = "red", lty = "dashed", lwd = 5)
abline(h = 300, col = 'blue', lwd = '10', lty = "dotted")
text(10, 250, "Hoiven-Glaven", family = "courier", cex = 4, srt = 90)
palette(rainbow(11, end = 0.9))
symbols(rep(200, 11), seq(0, 400, 40), circles = runif(11, 5, 35),
  bg = 1:11, inches = FALSE, add = TRUE)
dev.off()
print(img)

# Vectorized example with custom coordinates
earth <- image_read("https://jeroen.github.io/images/earth.gif")
img <- image_draw(earth, xlim = c(0,1), ylim = c(0,1))
rect(.1, .1, .9, .9, border = "red", lty = "dashed", lwd = 5)
text(.5, .9, "Our planet", cex = 3, col = "white")
dev.off()
print(img)
```

editing *Image Editing*

Description

Read, write and join or combine images. All image functions are vectorized, meaning they operate either on a single frame or a series of frames (e.g. a collage, video, or animation). Besides paths and URLs, the `image_read()` function supports all commonly used bitmap and raster types.

Usage

```
image_read(path, density = NULL, depth = NULL)

image_write(image, path = NULL, format = NULL, quality = NULL,
  depth = NULL, density = NULL, comment = NULL, flatten = FALSE)

image_convert(image, format = NULL, type = NULL, colorspace = NULL,
  depth = NULL, antialias = NULL)

image_display(image, animate = TRUE)

image_browse(image, browser = getOption("browser"))

image_join(...)
```

Arguments

`path`	a file, url, or raster object or bitmap array
`density`	resolution to render pdf or svg
`depth`	color depth (either 8 or 16)
`image`	magick image object returned by `image_read()` or `image_graph()`
`format`	output format such as `"png"`, `"jpeg"`, `"gif"`, `"rgb"` or `"rgba"`.
`quality`	number between 0 and 100 for jpeg quality. Defaults to 75.
`comment`	text string added to the image metadata for supported formats
`flatten`	should image be flattened before writing? This also replaces transparency with background color.
`type`	a magick ImageType classification for example `grayscale` to convert image to black/white
`colorspace`	string with a magick ColorspaceType for example `"gray"`, `"rgb"` or `"cmyk"`
`antialias`	enable anti-aliasing for text and strokes
`animate`	support animations in the X11 display
`browser`	argument passed to browseURL
`...`	several images or lists of images to be combined

Details

Besides functions above, all standard base vector methods such as [, [[, c(), as.list(), as.raster(), rev(), length(), and print() can be used with magick images.

Use the standard img[i] syntax to extract a subset of the frames from an image. The img[[i]] method is used to extract a single frame as a bitmap object, i.e. a raw matrix with pixel values.

X11 is required for image_display() which is only works on some platforms. A more portable method is image_browse() which opens the image in a browser. RStudio has an embedded viewer that does this automatically which is quite nice.

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, effects, ocr, painting, transform

Examples

```
# Download image from the web
frink <- image_read("https://jeroen.github.io/images/frink.png")
worldcup_frink <- image_fill(frink, "orange", "+100+200", 30000)
image_write(worldcup_frink, "output.png")

# extract raw bitmap array
bitmap <- frink[[1]]

# replace pixels with #FF69B4 ('hot pink') and convert back to image
bitmap[,50:100, 50:100] <- as.raw(c(0xff, 0x69, 0xb4, 0xff))
image_read(bitmap)

# Plot to graphics device via legacy raster format
raster <- as.raster(frink)
par(ask=FALSE)
plot(raster)

# Read bitmap arrays
curl::curl_download("https://www.r-project.org/logo/Rlogo.png", "Rlogo.png")
image_read(png::readPNG("Rlogo.png"))

curl::curl_download("https://jeroen.github.io/images/example.webp", "example.webp")
image_read(webp::read_webp("example.webp"))

curl::curl_download("http://jeroen.github.io/images/tiger.svg", "tiger.svg")
image_read(rsvg::rsvg("tiger.svg"))
```

---

effects *Image Effects*

Description

High level effects applied to an entire image. These are mostly just for fun.

Usage

```
image_despeckle(image, times = 1L)

image_reducenoise(image, radius = 1L)

image_noise(image, noisetype = "gaussian")

image_blur(image, radius = 1, sigma = 0.5)

image_charcoal(image, radius = 1, sigma = 0.5)

image_edge(image, radius = 1)

image_oilpaint(image, radius = 1)

image_emboss(image, radius = 1, sigma = 0.5)

image_implode(image, factor = 0.5)

image_negate(image)
```

Arguments

image	magick image object returned by image_read() or image_graph()
times	number of times to repeat the despeckle operation
radius	radius, in pixels, for various transformations
noisetype	integer betwee 0 and 5 with noisetype
sigma	the standard deviation of the Laplacian, in pixels.
factor	image implode factor (special effect)

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, editing, ocr, painting, transform

Examples

```
logo <- image_read("logo:")
image_despeckle(logo)
image_reducenoise(logo)
image_noise(logo)
image_blur(logo, 10, 10)
image_charcoal(logo)
image_edge(logo)
image_oilpaint(logo, radius = 3)
image_emboss(logo)
image_implode(logo)
image_negate(logo)
```

ocr *Image Text OCR*

Description

Extract text from an image using the tesseract package.

Usage

```
image_ocr(image, language = "eng", ...)
```

Arguments

image	magick image object returned by image_read() or image_graph()
language	passed to tesseract. To install additional languages see instructions in tesseract_download().
...	additional parameters passed to tesseract

Details

To use this function you need to tesseract first:

```
install.packages("tesseract")
```

Best results are obtained if you set the correct language in tesseract. To install additional languages see instructions in tesseract_download().

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, editing, effects, painting, transform

Examples

```
if(require("tesseract")){
img <- image_read("http://jeroen.github.io/images/testocr.png")
image_ocr(img)
}
```

painting *Image Painting*

Description

The image_fill() function performs flood-fill by painting starting point and all neighboring pixels of approximately the same color. Annotate simply prints some text on the image.

Usage

```
image_fill(image, color, point = "1x1", fuzz = 0)

image_annotate(image, text, gravity = "northwest", location = "+0+0",
  degrees = 0, size = 10, font = NULL, color = NULL,
  strokecolor = NULL, boxcolor = NULL)
```

Arguments

image	magick image object returned by image_read() or image_graph()
color	a valid color string such as "navyblue" or "#000080"
point	string indicating the flood-fill starting point
fuzz	Colors within this distance are considered equal. Use this option to match colors that are close to the target color in RGB space. I think max distance (from #000000 to #FFFFFF) is 256^2.
text	annotation text
gravity	string with gravity type
location	geometry string with location relative to gravity
degrees	rotates text around center point
size	font-size in pixels
font	rendering font. To use a TrueType font, precede the TrueType filename with an @.
strokecolor	adds a stroke (border around the text)
boxcolor	background color that annotation text is rendered on.

Details

Note that more sophisticated drawing mechanisms are available via the graphics device using image_draw.

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, editing, effects, ocr, transform

Examples

```
logo <- image_read("logo:")
logo <- image_transparent(logo, 'white')
image_fill(image_flatten(logo), "red")
image_fill(image_flatten(logo), "red", fuzz = 25600)
# Add some text to an image
image_annotate(logo, "This is a test")
image_annotate(logo, "CONFIDENTIAL", size = 50, color = "red", boxcolor = "pink",
 degrees = 30, location = "+100+100")

# Setting fonts requires fontconfig support (and that you have the font)
myfont <- ifelse(identical("windows", .Platform$OS.type), "courier-new", "courier")
try(image_annotate(logo, "The quick brown fox", font = myfont, size = 50))
```

`transform` *Image Transform*

Description

Basic transformations like rotate, resize, crop and flip. Details below.

Usage

```
image_trim(image)

image_chop(image, geometry)

image_rotate(image, degrees)

image_resize(image, geometry = NULL, filter = NULL)

image_scale(image, geometry = NULL)

image_sample(image, geometry = NULL)

image_crop(image, geometry = NULL)

image_flip(image)

image_flop(image)

image_deskew(image, treshold = 40)

image_page(image, pagesize = NULL, density = NULL)
```

Arguments

image	magick image object returned by image_read() or image_graph()
geometry	a string with geometry syntax specifying width+height and/or position offset. See details and examples below.
degrees	value between 0 and 360 for how many degrees to rotate
filter	string with a filtertype.
treshold	straightens an image. A threshold of 40 works for most images.
pagesize	geometry string with preferred size and location of an image canvas
density	geometry string with vertical and horizontal resolution in pixels of the image. Specifies an image density when decoding a Postscript or PDF.

Details

For details see Magick++ STL documentation. Short descriptions:

- image_trim removes edges that are the background color from the image.
- image_chop removes vertical or horizontal subregion of image.
- image_crop cuts out a subregion of original image
- image_rotate rotates and increases size of canvas to fit rotated image.
- image_deskew auto rotate to correct skewed images
- image_resize resizes using custom filterType
- image_scale and image_sample resize using simple ratio and pixel sampling algorithm.
- image_flip and image_flop invert image vertically and horizontally

The most powerful resize function is image_resize which allows for setting a custom resize filter. Output of image_scale is similar to `image_resize(img, filter = "point")`.

For resize operations it holds that if no `geometry` is specified, all frames are rescaled to match the top frame. Examples of `geometry` strings:

- `"500x300"` – *Resize image keeping aspect ratio, such that width does not exceed 500 and the height does not exceed 300.*
- `"500x300!"` – *Resize image to 500 by 300, ignoring aspect ratio*
- `"500x"` – *Resize width to 500 keep aspect ratio*
- `"x300"` – *Resize height to 300 keep aspect ratio*
- `"50%x20%"` – *Resize width to 50 percent and height to 20 percent of original*
- `"500x300+10+20"` – *Crop image to 500 by 300 at position 10,20*

See Also

Other image: _index_, analysis, animation, attributes, color, composite, device, editing, effects, ocr, painting

Examples

```
logo <- image_read("logo:")
logo <- image_scale(logo, "400")
image_trim(logo)
image_chop(logo, "100x20")
image_rotate(logo, 45)
# Small image
rose <- image_convert(image_read("rose:"), "png")

# Resize to 400 width or height:
image_resize(rose, "400x")
image_resize(rose, "x400")

# Resize keeping ratio
image_resize(rose, "400x400")

# Resize, force size losing ratio
image_resize(rose, "400x400!")

# Different filters
image_resize(rose, "400x", filter = "Triangle")
image_resize(rose, "400x", filter = "Point")
# simple pixel resize
image_scale(rose, "400x")
image_sample(rose, "400x")
image_crop(logo, "400x400+200+200")
image_flip(logo)
image_flop(logo)
```

---

index *Magick Image Processing*

Description

The magick package for graphics and image processing in R. Important resources:

- R introduction vignette: getting started
- Magick++ API and Magick++ STL detailed descriptions of methods and parameters

Details

Documentation is split into the following pages:

- analysis - metrics and calculations: compare, fft
- animation - manipulate or combine multiple frames: animate, morph, mosaic, montage, average, append, apply
- attributes - image properties: comment, info
- color - contrast, brightness, colors: modulate, quantize, map, transparent, background, colorize, contrast, normalize, enhance, equalize, median

- composite - advanced joining: `composite`, `border`, `frame`
- device - creating graphics and drawing on images
- editing - basice image IO: `read`, `write`, `convert`, `join`, `display`, `brose`
- effects - fun effects: `despecle`, `reducenoise`, `noise`, `blur`, `charcoal`, `edge`, `oilpaint`, `emboss`, `implode`, `negate`
- ocr - extract text from image using tesseract package
- painting - flood fill and annotating text
- transform - shape operations: `trim`, `chop`, `rotate`, `resize`, `scale`, `sample` `crop`, `flip`, `flop`, `deskew`, `page`

See Also

Other image: `analysis`, `animation`, `attributes`, `color`, `composite`, `device`, `editing`, `effects`, `ocr`, `painting`, `transform`

Index