Package 'imagefluency'

May 8, 2026

Type Package

Title Image Statistics Based on Processing Fluency

Version 1.0.0

Description Get image statistics based on processing fluency theory. The functions provide scores for several basic aesthetic principles that facilitate fluent cognitive processing of images: contrast, complexity / simplicity, self-similarity, symmetry, and typicality. See Mayer & Landwehr (2018) <doi:10.1037/aca0000187> and Mayer & Landwehr (2018) <doi:10.31219/osf.io/gtbhw> for the theoretical background of the methods.

License GPL-3

Encoding UTF-8

URL https://imagefluency.com, https://github.com/stm/imagefluency/, https://doi.org/10.5281/zenodo.5614665

BugReports https://github.com/stm/imagefluency/issues/

Depends R (>= 4.1.0)

Imports R.utils, readbitmap, pracma, magick, OpenImageR

Suggests grid, ggplot2, scales, shiny, bslib, testthat, mockery, knitr, rmarkdown, furrr, future, pbmcapply, tictoc, dplyr, collapse

RoxygenNote 7.3.3

Config/testthat/edition 3

Collate 'utils.R' 'complexity.R' 'contrast.R' 'imagefluency-package.R' 'imagefluencyApp.R' 'self-similarity.R' 'simplicity.R' 'symmetry.R' 'typicality.R'

VignetteBuilder knitr

NeedsCompilation no

Author Stefan Mayer [aut, cre] (ORCID: <https://orcid.org/0000-0003-0034-7090>)

Maintainer Stefan Mayer <stefan@mayer-de.com>

Repository CRAN

Date/Publication 2026-03-31 22:40:15 UTC

Contents

`img_complexity` *Image complexity*

Description

`img_complexity` returns the complexity of an image via image compression. Higher values indicate higher image complexity.

Usage

```
img_complexity(imgfile, algorithm = "zip", rotate = FALSE)
```

Arguments

`imgfile`	Either a character string containing the path to the image file (or URL) or an an image in form of a matrix (grayscale image) or array (color image) of numeric values representing the pre-loaded image (e.g. by using `img_read()`).
`algorithm`	Character string that specifies which image compression algorithm to use. Currently implemented are `zip` with deflate compression (default), `jpg`, `gif`, and `png`.
`rotate`	logical. Should the compressed file size of the rotated image also be computed? (see details)

Details

The function returns the visual complexity of an image. Visual complexity is calculated as ratio between the compressed and uncompressed image file size. Preferably, the original image is an uncompressed image file.

The function takes the file path of an image file (or URL) or a pre-loaded image as input argument (`imgfile`) and returns the ratio of the compressed divided by the uncompressed image file size. Values can range between almost 0 (virtually completely compressed image, thus extremely simple image) and 1 (no compression possible, thus extremely complex image).

You can choose between different image compression algorithms. Currently implemented are `zip` with deflate compression (default), `jpg`, `gif`, and `png`. See Mayer & Landwehr (2018) for a discussion of different image compression algorithms for measuring visual complexity.

As most compression algorithms do not depict horizontal and vertical redundancies equally, the function includes an optional `rotate` parameter (default: `FALSE`). Setting this parameter to `TRUE` has the following effects: first, the image is rotated by 90 degrees. Second, a compressed version of the rotated image is created. Finally, the overall compressed image's file size is computed as the minimum of the original image's file size and the file size of the rotated image.

As `R`'s built-in `bmp` device creates (a) indexed instead of True Color images and (b) creates files with different file sizes depending on the operating system, the function relies on the `magick` package to write (and read) images.

Value

a numeric value: the ratio of the compressed divided by the uncompressed image file size

References

Donderi, D. C. (2006). Visual complexity: A Review. *Psychological Bulletin*, *132*, 73–97. doi:10.1037/00332909.132.1.73

Forsythe, A., Nadal, M., Sheehy, N., Cela-Conde, C. J., & Sawey, M. (2011). Predicting Beauty: Fractal Dimension and Visual Complexity in Art. *British Journal of Psychology*, *102*, 49–70. doi:10.1348/000712610X498958

Mayer, S. & Landwehr, J, R. (2018). Quantifying Visual Aesthetics Based on Processing Fluency Theory: Four Algorithmic Measures for Antecedents of Aesthetic Preferences. *Psychology of Aesthetics, Creativity, and the Arts*, *12*(4), 399–431. doi:10.1037/aca0000187

See Also

`img_read`, `img_contrast`, `img_self_similarity`, `img_simplicity`, `img_symmetry`, `img_typicality`,

Examples

```
# Example image with high complexity: trees
trees <- img_read(system.file("example_images", "trees.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(trees)
#
# get complexity
img_complexity(trees)


# Example image with low complexity: sky
sky <- img_read(system.file("example_images", "sky.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(sky)
#
```

```
# get complexity
img_complexity(sky)
```

`img_contrast` *Image contrast*

Description

`img_contrast` returns the RMS contrast of an image `img`. A higher value indicates higher contrast.

Usage

```
img_contrast(img)
```

Arguments

`img`	An image in form of a matrix or array of numeric values. Use e.g. `img_read()` to read an image file into R.

Details

The function returns the RMS contrast of an image `img`. The RMS contrast is defined as the standard deviation of the normalized pixel intensity values. A higher value indicates higher contrast. The image is automatically normalized if necessary (i.e., normalization into range [0, 1]).

For color images, the weighted average between each color channel's values is computed.

Value

a numeric value (RMS contrast)

References

Peli, E. (1990). Contrast in complex images. *Journal of the Optical Society of America A*, *7*, 2032–2040. doi:10.1364/JOSAA.7.002032

See Also

`img_read`, `img_complexity`, `img_self_similarity`, `img_simplicity`, `img_symmetry`, `img_typicality`,

Examples

```
# Example image with relatively high contrast: berries
berries <- img_read(system.file("example_images", "berries.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(berries)
#
# get contrast
```

```
img_contrast(berries)

# Example image with relatively low contrast: bike
bike <- img_read(system.file("example_images", "bike.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(bike)
#
# get contrast
img_contrast(bike)
```

`img_read` *Read bitmap image (bmp, jpg, png, tiff)*

Description

Wrapper for readbitmap's `read.bitmap` function. The function currently allows reading in images in `bmp`, `jpg` / `jpeg`, `png`, or `tif` / `tiff` format.

Usage

```
img_read(path, ...)
```

Arguments

`path`	Path to the image file.
`...`	Additional parameters that are passed to `read.bitmap` and the underlying image reader packages.

Details

For details, see the `read.bitmap` documentation.

Value

Objects returned by `read.bmp`, `readJPEG`, `readPNG`, or `readTIFF`. See their documentation for details.

See Also

`read.bitmap`, `read.bmp`, `readJPEG`, `readPNG`, `readTIFF`

Examples

```
## Example image with high vertical symmetry: rails
rails <- img_read(system.file("example_images", "rails.jpg", package = "imagefluency"))
```

`img_self_similarity` *Image self-similarity*

Description

`img_self_similarity` returns the self-similarity of an image (i.e., the degree to which the log-log power spectrum of the image falls with a slope of -2). Higher values indicate higher image self-similarity.

Usage

```
img_self_similarity(img, full = FALSE, logplot = FALSE, raw = FALSE)
```

Arguments

`img`	An image in form of a matrix or array of numeric values, preferably by square size. If the input is not square, bilinear resizing to a square size is performed using the `OpenImageR` package. Use e.g. `img_read()` to read an image file into R.
`full`	logical. Should the full frequency range be used for interpolation? (default: `FALSE`)
`logplot`	logical. Should the log-log power spectrum of the image be plotted? (default: `FALSE`)
`raw`	logical. Should the raw value of the regression slope be returned? (default: `FALSE`)

Details

The function takes a (square) array or matrix of numeric or integer values representing an image as input and returns the self-similarity of the image. Self-similarity is computed via the slope of the log-log power spectrum using OLS. A slope near `-2` indicates fractal-like properties (see Redies et al., 2007; Simoncelli & Olshausen, 2001). Thus, value for self-similarity that is return by the function calculated as `self-similarity = abs(slope + 2) * (-1)`. That is, the measure reaches its maximum value of 0 for a slope of -2, and any deviation from -2 results in negative values that are more negative the higher the deviation from -2. For color images, the weighted average between each color channel's values is computed (cf. Mayer & Landwehr 2018).

Per default, only the frequency range betwen 10 and 256 cycles per image is used for interpolation. Computation for the full range can be set via the parameter `full = TRUE`.

If `logplot` is set to `TRUE` then a log-log plot of the power spectrum is additionally shown. If the package `ggplot2` is installed the plot includes the slope of the OLS regression. Note that this option is currently implemented for grayscale images.

It is possible to get the raw regression slope (instead of the transformed value which indicates self-similarity) by using the option `raw = TRUE`.

For color images, the weighed average between each color channel's values is computed.

Value

a numeric value (self-similarity)

Note

The function inspired by Matlab's sfPlot (by Diederick C. Niehorster).

References

Mayer, S. & Landwehr, J, R. (2018). Quantifying Visual Aesthetics Based on Processing Fluency Theory: Four Algorithmic Measures for Antecedents of Aesthetic Preferences. *Psychology of Aesthetics, Creativity, and the Arts*, *12*(4), 399–431. doi:10.1037/aca0000187

Redies, C., Hasenstein, J., & Denzler, J. (2007). Fractal-like image statistics in visual art: Similarity to natural scenes. *Spatial Vision*, *21*, 137–148. doi:10.1163/156856807782753921

Simoncelli, E. P., & Olshausen, B. A. (2001). Natural image statistics and neural representation. *Annual Review of Neuroscience*, *24*, 1193–1216. doi:10.1146/annurev.neuro.24.1.1193

See Also

img_read, img_contrast, img_complexity, img_simplicity, img_symmetry, img_typicality,

Examples

```
# Example image with high self-similarity: romanesco
romanesco <- img_read(system.file("example_images", "romanesco.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(romanesco)
#
# get self-similarity
img_self_similarity(romanesco)

# Example image with low self-similarity: office
office <- img_read(system.file("example_images", "office.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(office)
#
# get self-similarity
img_self_similarity(office)
```

`img_simplicity` *Image simplicity*

Description

`img_simplicity` returns the simplicity of an image as 1 minus the complexity of the image. Higher values indicated higher image simplicity.

Usage

```
img_simplicity(imgfile, algorithm = "zip", rotate = FALSE)
```

Arguments

`imgfile`	Either a character string containing the path to the image file (or URL) or an an image in form of a matrix (grayscale image) or array (color image) of numeric values representing the pre-loaded image (e.g. by using `img_read()`).
`algorithm`	Character string that specifies which image compression algorithm to use. Currently implemented are `zip` with deflate compression, `jpg`, `gif`, and `png`.
`rotate`	logical. Should the compressed file size of the rotated image also be computed? (see details)

Details

Image simplicity is calculated as 1 minus the ratio between the compressed and uncompressed file size (i.e., the compression rate). Values can range between 0 (no compression possible, thus extremely complex image) and almost 1 (virtually completely compressed image, thus extremely simple image). Different compression algorithms are implemented. For details, see `img_complexity`.

Value

a numeric value: 1 minus the ratio of compressed divided by uncompressed file size (i.e., the compression rate)

References

Donderi, D. C. (2006). Visual complexity: A Review. *Psychological Bulletin*, *132*, 73–97. doi:10.1037/00332909.132.1.73

Forsythe, A., Nadal, M., Sheehy, N., Cela-Conde, C. J., & Sawey, M. (2011). Predicting Beauty: Fractal Dimension and Visual Complexity in Art. *British Journal of Psychology*, *102*, 49–70. doi:10.1348/000712610X498958

Mayer, S. & Landwehr, J, R. (2018). Quantifying Visual Aesthetics Based on Processing Fluency Theory: Four Algorithmic Measures for Antecedents of Aesthetic Preferences. *Psychology of Aesthetics, Creativity, and the Arts*, *12*(4), 399–431. doi:10.1037/aca0000187

See Also

img_read, img_complexity, img_contrast, img_self_similarity, img_symmetry, img_typicality,

Examples

```
# Example image with low simplicity: trees
trees <- img_read(system.file("example_images", "trees.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(trees)
#
# get simplicity
img_simplicity(trees)

# Example image with high simplicity: sky
sky <- img_read(system.file("example_images", "sky.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(sky)
#
# get simplicity
img_simplicity(sky)
```

---

img_symmetry *Image symmetry*

Description

img_symmetry returns the vertical and horizontal mirror symmetry of an image. Higher values indicate higher image symmetry.

Usage

```
img_symmetry(img, vertical = TRUE, horizontal = TRUE, ...)
```

Arguments

img	An image in form of a matrix or array of numeric values. Use e.g. img_read() to read an image file into R.
vertical	logical. Should the vertical symmetry be computed? (default: TRUE)
horizontal	logical. Should the horizontal symmetry be computed? (default: TRUE)
...	Further options: shift_range to shift the mirror axis, per_channel to switch between a maximal per channel vs. per image symmetry (see details).

Details

The function returns the vertical and horizontal mirror symmetry of an image `img`. Symmetry values can range between 0 (not symmetrical) and 1 (fully symmetrical). If `vertical` or `horizontal` is set to `FALSE` then vertical or horizontal symmetry is not computed, respectively.

As the perceptual mirror axis is not necessarily exactly in the middle of a picture, the function estimates in a first step several symmetry values with different positions for the mirror axis. To this end, the mirror axis is automatically shifted up to 5% (default) of the image width to the left and to the right (in the case of vertical symmetry; analogously for horizontal symmetry). In the second step, the overall symmetry score is computed as the maximum of the symmetry scores given the different mirror axes. See Mayer & Landwehr (2018) for details.

Advanced users can change the shift range with the optional parameter `shift_range`, which takes a numeric decimal as input. The default `shift_range = 0.05` (i.e., 5%).

For color images, the default is that first a maximal symmetry score (as explained above) is obtained per color channel (parameter `per_channel = TRUE`). Subsequently, a weighted average between each color channel's maximal score is computed as the image's overall symmetry. Advanced users can reverse this order by setting `per_channel = FALSE`. This results in first computing the weighted averages for each position of the mirror axis separately, and afterwards finding the maximal overall symmetry score.

Value

a named vector of numeric values (vertical and horizontal symmetry)

References

Mayer, S. & Landwehr, J, R. (2018). Quantifying Visual Aesthetics Based on Processing Fluency Theory: Four Algorithmic Measures for Antecedents of Aesthetic Preferences. *Psychology of Aesthetics, Creativity, and the Arts*, *12*(4), 399–431. doi:10.1037/aca0000187

See Also

`img_read`, `img_complexity`, `img_contrast`, `img_self_similarity` `img_simplicity`, `img_typicality`

Examples

```
# Example image with high vertical symmetry: rails
rails <- img_read(system.file("example_images", "rails.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(rails)
#
# get symmetry
img_symmetry(rails)

# Example image with low vertical symmetry: bridge
bridge <- img_read(system.file("example_images", "bridge.jpg", package = "imagefluency"))
#
# display image
grid::grid.raster(bridge)
```

```
#
# get symmetry
img_symmetry(bridge)
```

`img_typicality` *Typicality of images relative to each other*

Description

`img_typicality` returns the visual typicality of a list of images relative to each other. Higher values indicate larger typicality.

Usage

```
img_typicality(imglist, rescale = NULL)
```

Arguments

`imglist`	A *list* of arrays or matrices with numeric values. Use e.g. `img_read()` to read image files into R (see example).
`rescale`	numeric. Rescales the images prior to computing the typicality scores (per default no rescaling is performed). Rescaling is performed by `OpenImageR`'s `resizeImage` function (bilinear rescaling)

Details

The function returns the visual typicality of a *list* of image arrays or matrices `imglist` relative to each other. Values can range between -1 (inversely typical) over 0 (not typical) to 1 (perfectly typical). That is, higher absolute values indicate a larger typicality.

The typicality score is computed as the correlation of a particular image with the average representation of all images, i.e. the mean of all images. For color images, the weighted average between each color channel's values is computed. If the images have different dimensions they are automatically resized to the smallest height and width.

Rescaling of the images prior to computing the typicality scores can be specified with the optional rescaling parameter (must be a numeric value). Most users won't need any rescaling and can use the default (`rescale = NULL`). See Mayer & Landwehr (2018) for more details.

Value

a named matrix of numeric values (typicality scores)

References

Mayer, S. & Landwehr, J. R. (2018). Objective measures of design typicality. *Design Studies*, *54*, 146–161. doi:10.1016/j.destud.2017.09.004

See Also

img_read, img_contrast, img_complexity, img_self_similarity img_simplicity, img_symmetry

Examples

```
# Example images depicting valleys: valley_green, valley_white
# Example image depicting fireworks: fireworks
valley_green <- img_read(
    system.file("example_images", "valley_green.jpg", package = "imagefluency")
  )
valley_white <- img_read(
    system.file("example_images", "valley_white.jpg", package = "imagefluency")
  )
fireworks <- img_read(
    system.file("example_images", "fireworks.jpg", package = "imagefluency")
  )
#
# display images
grid::grid.raster(valley_green)
grid::grid.raster(valley_white)
grid::grid.raster(fireworks)

# create image set as list
imglist <- list(fireworks, valley_green, valley_white)

# get typicality
img_typicality(imglist)
```

---

rgb2gray *RGB to Gray Conversion*

Description

rgb2gray transforms colors from RGB space (red/green/blue) into an matrix of grayscale values.

Usage

```
rgb2gray(img)
```

Arguments

img	3-dimensional array of numeric or integer values

Details

The function takes a 3-dimensional array of numeric or integer values as input (img) and returns a matrix of grayscale values as output. The grayscale values are computed as GRAY = 0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE. If the array has a fourth dimension (i.e., alpha channel), the fourth dimension is ignored.

Value

A matrix of grayscale values.

Examples

```
# construct a sample RGB image as array of random integers
imgRed <- matrix(runif(100, min = 0, max = 255), 10, 10)
imgGreen <- matrix(runif(100, min = 0, max = 255), 10, 10)
imgBlue <- matrix(runif(100, min = 0, max = 255), 10, 10)
imgColor <- array(c(imgRed, imgGreen, imgBlue), dim = c(10, 10, 3))

# convert to gray
img <- rgb2gray(imgColor)
```

`rotate90` *Matrix or Array Rotation by 90 Degrees*

Description

Matrix or Array Rotation by 90 Degrees

Usage

```
rotate90(img, direction = "positive")
```

Arguments

`img`	an array or a matrix
`direction`	The direction of rotation by 90 degrees. The value can be `"positive"` (default) or `"negative"`. Aliases are `"counterclockwise"` and `"clockwise"`, respectively.

Details

The function takes an array or matrix as input object (`img`) and returns the object rotated by 90 degrees. Per default, the rotation is done in the mathematically positive direction (i.e., counterclockwise). Clockwise rotation (i.e., mathematically negative) can be specified by passing the value `"negative"` to the `direction` argument.

Value

an array or a matrix (rotated by 90 degrees)

Examples

```
# sample matrix
img <- matrix(1:6, ncol = 2)
img

rotate90(img) # counterclockwise
rotate90(img, direction = "negative") # clockwise
```

`run_imagefluency` *Run imagefluency app*

Description

Launches a Shiny app that shows a demo of what can be done with the `imagefluency` package.

Usage

```
run_imagefluency()
```

Examples

```
## Only run this example in interactive R sessions
if (interactive()) {
  run_imagefluency()
}
```

Index