Package: pliman (via r-universe)

October 8, 2026

Title Tools for Plant Image Analysis

Version 3.2.0

Description Tools for both single and batch image manipulation and analysis (Olivoto, 2022 <doi:10.1111/2041-210X.13803>) and phytopathometry (Olivoto et al., 2022 <doi:10.1007/S40858-021-00487-5>). The tools can be used for the quantification of leaf area, object counting, extraction of image indexes, shape measurement, object landmark identification, and Elliptical Fourier Analysis of object outlines (Claude (2008) <doi:10.1007/978-0-387-77789-4>). The package also provides a comprehensive pipeline for generating shapefiles with complex layouts and supports high-throughput phenotyping of RGB, multispectral, and hyperspectral orthomosaics. This functionality facilitates field phenotyping using UAV- or satellite-based imagery.

License GPL (>= 3)

URL https://nepem-ufsc.github.io/pliman/, https://github.com/nepem-ufsc/pliman

BugReports https://github.com/nepem-ufsc/pliman/issues

Depends R (>= 4.1)

Imports cli, dplyr, graphics, grDevices, methods, mirai, Rcpp, sf, stats, terra, utils

Suggests av, BiocManager, curl, fields, jsonlite, knitr, leafem (>= 0.2.0), leaflet (>= 2.1.2), mapedit (>= 0.6.0), mapview (>= 2.11.0), pak, pkgload, rmarkdown, rprojroot, rstudioapi, tidyr, tiff, webp

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev

Repository https://nepem-ufsc.r-universe.dev

Date/Publication 2026-10-08 15:59:03 UTC

RemoteUrl https://github.com/nepem-ufsc/pliman

RemoteRef HEAD

RemoteSha d1dd7c40ba8c85157f6925eda750021440de6fde

Contents

[.image . . . 7
align_stack_cpp . . . 8
analyze_objects . . . 9
analyze_objects_minimal . . . 19
analyze_objects_shp . . . 22
apply_fun_to_imgs . . . 25
as.numeric.image . . . 27
as.raster.image . . . 27
c.yolo_fewshot_model . . . 28
calibrate . . . 28
ccc . . . 29
clear_pliman_cache . . . 31
colorchecker_ref . . . 32
contour . . . 32
contours . . . 33
cpp_extract_raster . . . 34
custom_palette . . . 35
dilate_cpp . . . 35
dist_transform . . . 36
dl_extract_phenotypes . . . 37
efourier . . . 38
efourier_coefs . . . 40
efourier_error . . . 41
efourier_inv . . . 42
efourier_norm . . . 43
efourier_power . . . 44
efourier_shape . . . 45
ellipse . . . 47
entropy . . . 48
erode_cpp . . . 49
erode_external_cpp . . . 49
erosao_conservativa . . . 50
fill_holes_cpp . . . 50
focus_stack . . . 51
fuse_laplacian_cpp . . . 56

get_card_colors . . . 56
get_free_ram . . . 59
get_free_ram_cpp . . . 59
get_pliman_viewer . . . 60
get_uuid . . . 60
ggplot_color . . . 61
gpa . . . 61
hist.image . . . 62
image . . . 63
image_align . . . 64
image_alpha . . . 65
image_augment . . . 66
image_binary . . . 67
image_binary_dl . . . 70
image_bwlabel . . . 75
image_canny_edge . . . 75
image_classify_dl . . . 76
image_color_labels . . . 78
image_combine . . . 78
image_contour_line . . . 79
image_correction . . . 80
image_correction_pick . . . 82
image_create . . . 83
image_data . . . 84
image_depth_dl . . . 85
image_detect_dl . . . 86
image_detect_sahi . . . 88
image_detect_world . . . 90
image_embed_dl . . . 92
image_expand . . . 93
image_features_dl . . . 94
image_generate_dl . . . 96
image_index . . . 97
image_inpaint_dl . . . 100
image_label . . . 101
image_line_segment . . . 102
image_pose_dl . . . 103
image_prepare . . . 106
image_remove_bg_dl . . . 107
image_remove_reflection . . . 112
image_resize . . . 112
image_segment . . . 114
image_segment_dl . . . 117
image_segment_kmeans . . . 125
image_segment_manual . . . 127
image_segment_mask . . . 128
image_segment_sahi . . . 129
image_shp . . . 131

image_skeleton_mat . . . 132
image_square . . . 134
image_stardist_dl . . . 135
image_stitch . . . 136
image_superres_dl . . . 138
image_thinning_guo_hall . . . 139
image_to_mat . . . 140
image_transpose . . . 141
image_view . . . 141
image_watershed . . . 143
landmarks . . . 144
landmarks_add . . . 145
landmarks_angle . . . 146
landmarks_dist . . . 147
landmarks_regradi . . . 148
leading_zeros . . . 149
line_on_halfplot . . . 150
list_cameras . . . 151
make_brush . . . 151
make_mask . . . 152
mean.image . . . 153
measure_disease . . . 153
measure_disease_byl . . . 160
measure_disease_shp . . . 163
measure_injury . . . 165
model_to_int8 . . . 166
mosaic_aggregate . . . 168
mosaic_analyze . . . 169
mosaic_analyze_iter . . . 175
mosaic_chm . . . 177
mosaic_chm_extract . . . 178
mosaic_chm_mask . . . 179
mosaic_classify . . . 180
mosaic_clip . . . 181
mosaic_crop . . . 182
mosaic_draw . . . 184
mosaic_epsg . . . 187
mosaic_extract . . . 188
mosaic_hist . . . 190
mosaic_index . . . 190
mosaic_index2 . . . 192
mosaic_input . . . 193
mosaic_interpolate . . . 195
mosaic_lonlat2epsg . . . 196
mosaic_plot . . . 196
mosaic_plot_rgb . . . 197
mosaic_prepare . . . 198
mosaic_project . . . 199

mosaic_resample . . . 200
mosaic_rotate . . . 201
mosaic_segment . . . 202
mosaic_segment_pick . . . 203
mosaic_to_pliman . . . 204
mosaic_to_rgb . . . 205
mosaic_vectorize . . . 206
mosaic_view . . . 208
object_bbox . . . 210
object_edge . . . 211
object_export . . . 212
object_export_shp . . . 214
object_haralick . . . 216
object_label . . . 219
object_map . . . 221
object_mark . . . 222
object_rgb . . . 223
object_scatter . . . 224
object_split . . . 226
object_split_shp . . . 227
object_to_color . . . 229
object_veins . . . 230
onnx_install . . . 232
onnx_quantize . . . 232
Ops.image . . . 233
order_quad_corners . . . 234
otsu . . . 234
palettes . . . 235
pipe . . . 237
pixel_index . . . 238
pliman_available_models . . . 239
pliman_clear_sessions . . . 239
pliman_configure_dl . . . 240
pliman_download_model . . . 241
pliman_download_onnx . . . 242
pliman_gpu_info . . . 243
pliman_images . . . 243
pliman_indexes_ican_compute . . . 245
pliman_model_dir . . . 245
pliman_models_base_url . . . 246
pliman_onnx_dir . . . 246
pliman_onnx_lib_path . . . 247
pliman_viewer . . . 247
plot.image . . . 247
plot.image_shp . . . 249
plot.root_analysis . . . 250
plot.yolo_fewshot_model . . . 251
plot.yolo_train . . . 253

plot_bbox . . . 256
plot_id . . . 257
plot_index . . . 258
plot_index_shp . . . 260
plot_line_segment . . . 262
plot_lw . . . 262
plot_yolo_heatmap . . . 263
poly_apex_base_angle . . . 265
poly_pcv . . . 266
poly_width_at . . . 267
predict_binary_glm . . . 268
prepare_to_shp . . . 269
print.image . . . 269
print.pliman_video_frames . . . 270
print.root_analysis . . . 270
print.yolo_train . . . 271
random_color . . . 271
remove_reflection_cpp . . . 272
rgb_image . . . 272
root_analyze . . . 273
sad . . . 277
sentinel_to_tif . . . 279
separate_col . . . 279
set_pliman_viewer . . . 280
shapefile_build . . . 280
shapefile_edit . . . 284
shapefile_interpolate . . . 285
shapefile_measures . . . 286
shapefile_operations . . . 287
shapefile_plot . . . 288
shapefile_surface . . . 289
shapefile_transform . . . 290
Summary.image . . . 291
summary_index . . . 292
text_embed_dl . . . 293
to_numeric . . . 294
utils_colorspace . . . 295
utils_dpi . . . 296
utils_file . . . 298
utils_image . . . 300
utils_indexes . . . 302
utils_measures . . . 303
utils_neural . . . 305
utils_objects . . . 308
utils_pca . . . 311
utils_pick . . . 313
utils_polygon . . . 316
utils_polygon_plot . . . 322

utils_rows_cols . . . 324
utils_shapefile . . . 325
utils_shapes . . . 326
utils_stats . . . 329
utils_transform . . . 329
utils_wd . . . 337
uuid . . . 338
video_detect_dl . . . 339
video_split . . . 350
view . . . 352
view.image . . . 352
yolo_dataset_add_background . . . 353
yolo_dataset_audit . . . 355
yolo_dataset_augment . . . 356
yolo_dataset_autolabel . . . 358
yolo_dataset_balance . . . 359
yolo_dataset_classify . . . 360
yolo_dataset_convert . . . 363
yolo_dataset_export . . . 364
yolo_dataset_filter . . . 369
yolo_dataset_from_ndjson . . . 370
yolo_dataset_merge . . . 372
yolo_dataset_preview . . . 373
yolo_dataset_split . . . 375
yolo_dataset_stats . . . 376
yolo_export . . . 376
yolo_fewshot_fit . . . 377
yolo_fewshot_predict . . . 379
yolo_plot . . . 380
yolo_results . . . 381
yolo_train . . . 383

Index **388**

`[.image` *Subsetting method for* `image` *objects*

Description

Subsetting method for `image` objects

Usage

```
## S3 method for class 'image'
x[i, j, k, ..., drop = FALSE]
```

Arguments

x	An image object.
i	Row/X index or missing.
j	Column/Y index or missing.
k	Channel index or missing.
...	Additional arguments.
drop	Logical; whether to drop dimensions.

`align_stack_cpp` *Native C++ Sub-Pixel Image Stack Alignment (ECC Maximization)*

Description

High-performance C++ implementation of Enhanced Correlation Coefficient (ECC) maximization with multi-scale Gaussian pyramids for sub-pixel image alignment.

Usage

```
align_stack_cpp(
  images,
  ref_idx = 0L,
  method = "ecc",
  crop = TRUE,
  levels = 3L,
  max_iter = 50L,
  eps = 1e-04,
  verbose = TRUE
)
```

Arguments

images	List of Image objects or arrays.
ref_idx	0-indexed reference image index.
method	Character: "ecc" (affine) or "translation".
crop	Logical: automatically crop empty affine borders.
levels	Integer: number of pyramid levels (default: 3).
max_iter	Integer: max iterations per level (default: 50).
eps	Numeric: convergence tolerance (default: 1e-4).
verbose	Logical: print progress.

Value

A named list containing aligned, matrices, statuses, and crop_box.

`analyze_objects` *Analyzes objects in an image*

Description

- `analyze_objects()` provides tools for counting and extracting object features (e.g., area, perimeter, radius, pixel intensity, Haralick texture, veins, and Fourier descriptors) in an image using an ultra-fast, cache-optimized C++ computer vision engine. See more at the **Details** section.
- `analyze_objects_iter()` provides an iterative section to measure object features using an object with a known area.
- `plot.anal_obj()` produces a histogram or density plot for object measures or R, G, and B values when argument `object_index` is used in `analyze_objects()`.
- `analyze_objects2()` is maintained as an exact alias for `analyze_objects()`.

Usage

```
analyze_objects(
  img,
  foreground = NULL,
  background = NULL,
  pick_palettes = FALSE,
  segment_objects = TRUE,
  viewer = get_pliman_viewer(),
  reference = FALSE,
  reference_area = NULL,
  back_fore_index = "R/(G/B)",
  fore_ref_index = "B-R",
  reference_img = NULL,
  reference_larger = FALSE,
  reference_smaller = FALSE,
  pattern = NULL,
  parallel = FALSE,
  workers = NULL,
  watershed = TRUE,
  veins = FALSE,
  sigma_veins = 1,
  veins_thinning = FALSE,
  veins_rel_erode = 0.005,
  ab_angles = FALSE,
  ab_angles_percentiles = c(0.25, 0.75),
  width_at = FALSE,
  width_at_percentiles = c(0.05, 0.25, 0.5, 0.75, 0.95),
  haralick = FALSE,
  har_nbins = 32,
  har_scales = 1,
```

```
har_band = 1,
smooth = FALSE,
pcv = FALSE,
pcv_niter = 100,
resize = FALSE,
trim = FALSE,
fill_hull = FALSE,
erode = FALSE,
dilate = FALSE,
opening = FALSE,
closing = FALSE,
filter = FALSE,
return_exact = FALSE,
filter_order = c("erode", "dilate", "opening", "closing", "filter", "fill_hull"),
invert = FALSE,
object_size = "medium",
sensitivity = "medium",
area_mode = c("contour", "pixel"),
index = "NB",
r = 1,
g = 2,
b = 3,
re = 4,
nir = 5,
swir = 6,
object_index = NULL,
pixel_level_index = FALSE,
return_mask = FALSE,
efourier = FALSE,
nharm = 10,
threshold = "Otsu",
k = 0.1,
windowsize = NULL,
tolerance = NULL,
extension = NULL,
lower_noise = 0.1,
rel_size = NULL,
lower_size = NULL,
upper_size = NULL,
topn_lower = NULL,
topn_upper = NULL,
lower_eccent = NULL,
upper_eccent = NULL,
lower_circ = NULL,
upper_circ = NULL,
randomize = TRUE,
nrows = 1000,
plot = TRUE,
```

```
  show_original = TRUE,
  show_chull = FALSE,
  show_contour = FALSE,
  show_bbox = FALSE,
  contour_col = "red",
  contour_size = 1,
  show_lw = FALSE,
  show_background = TRUE,
  show_segmentation = FALSE,
  col_foreground = NULL,
  col_background = NULL,
  marker = "point",
  marker_col = "red",
  marker_size = NULL,
  save_image = FALSE,
  max_pixels = 2e+06,
  prefix = "proc_",
  dir_original = NULL,
  dir_processed = NULL,
  verbose = TRUE
)

analyze_objects2(
  img,
  foreground = NULL,
  background = NULL,
  pick_palettes = FALSE,
  segment_objects = TRUE,
  viewer = get_pliman_viewer(),
  reference = FALSE,
  reference_area = NULL,
  back_fore_index = "R/(G/B)",
  fore_ref_index = "B-R",
  reference_img = NULL,
  reference_larger = FALSE,
  reference_smaller = FALSE,
  pattern = NULL,
  parallel = FALSE,
  workers = NULL,
  watershed = TRUE,
  veins = FALSE,
  sigma_veins = 1,
  veins_thinning = FALSE,
  veins_rel_erode = 0.005,
  ab_angles = FALSE,
  ab_angles_percentiles = c(0.25, 0.75),
  width_at = FALSE,
  width_at_percentiles = c(0.05, 0.25, 0.5, 0.75, 0.95),
```

```
  haralick = FALSE,
  har_nbins = 32,
  har_scales = 1,
  har_band = 1,
  smooth = FALSE,
  pcv = FALSE,
  pcv_niter = 100,
  resize = FALSE,
  trim = FALSE,
  fill_hull = FALSE,
  erode = FALSE,
  dilate = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  return_exact = FALSE,
  filter_order = c("erode", "dilate", "opening", "closing", "filter", "fill_hull"),
  invert = FALSE,
  object_size = "medium",
  sensitivity = "medium",
  area_mode = c("contour", "pixel"),
  index = "NB",
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  swir = 6,
  object_index = NULL,
  pixel_level_index = FALSE,
  return_mask = FALSE,
  efourier = FALSE,
  nharm = 10,
  threshold = "Otsu",
  k = 0.1,
  windowsize = NULL,
  tolerance = NULL,
  extension = NULL,
  lower_noise = 0.1,
  rel_size = NULL,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  lower_eccent = NULL,
  upper_eccent = NULL,
  lower_circ = NULL,
  upper_circ = NULL,
```

```
  randomize = TRUE,
  nrows = 1000,
  plot = TRUE,
  show_original = TRUE,
  show_chull = FALSE,
  show_contour = FALSE,
  show_bbox = FALSE,
  contour_col = "red",
  contour_size = 1,
  show_lw = FALSE,
  show_background = TRUE,
  show_segmentation = FALSE,
  col_foreground = NULL,
  col_background = NULL,
  marker = "point",
  marker_col = "red",
  marker_size = NULL,
  save_image = FALSE,
  max_pixels = 2e+06,
  prefix = "proc_",
  dir_original = NULL,
  dir_processed = NULL,
  verbose = TRUE
)

## S3 method for class 'anal_obj'
plot(
  x,
  which = "measure",
  measure = "area",
  type = c("density", "histogram"),
  ...
)

## S3 method for class 'anal_obj_ls'
plot(
  x,
  which = "measure",
  measure = "area",
  type = c("density", "histogram"),
  ...
)

analyze_objects_iter(pattern, known_area, verbose = TRUE, ...)
```

Arguments

`img`	The image to be analyzed.

`foreground`, `background`, `reference_img`
: A color palette for the foreground, background, and reference object, respectively (optional). If a character is used (e.g., `foreground = "fore"`), the function will search in the current working directory a valid image named "fore".

`pick_palettes`
: Logical argument indicating whether the user needs to pick up the color palettes for foreground and background for the image.

`segment_objects`
: Segment objects in the image? Defaults to `TRUE`. If `segment_objects = FALSE`, the objects are not segmented and the entire image is analyzed (useful for whole-canopy indices).

`viewer`
: The viewer option for interactive plotting.

`reference`
: Logical to indicate if a reference object is present in the image.

`reference_area`
: The known area of the reference object.

`back_fore_index`
: A character value to indicate the index to segment the foreground (objects and reference) from the background. Defaults to `"R/(G/B)"`.

`fore_ref_index`
: A character value to indicate the index to segment objects and the reference object. Defaults to `"B-R"`.

`reference_larger`, `reference_smaller`
: Logical argument indicating when the larger/smaller object in the image must be used as the reference object.

`pattern`
: A pattern of file name used to identify images to be imported.

`parallel`
: If `TRUE` processes the images asynchronously in parallel.

`workers`
: Number of parallel processes.

`watershed`
: If `TRUE` (default) performs watershed-based object detection.

`veins`
: Logical argument indicating whether vein features are computed per object.

`sigma_veins`
: Fine Gaussian scale parameter for vein detection (default `1`).

`veins_thinning`
: Logical value indicating whether Guo-Hall skeleton thinning should be applied to detected veins (default `FALSE`).

`veins_rel_erode`
: Relative boundary erosion fraction of `min(width, height)` (default `0.005`).

`ab_angles`
: Logical argument indicating whether apex and base angles should be computed.

`ab_angles_percentiles`
: The percentiles indicating the heights of the object for angles.

`width_at`
: Logical. If `TRUE`, widths at quantiles of height are computed.

`width_at_percentiles`
: A vector of heights along the vertical axis of the object.

`haralick`
: Logical value indicating whether Haralick features are computed.

`har_nbins` An integer indicating the number of bins for the Haralick matrix. Defaults to 32.

`har_scales` An integer vector indicating the number of scales for Haralick features.

`har_band` The band to compute Haralick features (1 = R, 2 = G, 3 = B, or `"GRAY"`).

`smooth` Whether the object contours should be smoothed with `poly_smooth()`.

`pcv` Computes the Perimeter Complexity Value? Defaults to `FALSE`.

`pcv_niter` An integer specifying the number of smoothing iterations for PCV.

`resize` Resize the image before processing?

`trim` Number of pixels removed from edges in the analysis.

`fill_hull` Fill holes in the binary image? Defaults to `FALSE`.

`opening`, `closing`, `filter`, `erode`, `dilate`
Morphological operations (brush size), applied in hierarchical order.

`return_exact` Logical. If `TRUE`, exact morphological reconstruction is performed.

`filter_order` Hierarchical order of morphological operations.

`invert` Inverts the binary image if desired.

`object_size` The size of the object (`"small"`, `"medium"`, `"large"`, `"elarge"`). Used to automatically set up `tolerance` and `extension` parameters.

`sensitivity` Sensitivity level for thresholding (`"low"`, `"medium"`, `"high"`, `"extreme"`, or numeric). Defaults to `"medium"`.

`area_mode` The method to compute object area (`"contour"` or `"pixel"`).

`index` A character value specifying the index for conversion to binary image.

`r`, `g`, `b`, `re`, `nir` The red, green, blue, red-edge, and near-infrared bands, respectively.

`swir` The short-wave infrared band.

`object_index` If informed, the average index value for each object is returned.

`pixel_level_index`
Return indexes computed in `object_index` at pixel level?

`return_mask` Returns the mask for the analyzed image?

`efourier` Logical argument indicating if Elliptical Fourier should be computed.

`nharm` Number of harmonics for Fourier analysis. Defaults to 10.

`threshold` The threshold method (`"Otsu"`, `"Adaptive"`, or numeric).

`k`, `windowsize` Parameters for adaptive thresholding.

`tolerance` Minimum height for watershed segmentation.

`extension` Neighborhood radius for watershed segmentation.

`lower_noise` Objects with area less than `lower_noise` * mean area are removed.

`rel_size` Relative size threshold as a fraction of mean object area.

`lower_size`, `upper_size`
Lower and upper size limits for object filtering.

`topn_lower`, `topn_upper`
Select top `n` objects based on area.

`lower_eccent`, `upper_eccent`, `lower_circ`, `upper_circ`
: Eccentricity and circularity limits.

`randomize`
: Randomize rows before training GLM model?

`nrows`
: Number of rows to sample for GLM training.

`plot`
: Show image after processing?

`show_original`
: Show count objects on the original image?

`show_chull`
: Show convex hull around objects?

`show_contour`
: Show contour line around objects?

`show_bbox`
: Show bounding box around objects?

`contour_col`, `contour_size`
: Color and size for contour line.

`show_lw`
: Plot length and width lines on each object?

`show_background`
: Show the background?

`show_segmentation`
: Shows object segmentation colored with random permutations?

`col_foreground`, `col_background`
: Foreground and background colors.

`marker`, `marker_col`, `marker_size`
: Type, color, and size of object marker.

`save_image`
: Save the image after processing?

`max_pixels`
: Maximum number of pixels to render when plotting or saving.

`prefix`
: Prefix for processed images.

`dir_original`, `dir_processed`
: Directories for original and processed images.

`verbose`
: If `TRUE` (default) shows progress summary in console.

`x`
: An object of class `anal_obj`.

`which`
: Which to plot. Either 'measure' (object measures) or 'index' (object index). Defaults to `"measure"`.

`measure`
: The measure to plot. Defaults to `"area"`.

`type`
: The type of plot. Either `"hist"` or `"density"`. Partial matches are recognized.

`...`
: Additional arguments passed to methods.

`known_area`
: Known area of template object in `analyze_objects_iter()`.

Details

A binary image is first generated to segment the foreground and background. The argument `index` is useful to choose a proper index to segment the image (see `image_binary()` for more details). It is also possible to provide color palettes for background and foreground (arguments `background` and `foreground`, respectively). When this is used, a general linear model (binomial family) is fitted to the RGB values to segment fore- and background.

Then, the number of objects in the foreground is counted. By setting up arguments such as `lower_size` and `upper_size`, it is possible to set a threshold for lower and upper sizes of the objects, respectively. The argument `object_size` or `sensitivity` can be used to set up pre-defined values of `tolerance` and `extension` depending on the image resolution. This will influence the watershed-based object segmentation. Users can also tune up `tolerance` and `extension` explicitly for a better precision of watershed segmentation.

If `watershed = FALSE` is used, all pixels for each connected set of foreground pixels in `img` are set to a unique object. This is faster, especially for a large number of objects, but it is not able to segment touching objects.

There are some ways to correct the measures based on a reference object. If a reference object with a known area (`reference_area`) is used in the image and `reference = TRUE` is used, the measures of the objects will be corrected, considering the unit of measure informed in `reference_area`. There are two main ways to work with reference objects: * The first is to provide a reference object that has a contrasting color with both the background and object of interest. In this case, the arguments `back_fore_index` and `fore_ref_index` can be used to define an index to first segment the reference object and objects to be measured from the background, then the reference object from objects to be measured.

`* The second one is to use a reference object that has a similar color to the`

objects to be measured, but has a contrasting size. For example, if we are counting small brown grains, we can use a brown reference template that has an area larger (says 3 times the area of the grains) and then uses `reference_larger = TRUE`. With this, the larger object in the image will be used as the reference object. This is particularly useful when images are captured with background light. The argument `reference_smaller` can be used in the same way.

By using `pattern`, it is possible to process several images with common pattern names that are stored in the current working directory or in the subdirectory informed in `dir_original`. To speed up the computation time, one can set `parallel = TRUE`.

`analyze_objects_iter()` can be used to process several images using an object with a known area as a template. In this case, all the images in the current working directory that match the `pattern` will be processed.

- Additional measures:
  - If `haralick = TRUE`, the function computes 13 Haralick texture features for each object based on a gray-level co-occurrence matrix (Haralick et al. 1973).
  - If `efourier = TRUE` is used, an Elliptical Fourier Analysis (Kuhl and Giardina, 1982) is computed for each object contour using `efourier()`.
  - If `veins = TRUE`, vein features are computed for each object using Difference of Gaussians (DoG) bandpass filtering.
  - If `ab_angles = TRUE`, the apex and base angles of each object are computed with `poly_apex_base_angle()`.
  - If `width_at = TRUE`, the widths at given percentiles of object height are computed.
  - If `pcv = TRUE`, the Perimeter Complexity Value is computed.

Value

An object of class `anal_obj` (single image) or `anal_obj_ls` (batch).

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Haralick, R.M., K. Shanmugam, and I. Dinstein. 1973. Textural Features for Image Classification. IEEE Transactions on Systems, Man, and Cybernetics SMC-3(6): 610–621. doi:10.1109/TSMC.1973.4309314

Kuhl, F. P., and Giardina, C. R. (1982). Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18, 236-258.

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
obj <- analyze_objects(img)
obj$statistics

# Top-5 grains with largest area
top <- analyze_objects(img, marker = "id", topn_upper = 5)
top$results

# Reference object correction
img <- image_pliman("flax_grains.jpg")
res <- analyze_objects(img,
                       index = "GRAY",
                       reference = TRUE,
                       reference_area = 6,
                       reference_larger = TRUE,
                       lower_noise = 0.3)
}

if (interactive()) {
library(pliman)

img <- image_pliman("soy_green.jpg")
rgb <- analyze_objects(img,
                       marker = "id",
                       object_index = "B",
                       pixel_level_index = TRUE)
# density of area
plot(rgb)

# histogram of perimeter
plot(rgb, measure = "perimeter", type = "histogram")

# density of the blue (B) index
plot(rgb, which = "index")
}
```

`analyze_objects_minimal` *Analyzes objects in an image*

Description

A lighter option to `analyze_objects()`

Usage

```
analyze_objects_minimal(
  img,
  segment_objects = TRUE,
  reference = FALSE,
  reference_area = NULL,
  back_fore_index = "R/(G/B)",
  fore_ref_index = "B-R",
  reference_larger = FALSE,
  reference_smaller = FALSE,
  pattern = NULL,
  parallel = FALSE,
  workers = NULL,
  watershed = TRUE,
  fill_hull = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  invert = FALSE,
  object_size = "medium",
  index = "NB",
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  threshold = "Otsu",
  tolerance = NULL,
  extension = NULL,
  lower_noise = 0.1,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  lower_eccent = NULL,
  upper_eccent = NULL,
```

```
  lower_circ = NULL,
  upper_circ = NULL,
  plot = TRUE,
  show_original = TRUE,
  show_contour = TRUE,
  contour_col = "red",
  contour_size = 1,
  col_foreground = NULL,
  col_background = NULL,
  marker = FALSE,
  marker_col = NULL,
  marker_size = NULL,
  save_image = FALSE,
  max_pixels = 4e+06,
  prefix = "proc_",
  dir_original = NULL,
  dir_processed = NULL,
  verbose = TRUE
)

## S3 method for class 'anal_obj_minimal'
plot(
  x,
  which = "measure",
  measure = "area",
  type = c("density", "histogram"),
  ...
)

## S3 method for class 'anal_obj_ls_minimal'
plot(
  x,
  which = "measure",
  measure = "area",
  type = c("density", "histogram"),
  ...
)
```

Arguments

`img`	The image to be analyzed.
`segment_objects`	Segment objects in the image? Defaults to `TRUE`. If `segment_objects = FALSE`, the objects are not segmented and the entire image is analyzed (useful for whole-canopy indices).
`reference`	Logical to indicate if a reference object is present in the image.
`reference_area`	The known area of the reference object.

`back_fore_index`
: A character value to indicate the index to segment the foreground (objects and reference) from the background. Defaults to `"R/(G/B)"`.

`fore_ref_index`
: A character value to indicate the index to segment objects and the reference object. Defaults to `"B-R"`.

`reference_larger`, `reference_smaller`
: Logical argument indicating when the larger/smaller object in the image must be used as the reference object.

`pattern`
: A pattern of file name used to identify images to be imported.

`parallel`
: If `TRUE` processes the images asynchronously in parallel.

`workers`
: Number of parallel processes.

`watershed`
: If `TRUE` (default) performs watershed-based object detection.

`fill_hull`
: Fill holes in the binary image? Defaults to `FALSE`.

`opening`, `closing`, `filter`, `erode`, `dilate`
: **Morphological operations (brush size)**, applied in hierarchical order.

`invert`
: Inverts the binary image if desired.

`object_size`
: The size of the object (`"small"`, `"medium"`, `"large"`, `"elarge"`). Used to automatically set up `tolerance` and `extension` parameters.

`index`
: A character value specifying the index for conversion to binary image.

`r`, `g`, `b`, `re`, `nir`
: The red, green, blue, red-edge, and near-infrared bands, respectively.

`threshold`
: The threshold method (`"Otsu"`, `"Adaptive"`, or numeric).

`tolerance`
: Minimum height for watershed segmentation.

`extension`
: Neighborhood radius for watershed segmentation.

`lower_noise`
: Objects with area less than `lower_noise` * mean area are removed.

`lower_size`, `upper_size`
: Lower and upper size limits for object filtering.

`topn_lower`, `topn_upper`
: Select top `n` objects based on area.

`lower_eccent`, `upper_eccent`, `lower_circ`, `upper_circ`
: Eccentricity and circularity limits.

`plot`
: Show image after processing?

`show_original`
: Show count objects on the original image?

`show_contour`
: Show contour line around objects?

`contour_col`, `contour_size`
: Color and size for contour line.

`col_foreground`, `col_background`
: Foreground and background colors.

`marker`, `marker_col`, `marker_size`
: Type, color, and size of object marker.

`save_image`
: Save the image after processing?

`max_pixels`
: Maximum number of pixels to render when plotting or saving.

`prefix`	Prefix for processed images.
`dir_original, dir_processed`	Directories for original and processed images.
`verbose`	If `TRUE` (default) shows progress summary in console.
`x`	An object of class `anal_obj`.
`which`	Which to plot. Either 'measure' (object measures) or 'index' (object index). Defaults to `"measure"`.
`measure`	The measure to plot. Defaults to `"area"`.
`type`	The type of plot. Either `"hist"` or `"density"`. Partial matches are recognized.
`...`	Additional arguments passed to methods.

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
obj <- analyze_objects(img)
obj$statistics

}

if (interactive()) {
library(pliman)

img <- image_pliman("soy_green.jpg")
# Segment the foreground (grains) using the normalized blue index (NB, default)
# Shows the average value of the blue index in each object

rgb <- analyze_objects_minimal(img)
# density of area
plot(rgb)

# histogram of area
plot(rgb, type = "histogram") # or 'hist'
}
```

---

analyze_objects_shp *Analyzes objects using shapefiles*

Description

Analyzes objects using shapefiles

Usage

```
analyze_objects_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  prepare = FALSE,
  segment_objects = TRUE,
  viewer = get_pliman_viewer(),
  index = "R",
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  shapefile = NULL,
  interactive = FALSE,
  plot = FALSE,
  parallel = FALSE,
  workers = NULL,
  watershed = TRUE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  object_size = "medium",
  efourier = FALSE,
  object_index = NULL,
  veins = FALSE,
  width_at = FALSE,
  verbose = TRUE,
  invert = FALSE,
  ...
)
```

Arguments

`img`	An `image` object
`nrow`, `ncol`	The number of rows and columns to generate the shapefile when `shapefile` is not declared. Defaults to `1`.
`buffer_x`, `buffer_y`	Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.

`prepare` Logical value indicating whether to prepare the image for analysis using `image_prepare()` function. Defaults to `FALSE`. Set to `TRUE` to interactively align and crop the image before processing.

`segment_objects` Segment objects in the image? Defaults to `TRUE`. If `segment_objects = FALSE`, the objects are not segmented and the entire image is analyzed (useful for whole-canopy indices).

`viewer` The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

`index` A character value specifying the index for conversion to binary image.

`r`, `g`, `b`, `re`, `nir` The red, green, blue, red-edge, and near-infrared bands, respectively.

`shapefile` (Optional) An object created with `image_shp()`. If `NULL` (default), both `nrow` and `ncol` must be declared.

`interactive` If `FALSE` (default) the grid is created automatically based on the image dimension and number of nrow/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.

`plot` Plots the processed images? Defaults to `FALSE`.

`parallel` If `TRUE` processes the images asynchronously in parallel.

`workers` Number of parallel processes.

`watershed` If `TRUE` (default) performs watershed-based object detection.

`opening`, `closing`, `filter`, `erode`, `dilate` **Morphological operations (brush size)**, applied in hierarchical order.

`object_size` Argument to control control the watershed segmentation. See `analyze_objects()` for more details.

`efourier` Logical argument indicating if Elliptical Fourier should be computed.

`object_index` If informed, the average index value for each object is returned.

`veins` Logical argument indicating whether vein features are computed per object.

`width_at` Logical. If `TRUE`, widths at quantiles of height are computed.

`verbose` If `TRUE` (default) shows progress summary in console.

`invert` Inverts the binary image if desired.

`...` Aditional arguments passed on to analyze_objects.

Details

The `analyze_objects_shp` function performs object analysis on an image and generates shapefiles representing the analyzed objects. The function first prepares the image for analysis using the `image_prepare()` function if the `prepare` argument is set to `TRUE`. If a shapefile object is provided, the number of rows and columns for splitting the image is obtained from the shapefile. Otherwise, the image is split into multiple sub-images based on the specified number of rows and columns using the `object_split_shp()` function. The objects in each sub-image are analyzed using the `analyze_objects()` function, and the results are stored in a list. If parallel processing is enabled, the analysis is performed in parallel using multiple workers.

The output object provides access to various components of the analysis results, such as the analyzed object coordinates and properties. Additionally, the shapefiles representing the analyzed objects are included in the output object for further analysis or visualization.

Value

An object of class `anal_obj`. See more details in the `Value` section of `analyze_objects()`.

Examples

```
if (interactive()) {
library(pliman)

# Computes the DGCI index for each flax leaf
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects_shp(flax,
                       nrow = 3,
                       ncol = 5,
                       plot = FALSE,
                       object_index = "DGCI")
plot(flax)
plot(res$shapefiles)
plot_measures(res, measure = "DGCI")
}
```

---

apply_fun_to_imgs *Apply a function to images*

Description

Most of the functions in pliman can be applied to a list of images, but this can be not ideal to deal with lots of images, mainly if they have a high resolution. For curiosity, a 6000 x 4000 image use nearly 570 Megabytes of RAM. So, it would be impossible to deal with lots of images within R. `apply_fun_to_img()` applies a function to images stored in a given directory as follows:

- Create a vector of image names that contain a given pattern of name.

- Import each image of such a list.
- Apply a function to the imported image.
- Export the mutated image to the computer.

If parallel is set to `FALSE` (default), the images are processed sequentially, which means that one image needs to be imported, processed, and exported so that the other image can be processed. If parallel is set to `TRUE`, the images are processed asynchronously (in parallel) in separate R sessions (3) running in the background on the same machine. It may speed up the processing time when lots of images need to be processed.

Usage

```
apply_fun_to_imgs(
  pattern,
  fun,
  ...,
  dir_original = NULL,
  dir_processed = NULL,
  prefix = "",
  suffix = "",
  parallel = FALSE,
  workers = 3,
  verbose = TRUE
)
```

Arguments

`pattern`	A pattern to match the images' names.
`fun`	A function to apply to the images.
`...`	Arguments passed on to `fun`.
`dir_original`, `dir_processed`	The directory containing the original and processed images. Defaults to `NULL`, which means that the current working directory will be considered. **The processed image will overwrite the original image unless a prefix/suffix be used or a subfolder is informed in dir_processed argument**.
`prefix`, `suffix`	A prefix and/or suffix to be included in the name of processed images. Defaults to `""`.
`parallel`	If `TRUE` processes the images asynchronously (in parallel) in separate R sessions (3 by default) running in the background on the same machine. It may speed up the processing time, especially when pattern is used is informed.
`workers`	A positive numeric scalar or a function specifying the number of parallel processes that can be active at the same time. Defaults to 3.
`verbose`	Shows the progress in console? Defaults to `TRUE`.

Value

Nothing. The processed images are saved to the current working directory.

Examples

```
# apply_fun_to_imgs("pattern", image_resize, rel_size = 50)
```

---

as.numeric.image *Coerce* image *pixel values to numeric vector*

Description

Coerce `image` pixel values to numeric vector

Usage

```
## S3 method for class 'image'
as.numeric(x, raw_to_scale = TRUE, ...)
```

Arguments

`x`	An `image` object.
`raw_to_scale`	Logical; if `TRUE` and image is `raw`, scales 0..255 to 0..1. Default `TRUE`.
`...`	Unused.

---

as.raster.image *Convert* image *to standard R raster matrix*

Description

Convert `image` to standard R raster matrix

Usage

```
## S3 method for class 'image'
as.raster(x, ...)
```

Arguments

`x`	An `image` object.
`...`	Unused.

c.yolo_fewshot_model *Combine Few-Shot Models*

Description

Combines multiple `yolo_fewshot_model` objects into a multi-class few-shot model.

Usage

```
## S3 method for class 'yolo_fewshot_model'
c(...)
```

Arguments

`...`	`yolo_fewshot_model` objects to combine.

Value

A `yolo_fewshot_multimodel` object.

calibrate *Calibrates distances of landmarks*

Description

Calibrating the actual size is possible if any interlandmark distance on the image is known. calibrate() can be used to determine the size of a known distance (cm) on the graph. I invite users to photograph the object together with a scale (e.g., ruler, micrometer...).

Usage

```
calibrate(img, viewer = get_pliman_viewer())
```

Arguments

`img`	An `image` object
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

Value

A numeric (`double`) scalar value indicating the scale (in pixels per unit of known distance).

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
#### compute scale (dots per unit of known distance) ####
# only works in an interactive section
# objects_300dpi.jpg has a known resolution of 300 dpi
img <- image_pliman("objects_300dpi.jpg")
# Larger square: 10 x 10 cm
# 1) Run the function calibrate()
# 2) Use the left mouse button to create a line in the larger square
# 3) Declare a known distance (10 cm)
# 4) See the computed scale (pixels per cm)
calibrate(img)

# scale ~118
# 118 * 2.54 ~300 DPI
}
```

`ccc` *Model performance statistics*

Description

Computes several statistics to evaluate the agreement and accuracy between observed (real) and predicted values. It includes Lin's Concordance Correlation Coefficient (CCC), Pearson's correlation, Willmott's index of agreement, and error metrics like RMSE and Bias.

Usage

```
ccc(data, real, predito)
```

Arguments

`data`	A data frame or a grouped data frame (`grouped_df`).
`real`	The column containing the observed/reference values.
`predito`	The column containing the predicted/estimated values.

Details

The function evaluates model performance through different lenses:

- **Precision**: Pearson's `r` and `r2` measure how well the predicted values follow a linear relationship with the observed values.
- **Accuracy/Agreement**: `ccc` and `d` measure how close the predictions are to the 1:1 line (perfect agreement).
- **Error magnitude**: `rmse`, `mae`, and `rrmse` quantify the average deviation in absolute and relative terms.
- **Directional Bias**: `bias` indicates if the model consistently overestimates (positive) or underestimates (negative) the real values.

Lin's CCC is defined as:

$$\rho_c = \frac{2 \cdot \text{Cov}(x, y)}{\text{Var}(x) + \text{Var}(y) + (\bar{x} - \bar{y})^2}$$

Willmott's index of agreement (d) is defined as:

$$d = 1 - \frac{\sum(y - x)^2}{\sum(|y - \bar{x}| + |x - \bar{x}|)^2}$$

Value

A data frame with the following columns:

- `r`: Pearson correlation coefficient.
- `r2`: Coefficient of determination (R-squared).
- `ccc`: Lin's Concordance Correlation Coefficient.
- `d`: Willmott's Index of Agreement.
- `rmse`: Root Mean Squared Error.
- `rrmse`: Relative Root Mean Squared Error (%).
- `mae`: Mean Absolute Error.
- `bias`: Mean bias error.

Examples

```
library(dplyr)
library(pliman)
df <- data.frame(
  group = rep(c("A", "B"), each = 5),
  real = c(1:5, 2:6),
  predicted = c(1.1, 2, 2.9, 4.1, 5, 2.2, 3.1, 4, 4.8, 6.1)
)

# Without grouping
ccc(df, real, predicted)

# With grouping
```

```
df |>
  group_by(group) |>
  ccc(real, predicted)
```

`clear_pliman_cache` *Clear cached files created by pliman*

Description

Deletes cached `.rds` files used by functions such as `object_scatter()`. You can either remove the entire cache directory or only files older than a given number of days.

Usage

```
clear_pliman_cache(all = TRUE, days = NULL)
```

Arguments

`all`	Logical. If `TRUE` (default), deletes the entire cache directory.
`days`	Integer (optional). If provided, removes only files older than `days`. Ignored if `all = TRUE`.

Value

Invisibly returns `TRUE` if the operation was successful.

Examples

```
if(interactive()){
# Clear everything
clear_pliman_cache()

# Clear only files older than 7 days
clear_pliman_cache(all = FALSE, days = 7)
}
```

colorchecker_ref *Standard ColorChecker Reference Chart*

Description

Provides the standard reference RGB values for the 24-patch ColorChecker Classic (Macbeth) chart in sRGB D65 color space.

Usage

```
colorchecker_ref(chart = c("colorchecker24", "macbeth"))
```

Arguments

chart	The reference chart standard. Currently "colorchecker24" (default) or "macbeth".

Value

A data.frame with 24 rows and columns:

- id: Patch identifier (1 to 24).
- code: Patch code (No.001 to No.024).
- name: Descriptive name of the color patch.
- R, G, B: Standard reference sRGB values (0-255 scale).
- ref_R, ref_G, ref_B: Alias reference columns.

Examples

```
ref <- colorchecker_ref()
head(ref)
```

contour *Extract object contours from a labeled image*

Description

Extracts the external contour of each object in a labeled matrix (e.g., from watershed segmentation). It returns a list of coordinates for each object.

Usage

```
contour(labels)
```

Arguments

`labels`	A labeled `image` object or a matrix containing integer labels. Typically the result of `image_watershed()` or similar segmentation.

Value

A list where each element is a matrix of (x, y) coordinates for the contour of the respective object.

Examples

```
library(pliman)
# generate a binary image
img <- image_pliman("soybean_touch.jpg")
bin <- image_binary(img, "B")[[1]]
# watershed segmentation
wat <- image_watershed(bin)
# extract contours
cont <- contour(wat)
plot(img)
plot_contour(cont, col = "red", lwd = 2)
```

`contours` *Contour outlines from five leaves*

Description

A list of contour outlines from five leaves. It may be used as example in some functions such as `efourier()`

Format

A list with five objects

- `leaf_1`
- `leaf_2`
- `leaf_3`
- `leaf_4`
- `leaf_5`

Each object is a `data.frame` with the coordinates for the outline perimeter

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Source

Personal data. The images were obtained in the Flavia data set downlodable at https://flavia.sourceforge.net/

`cpp_extract_raster` *Fast C++ raster extraction engine without GDAL*

Description

Fast C++ raster extraction engine without GDAL

Usage

```
cpp_extract_raster(
  values,
  n_row,
  n_col,
  n_lyr,
  bbox_raster,
  geoms_r,
  fun = as.character(c("none")),
  exact = FALSE,
  return_coverage_area = FALSE,
  subdiv = 10L,
  summarize_quantiles = as.numeric(c(0.05, 0.975))
)
```

Arguments

`values`	Matrix of raster values (ncell x nlyr).
`n_row`	Number of raster rows.
`n_col`	Number of raster columns.
`n_lyr`	Number of raster layers.
`bbox_raster`	Vector c(xmin, xmax, ymin, ymax).
`geoms_r`	Rcpp::List of polygon geometries for each feature.
`fun`	CharacterVector of summary functions (”none”, ”mean”, ”median”, ”sum”, ”min”, ”max”, ”sd”, ”count”, ”quantile”, ”quantiles”).
`exact`	Logical, whether to compute exact pixel area fraction coverage.
`return_coverage_area`	Logical, whether to include coverage_area column in raw pixel output.
`subdiv`	Number of subdivisions per dimension when exact = TRUE.
`summarize_quantiles`	NumericVector of quantile probabilities (between 0 and 1) when ”quantiles” is requested.

custom_palette *Generate Custom Color Palette*

Description

This function generates a custom color palette using the specified colors and number of colors.

Usage

```
custom_palette(
  colors = c("yellow", "#53CC67", "#009B95", "#00588B", "#4B0055"),
  n = 5
)
```

Arguments

colors	A vector of colors to create the color palette. Default is c("steelblue", "salmon", "forestgreen").
n	The number of gradient colors in the color palette. Default is 100.

Value

A vector of colors representing the custom color palette.

Examples

```
# Generate a custom color palette with default colors and 10 colors
custom_palette()

# Generate a custom color palette with specified colors and 20 colors
custom_palette(colors = c("blue", "red"), n = 20)

# example code
library(pliman)
custom_palette(n = 5)
```

dilate_cpp *Fast morphological dilation*

Description

Fast morphological dilation

Usage

```
dilate_cpp(img, raio = 3L, forma = "disc")
```

Arguments

`img`	A logical matrix representing the binary image.
`raio`	Radius of the structuring element (integer >= 1).
`forma`	Shape of the structuring element ("disc" or "square").

`dist_transform` *Distance map transform*

Description

Computes the distance map transform of a binary image. The distance map is a matrix which contains for each pixel the distance to its nearest background pixel.

Usage

```
dist_transform(binary, normalize = FALSE)
```

Arguments

`binary`	A binary `image` object or logical matrix.
`normalize`	Logical argument indicating whether the distance map should be normalized to the 0-1 range. Defaults to `FALSE`.

Value

An `image` object or an array, with pixels containing the distances to the nearest background points

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
binary <- image_binary(img, "B")[[1]]
wts <- dist_transform(binary)
range(wts)
}
```

`dl_extract_phenotypes` *Extract Morphological, Spectral, and Texture Phenotypes from Deep Learning Predictions*

Description

`dl_extract_phenotypes()` bridges deep learning inference with botanical, agronomic, and biological measurement. It extracts granular phenotypes for every detected instance (seeds, fruits, leaves, lesions, roots) predicted by `image_detect_dl()`, `image_segment_dl()`, or `image_detect_sahi()`.

Extracted traits include:

- **Morphometry:** Area (pixels and physical units like cm^2), perimeter, circularity, solidity, eccentricity, aspect ratio, and major/minor axes.
- **Spectral Color Profiles:** Mean R, G, B channels and user-selected vegetation/color indices (e.g., VARI, NGRDI, GLI, BGI).
- **Texture Features:** Haralick contrast, entropy, correlation, and homogeneity.
- **Phytopathology & Severity:** Automated disease severity percentage ((Lesion Area / Host Area) * 100) grouped per leaf.

Usage

```
dl_extract_phenotypes(
  data,
  img = NULL,
  pixel_size = NULL,
  extract_indexes = c("R", "G", "B", "VARI", "NGRDI"),
  haralick = FALSE,
  severity = FALSE,
  host_class = "leaf",
  lesion_class = "lesion",
  verbose = TRUE
)
```

Arguments

`data`	A prediction object returned by `image_detect_dl()`, `image_segment_dl()`, or `image_detect_sahi()`, or a bounding box `data.frame`.
`img`	An optional `image` object. If `NULL` (default), extracted from `attr(data, "image")`.
`pixel_size`	Optional numeric scalar representing physical size per pixel (e.g. `0.05` for 0.05 cm/px). If provided, physical areas and perimeters are automatically computed.

`extract_indexes`
: Character vector of color indices to calculate per instance. Defaults to `c("R", "G", "B", "VARI", "NGRDI")`.

`haralick`
: Logical. Whether to extract Haralick texture features. Defaults to `FALSE`.

`severity`
: Logical. Whether to calculate disease severity percentages. Defaults to `FALSE`.

`host_class`
: Character string naming the host plant class (e.g., `"leaf"`, `"folha"`). Defaults to `"leaf"`.

`lesion_class`
: Character string naming the symptom/lesion class (e.g., `"lesion"`, `"sintoma"`). Defaults to `"lesion"`.

`verbose`
: Logical. Show progress messages. Defaults to `TRUE`.

Value

An S3 object of class `c("dl_phenotypes", "data.frame")` containing per-object phenotypic traits.

Examples

```
## Not run:
library(pliman)

# Segment leaves and lesions
res <- image_segment_dl("bean_rust.jpg", model = "bean_rust_yolo_seg.onnx")

# Extract comprehensive phenotypes & disease severity
pheno <- dl_extract_phenotypes(res, pixel_size = 0.02, severity = TRUE)
head(pheno)

## End(Not run)
```

---

efourier *Elliptical Fourier Analysis*

Description

Computes Elliptical Fourier Analysis of closed outlines based on `x` and `y`-coordinates coordinates.

Usage

```
efourier(x, nharm = 10, align = FALSE, center = FALSE, smooth_iter = 0)
```

Arguments

x	A matrix, a data.frame a list of perimeter coordinates, often produced with object_contour() or a vector of landmarks produced with landmarks() or landmarks_regradi().
nharm	An integer indicating the number of harmonics to use. Defaults to 10.
align	Align the objects before computing Fourier analysis? Defaults to FALSE. If TRUE, the object is first aligned along the major caliper with poly_align().
center	Center the objects on the origin before computing Fourier analysis? Defaults to FALSE. If TRUE, the object is first centered on the origin with poly_center().
smooth_iter	The number of smoothing iterations to perform. This will smooth the perimeter of the objects using poly_smooth().

Details

Adapted from Claude (2008). pp. 222-223.

Value

A list of class efourier with:

- the harmonic coefficients (an, bn, cn and dn)
- the estimates of the coordinates of the centroid of the configuration (a0 and c0).
- The number of rows (points) of the perimeter outline (nr).
- The number of harmonics used (nharm).
- The original coordinates (coords).

If x is a list of perimeter coordinates, a list of efourier objects will be returned as an object of class iefourier_lst.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Kuhl, F. P., and Giardina, C. R. (1982). Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18, 236-258. doi: doi:10.1016/0146664X(82)90034X

Examples

```
if (interactive()) {
library(pliman)
leaf1 <- contours[[4]]
plot_polygon(leaf1)

#### default options
# 10 harmonics (default)
# without alignment

ef <- efourier(leaf1)
```

```
efourier_coefs(ef)

# object is aligned along the major caliper with `poly_align()`
# object is centered on the origin with `poly_center()`
# using a list of object coordinates
ef2 <- efourier(contours, align = TRUE, center = TRUE)
efourier_coefs(ef2)

# reconstruct the perimeter of the object
# Use only the first one for simplicity
plot_polygon(contours[[1]] |> poly_align() |> poly_center())
efourier_inv(ef2[[1]]) |> plot_contour(col = "red", lwd = 4)
}
```

`efourier_coefs` *Get Fourier coefficients*

Description

Extracts the Fourier coefficients from objects computed with `efourier()` and `efourier_norm()` returning a 'ready-to-analyze' data frame.

Usage

```
efourier_coefs(x)
```

Arguments

`x`	An object computed with `efourier()` or `efourier_norm()`.

Value

A `data.frame` object

Examples

```
if (interactive()) {
library(pliman)

# a list of objects
efourier(contours) |> efourier_coefs()

# one object, normalized coefficients
efourier(contours[[4]]) |>
  efourier_norm() |>
  efourier_coefs()
}
```

`efourier_error` *Erros between the original and reconstructed outline*

Description

Computes the sum of squared distances between the original data and reconstructed outline. It allows examining reconstructed outlines with the addition of successive contributing harmonics indicated in the argument `nharm`.

Usage

```
efourier_error(
  x,
  nharm = NULL,
  type = c("error", "outline", "deviations"),
  plot = TRUE,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

`x`	An object computed with `efourier()`.
`nharm`	An integer or vector of integers indicating the number of harmonics to use. If not specified the number of harmonics used in `x` is used.
`type`	The type of plot to produce. By default, a line plot with the sum of squared distances (y-axis) and the number of harmonics (x-axis) is produced. If `type = "outline"` is used, a plot with the original polygon and the constructed outline is produced. If `type = "deviations"` is used, a plot with the deviations from the original outline and reconstructed outline (y-axis) and points along the outline (x-axis) is produced.
`plot`	A logical to inform if a plot should be produced. Defaults to `TRUE`.
`ncol`, `nrow`	The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.

Value

A list with the objects:

- `dev_points` A list with the deviations (distances) from original and predicted outline for each pixel of the outline.
- `data.frame` object with the minimum, maximum and average deviations (based on the outline points).

If `x` is an object of class `efourier_lst`, a list will be returned.

Examples

```
if (interactive()) {
library(pliman)
ef <-
  contours[[1]] |>
  efourier(nharm = 30)

efourier_error(ef)

efourier_error(ef,
               nharm = 30,
               type = "outline")

efourier_error(ef,
               nharm = c(1, 4, 20),
               type = "deviations")
}
```

`efourier_inv` *Inverse Elliptical Fourier Analysis*

Description

Performs an inverse elliptical Fourier transformation to construct a shape, given a list with Fourier coefficients computed with `efourier()`.

Usage

```
efourier_inv(x, nharm = NULL, a0 = NULL, c0 = NULL, npoints = 500)
```

Arguments

`x`	An object of class `efourier` or `efourier_lst` computed with `efourier()`.
`nharm`	An integer indicating the number of harmonics to use. If not specified the number of harmonics used in `x` is used.
`a0`, `c0`	the estimates of the coordinates of the centroid of the configuration. If `NULL` (default), the generated coordinates will be centered on the position of the original shape given by `efourier()`.
`npoints`	The number of interpolated points on the constructed outline. Defaults to 500.

Details

Adapted from Claude (2008). pp. 223.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if (interactive()) {
library(pliman)
plot_polygon(contours, aspect_ratio = 1)
# without alignment
ef <- efourier(contours, nharm = 10, align = FALSE)
ief <- efourier_inv(ef)
plot_contour(ief, col = "red", lwd = 2)
}
```

---

efourier_norm *Normalized Fourier coefficients*

Description

The first harmonic defines an ellipse that best fits the outlines. One can use the parameters of the first harmonic to "normalize" the data so that they can be invariant to size, rotation, and starting position of the outline trace. This approach is referred to in the literature as the normalized elliptic Fourier. `efourier_norm()` calculates a new set of Fourier coefficients `An`, `Bn`, `Cn`, `Dn` that one can use for further multivariate analyses (Claude, 2008).

Usage

```
efourier_norm(x, start = FALSE)
```

Arguments

`x`	An object computed with `efourier()`.
`start`	Logical value telling whether the position of the starting point has to be preserved or not.

Details

Adapted from Claude (2008). pp. 226.

Value

A list with the following components:

- `A`, `B`, `C`, `D` for harmonic coefficients.
- `size` the magnitude of the semi-major axis of the first fitting ellipse.
- `theta` angle, in radians, between the starting and the semi-major axis of the first fitting ellipse.
- `psi` orientation of the first fitting ellipse
- `a0` and `c0`, harmonic coefficients.
- `lnef` the concatenation of coefficients.
- `nharm` the number of harmonics used.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if (interactive()) {
library(pliman)
leaf1 <- contours[[4]]
plot_polygon(leaf1)

# compute the Fourier coefficients
ef <- efourier(leaf1)
efourier_coefs(ef)

# Normalized Fourier coefficients

efn <- efourier_norm(ef)
efourier_coefs(efn)
}
```

`efourier_power` *Power in Fourier Analysis*

Description

Computes an spectrum of harmonic Fourier power. The power is proportional to the harmonic amplitude and can be considered as a measure of shape information. As the rank of harmonic increases, the power decreases and adds less and less information. We can evaluate the number of harmonics that we must select, so their cumulative power gathers 99% of the total cumulative power (Claude, 2008).

Usage

```
efourier_power(
  x,
  first = TRUE,
  thresh = c(0.8, 0.85, 0.9, 0.95, 0.99, 0.999),
  plot = TRUE,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

`x`	An object of class `efourier`computed with `efourier()`.
`first`	Logical argument indicating whether to include the first harmonic for computing the power. See `Details`.

`thresh`	A numeric vector indicating the threshold power. The number of harmonics needed for such thresholds will then be computed.
`plot`	Logical argument indicating whether to produce a plot.
`ncol`, `nrow`	The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.

Details

Most of the shape "information" is contained in the first harmonic. This is not surprising because this is the harmonic that best fits the outline, and the size of ellipses decreases as for explaining successive residual variation. However, one may think that the first ellipse does not contain relevant shape information, especially when differences one wants to investigate concern complex outlines. By using `first = FALSE` it is possible to remove the first harmonic for this computation. When working on a set of outlines, high-rank-harmonics can contain information that may allow groups to be distinguished (Claude, 2008).

Adapted from Claude (2008). pp. 229.

Value

A list with the objects:

- `cum_power`, a `data.frame` object with the accumulated power depending on the number of harmonics
- `min_harm` The minimum number of harmonics to achieve a given power.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if (interactive()) {
library(pliman)
pw <- efourier(contours) |> efourier_power()
}
```

`efourier_shape` *Draw shapes based on Fourier coefficients*

Description

Calculates a 'Fourier elliptical shape' given Fourier coefficients

Usage

```
efourier_shape(
  an = NULL,
  bn = NULL,
  cn = NULL,
  dn = NULL,
  n = 1,
  nharm = NULL,
  npoints = 150,
  alpha = 4,
  plot = TRUE
)
```

Arguments

an	The a_n Fourier coefficients on which to calculate a shape.
bn	The b_n Fourier coefficients on which to calculate a shape.
cn	The c_n Fourier coefficients on which to calculate a shape.
dn	The d_n Fourier coefficients on which to calculate a shape.
n	The number of shapes to generate. Defaults to 1. If more than one shape is used, a list of coordinates is returned.
nharm	The number of harmonics to use. It must be less than or equal to the length of `*_n` coefficients.
npoints	The number of points to calculate.
alpha	The power coefficient associated with the (usually decreasing) amplitude of the Fourier coefficients.
plot	Logical indicating Whether to plot the shape. Defaults to `TRUE`

Details

`efourier_shape` can be used by specifying `nharm` and `alpha`. The coefficients are then sampled in an uniform distribution $(-\pi; \pi)$ and this amplitude is then divided by $harmonicrank^{a}lpha$. If `alpha` is lower than 1, consecutive coefficients will thus increase. See Claude (2008) pp.223 for the maths behind inverse ellipitical Fourier

Adapted from Claude (2008). pp. 223.

Value

A list with components:

- `x` vector of x-coordrdinates
- `y` vector of y-coordrdinates.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if (interactive()) {
library(pliman)
# approximation of the third leaf's perimeter
# 4 harmonics
image_pliman("potato_leaves.jpg", plot = TRUE)

efourier_shape(an = c(-7.34,  1.81,  -1.32, 0.50),
               bn = c(-113.88, 21.90, -0.31, -6.14),
               cn = c(-147.51, -20.89, 0.66, -14.06),
               dn = c(-0.48, 2.36, -4.36, 3.03))
}
```

---

ellipse *Confidence ellipse*

Description

Produces a confidence ellipse that is an iso-contour of the Gaussian distribution, allowing to visualize a 2D confidence interval.

Usage

```
ellipse(
  x,
  conf = 0.95,
  np = 100,
  plot = TRUE,
  fill = "green",
  alpha = 0.3,
  random_fill = TRUE
)
```

Arguments

`x`	A `matrix`, a `data.frame` or a `list` of perimeter coordinates, often produced with `object_contour()`.
`conf`	The confidence level. Defaults to `0.95`
`np`	Number of sampled points on the ellipse.
`plot`	Create a plot? Defaults to `TRUE`.
`fill`	The color to fill the ellipse. Defaults to `"green"`.
`alpha`	The alpha value to define the opacity of ellipse. Defaults to `0.3`
`random_fill`	Fill multiple ellipses with random colors? Defaults to `TRUE`.

Value

A matrix with coordinates of points sampled on the ellipse.

Note

Borrowed from Claude (2008), pp. 85

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if (interactive()) {
library(pliman)
ellipse(contours)
}
```

---

entropy *Compute Shannon Entropy*

Description

This function calculates the Shannon entropy of a numeric vector.

Usage

```
entropy(x)
```

Arguments

x	A numeric vector containing the values for which entropy will be computed.

Value

A numeric value representing the entropy.

Examples

```
library(pliman)
x <- c(1, 2, 2, 3, 3, 3, 4, 4, 4, 4)
entropy(x)
```

erode_cpp *Standard morphological erosion*

Description

Standard morphological erosion

Usage

```
erode_cpp(img, raio = 3L, forma = "disc")
```

Arguments

img	A logical matrix representing the binary image.
raio	Radius of the structuring element (integer >= 1).
forma	Shape of the structuring element ("disc" or "square").

erode_external_cpp *Fast external erosion*

Description

Fast external erosion

Usage

```
erode_external_cpp(img, raio = 3L, forma = "disc")
```

Arguments

img	A logical matrix representing the binary image.
raio	Radius of the structuring element (integer >= 1).
forma	Shape of the structuring element ("disc" or "square").

`erosao_conservativa` *Fast conservative erosion pipeline*

Description

Fast conservative erosion pipeline

Usage

```
erosao_conservativa(img, raio_erosao = 3L, raio_dilatacao = 3L, forma = "disc")
```

Arguments

`img`	A logical matrix representing the binary image.
`raio_erosao`	Radius for erosion step (integer >= 1).
`raio_dilatacao`	Radius for dilation step (integer >= 1).
`forma`	Shape of the structuring element ("disc" or "square").

`fill_holes_cpp` *Fast internal hole filling (with optional maximum hole size and neck distance check)*

Description

Fast internal hole filling (with optional maximum hole size and neck distance check)

Usage

```
fill_holes_cpp(img, max_size = -1, min_neck_dist = 0)
```

Arguments

`img`	A logical matrix representing the binary image.
`max_size`	Maximum area (in pixels) of internal holes to fill. If <= 0 (default), fills all holes.
`min_neck_dist`	Minimum foreground thickness (distance to background) surrounding a hole to allow filling. Default = 0.0 (fill all internal holes).

`focus_stack` *Multi-Focus Image Alignment, Registration and Fusion (MFIF)*

Description

High-performance native tools for focus stacking in digital imaging and microscopy:

- `align_stack()`: Sub-pixel image alignment and registration implemented natively in C++ using multi-scale Gaussian pyramid Enhanced Correlation Coefficient (ECC) maximization (affine motion or translation) to eliminate focus breathing, translation, and rotation, with optional automatic border cropping. Zero external Python/OpenCV dependencies.
- `fuse_multifocus()`: Multi-Focus Image Fusion (MFIF) supporting:
  1. Deep learning ONNX models (e.g., IFCNN) with either hierarchical tournament tree (`strategy = "pyramid"`) or linear (`strategy = "sequential"`) fusion.
  2. Ultra-fast native C++ Modified Laplacian Extended Depth of Field (`method = "laplacian"`), ideal for massive stacks of 50-500 images in < 1 second.
- `process_focus_stack()`: Orchestrates the complete end-to-end focus stacking pipeline, taking raw images, performing alignment, keyframe subsampling (`max_frames`), and fusion.

Usage

```
align_stack(
  images,
  reference_index = NULL,
  method = c("ecc", "translation", "orb"),
  crop = TRUE,
  levels = 3,
  max_iter = 50,
  eps = 1e-04,
  max_frames = NULL,
  verbose = TRUE,
  ...
)

image_align_stack(
  images,
  reference_index = NULL,
  method = c("ecc", "translation", "orb"),
  crop = TRUE,
  levels = 3,
  max_iter = 50,
  eps = 1e-04,
  max_frames = NULL,
  verbose = TRUE,
```

```
  ...
)

fuse_multifocus(
  images,
  method = c("ifcnn", "laplacian"),
  strategy = c("pyramid", "sequential"),
  blend = c("soft", "hard"),
  radius = 3,
  power = 6,
  max_frames = NULL,
  model = "ifcnn",
  engine = c("gpu", "cpu"),
  threads = 0L,
  device_id = -1L,
  verbose = TRUE,
  ...
)

image_fuse_multifocus(
  images,
  method = c("ifcnn", "laplacian"),
  strategy = c("pyramid", "sequential"),
  blend = c("soft", "hard"),
  radius = 3,
  power = 6,
  max_frames = NULL,
  model = "ifcnn",
  engine = c("gpu", "cpu"),
  threads = 0L,
  device_id = -1L,
  verbose = TRUE,
  ...
)

process_focus_stack(
  images,
  align = TRUE,
  align_method = c("ecc", "translation", "orb"),
  reference_index = NULL,
  crop = TRUE,
  levels = 3,
  max_iter = 50,
  eps = 1e-04,
  max_frames = NULL,
  method = NULL,
  fusion_method = c("ifcnn", "laplacian"),
  strategy = c("pyramid", "sequential"),
```

```
  blend = c("soft", "hard"),
  radius = 3,
  power = 6,
  model = "ifcnn",
  engine = c("gpu", "cpu"),
  threads = 0L,
  device_id = -1L,
  return_aligned = FALSE,
  verbose = TRUE,
  ...
)

image_focus_stack(
  images,
  align = TRUE,
  align_method = c("ecc", "translation", "orb"),
  reference_index = NULL,
  crop = TRUE,
  levels = 3,
  max_iter = 50,
  eps = 1e-04,
  max_frames = NULL,
  method = NULL,
  fusion_method = c("ifcnn", "laplacian"),
  strategy = c("pyramid", "sequential"),
  blend = c("soft", "hard"),
  radius = 3,
  power = 6,
  model = "ifcnn",
  engine = c("gpu", "cpu"),
  threads = 0L,
  device_id = -1L,
  return_aligned = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`images`	A list of `Image` objects, an array of images, or a character vector of file paths to images in the stack.
`reference_index`	Integer specifying the index of the base reference image for alignment. If `NULL` (default), the central image of the stack (`ceiling(length(images) / 2)`) is selected as the optical reference.
`method`	Character specifying the method: • In `align_stack()`: Alignment model (`"ecc"` for 2D Affine motion or `"translation"` for 2D shifts).

- In `fuse_multifocus()`: Fusion algorithm (`"ifcnn"` for deep learning or `"laplacian"` for native C++ EDF).

`crop` Logical. Should black/empty borders introduced by affine warping be automatically cropped out? Defaults to `TRUE`.

`levels` Integer specifying number of pyramid levels for coarse-to-fine ECC alignment (default: 3).

`max_iter` Integer specifying maximum iterations per pyramid level for ECC convergence (default: 50).

`eps` Numeric threshold for ECC convergence (default: 1e-4).

`max_frames` Optional integer. If provided and the image stack exceeds this number of frames, automatically subsamples the stack to `max_frames` evenly spaced focal planes.

`verbose` Logical. If `TRUE` (default), displays progress messages.

`...` Additional arguments passed to underlying functions.

`strategy` Multi-image fusion strategy for deep learning (`"pyramid"` or `"sequential"`). Defaults to `"pyramid"`. Hierarchical tournament tree reduction pairs images at each level, bounding the number of sequential convolution passes per pixel to $\lceil \log_2(N) \rceil$ and preventing contrast loss and blur over large image stacks.

`blend` Character specifying blending mode for Laplacian fusion: `"soft"` (default, weighted softmax power blending) or `"hard"` (maximum sharpness pixel selection).

`radius` Integer specifying the local window radius for Laplacian sharpness aggregation (default: 3).

`power` Numeric exponent for soft Laplacian blending weights (default: 6.0).

`model` Character specifying the fusion ONNX model. Defaults to `"ifcnn"`. Can also be a direct path to a custom ONNX model file.

`engine` Execution engine: `"cpu"` (default) or `"gpu"` (DirectML acceleration).

`threads` Integer number of threads for ONNX inference (0 for auto).

`device_id` GPU device ID (-1 for default best detected GPU).

`align` Logical. Should the image stack be aligned before fusion? Defaults to `TRUE`.

`align_method` Alignment method for `process_focus_stack()`. One of `"ecc"` or `"translation"`.

`fusion_method` Multi-focus fusion method for `process_focus_stack()`. One of `"ifcnn"` (deep learning) or `"laplacian"` (native C++ Modified Laplacian Extended Depth of Field).

`return_aligned`

Logical. If `TRUE`, returns a named list with both `fused` and `aligned` images. If `FALSE` (default), returns only the fused `Image`.

Value

- `align_stack()`: A list of aligned `Image` objects with identical spatial dimensions.
- `fuse_multifocus()`: An all-in-focus `Image` object.
- `process_focus_stack()`: The fused `Image` object (or a list containing `fused` and `aligned` if `return_aligned = TRUE`).

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Zhang, Y., Liu, Y., Sun, P., Yan, H., Zhao, X., & Zhang, L. (2020). IFCNN: A general image fusion framework based on convolutional neural network. Information Fusion, 54, 99-118.

Evangelidis, G. D., & Psarakis, E. Z. (2008). Parametric image alignment using enhanced correlation coefficient maximization. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30(10), 1858-1865.

Examples

```
## Not run:
library(pliman)

# Load or read a stack of multi-focus images
imgs <- list(
  image_read("focus_frame1.jpg"),
  image_read("focus_frame2.jpg"),
  image_read("focus_frame3.jpg")
)

# Fast automated pipeline with native C++ Modified Laplacian (zero-AI, ultra-fast)
fused_lap <- process_focus_stack(imgs, fusion_method = "laplacian", blend = "soft")
plot(fused_lap)

# Deep learning IFCNN with hierarchical tournament pyramid reduction
fused_dl <- process_focus_stack(imgs, fusion_method = "ifcnn", strategy = "pyramid")
plot(fused_dl)

# Handling large stacks (e.g. 200 frames from a focus rack video)
# Automatically extract 25 key focal planes for alignment and fusion:
fused_rack <- process_focus_stack(imgs, max_frames = 25, fusion_method = "laplacian")

## End(Not run)
```

fuse_laplacian_cpp — *Native C++ Extended Depth of Field (EDF) Focus Stacking via Modified Laplacian*

Description

Ultra-fast analytical multi-focus image fusion using spatial Modified Laplacian (ML) energy maps. Ideal for large image stacks (50-500 images) where deep learning networks are computationally heavy.

Usage

```
fuse_laplacian_cpp(images, blend = "soft", radius = 3L, power = 6)
```

Arguments

`images`	List of aligned `Image` objects or arrays.
`blend`	Character: `"soft"` (weighted softmax blending) or `"hard"` (maximum sharpness selection).
`radius`	Integer: smoothing window radius for local sharpness aggregation (default: 3).
`power`	Numeric: exponent weight for soft blending (default: 6.0).

Value

A fused `Image` object.

get_card_colors — *Extract ColorChecker Palettes using Bilinear Homography and Negative Buffering*

Description

Automatically detects a ColorChecker chart in an image (or uses user-specified corners), maps each color patch using perspective-aware bilinear homography, and extracts pure nucleus color statistics using a dynamic negative buffer.

Usage

```
get_card_colors(
  img,
  reference = "colorchecker24",
  nrow = NULL,
  ncol = NULL,
  buffer = 0.45,
```

```
  margin_x = 0.05,
  margin_y = 0.07,
  corners = NULL,
  auto_orient = TRUE,
  index = "GRAY",
  erode = NULL,
  fill_hull = TRUE,
  stat = c("mean", "median"),
  plot = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An `image` object (S3 pliman image or array).
`reference`	Reference chart standard for comparison. Default `"colorchecker24"`. Can be `"colorchecker24"`, a custom `data.frame` with reference `R, G, B` columns, or `NULL` to extract observed colors without reference metrics.
`nrow`	The number of rows of color patches. If `NULL` (default), automatically determined based on card orientation (4 for landscape, 6 for portrait for 24-patch charts).
`ncol`	The number of columns of color patches. If `NULL` (default), automatically determined based on card orientation (6 for landscape, 4 for portrait).
`buffer`	Negative buffer shrinkage fraction (0 to 1, default `0.45` for 45% shrink). If provided as a percentage > 1 (e.g. 45), it is automatically scaled to fraction.
`margin_x`	Outer horizontal border margin fraction (default `0.05` / 5%).
`margin_y`	Outer vertical border margin fraction (default `0.07` / 7%).
`corners`	Optional quadrilateral corners. Can be `NULL` (automatic detection), a 4x2 matrix, a list with `$TL, $TR, $BR, $BL`, or an 8-element numeric vector.
`auto_orient`	Logical. If `TRUE` (default for 24-patch ColorChecker charts), automatically detects the physical chart orientation (0, 90, 180, 270 degrees) by identifying the White (patch 19) and Black (patch 24) corners, ensuring that patch 1 (Dark skin) to patch 24 (Black) are always indexed correctly regardless of camera tilt or card rotation.
`index`	Segmentation index used for auto-detecting the card (default `"GRAY"`).
`erode`	Erosion kernel size for segmentation mask. If `NULL` (default), automatically computed based on card coverage.
`fill_hull`	Logical. Fill holes in the segmentation mask (default `TRUE`).
`stat`	Primary statistic assigned to `R, G, B` columns: `"mean"` (default) or `"median"`.
`plot`	Logical. If `TRUE` (default), displays the image with overlay showing the detected card boundary, cell grid, negative buffer sampling polygons, and patch badges.
`verbose`	Logical. If `TRUE` (default), prints diagnostic summary.

... Legacy arguments (`xpix`, `ypix`, `filter_order`) supported for backward compatibility.

Details

The extraction employs a bilinear interpolation mapping:

$$P(u, v) = (1 - u)(1 - v)\mathbf{TL} + u(1 - v)\mathbf{TR} + uv\mathbf{BR} + (1 - u)v\mathbf{BL}$$

Outer card margins (`margin_x`, `margin_y`) exclude the surrounding plastic/cardboard frame. A dynamic negative buffer (`buffer`, default 45%) shrinks each nominal cell inwards towards its center, ensuring pixel sampling is strictly restricted to the homogeneous nucleus of the color chip, immune to edge glare, shadows, and plastic borders.

Value

An object of class `c("pliman_card_colors", "data.frame")` containing:

- `id`: Patch identifier (1 to `nrow * ncol`).
- `row`, `col`: Grid row and column positions.
- `center_x`, `center_y`: Pixel center coordinates.
- `R`, `G`, `B`: Sampled primary RGB values (0-255 scale).
- `R_med`, `G_med`, `B_med`: Median RGB values.
- `R_sd`, `G_sd`, `B_sd`: Standard deviations per channel.
- `n_pixels`: Number of pixels sampled in the nucleus.
- If reference comparison is active: `code`, `name`, `ref_R`, `ref_G`, `ref_B`, `Delta_R`, `Delta_G`, `Delta_B`, `Delta_E`, and `Delta_E_Calib`.

The object also stores attributes:

- `attr(res, "patches")`: List of patch bounding boxes and polygon vertices.
- `attr(res, "corners")`: Sorted corner coordinates (`TL, TR, BR, BL`).
- `attr(res, "wb_gains")`: Calculated White Balance gains.
- `attr(res, "ccm")`: Calculated 3x3 Color Correction Matrix.
- `attr(res, "avg_delta_raw")`: Mean Euclidean color error (ΔE) before calibration.
- `attr(res, "avg_delta_calib")`: Mean ΔE after CCM 3x3 calibration.

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("colorcheck.jpg")

# Auto-detect card, auto-orient, and extract 24 patches with 45% negative buffer
card <- get_card_colors(img)
head(card)

# View patch overlay
plot(card)
}
```

get_free_ram — *Get Available System Physical Memory (RAM)*

Description

`get_free_ram()` retrieves the current available physical RAM of the system in real-time using native OS system calls in C++ (`GlobalMemoryStatusEx` on Windows, `sysconf` on Linux, and Mach Kernel `host_statistics` on macOS). It executes in microseconds with zero memory allocation.

Usage

```
get_free_ram(unit = c("GB", "MB", "KB", "B"))
```

Arguments

`unit`	Character string specifying memory unit: `"GB"` (default), `"MB"`, `"KB"`, or `"B"` (bytes).

Value

A numeric value representing available system physical memory in the specified unit.

Examples

```
## Not run:
library(pliman)
# Get free RAM in GB
get_free_ram()

# Get free RAM in MB
get_free_ram("MB")

## End(Not run)
```

get_free_ram_cpp — *Fast C++ retrieval of available system physical RAM in bytes (Windows, Linux, macOS)*

Description

Fast C++ retrieval of available system physical RAM in bytes (Windows, Linux, macOS)

Usage

```
get_free_ram_cpp()
```

get_pliman_viewer *Get the value of the pliman_viewer option*

Description

Retrieves the current value of the pliman_viewer option used in the package.

Usage

```
get_pliman_viewer()
```

Value

The current value of the pliman_viewer option.

get_uuid *Extract UUID from filenames*

Description

This function extracts a UUID (Universal Unique Identifier) from the filename, using a regular expression that specifically identifies the standard UUID format.

Usage

```
get_uuid(filename)
```

Arguments

filename	A character vector containing filenames or strings

Value

A character vector with extracted UUIDs (or NA if none found)

Examples

```
library(pliman)
file <- "Grãos - contagem_f68bca60-c8cf-4272-9448-3f28891a97cd.jpg"
file2 <- "Grãos - contagem_f68bca60-c8cf-4272-9448-3f8891a97cd.jpg"
get_uuid(file)
```

ggplot_color *ggplot2-like colors generation*

Description

Generate ggplot2

Usage

```
ggplot_color(n = 1)
```

Arguments

n	The number of colors. This works well for up to about eight colours, but after that it becomes hard to tell the different colours apart.

Examples

```
library(pliman)
ggplot_color(n = 3)
```

gpa *Generalized Procrustes Analysis*

Description

Performs Generalized Procrustes Analysis (GPA) to align a set of shapes.

Usage

```
gpa(x, tol = 1e-05, max_iter = 100)
```

Arguments

x	A list of coordinate matrices (e.g., from `landmarks()` or outline coordinates).
tol	Tolerance for convergence.
max_iter	Maximum number of iterations.

Value

A list with components:

- `aligned`: A list of the aligned shapes.
- `consensus`: The consensus (mean) shape.
- `centroid_sizes`: The centroid sizes of the original shapes.
- `iterations`: The number of iterations until convergence.

`hist.image` *Compute and Plot Histogram for an* `image` *Object*

Description

Computes an ultra-fast C++ pixel intensity histogram for `image` objects of any storage mode (`raw`, `logical`, `double`, `integer`).

Usage

```
## S3 method for class 'image'
hist(
  x,
  nbins = 256,
  plot = TRUE,
  col = NULL,
  main = NULL,
  xlab = "Pixel Value",
  ...
)
```

Arguments

`x`	An `image` object.
`nbins`	Integer specifying the number of bins (default 256 for uint8/normalized data).
`plot`	Logical; if `TRUE` (default), plots the histogram.
`col`	Color(s) used to fill the histogram bars.
`main`	Main title for the plot.
`xlab`	X-axis label.
`...`	Additional graphical arguments passed to `plot()`.

Value

A `histogram` S3 object (or list of `histogram` objects for multi-channel images) invisibly.

`image` *The* `image` *class constructor and utilities*

Description

`image` creates or coerces objects into a memory-optimized `image` S3 object in `pliman`. Storing pixel data efficiently (using 8-bit `raw` vectors by default) provides up to 8x memory reduction compared to standard float64 double arrays.

Usage

```
image(
  data = raw(),
  colormode = NULL,
  colspace = "RGB",
  gamma = 1,
  storage = c("raw", "double", "integer", "logical"),
  ...
)

as_image(
  data,
  colormode = NULL,
  colspace = "RGB",
  gamma = 1,
  storage = c("raw", "double", "integer", "logical"),
  dim = NULL,
  ...
)

is_image(x)
```

Arguments

`data`	An array, matrix, raw vector, or existing image object.
`colormode`	Character specifying color mode: `"Color"` (default for 3D arrays) or `"Grayscale"` (default for 2D matrices).
`colspace`	Character specifying color space (default `"RGB"`).
`gamma`	Numeric gamma value (default `1.0`).
`storage`	Character specifying storage type: `"raw"` (uint8 0..255, memory optimized) or `"double"` (0..1 floating point).
`...`	Additional arguments.
`dim`	Optional dimensions vector to reshape data.
`x`	An object to check if it inherits from class `image`.

Value

An S3 object of class `c("image", "array")`.

Examples

```
## Not run:
img <- image(array(as.raw(sample(0:255, 300, replace = TRUE)), dim = c(10, 10, 3)))
print(img)
summary(img)
is_image(img)

## End(Not run)
```

---

image_align *Aligns an* `image` *object by hand*

Description

`image_align()` rotate an image given a line of desired aligment along the y axis that corresponds to the alignment of the objects (e.g., field plots). By default, the aligment will be to the vertical, which means that if the drawed line have an angle < 90 degrees parallel to the x axis, the rotation angle wil be negative (anticlocwise rotation).

Usage

```
image_align(
  img,
  align = c("vertical", "horizontal"),
  viewer = get_pliman_viewer(),
  plot = TRUE
)
```

Arguments

`img`	An `image` object
`align`	The desired alignment. Either `"vertical"` (default) or `"horizontal"`.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`plot`	Plots the aligned image? Defaults to `TRUE`.

Details

The `image_align` function aligns an image along the vertical or horizontal axis based on user-selected points. The alignment can be performed in either the base plotting system or using the mapview package for interactive visualization. If the viewer option is set to "base", the function prompts the user to select two points on the image to define the alignment line. If the viewer option is set to "mapview", the function opens an interactive map where the user can draw a polyline to define the alignment line. The alignment angle is calculated based on the selected points, and the image is rotated accordingly using the `image_rotate` function. The function returns the aligned image object.

Value

The `img` aligned

Examples

```
if (interactive()) {
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot = TRUE)
aligned <- image_align(flax)
}
```

`image_alpha` *Add Alpha Layer to an RGB Image*

Description

This function adds an alpha (transparency) layer to an RGB image. The alpha layer can be specified as a single numeric value for uniform transparency or as a matrix/array matching the dimensions of the image for varying transparency.

Usage

```
image_alpha(img, mask)
```

Arguments

`img`	An RGB image of class `image` ..
`mask`	A numeric value or matrix/array specifying the alpha layer: • If `mask` is a single numeric value, it sets a uniform transparency level (0 for fully transparent, 1 for fully opaque). • If `mask` is a matrix or array, it must have the same dimensions as the image channels, allowing for varying transparency.

Value

An `image` object with an added alpha layer, maintaining the RGBA format.

Examples

```
if (interactive()) {
library(pliman)

# Load a sample RGB image
img <- image_pliman("soybean_touch.jpg")

# 50% transparency
image_alpha(img, 0.5) |> plot()

# transparent background
mask <- image_binary(img, "NB")[[1]]
img_tb <- image_alpha(img, mask)
plot(img_tb)

}
```

---

image_augment *Augment Images*

Description

This function takes an image and augments it by rotating it multiple times.

Usage

```
image_augment(
  img,
  pattern = NULL,
  times = 12,
  type = "export",
  dir_original = NULL,
  dir_processed = NULL,
  parallel = FALSE,
  verbose = TRUE,
  workers = NULL
)
```

Arguments

`img`	An `image` object.
`pattern`	A regular expression pattern to select multiple images from a directory.
`times`	The number of times to rotate the image.
`type`	The type of output: "export" to save images or "return" to return a list of augmented images.
`dir_original`	The directory where original images are located.

`dir_processed`	The directory where processed images will be saved.
`parallel`	Whether to perform image augmentation in parallel.
`verbose`	Whether to display progress messages.
`workers`	Number of parallel processes.

Value

If type is "export," augmented images are saved. If type is "return," a list of augmented images is returned.

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf.jpg")
imgs <- image_augment(img, type = "return", times = 4)
image_combine(imgs)
}
```

`image_binary` *Creates a binary image*

Description

Reduce a color, color near-infrared, or grayscale images to a binary image using a given color channel (red, green blue) or even color indexes. The Otsu's thresholding method (Otsu, 1979) is used to automatically perform clustering-based image thresholding.

Usage

```
image_binary(
  img,
  index = "R",
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  return_class = "image",
  threshold = c("Otsu", "adaptive"),
  k = 0.15,
  windowsize = NULL,
  has_white_bg = FALSE,
  resize = FALSE,
  fill_hull = FALSE,
  max_size = NULL,
```

```
  remove_reflection = FALSE,
  erode = FALSE,
  dilate = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  filter_order = c("erode", "dilate", "opening", "closing", "filter", "fill_hull"),
  invert = FALSE,
  return_exact = FALSE,
  plot = TRUE,
  nrow = NULL,
  ncol = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)
```

Arguments

`img`	An image object.
`index`	A character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with `pliman_indexes()` and `image_index()` for more details.
`r`, `g`, `b`, `re`, `nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.
`return_class`	The class of object to be returned. If `"terra` returns a SpatRaster object with the number of layers equal to the number of indexes computed. If `"ebimage"` (default) returns a list of `image` objects, where each element is one index computed.
`threshold`	The theshold method to be used. • By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. • If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments. • If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.
`k`	a numeric in the range 0-1. when `k` is high, local threshold values tend to be lower. when `k` is low, local threshold value tend to be higher.
`windowsize`	windowsize controls the number of local neighborhood in adaptive thresholding. By default it is set to `1/3 * minxy`, where `minxy` is the minimum dimension of the image (in pixels).

`has_white_bg` Logical indicating whether a white background is present. If `TRUE`, pixels that have R, G, and B values equals to 1 will be considered as `NA`. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.

`resize` Resize the image before processing? Defaults to `FALSE`. Use a numeric value as the percentage of desired resizing. For example, if `resize = 30`, the resized image will have 30% of the size of original

`fill_hull` Fill holes in the objects? Defaults to `FALSE`.

`max_size` Maximum size of objects to keep. Larger objects are removed. Default is `NULL`.

`remove_reflection`
Logical or numeric. If `TRUE` or numeric, removes internal reflection holes from binary objects. If numeric, specifies the neck distance threshold. Default is `FALSE`.

`erode`, `dilate`, `opening`, `closing`, `filter`
Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.

Hierarchically, the operations are performed as opening $>$ closing $>$ filter. The value declared in each argument will define the brush size.

`filter_order` A character vector indicating the order in which morphological operations are applied. Defaults to `c("erode", "dilate", "opening", "closing", "filter", "fill_hull")`.

`invert` Inverts the binary image, if desired.

`return_exact` Logical. If `TRUE`, exact morphological reconstruction via watershed-partitioned seed matching is performed when `opening > 0`.

`plot` Show image after processing?

`nrow`, `ncol` The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.

`parallel`	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 70% of available cores.
`workers`	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
`verbose`	If `TRUE` (default) a summary is shown in the console.

Value

A list containing binary images. The length will depend on the number of indexes used.

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

References

Otsu, N. 1979. Threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern SMC-9(1): 62-66. doi:10.1109/tsmc.1979.4310076

Shafait, F., D. Keysers, and T.M. Breuel. 2008. Efficient implementation of local adaptive thresholding techniques using integral images. Document Recognition and Retrieval XV. SPIE. p. 317-322 doi:10.1117/12.767755

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_binary(img, index = c("R, G"))
}
```

`image_binary_dl` *Deep Learning Binary Mask Extraction*

Description

Extracts a pixel-accurate binary mask (foreground = 1, background = 0) from an image using state-of-the-art Deep Learning neural networks executed in ONNX format via native C++ bindings. Supports salient object detection, dichotomous image segmentation, high-resolution boundary matting, foundation segmentation models (SAM 2.1 / SAM 3.1), zero-shot text-prompted instance segmentation (Grounded-SAM), and one-shot visual exemplar segmentation (PerSAM).

Usage

```
image_binary_dl(
  img,
  model = c("u2netp", "ben2", "grounded-sam", "sam2.1", "persam", "rmbg-1.4", "rmbg-2.0",
    "withoutbg", "birefnet-lite", "sam3.1", "isnet-general-use", "silueta", "u2net"),
  threshold = 0.5,
  fill_hull = TRUE,
  filter = 0,
  erode = 0,
  dilate = 0,
  opening = 0,
  closing = 0,
  min_area = 0,
  invert = FALSE,
  pick_object = FALSE,
  prompt = NULL,
  exemplar = FALSE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img` An `image` object, 2D grayscale matrix, 3D color array (RGB/RGBA), or a `list` of images/arrays. When a list is passed, each element is automatically processed in batch, returning a corresponding list of binary masks.

`model` Character string specifying the pre-trained neural network architecture to use:

- **General Salient Object Detection & Background Extraction:**
  - `"u2netp"` (default): Ultra-lightweight U2-Net Portable (~4.6 MB, 320x320 input). Highly recommended for rapid CPU batch processing on standard laptops without requiring a dedicated GPU.
  - `"ben2"` (or `"ben"`): Boundary-aware Extraction Network Base (~212.6 MB, 1024x1024 input). High-accuracy crisp boundary extraction, exceptional for fine leaf contours, intricate roots, thin petioles, and plant serrations.
  - `"rmbg-1.4"` (or `"rmbg"`): BRIA AI RMBG 1.4 (~176.1 MB, 1024x1024 input). Enterprise-grade salient foreground extractor trained on diverse commercial datasets, exceptionally robust against studio lighting, reflections, and shadows.

  - `"rmbg-2.0"`: BRIA AI RMBG 2.0 (~976.9 MB, 1024x1024 input). Next-generation BiRefNet-based model for maximum boundary detail, fine textures, and translucent edge matting.
  - `"isnet-general-use"`: Intermediate Supervision Network (~178.6 MB, 1024x1024 input). High-precision boundary matting, ideal for subtle leaf lesions, chlorotic halos, and fine botanical structures.
  - `"withoutbg"`: withoutBG Open Weights (~433.4 MB, 448x448 input). Combines DepthAnythingV2 3D depth features with ConvNeXt-fused U-Net matting, excelling in depth-cluttered backgrounds.
  - `"birefnet-lite"` (or `"birefnet"`): Bilateral Reference Network Lite (~213.6 MB, 1024x1024 input). Bilateral reference architecture specialized in dichotomous image segmentation and complex shapes.
  - `"silueta"`: Compact silhouette extraction model (~44.2 MB, 320x320 input). Fast inference with solid accuracy on edge/CPU hardware.
  - `"u2net"`: Full-depth U2-Net architecture (~176.3 MB, 320x320 input). Deep multi-scale salient object detection model.
- **Zero-Shot & Interactive Foundation Models:**
  - `"grounded-sam"`: Zero-shot open-vocabulary text-prompted instance segmentation (~314.8 MB: Grounding DINO Tiny + SAM 2.1). Detects and segments specific concepts described in natural language (e.g., `prompt = "leaf"`, `prompt = "fruit"`, `prompt = "insect"`).
  - `"persam"`: Personalize Segment Anything via SAM 2.1 embeddings (~120.2 MB, 1024x1024 input). Segments all instances visually similar to one or more user-clicked exemplar objects using Multi-Prototype Cosine Similarity Max-Pooling.
  - `"sam2.1"` (or `"sam2"`): Meta AI Segment Anything 2.1 (Hiera-Tiny, ~120.2 MB, 1024x1024 input). Foundation vision transformer supporting point prompts, box prompts, or interactive clicking (`pick_object = TRUE`).
  - `"sam3.1"` (or `"sam3"`): Meta AI Segment Anything 3.1 (~868.1 MB, 1024x1024 input). High-capacity concept-driven foundation segmentation model.

`threshold` Numeric value in `[0, 1]` specifying the cutoff threshold applied to the model's continuous probability/logit output map. Default is `0.5`. Increasing this threshold (e.g., `0.7`) makes segmentation more conservative (avoiding background noise), whereas decreasing it (e.g., `0.3`) includes faint, thin, or translucent object edges.

`fill_hull` Logical. If `TRUE` (default), automatically fills internal holes and voids (such as specular glare or reflective spots) within the segmented foreground object using morphological hole reconstruction (`fill_holes_cpp()`).

`filter` Integer specifying the window radius for binary median filtering (`median_filter_binary_cpp()`). Default is `0` (disabled). A positive value (e.g., `2` or `3`) eliminates salt-and-

pepper noise and smoothes jagged boundaries without blurring sharp object corners.

`erode` Integer specifying the radius for morphological erosion (`erode_cpp()`). Default is `0` (disabled). Shrinks foreground boundaries inward by the given pixel radius, useful for severing thin touching bridges between adjacent objects or removing outer halo artifacts.

`dilate` Integer specifying the radius for morphological dilation (`dilate_cpp()`). Default is `0` (disabled). Expands foreground boundaries outward by the given pixel radius, useful for restoring peripheral details or slightly undersized masks.

`opening` Integer specifying the radius for morphological opening (erosion followed by dilation). Default is `0` (disabled). Removes small stray islands and thin protrusions while preserving overall object geometry.

`closing` Integer specifying the radius for morphological closing (dilation followed by erosion). Default is `0` (disabled). Fuses narrow internal cracks, gaps, and small indentations along the perimeter.

`min_area` Integer minimum object size in pixels evaluated via connected component labeling (`bwlabel_cpp()`). If positive, discards all foreground connected components smaller than this area. If `0` (default), automatically isolates and retains only the single largest connected component in the image.

`invert` Logical. If `TRUE`, inverts the output binary mask such that foreground becomes `0` and background becomes `1`. Default is `FALSE`.

`pick_object` Logical. If `TRUE`, activates an interactive graphics prompt via `graphics::locator()`, allowing the user to directly click on the object(s) of interest in the plot window to guide SAM models. Press `<Esc>` or right-click when finished. Default is `FALSE`.

`prompt` Flexible prompt specification for SAM, Grounded-SAM, and PerSAM models:

- `NULL` (default): Defaults to the center of the image `c(width / 2, height / 2)` for SAM models.
- Character string (e.g., `"leaf"`, `"fruit"`, `"weed"`): Triggers Grounded-SAM open-vocabulary object detection.
- Numeric vector `c(x, y)`: Coordinates of a single foreground point prompt or single visual exemplar.
- Two-column numeric matrix or data frame `cbind(x, y)`: Specifies multiple positive point prompts or multiple exemplar prototypes.
- Numeric vector of length 4 `c(xmin, ymin, xmax, ymax)`: Specifies a bounding box prompt in pixel coordinates.
- `"center"`: Uses the image center coordinates `c(width / 2, height / 2)`.
- `"box"`: Uses a bounding box spanning the full image canvas.

`exemplar` Logical. If `TRUE` (or if `model = "persam"`), activates one-shot visual exemplar segmentation (PerSAM) using SAM 2.1 embeddings. Default is `FALSE`.

threads	Integer specifying the number of threads for parallel ONNX Runtime execution. Default is 0, which automatically detects and utilizes all available logical CPU cores on the system.
engine	Character specifying the backend execution engine: "cpu" (default) or "gpu". On Windows, "gpu" utilizes Microsoft DirectML over DirectX 12 hardware (compatible with NVIDIA, AMD, and Intel GPUs without requiring CUDA toolkit installation).
device_id	Integer GPU device adapter index. Default is -1, which automatically selects the high-performance discrete GPU adapter with the maximum dedicated video memory (VRAM).
verbose	Logical. If TRUE (default), displays step-by-step progress feedback and timing indicators via cli::cli_progress_step(). Set to FALSE for silent execution in automated pipelines.
plot	Logical. If TRUE (default), renders the resulting binary mask to the active graphics device.
dir	Character string specifying the local directory where ONNX model files are stored and cached. Defaults to pliman_model_dir().
...	Additional arguments passed down to internal plotting methods.

Value

A binary image object (Grayscale, values 0 and 1) or a list of binary image objects.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("leaf.png")

  # Fast binarization on CPU (ultra-lightweight, 4.6 MB)
  mask <- image_binary_dl(img, model = "u2netp")

  # Sub-pixel boundary segmentation for fine veins and serrations
  mask_ben <- image_binary_dl(img, model = "ben2")

  # Interactive prompt: click directly on objects of interest
  mask_interactive <- image_binary_dl(img, model = "sam2.1", pick_object = TRUE)

  # Open-vocabulary text-prompted binarization
  mask_weed <- image_binary_dl(img, model = "grounded-sam", prompt = "weed")

  # One-shot visual exemplar segmentation (PerSAM)
  mask_exemplar <- image_binary_dl(img, model = "persam", prompt = c(150, 200))

## End(Not run)
```

image_bwlabel *Label connected components in a binary image*

Description

This function performs Connected Component Labeling (CCL) on a binary image. It implements a highly optimized 2-pass Union-Find algorithm, providing lightning-fast execution even for massive images.

Usage

```
image_bwlabel(binary)
```

Arguments

`binary`	A binary `image` object or a logical matrix/array.

Value

A grayscale `image` object containing the labels, where each isolated object is assigned a unique integer value. Background pixels are 0.

Examples

```
## Not run:
  library(pliman)
  bin <- image_pliman("soybean_touch.jpg") |> image_binary(index = "B")
  labels <- image_bwlabel(bin$B)

## End(Not run)
```

image_canny_edge *Canny Edge Detector*

Description

Canny Edge Detector for Images. Adapted from https://github.com/bnosac/image/tree/master/image.CannyEdges.

Usage

```
image_canny_edge(img, index = "GRAY", s = 5, low_thr = 10, high_thr = 20)
```

Arguments

`img`	An `image` object.
`index`	A character string with the index to be used. Defaults to `"GRAY"`.
`s`	sigma, the Gaussian filter variance. Defaults to 5.
`low_thr`	lower threshold value of the algorithm. Defaults to 10.
`high_thr`	upper threshold value of the algorithm. Defaults to 20

Value

a list with an `image` object with values 0 or 255, and the number of pixels which have value 255 (pixels_nonzero).

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("sev_leaf.jpg")
conts <- image_canny_edge(img, index = "B")
image_combine(img, conts$edges)
}
```

`image_classify_dl` *Real-Time and Zero-Shot Image Classification*

Description

Classifies images using pre-trained YOLO26 classification models across 1,000 ImageNet categories, or open-vocabulary text candidates using OpenAI CLIP (ViT-B/32) cross-modal embeddings and cosine similarity softmax.

Usage

```
image_classify_dl(
  img,
  candidates = NULL,
  model = NULL,
  top_k = NULL,
  labels = NULL,
  format_prompt = TRUE,
  plot = FALSE,
  col = "#2b5c8f",
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`candidates`	An optional character vector of candidate class names or natural language descriptions. When supplied, performs zero-shot open-vocabulary classification using CLIP. Defaults to `NULL`.
`model`	Model name or path to a custom ONNX file. Defaults to `"yolo26n-cls"` when `candidates = NULL`, or `"clip-vit-b32"` when `candidates` is provided.
`top_k`	Integer. Number of top class predictions to return (defaults to 5 for YOLO models, or `length(candidates)` for CLIP zero-shot classification).
`labels`	Optional character vector of class names (defaults to 1,000 standard ImageNet classes).
`format_prompt`	Logical. If `TRUE` (default), candidate strings that do not contain `"photo"` or `"a "` are automatically formatted as `"a photo of a {candidate}"` for optimal CLIP zero-shot accuracy.
`plot`	Logical. Whether to display the image with top predicted class labels (or horizontal bar chart for CLIP).
`col`	Fill color for probability bar chart when `candidates` is supplied (default `"#2b5c8f"`).
`threads`	Number of CPU threads (default 0 for auto-tuning).
`engine`	Execution engine: `"cpu"` or `"gpu"` (DirectML).
`device_id`	GPU device ID (default -1 for auto).
`verbose`	Logical. Whether to show progress messages.
`dir`	Model directory (default `pliman_model_dir()`).
`...`	Additional arguments passed to `graphics::plot()` or `graphics::barplot()`.

Value

A data frame containing top predictions.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("sample.png")
  # YOLO classification
  image_classify_dl(img)

  # CLIP zero-shot classification
  res <- image_classify_dl(img, candidates = c("healthy leaf", "rust lesion", "insect damage"))
  print(res)

## End(Not run)
```

image_color_labels *Colorize Labeled Images*

Description

Colorizes a labeled image by allocating a different color to each object. Background pixels (value 0) are colorized with black.

Usage

```
image_color_labels(labels)
```

Arguments

`labels`	A labeled image (an `image` object or a matrix containing integer labels).

Value

A color `image` object.

Examples

```
## Not run:
  library(pliman)
  bin <- image_pliman("soybean_touch.jpg") |> image_binary(index = "B")
  lbl <- image_bwlabel(bin$B)
  color_lbl <- image_color_labels(lbl)
  plot(color_lbl)

## End(Not run)
```

image_combine *Combines images to a grid*

Description

Combines several images to a grid

Usage

```
image_combine(
  ...,
  labels = NULL,
  nrow = NULL,
  ncol = NULL,
  col = "black",
  verbose = TRUE,
```

```
  mar = c(1.5, 1.5, 1.5, 1.5),
  oma = c(0, 0, 0, 0)
)
```

Arguments

`...`	a comma-separated name of image objects or a list containing image objects.
`labels`	A character vector with the same length of the number of objects in `...` to indicate the plot labels.
`nrow`, `ncol`	The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.
`col`	The color for the plot labels. Defaults to `col = "black"`.
`verbose`	Shows the name of objects declared in `...` or a numeric sequence if a list with no names is provided. Set to `FALSE` to supress the text.
`mar`, `oma`	Margins for each panel and outer margins for the combined plot. Defaults to `mar = c(1.5, 1.5, 1.5, 1.5)` and `oma = c(0, 0, 0, 0)`. Set `mar = 0` or `mar = c(0, 0, 0, 0)` to eliminate margins and maximize the plotting area.

Value

A grid with the images in `...`

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Examples

```
if (interactive()) {
library(pliman)
img1 <- image_pliman("sev_leaf.jpg")
img2 <- image_pliman("sev_leaf_nb.jpg")
image_combine(img1, img2)

# Remove margins to optimize plotting area:
image_combine(img1, img2, mar = 0)
}
```

---

image_contour_line *Smooth Contour Line Detection*

Description

Smooth Contour Line Detection

Usage

```
image_contour_line(img, index = "GRAY", Q = 2)
```

Arguments

img	An image object.
index	A character string with the index to be used. Defaults to "GRAY".
Q	numeric value with the pixel quantization step

Value

A list with the contour lines.

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("sev_leaf.jpg")
conts <- image_contour_line(img, index = "B")
plot(img)
plot_contour(conts, col = "black")
}
```

---

image_correction *Correct Image Colors using a Color Checker*

Description

Calibrates the color of an image using a set of known color references (e.g., from a color checker) by implementing multiple calibration models including industry-standard **CCM (Color Correction Matrix 3x3)**, Affine (4x3), White Balance, Cubic (9-term polynomial), and Root-Polynomial (20-term).

Usage

```
image_correction(
  img,
  card_colors = NULL,
  known_colors = NULL,
  k_mat = NULL,
  model = c("ccm", "affine", "white_balance", "cubic", "root_polynomial"),
  reference = "colorchecker24",
  lambda = 1e-04,
  plot = FALSE,
  verbose = TRUE
)
```

Arguments

`img`	An `image` object to be corrected.
`card_colors`	A data frame of observed colors from the color checker (typically output directly from `get_card_colors()`). Must contain `R, G, B` (or `R_obs, G_obs, B_obs`) columns. If `NULL`, `k_mat` must be provided.
`known_colors`	A data frame of known reference colors. Must contain `R, G, B` (or `ref_R, ref_G, ref_B`) columns. If `NULL` and `card_colors` contains reference data or `reference = "colorchecker24"`, the built-in reference chart is used automatically.
`k_mat`	A pre-calculated transformation matrix (**K**). If provided, `card_colors` and `known_colors` are optional, and the correction is applied directly to `img`. Useful for batch calibration of image series.
`model`	The calibration model to use: • `"ccm"` (default): 3x3 linear Color Correction Matrix (robust, standard in computer vision). • `"affine"`: 4x3 matrix with intercept. • `"white_balance"`: Channel gains derived from neutral white patch. • `"cubic"`: 9-term 3rd-order polynomial model. • `"root_polynomial"`: 20-term root-polynomial model.
`reference`	Reference chart standard to use when `known_colors = NULL` (default `"colorchecker24"`).
`lambda`	Ridge regularization factor for matrix inversion (default `1e-4`).
`plot`	Logical. If `TRUE` (default `FALSE`), displays a side-by-side comparison of the original and calibrated image.
`verbose`	Logical. If `TRUE` (default), prints calibration diagnostics.

Details

Color calibration finds an optimal transformation matrix **K** mapping observed colors **S** to reference target colors **T**:

- `"ccm"` (default): Linear 3x3 Color Correction Matrix computed via Ridge-regularized least squares:
$$\mathbf{K} = (\mathbf{S}^T\mathbf{S} + \lambda\mathbf{I})^{-1}\mathbf{S}^T\mathbf{T}$$
- `"affine"`: 4x3 linear transformation with translation offset (intercept) $[1, R, G, B]$.
- `"white_balance"`: Channel-wise gain correction derived from neutral/white patches.
- `"cubic"`: 9-term polynomial mapping $(R, G, B, R^2, G^2, B^2, R^3, G^3, B^3)$.
- `"root_polynomial"`: 20-term root-polynomial mapping with cross-channel interactions.

Value

If `card_colors` or `known_colors` are provided, returns a list containing:

- `img`: An `image` object with the color correction applied.

- `k`: The calculated transformation matrix (**K**).
- `model`: The model used.
- `avg_delta_raw`: Mean Euclidean color error (ΔE) before calibration.
- `avg_delta_calib`: Mean ΔE after calibration.
- `wb_gains`: White balance channel gains.
- `df`: Comparison data frame with observed, calibrated, and reference colors.

If only `img` and a pre-calculated `k_mat` are supplied, returns the corrected `image` directly.

Examples

```
if(interactive()){
library(pliman)
img <- image_pliman("colorcheck.jpg")

# 1. Extract card colors
card <- get_card_colors(img)

# 2. Calibrate image using default CCM 3x3 model
calib <- image_correction(img, card, model = "ccm", plot = TRUE)

# 3. Or using polynomial models
calib_poly <- image_correction(img, card, model = "root_polynomial")
}
```

---

image_correction_pick

Interactive Color Correction

Description

Interactively calibrates the color of an image by prompting the user to sample color patches. It calls `pliman::pick_rgb_area()` to pause and prompt the user to click on `color_chips` patches in the image. These sampled colors are then mapped to `known_colors` using the specified `model`.

Usage

```
image_correction_pick(
  img,
  known_colors = NULL,
  color_chips = 24,
  model = c("ccm", "affine", "white_balance", "cubic", "root_polynomial"),
  lambda = 1e-04
)
```

Arguments

`img`	An `image` object to be corrected.
`known_colors`	A `data.frame` containing target reference values with `R, G, B` columns. If `NULL` and `color_chips == 24`, defaults to `colorchecker_ref()`.
`color_chips`	The number of color patches to sample interactively (default `24`).
`model`	The calibration model to use (`"ccm"`, `"affine"`, `"white_balance"`, `"cubic"`, or `"root_polynomial"`). Default `"ccm"`.
`lambda`	Ridge regularization factor (default `1e-4`).

Value

An `image` object with corrected colors.

Examples

```
if(interactive()){
library(pliman)

# Sample 4 known chips (e.g. blue, red, yellow, white)
known_colors <- data.frame(
 id = c(1, 2, 3, 4),
 R = c(25, 186, 245, 249),
 G = c(55, 26, 205, 242),
 B = c(135, 51, 0, 238)
)
img <- image_pliman("colorcheck.jpg")
img_cor <- image_correction_pick(img, known_colors, color_chips = 4)
image_combine(img, img_cor)
}
```

`image_create` *Create an* `image` *object of a given color*

Description

image_create() can be used to create an `image` object with a desired color and size.

Usage

```
image_create(color, width = 200, heigth = 200, plot = FALSE)
```

Arguments

`color`	either a color name (as listed by `grDevices::colors()`), or a hexadecimal string of the form `"#rrggbb"`.
`width`, `heigth`	The width and heigth of the image in pixel units.
`plot`	Plots the image after creating it? Defaults to `FALSE`.

Value

An object of class `image`.

Examples

```
if (interactive()) {
image_create("red")
image_create("#009E73", width = 300, heigth = 100)
}
```

---

image_data *Extract raw or numeric image data array/matrix*

Description

Extract raw or numeric image data array/matrix

Usage

```
image_data(x, ...)

## S3 method for class 'image'
image_data(x, type = c("same", "numeric", "normalized", "raw"), ...)
```

Arguments

`x`	An `image` or `image` object.
`...`	Additional arguments passed to methods.
`type`	Storage format to extract: • `"same"` — unmodified array (raw bytes or double as-is; default, fastest). • `"numeric"` — integer `0..255` regardless of internal storage (useful for arithmetic like `image_data(img, "numeric") / 2` or index calculation). • `"normalized"` — double `0..1` (raw images are divided by 255, double images returned as-is). • `"raw"` — raw bytes `0..255` (double images are scaled and coerced to raw).

`image_depth_dl` *Estimate Relative 3D Depth with Depth Anything V2*

Description

Estimates a metric-aligned relative 3D depth map from any 2D RGB image using the state-of-the-art Depth Anything V2 foundation model running purely via ONNX Runtime.

Usage

```
image_depth_dl(
  img,
  model = "depth-anything-v2",
  col_palette = "magma",
  invert = FALSE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`model`	Model name or path to a custom ONNX file. Defaults to `"depth-anything-v2"`.
`col_palette`	Color palette used to visualize the depth map when `plot = TRUE`. Options include `"viridis"`, `"magma"`, `"inferno"`, `"plasma"`, `"spectral"`, or a custom vector of colors.
`invert`	Logical. If `TRUE`, inverts the depth map so near objects are darker. Defaults to `FALSE`.
`threads`	Number of CPU threads (default 0 for auto-tuning).
`engine`	Execution engine: `"cpu"` or `"gpu"` (DirectML).
`device_id`	GPU device ID (default -1 for auto).
`verbose`	Logical. Whether to show progress messages.
`plot`	Logical. Whether to plot the resulting depth heatmap.
`dir`	Model directory (default `pliman_model_dir()`).
`...`	Additional arguments passed to `graphics::plot()`.

Value

An `Image` object representing the colorized depth map, with the raw normalized depth matrix stored in the `"depth"` attribute (`attr(res, "depth")`).

Examples

```
## Not run:
  library(pliman)
  img <- image_import("leaf.png")
  depth <- image_depth_dl(img)

## End(Not run)
```

---

image_detect_dl *Universal Object Detection with YOLO26*

Description

Performs real-time object detection using YOLO26 (or custom YOLO ONNX models) with bounding box regression and End-to-End detection.

Usage

```
image_detect_dl(
  img,
  model = "yolo26n",
  conf_threshold = 0.25,
  iou_threshold = 0.45,
  labels = NULL,
  col = NULL,
  rainbow = TRUE,
  lwd = 2,
  cex = 1,
  pad = 1,
  show_text = TRUE,
  show_conf = TRUE,
  show_class = TRUE,
  show_id = FALSE,
  badge = TRUE,
  return_features = FALSE,
  feature_model = c("yolo", "clip-vit-b32", "dinov2"),
  plot_features = FALSE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`model`	Model name or path to a custom ONNX file. Defaults to `"yolo26n"`. Options include `"yolo26n"`, `"yolo26s"`, `"yolo26m"`, `"yolo26l"`, `"yolo26x"`.
`conf_threshold`	Minimum confidence score for candidate boxes (default 0.25).
`iou_threshold`	IoU threshold for Non-Maximum Suppression (default 0.45).
`labels`	Optional character vector of class names (defaults to standard 80 COCO classes).
`col`	Optional color or palette for bounding boxes.
`rainbow`	Logical. If `TRUE`, assigns a distinct, unique color to each detected object. If `FALSE` (the default), objects of the same class share the exact same color. Defaults to `FALSE`.
`lwd`	Line width for bounding boxes (default 2).
`cex`	Size factor for label text (default `1.0`). Use smaller values (e.g., `cex = 0.2` or `0.5`) to reduce text size for dense scenes or many objects.
`pad`	Padding multiplier controlling the area/background badge size around label text (default `1.0`). Can be a single value or a length-2 vector `c(pad_x, pad_y)`. Use `pad = 0` or smaller values for a compact area.
`show_text`	Logical. Whether to display text labels on bounding boxes (default `TRUE`).
`show_conf`	Logical. Whether to display confidence scores in labels (default `TRUE`).
`show_class`	Logical. Whether to display class names in labels (default `TRUE`).
`show_id`	Logical. Whether to display object IDs (`#1`, `#2`, ...) in labels (default `FALSE`).
`badge`	Logical. Whether to draw a solid background badge behind label text (default `TRUE`). If `FALSE`, only the text is drawn directly.
`return_features`	Logical. If `TRUE`, extracts deep feature representations for detected bounding boxes and the global image. Attaches `"features"` (an $N \times D$ matrix of per-object deep embeddings, 512-D for CLIP or 384-D for DINOv2) and `"feature_map"` (dense spatial 3-component PCA false-color `Image`) as attributes to the returned object. Defaults to `FALSE`.
`feature_model`	Foundation model used for feature representation when `return_features = TRUE`. Options are `"clip-vit-b32"` (default, OpenAI CLIP ViT-B/32 producing 512-D semantic embeddings) or `"dinov2"` (Meta DINOv2-S ViT producing 384-D self-supervised patch embeddings).
`plot_features`	Logical. If `TRUE` and `return_features = TRUE`, visualizes the dense spatial PCA feature map. Defaults to `FALSE`.
`threads`	Number of CPU threads (default 0 for auto-tuning).
`engine`	Execution engine: `"cpu"` or `"gpu"` (DirectML).
`device_id`	GPU device ID (default -1 for auto).

`verbose`	Logical. Whether to show progress messages.
`plot`	Logical. Whether to plot the original image with bounding boxes.
`dir`	Model directory (default `pliman_model_dir()`).
`...`	Additional arguments passed to `graphics::plot()`.

Value

A data frame containing detected bounding boxes (`id`, `xmin`, `ymin`, `xmax`, `ymax`, `label`, `score`), with `summary` and `counts` stored as attributes. If `return_features = TRUE`, `features` (an $N \times D$ matrix) and `feature_map` (dense spatial PCA `Image`) are also attached as attributes.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("objects.png")
  boxes <- image_detect_dl(img)
  # Rainbow coloring per box:
  boxes_rb <- image_detect_dl(img, rainbow = TRUE)

## End(Not run)
```

`image_detect_sahi` *Slicing Aided Hyper Inference (SAHI) for Object Detection*

Description

`image_detect_sahi()` performs high-resolution sliding-window hyper inference on ultra-large images, drone orthomosaics, and macro scans. Large images (e.g. 20–50 Megapixels) are dynamically sliced into overlapping tiles of size `slice_width x slice_height`.

Deep learning inference is executed per slice, and all bounding boxes are transformed to global coordinates and merged using Global Cross-Slice Non-Maximum Suppression (NMS). This prevents small objects (seeds, aphids, stomata, seedlings) from vanishing when downscaled.

Usage

```
image_detect_sahi(
  img,
  model = "yolo26n",
  slice_height = 640,
  slice_width = 640,
  overlap_height_ratio = 0.2,
  overlap_width_ratio = 0.2,
  conf_threshold = 0.25,
  iou_threshold = 0.45,
```

```
  full_image_inference = TRUE,
  rainbow = TRUE,
  plot = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An `image` object, file path, or 3D array.
`model`	YOLO ONNX model name or file path (e.g., `"yolo26n"`, `"yolov8s.onnx"`).
`slice_height`	Integer. Height of sliding window slices. Defaults to `640`.
`slice_width`	Integer. Width of sliding window slices. Defaults to `640`.
`overlap_height_ratio`	Numeric scalar between 0 and 0.5. Height overlap fraction. Defaults to `0.20`.
`overlap_width_ratio`	Numeric scalar between 0 and 0.5. Width overlap fraction. Defaults to `0.20`.
`conf_threshold`	Numeric scalar between 0 and 1. Minimum detection confidence. Defaults to `0.25`.
`iou_threshold`	Numeric scalar between 0 and 1. IoU threshold for cross-slice NMS. Defaults to `0.45`.
`full_image_inference`	Logical. If `TRUE` (default), runs an additional detection pass on the resized full canvas to capture objects larger than individual slices.
`rainbow`	Logical. If `TRUE` (default), bounding boxes are rendered in rainbow colors.
`plot`	Logical. Display global detections overlaid on the full image. Defaults to `TRUE`.
`verbose`	Logical. Show progress bar and tile metrics. Defaults to `TRUE`.
`...`	Additional arguments passed to `image_detect_dl()`.

Value

A `data.frame` of global bounding boxes with attributes `image`, `counts`, and `summary`.

Examples

```
## Not run:
library(pliman)

# Run SAHI on a 4000x3000 drone orthophoto
res <- image_detect_sahi("drone_field.jpg", model = "seedlings.onnx", slice_width = 640)

## End(Not run)
```

`image_detect_world` *Open-Vocabulary Zero-Shot Object Detection with YOLO-World*

Description

Detects arbitrary objects described in natural language without any model retraining, powered by YOLO-World (v8s) and Ultralytics.

Usage

```
image_detect_world(
  img,
  classes = c("object"),
  conf_threshold = 0.25,
  iou_threshold = 0.45,
  model = "yolov8s-world",
  rainbow = TRUE,
  col = NULL,
  lwd = 2,
  cex = 1,
  pad = 1,
  show_text = TRUE,
  show_conf = TRUE,
  show_class = TRUE,
  show_id = FALSE,
  badge = TRUE,
  return_features = FALSE,
  feature_model = c("yolo", "clip-vit-b32", "dinov2"),
  plot_features = FALSE,
  plot = TRUE,
  auto_install = TRUE,
  verbose = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`classes`	A character vector of object classes or a comma-separated string (e.g. `c("dog", "frisbee")`).
`conf_threshold`	Minimum confidence threshold (default 0.25).
`iou_threshold`	IoU threshold for NMS (default 0.45).
`model`	Model name or weights path (default `"yolov8s-world"`).

`rainbow` Logical. If `TRUE`, gives each detected object instance a distinct, vibrant color. If `FALSE` (default), objects of the same class share the exact same color.

`col` Optional color or palette for bounding boxes.

`lwd` Bounding box line width (default 2).

`cex` Size factor for bounding box label text (default `1.0`).

`pad` Padding multiplier controlling the background badge size around label text (default `1.0`).

`show_text` Logical. Whether to display text labels on bounding boxes (default `TRUE`).

`show_conf` Logical. Whether to display confidence scores in labels (default `TRUE`).

`show_class` Logical. Whether to display class names in labels (default `TRUE`).

`show_id` Logical. Whether to display object IDs (`#1`, `#2`, ...) in labels (default `FALSE`).

`badge` Logical. Whether to draw a solid background badge behind label text (default `TRUE`). If `FALSE`, only the text is drawn directly.

`return_features` Logical. If `TRUE`, extracts deep feature representations for detected bounding boxes and the global image. Attaches `"features"` (an $N \times D$ matrix of per-object deep embeddings, 512-D for CLIP or 384-D for DINOv2) and `"feature_map"` (dense spatial 3-component PCA false-color `Image`) as attributes to the returned object. Defaults to `FALSE`.

`feature_model` Foundation model used for feature representation when `return_features = TRUE`. Options are `"clip-vit-b32"` (default, OpenAI CLIP ViT-B/32 producing 512-D semantic embeddings) or `"dinov2"` (Meta DINOv2-S ViT producing 384-D self-supervised patch embeddings).

`plot_features` Logical. If `TRUE` and `return_features = TRUE`, visualizes the dense spatial PCA feature map. Defaults to `FALSE`.

`plot` Logical. If `TRUE` (default), displays the image with annotated bounding boxes.

`auto_install` Logical. If `TRUE`, sets up required Python libraries if not already available.

`verbose` Logical. Whether to display progress messages.

`dir` Directory where models are stored (default `pliman_model_dir()`).

`...` Additional plotting parameters.

Value

A data frame containing detected bounding boxes (`id`, `xmin`, `ymin`, `xmax`, `ymax`, `class`, `score`). If `return_features = TRUE`, `features` (an $N \times D$ matrix) and `feature_map` (dense spatial PCA `Image`) are attached as attributes.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("field.jpg")
  dets <- image_detect_world(img, classes = c("corn plant", "weed", "soil"), rainbow = TRUE)

## End(Not run)
```

---

image_embed_dl *Extract Image Embeddings with OpenAI CLIP ViT-B/32*

Description

Computes L2-normalized 512-dimensional semantic embeddings for one or more images using OpenAI's CLIP (ViT-B/32) vision encoder directly via C++ ONNX Runtime.

Usage

```
image_embed_dl(
  img,
  model = "clip-vit-b32",
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = FALSE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`model`	Model name or path to a custom ONNX vision model file (default `"clip-vit-b32"`).
`threads`	Number of CPU threads (default 0 for auto-tuning).
`engine`	Execution engine: `"cpu"` or `"gpu"` (DirectML).
`device_id`	GPU device ID (default -1 for auto).
`verbose`	Logical. Whether to show progress messages.
`dir`	Directory where models are stored (default `pliman_model_dir()`).
`...`	Additional arguments.

Value

A numeric vector (if single image) or numeric matrix of dimensions `N x 512` containing the L2-normalized image embeddings.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("leaf.png")
  emb <- image_embed_dl(img)
  length(emb) # 512

## End(Not run)
```

`image_expand` *Expands an image*

Description

Expands an image towards the left, top, right, or bottom by sampling pixels from the image edge. Users can choose how many pixels (rows or columns) are sampled and how many pixels the expansion will have.

Usage

```
image_expand(
  img,
  left = NULL,
  top = NULL,
  right = NULL,
  bottom = NULL,
  edge = NULL,
  sample_left = 10,
  sample_top = 10,
  sample_right = 10,
  sample_bottom = 10,
  random = FALSE,
  filter = NULL,
  plot = TRUE
)
```

Arguments

`img`	An `image` object.
`left`, `top`, `right`, `bottom`	The number of pixels to expand in the left, top, right, and bottom directions, respectively.
`edge`	The number of pixels to expand in all directions. This can be used to avoid calling all the above arguments
`sample_left`, `sample_top`, `sample_right`, `sample_bottom`	The number of pixels to sample from each side. Defaults to 20.
`random`	Randomly sampling of the edge's pixels? Defaults to `FALSE`.

filter	Apply a median filter in the sampled pixels? Defaults to FALSE.
plot	Plots the extended image? defaults to FALSE.

Value

An image object

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_expand(img, left = 200)
image_expand(img, right = 150, bottom = 250, filter = 5)
}
```

---

image_features_dl *Extract Foundation Features and Semantic Visualization with DINOv2*

Description

Extracts dense patch embeddings and global representation vectors from DINOv2 (Vision Transformer ViT-S/14), and computes top-3 PCA false-color semantic segmentation without any labels or supervised training.

Usage

```
image_features_dl(
  img,
  model = "dinov2",
  patch_size = 14,
  return_pca = TRUE,
  interpolate = TRUE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`model`	Model name or path to a custom ONNX file. Defaults to `"dinov2"`.
`patch_size`	Size of ViT patches in pixels (default 14).
`return_pca`	Logical. If `TRUE`, computes top-3 principal components of patch embeddings.
`interpolate`	Logical. If `TRUE`, interpolates the PCA map from patch grid (37x37) to original image resolution.
`threads`	Number of CPU threads (default 0 for auto-tuning).
`engine`	Execution engine: `"cpu"` or `"gpu"` (DirectML).
`device_id`	GPU device ID (default -1 for auto).
`verbose`	Logical. Whether to show progress messages.
`plot`	Logical. Whether to plot the PCA false-color semantic image.
`dir`	Model directory (default `pliman_model_dir()`).
`...`	Additional arguments passed to `graphics::plot()`.

Value

A list containing:

- `cls_token`: 384-dimensional global semantic representation vector.
- `pca_image`: An `Image` object representing the 3-component PCA false-color semantic map.
- `pca_r`, `pca_g`, `pca_b`: Matrices of normalized PCA coordinates.
- `num_patches`: Total number of spatial patches.
- `embed_dim`: Dimensionality of patch embeddings.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("leaf.png")
  feats <- image_features_dl(img)

## End(Not run)
```

`image_generate_dl` *Ultra-Fast Generative Image Synthesis with SD-Turbo / LCM*

Description

Generates photo-realistic synthetic images from natural language text prompts in 1-2 inference steps using Stability AI's SD-Turbo (or LCM), enhanced to high resolution with Real-ESRGAN super-resolution.

Usage

```
image_generate_dl(
  prompt,
  negative_prompt = NULL,
  aspect = c("square", "portrait", "landscape"),
  n_steps = 1,
  guidance_scale = 0,
  superres = TRUE,
  scale = 4,
  seed = NULL,
  plot = TRUE,
  output_dir = NULL,
  auto_install = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`prompt`	Character text prompt describing the image to generate.
`negative_prompt`	Optional negative prompt specifying elements to avoid.
`aspect`	Aspect ratio of the generated image: `"square"` (1:1, 512x512), `"portrait"` (9:16, 432x768), or `"landscape"` (16:9, 768x432).
`n_steps`	Number of diffusion inference steps (default 1 for SD-Turbo, or 2-4 for LCM).
`guidance_scale`	Classifier-Free Guidance scale (default 0.0 for SD-Turbo).
`superres`	Logical. If `TRUE` (default), enhances the generated image resolution using Real-ESRGAN deep learning upscaler (`image_superres_dl()`).
`scale`	Upscaling factor when `superres = TRUE` (default 4).
`seed`	Optional integer seed for reproducible generation.
`plot`	Logical. If `TRUE` (default), displays the generated image.
`output_dir`	Optional directory to save generated image file.

`auto_install`	Logical. Whether to automatically configure environment if needed.
`verbose`	Logical. Whether to show progress messages.
`...`	Additional arguments passed to `graphics::plot()`.

Value

A `pliman` `Image` object.

Examples

```
## Not run:
  library(pliman)
  img <- image_generate_dl("macro photo of a healthy soybean leaf with morning dew")

## End(Not run)
```

`image_index` *Image indexes*

Description

`image_index()` Builds image indexes using Red, Green, Blue, Red-Edge, and NIR bands. See this page for a detailed list of available indexes.

The S3 method `plot()` can be used to generate a raster or density plot of the index values computed with `image_index()`

Usage

```
image_index(
  img,
  index = NULL,
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  return_class = c("image", "terra"),
  resize = FALSE,
  has_white_bg = FALSE,
  plot = TRUE,
  nrow = NULL,
  ncol = NULL,
  max_pixels = 5e+05,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  ...
```

```
)

## S3 method for class 'image_index'
plot(x, type = c("raster", "density"), nrow = NULL, ncol = NULL, ...)
```

Arguments

`img`	An `image` object. Multispectral mosaics can be converted to an `image` object using `mosaic_as_ebimage()`.
`index`	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use `pliman_indexes()` or the `details` section to see the available indexes. Defaults to `NULL` (normalized Red, Green, and Blue). You can also use "RGB" for RGB only, "NRGB" for normalized RGB, "MULTISPECTRAL" for multispectral indices (provided NIR and RE bands are available) or "all" for all indexes. Users can also calculate their own index using the band names, e.g., `index = "R+B/G"`.
`r`, `g`, `b`, `re`, `nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.
`return_class`	The class of object to be returned. If `"terra` returns a SpatRaster object with the number of layers equal to the number of indexes computed. If `"ebimage"` (default) returns a list of `image` objects, where each element is one index computed.
`resize`	Resize the image before processing? Defaults to `resize = FALSE`. Use `resize = 50`, which resizes the image to 50% of the original size to speed up image processing.
`has_white_bg`	Logical indicating whether a white background is present. If TRUE, pixels that have R, G, and B values equals to 1 will be considered as NA. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.
`plot`	Show image after processing?
`nrow`, `ncol`	The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.
`max_pixels`	integer > 0. Maximum number of cells to plot the index. If `max_pixels < npixels(img)`, downsampling is performed before plotting the index. Using a large number of pixels may slow down the plotting time.
`parallel`	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 70% of available cores.
`workers`	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
`verbose`	If `TRUE` (default) a summary is shown in the console.

...	Additional arguments passed to plot_index() for customization.
x	An object of class image_index.
type	The type of plot. Use type = "raster" (default) to produce a raster plot showing the intensity of the pixels for each image index or type = "density" to produce a density plot with the pixels' intensity.

Details

When type = "raster" (default), the function calls plot_index() to create a raster plot for each index present in x. If type = "density", a for loop is used to create a density plot for each index. Both types of plots can be arranged in a grid controlled by the ncol and nrow arguments.

Value

A list containing Grayscale images. The length will depend on the number of indexes used.

A NULL object

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Nobuyuki Otsu, "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62-66. 1979. doi:10.1109/TSMC.1979.4310076

Karcher, D.E., and M.D. Richardson. 2003. Quantifying Turfgrass Color Using Digital Image Analysis. Crop Science 43(3): 943-951. doi:10.2135/cropsci2003.9430

Bannari, A., D. Morin, F. Bonn, and A.R. Huete. 1995. A review of vegetation indices. Remote Sensing Reviews 13(1-2): 95-120. doi:10.1080/02757259509532298

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
image_index(img, index = c("R, NR"))
}
if (interactive()) {
# Example for S3 method plot()
library(pliman)
img <- image_pliman("sev_leaf.jpg")
# compute the index
ind <- image_index(img, index = c("R, G, B, NGRDI"), plot = FALSE)
plot(ind)

# density plot
plot(ind, type = "density")
}
```

`image_inpaint_dl` *Generative AI Image Inpainting with SD-Turbo*

Description

Modifies or replaces masked regions of an image based on a natural language text prompt using SD-Turbo inpainting.

Usage

```
image_inpaint_dl(
  img,
  mask,
  prompt,
  negative_prompt = NULL,
  n_steps = 2,
  guidance_scale = 0,
  seed = NULL,
  plot = TRUE,
  output_dir = NULL,
  auto_install = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An `Image` object, matrix, or file path to the input image.
`mask`	An `Image` object, binary matrix, or file path where foreground (1/TRUE) indicates the area to inpaint.
`prompt`	Text prompt describing what to synthesize inside the masked area.
`negative_prompt`	Optional negative prompt.
`n_steps`	Number of inference steps (default 2).
`guidance_scale`	CFG scale (default 0.0).
`seed`	Optional random seed.
`plot`	Logical. If `TRUE` (default), plots the inpainted image.
`output_dir`	Optional directory to save the inpainted image.
`auto_install`	Logical. Whether to automatically configure environment if needed.
`verbose`	Logical. Whether to show progress messages.
`...`	Additional arguments passed to `graphics::plot()`.

Value

A `pliman Image` object containing the inpainted result.

Examples

```
## Not run:
  library(pliman)
  inpainted <- image_inpaint_dl(img, mask, "a fresh green sprout")

## End(Not run)
```

---

image_label *Label Connected Components in a Binary Image*

Description

This function labels connected components in a binary image while allowing for a specified maximum gap between pixels to still be considered part of the same object.

Usage

```
image_label(img, max_gap = 0)
```

Arguments

`img`	A binary image matrix where `1` represents foreground pixels and `0` represents background pixels.
`max_gap`	An integer specifying the maximum allowable gap (in pixels) between connected components to be considered as part of the same object. Default is `1`.

Value

An object of class `image`, where each connected component is assigned a unique integer label.

Examples

```
if(interactive()){
library(pliman)
img <- matrix(c(
  1, 1, 0, 0, 0, 1, 1, 1, 0,
  0, 0, 0, 0, 0, 1, 0, 0, 0,
  1, 1, 0, 0, 1, 1, 1, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 1
), nrow = 4, byrow = TRUE)

image_label(img, max_gap = 1) |> plot()
image_label(img, max_gap = 2)
```

```
image_label(img, max_gap = 3)
}
```

`image_line_segment` *Line Segment Detection in an Image*

Description

Detects line segments in a digital image using the Line Segment Detector (LSD), a linear-time method that controls false detections and requires no parameter tuning. Based on Burns, Hanson, and Riseman's method with an a-contrario validation approach.

Usage

```
image_line_segment(
  img,
  index = "GRAY",
  scale = 0.8,
  sigma_scale = 0.6,
  quant = 2,
  ang_th = 22.5,
  log_eps = 0,
  density_th = 0.7,
  n_bins = 1024,
  union = FALSE,
  union_min_length = 5,
  union_max_distance = 5,
  union_ang_th = 7,
  union_use_NFA = FALSE,
  union_log_eps = 0
)
```

Arguments

`img`	An `image` object.
`index`	A character string with the index to be used. Defaults to `"GRAY"`.
`scale`	A positive numeric value. Scales the input image before detection using Gaussian filtering. A value <1 downscales, >1 upscales. Default is 0.8.
`sigma_scale`	A positive numeric value determining the Gaussian filter sigma. If scale <1, sigma = sigma_scale / scale; otherwise, sigma = sigma_scale. Default is 0.6.
`quant`	A positive numeric value controlling gradient quantization error. Default is 2.0.
`ang_th`	A numeric value (0-180) defining the gradient angle tolerance in degrees. Default is 22.5.

`log_eps` A numeric detection threshold. Larger values make detection stricter. Default is 0.0.

`density_th` A numeric value (0-1) defining the minimum proportion of supporting points in a rectangle. Default is 0.7.

`n_bins` A positive integer specifying the number of bins for pseudo-ordering gradient modulus. Default is 1024.

`union` Logical. If TRUE, merges close line segments. Default is FALSE.

`union_min_length` Numeric. Minimum segment length to merge. Default is 5.

`union_max_distance` Numeric. Maximum distance between segments to merge. Default is 5.

`union_ang_th` Numeric. Angle threshold for merging segments. Default is 7.

`union_use_NFA` Logical. If TRUE, uses NFA in merging. Default is FALSE.

`union_log_eps` Numeric. Detection threshold for merging. Default is 0.0.

Value

A list of class `lsd` containing:

- `n` - Number of detected line segments.
- `lines` - A matrix with detected segments (columns: x1, y1, x2, y2, width, p, -log_nfa).
- `pixels` - A matrix assigning each pixel to a detected segment (0 = unused pixels).

References

Grompone von Gioi, R., Jakubowicz, J., Morel, J.-M., & Randall, G. (2010). LSD: A Fast Line Segment Detector with a False Detection Control. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(4), 722-732.doi:10.5201/ipol.2012.gjmrlsd

Examples

```
library(pliman)
```

---

image_pose_dl *Universal Human Pose Estimation with YOLO26*

Description

Performs real-time human pose estimation and 17 keypoint detection using YOLO26 (e.g., `yolo26n-pose`, `yolo26s-pose`, etc.) with bounding box regression and anatomical skeleton visualization.

Usage

```
image_pose_dl(
  img,
  model = "yolo26n-pose",
  conf_threshold = 0.25,
  iou_threshold = 0.45,
  kpt_threshold = 0.3,
  col = NULL,
  lwd = 2,
  cex = 1,
  pad = 1,
  show_text = TRUE,
  show_conf = TRUE,
  show_class = TRUE,
  show_id = FALSE,
  badge = TRUE,
  kpt_radius = 4,
  bbox = TRUE,
  skeleton = TRUE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`model`	Model name or path to a custom ONNX file. Defaults to `"yolo26n-pose"`. Options include `"yolo26n-pose"`, `"yolo26s-pose"`, `"yolo26m-pose"`, `"yolo26l-pose"`, `"yolo26x-pose"`.
`conf_threshold`	Minimum confidence score for candidate person detections (default 0.25).
`iou_threshold`	IoU threshold for Non-Maximum Suppression (default 0.45).
`kpt_threshold`	Minimum confidence threshold for rendering individual keypoints and skeleton limbs (default 0.3).
`col`	Optional color or palette for bounding boxes and keypoint markers.
`lwd`	Line width for bounding boxes and skeleton limbs (default 2).
`cex`	Size factor for bounding box label text (default `1.0`).
`pad`	Padding multiplier controlling the background badge size around label text (default `1.0`).
`show_text`	Logical. Whether to display text labels on bounding boxes (default `TRUE`).

show_conf	Logical. Whether to display confidence scores in labels (default TRUE).
show_class	Logical. Whether to display class names in labels (default TRUE).
show_id	Logical. Whether to display object IDs (#1, #2, ...) in labels (default FALSE).
badge	Logical. Whether to draw a solid background badge behind label text (default TRUE). If FALSE, only the text is drawn directly.
kpt_radius	Radius/size for keypoint markers (default 4).
bbox	Logical. Whether to draw bounding boxes around detected persons (default TRUE).
skeleton	Logical. Whether to render the 17-keypoint anatomical skeleton (default TRUE).
threads	Number of CPU threads (default 0 for auto-tuning).
engine	Execution engine: "cpu" or "gpu" (DirectML).
device_id	GPU device ID (default -1 for auto).
verbose	Logical. Whether to show progress messages.
plot	Logical. Whether to plot the image with pose overlays.
dir	Model directory (default pliman_model_dir()).
...	Additional arguments passed to graphics::plot().

Value

A list with class pliman_pose containing:

- boxes: Data frame containing detected person bounding boxes (id, xmin, ymin, xmax, ymax, label, score).
- keypoints: List of data frames (one per person) containing the 17 COCO keypoints (keypoint, x, y, conf).
- summary: Human-readable detection summary string.
- counts: Data frame of instance counts.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("person.png")
  res <- image_pose_dl(img)
  res$boxes
  res$keypoints[[1]]

## End(Not run)
```

`image_prepare` *Prepare an image*

Description

This function aligns and crops the image using either base or mapview visualization. This is useful to prepare the images to be analyzed with `analyze_objects_shp()`

Usage

```
image_prepare(
  img,
  viewer = get_pliman_viewer(),
  downsample = NULL,
  max_pixels = 1e+06
)
```

Arguments

`img`	An optional `image` object
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`downsample`	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to `NULL`, which will find the best downsampling factor to approximate the `max_pixels` value.
`max_pixels`	integer > 0. Maximum number of cells to use for the plot. If `max_pixels < npixels(img)`, regular sampling is used before plotting.

Value

The alighed/cropped image for further visualization or analysis.

Examples

```
# Example usage:
if (interactive()) {
img <- image_pliman("mult_leaves.jpg")
image_prepare(img, viewer = "mapview")
}
```

`image_remove_bg_dl` *Deep Learning Background Removal (Transparent / RGBA)*

Description

Removes the background of an image using pre-trained Deep Learning models, producing a 4-channel **RGBA** image with a transparent background by default, or replacing the background with a solid color.

Usage

```
image_remove_bg_dl(
  img,
  model = c("u2netp", "ben2", "grounded-sam", "sam2.1", "persam", "rmbg-1.4", "rmbg-2.0",
    "withoutbg", "birefnet-lite", "sam3.1", "isnet-general-use", "silueta", "u2net"),
  threshold = 0.5,
  fill_hull = TRUE,
  filter = 0,
  erode = 0,
  dilate = 0,
  opening = 0,
  closing = 0,
  min_area = 0,
  invert = FALSE,
  pick_object = FALSE,
  prompt = NULL,
  exemplar = FALSE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  transparent = TRUE,
  bg_color = "black",
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)

image_remove_bg(
  img,
  model = c("u2netp", "ben2", "grounded-sam", "sam2.1", "persam", "rmbg-1.4", "rmbg-2.0",
    "withoutbg", "birefnet-lite", "sam3.1", "isnet-general-use", "silueta", "u2net"),
  threshold = 0.5,
  fill_hull = TRUE,
  filter = 0,
  erode = 0,
  dilate = 0,
```

```
  opening = 0,
  closing = 0,
  min_area = 0,
  invert = FALSE,
  pick_object = FALSE,
  prompt = NULL,
  exemplar = FALSE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  transparent = TRUE,
  bg_color = "black",
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img` An `image` object, 2D grayscale matrix, 3D color array (RGB/RGBA), or a `list` of images/arrays. When a list is passed, each element is automatically processed in batch, returning a corresponding list of binary masks.

`model` Character string specifying the pre-trained neural network architecture to use:

- **General Salient Object Detection & Background Extraction:**
  - `"u2netp"` (default): Ultra-lightweight U2-Net Portable (~4.6 MB, 320x320 input). Highly recommended for rapid CPU batch processing on standard laptops without requiring a dedicated GPU.
  - `"ben2"` (or `"ben"`): Boundary-aware Extraction Network Base (~212.6 MB, 1024x1024 input). High-accuracy crisp boundary extraction, exceptional for fine leaf contours, intricate roots, thin petioles, and plant serrations.
  - `"rmbg-1.4"` (or `"rmbg"`): BRIA AI RMBG 1.4 (~176.1 MB, 1024x1024 input). Enterprise-grade salient foreground extractor trained on diverse commercial datasets, exceptionally robust against studio lighting, reflections, and shadows.
  - `"rmbg-2.0"`: BRIA AI RMBG 2.0 (~976.9 MB, 1024x1024 input). Next-generation BiRefNet-based model for maximum boundary detail, fine textures, and translucent edge matting.
  - `"isnet-general-use"`: Intermediate Supervision Network (~178.6 MB, 1024x1024 input). High-precision boundary matting, ideal for subtle leaf lesions, chlorotic halos, and fine botanical structures.
  - `"withoutbg"`: withoutBG Open Weights (~433.4 MB, 448x448 input). Combines DepthAnythingV2 3D depth features with

ConvNeXt-fused U-Net matting, excelling in depth-cluttered backgrounds.

- `"birefnet-lite"` (or `"birefnet"`): Bilateral Reference Network Lite (~213.6 MB, 1024x1024 input). Bilateral reference architecture specialized in dichotomous image segmentation and complex shapes.
- `"silueta"`: Compact silhouette extraction model (~44.2 MB, 320x320 input). Fast inference with solid accuracy on edge/CPU hardware.
- `"u2net"`: Full-depth U2-Net architecture (~176.3 MB, 320x320 input). Deep multi-scale salient object detection model.

- **Zero-Shot & Interactive Foundation Models:**
  - `"grounded-sam"`: Zero-shot open-vocabulary text-prompted instance segmentation (~314.8 MB: Grounding DINO Tiny + SAM 2.1). Detects and segments specific concepts described in natural language (e.g., `prompt = "leaf"`, `prompt = "fruit"`, `prompt = "insect"`).
  - `"persam"`: Personalize Segment Anything via SAM 2.1 embeddings (~120.2 MB, 1024x1024 input). Segments all instances visually similar to one or more user-clicked exemplar objects using Multi-Prototype Cosine Similarity Max-Pooling.
  - `"sam2.1"` (or `"sam2"`): Meta AI Segment Anything 2.1 (Hiera-Tiny, ~120.2 MB, 1024x1024 input). Foundation vision transformer supporting point prompts, box prompts, or interactive clicking (`pick_object = TRUE`).
  - `"sam3.1"` (or `"sam3"`): Meta AI Segment Anything 3.1 (~868.1 MB, 1024x1024 input). High-capacity concept-driven foundation segmentation model.

`threshold` Numeric value in `[0, 1]` specifying the cutoff threshold applied to the model's continuous probability/logit output map. Default is `0.5`. Increasing this threshold (e.g., `0.7`) makes segmentation more conservative (avoiding background noise), whereas decreasing it (e.g., `0.3`) includes faint, thin, or translucent object edges.

`fill_hull` Logical. If `TRUE` (default), automatically fills internal holes and voids (such as specular glare or reflective spots) within the segmented foreground object using morphological hole reconstruction (`fill_holes_cpp()`).

`filter` Integer specifying the window radius for binary median filtering (`median_filter_binary_cpp()`). Default is `0` (disabled). A positive value (e.g., `2` or `3`) eliminates salt-and-pepper noise and smoothes jagged boundaries without blurring sharp object corners.

`erode` Integer specifying the radius for morphological erosion (`erode_cpp()`). Default is `0` (disabled). Shrinks foreground boundaries inward by the given pixel radius, useful for severing thin touching bridges between adjacent objects or removing outer halo artifacts.

`dilate` Integer specifying the radius for morphological dilation (`dilate_cpp()`). Default is `0` (disabled). Expands foreground boundaries outward by the

given pixel radius, useful for restoring peripheral details or slightly undersized masks.

`opening` Integer specifying the radius for morphological opening (erosion followed by dilation). Default is `0` (disabled). Removes small stray islands and thin protrusions while preserving overall object geometry.

`closing` Integer specifying the radius for morphological closing (dilation followed by erosion). Default is `0` (disabled). Fuses narrow internal cracks, gaps, and small indentations along the perimeter.

`min_area` Integer minimum object size in pixels evaluated via connected component labeling (`bwlabel_cpp()`). If positive, discards all foreground connected components smaller than this area. If `0` (default), automatically isolates and retains only the single largest connected component in the image.

`invert` Logical. If `TRUE`, inverts the output binary mask such that foreground becomes `0` and background becomes `1`. Default is `FALSE`.

`pick_object` Logical. If `TRUE`, activates an interactive graphics prompt via `graphics::locator()`, allowing the user to directly click on the object(s) of interest in the plot window to guide SAM models. Press `<Esc>` or right-click when finished. Default is `FALSE`.

`prompt` Flexible prompt specification for SAM, Grounded-SAM, and PerSAM models:

- `NULL` (default): Defaults to the center of the image `c(width / 2, height / 2)` for SAM models.
- Character string (e.g., `"leaf"`, `"fruit"`, `"weed"`): Triggers Grounded-SAM open-vocabulary object detection.
- Numeric vector `c(x, y)`: Coordinates of a single foreground point prompt or single visual exemplar.
- Two-column numeric matrix or data frame `cbind(x, y)`: Specifies multiple positive point prompts or multiple exemplar prototypes.
- Numeric vector of length 4 `c(xmin, ymin, xmax, ymax)`: Specifies a bounding box prompt in pixel coordinates.
- `"center"`: Uses the image center coordinates `c(width / 2, height / 2)`.
- `"box"`: Uses a bounding box spanning the full image canvas.

`exemplar` Logical. If `TRUE` (or if `model = "persam"`), activates one-shot visual exemplar segmentation (PerSAM) using SAM 2.1 embeddings. Default is `FALSE`.

`threads` Integer specifying the number of threads for parallel ONNX Runtime execution. Default is `0`, which automatically detects and utilizes all available logical CPU cores on the system.

`engine` Character specifying the backend execution engine: `"cpu"` (default) or `"gpu"`. On Windows, `"gpu"` utilizes Microsoft DirectML over DirectX 12 hardware (compatible with NVIDIA, AMD, and Intel GPUs without requiring CUDA toolkit installation).

device_id	Integer GPU device adapter index. Default is -1, which automatically selects the high-performance discrete GPU adapter with the maximum dedicated video memory (VRAM).
verbose	Logical. If TRUE (default), displays step-by-step progress feedback and timing indicators via cli::cli_progress_step(). Set to FALSE for silent execution in automated pipelines.
transparent	Logical. If TRUE (default), returns a 4-channel RGBA image object where background pixels have an alpha channel value of 0 (100% transparent).
bg_color	Optional background color to fill when transparent = FALSE. Accepts standard R color names (e.g., "white", "black") or hex strings (e.g., "#FFFFFF"). Defaults to "black".
plot	Logical. If TRUE (default), renders the resulting binary mask to the active graphics device.
dir	Character string specifying the local directory where ONNX model files are stored and cached. Defaults to pliman_model_dir().
...	Additional arguments passed down to internal plotting methods.

Value

An image object (4-channel RGBA if transparent = TRUE, 3-channel RGB otherwise) or a list of images.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("leaf.png")

  # Fast background removal with transparent alpha channel (CPU friendly)
  img_trans <- image_remove_bg_dl(img, model = "u2netp")

  # Razor-sharp plant boundary cutout (leaf margins, serrations, petioles)
  img_ben <- image_remove_bg_dl(img, model = "ben2")

  # Robust background removal under complex studio shadows and reflections
  img_rmbg <- image_remove_bg_dl(img, model = "rmbg-1.4")

  # Replace background with solid white
  img_white <- image_remove_bg_dl(img, model = "u2netp", transparent = FALSE, bg_color = "white")

  # Extract specific object via natural language prompt (Grounded-SAM)
  img_fruit <- image_remove_bg_dl(img, model = "grounded-sam", prompt = "red fruit")

  # Extract instances matching a visual exemplar click (PerSAM)
  img_exemplar <- image_remove_bg_dl(img, model = "persam", prompt = c(120, 150))

## End(Not run)
```

`image_remove_reflection` *Remove Specular Light Reflections*

Description

Removes specular light reflections from grayscale index images prior to thresholding.

Usage

```
image_remove_reflection(img, radius = 5, has_white_bg = TRUE)
```

Arguments

`img`	Grayscale `Image` object or numeric matrix.
`radius`	Radius of structuring window for specular peak removal (default = 5).
`has_white_bg`	Logical indicating whether the background is white/light (default = TRUE).

Value

A filtered `Image` object or matrix.

`image_resize` *Resize an image*

Description

Resizes a 2D (grayscale) or 3D (RGB) image or a list of images. It supports three reconstruction filters: Lanczos-3 (highest quality), Bilinear, and Nearest-Neighbor.

Usage

```
image_resize(
  img,
  rel_size = 100,
  width,
  height,
  filter = c("lanczos", "bilinear", "nearest"),
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)
```

Arguments

`img`	An `Image` object or a list of `Image` objects.
`rel_size`	The relative size of the resized image (as a percentage of the original size). Defaults to 100. For example, `rel_size = 50` resizes a `1000 x 800` image to `500 x 400`.
`width`	The target width (number of rows in the output matrix) in pixels. If missing, it is computed automatically to preserve the aspect ratio based on `height` or `rel_size`.
`height`	The target height (number of columns in the output matrix) in pixels. If missing, it is computed automatically to preserve the aspect ratio.
`filter`	The interpolation filter to use. Options are: • `"lanczos"`: Lanczos-3 windowed sinc resampling (highest quality). • `"bilinear"`: Bilinear interpolation (fast, smooth preview). • `"nearest"`: Nearest-neighbor interpolation (use for binary masks or label images).
`parallel`	Logical value indicating whether to process lists of images in parallel.
`workers`	The maximum number of parallel processes to run when `parallel = TRUE`.
`verbose`	Logical value. If `TRUE` (default), displays step notifications in the console.
`plot`	Logical value. If `TRUE`, plots the resized image(s). Defaults to `FALSE`.

Value

A modified `Image` object or a list of modified `Image` objects.

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Examples

```
if (interactive()) {
library(pliman)

# Load example image
img <- image_pliman("sev_leaf.jpg")

# Resize to 50% using high-quality Lanczos filter
img_half <- image_resize(img, rel_size = 50, filter = "lanczos", plot = TRUE)

# Resize specifying custom width and height
img_custom <- image_resize(img, width = 400, height = 300, filter = "bilinear")

# Resize a list of images
img_list <- list(img, img)
resized_list <- image_resize(img_list, rel_size = 25, parallel = FALSE)
}
```

`image_segment` *Image segmentation*

Description

- `image_segment()` reduces a color, color near-infrared, or grayscale images to a segmented image using a given color channel (red, green blue) or even color indexes (See `image_index()` for more details). The Otsu's thresholding method (Otsu, 1979) is used to automatically perform clustering-based image thresholding.
- `image_segment_iter()` Provides an iterative image segmentation, returning the proportions of segmented pixels.

Usage

```
image_segment(
  img,
  index = NULL,
  r = 1,
  g = 2,
  b = 3,
  re = 4,
  nir = 5,
  threshold = c("Otsu", "adaptive"),
  k = 0.1,
  windowsize = NULL,
  col_background = NULL,
  na_background = FALSE,
  has_white_bg = FALSE,
  fill_hull = FALSE,
  erode = FALSE,
  dilate = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  invert = FALSE,
  plot = TRUE,
  nrow = NULL,
  ncol = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)

image_segment_iter(
  img,
  nseg = 2,
  index = NULL,
```

```
  invert = NULL,
  threshold = NULL,
  k = 0.1,
  windowsize = NULL,
  has_white_bg = FALSE,
  plot = TRUE,
  verbose = TRUE,
  nrow = NULL,
  ncol = NULL,
  parallel = FALSE,
  workers = NULL,
  ...
)
```

Arguments

`img`	An image object or a list of image objects.
`index`	• For `image_segment()`, a character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with `pliman_indexes()`. See `image_index()` for more details. • For `image_segment_iter()` a character or a vector of characters with the same length of `nseg`. It can be either an available index (described above) or any operation involving the RGB values (e.g., `"B/R+G"`).
`r`, `g`, `b`, `re`, `nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. Defaults to 1, 2, 3, 4, and 5, respectively. If a multispectral image is provided (5 bands), check the order of bands, which are frequently presented in the 'BGR' format.
`threshold`	The theshold method to be used. • By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. • If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments. • If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.
`k`	a numeric in the range 0-1. when `k` is high, local threshold values tend to be lower. when `k` is low, local threshold value tend to be higher.
`windowsize`	windowsize controls the number of local neighborhood in adaptive thresholding. By default it is set to `1/3 * minxy`, where `minxy` is the minimum dimension of the image (in pixels).
`col_background`	The color of the segmented background. Defaults to `NULL` (white background).
`na_background`	Consider the background as NA? Defaults to FALSE.

`has_white_bg` Logical indicating whether a white background is present. If `TRUE`, pixels that have R, G, and B values equals to 1 will be considered as `NA`. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.

`fill_hull` Fill holes in the objects? Defaults to `FALSE`.

`erode`, `dilate`, `opening`, `closing`, `filter`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`invert` Inverts the binary image, if desired. For `image_segmentation_iter()` use a vector with the same length of `nseg`.

`plot` Show image after processing?

`nrow`, `ncol` The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.

`parallel` Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 70% of available cores.

`workers` A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.

`verbose` If `TRUE` (default) a summary is shown in the console.

`nseg` The number of iterative segmentation steps to be performed.

`...` Additional arguments passed on to `image_segment()`.

Value

- `image_segment()` returns list containing `n` objects where `n` is the number of indexes used. Each objects contains:

  - `image` an image with the RGB bands (layers) for the segmented object.
  - `mask` A mask with logical values of 0 and 1 for the segmented image.
- `image_segment_iter()` returns a list with (1) a data frame with the proportion of pixels in the segmented images and (2) the segmented images.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Nobuyuki Otsu, "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber. 9 (1): 62-66. 1979. doi:10.1109/TSMC.1979.4310076

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg", plot = TRUE)
image_segment(img, index = c("R, G, B"))
}
```

`image_segment_dl` *Deep Learning Semantic & Instance Segmentation, Object Detection, and Visual Exemplar Counting*

Description

Segments foreground objects (such as leaves, fruits, seeds, grains, roots, animals, or plants) from complex, textured, or out-of-focus backgrounds using pre-trained Deep Learning models in **ONNX** format.

Usage

```
image_segment_dl(
  img,
  type = c("segment", "mask", "highlight"),
  col_background = "white",
  col_highlight = "salmon",
  alpha = 0.4,
  border = NA,
  lwd = 1,
  bbox = FALSE,
  mask = TRUE,
  cex = 1,
  pad = 1,
```

```
  show_text = TRUE,
  show_conf = TRUE,
  show_class = TRUE,
  show_id = FALSE,
  badge = TRUE,
  rainbow = TRUE,
  model = c("u2netp", "ben2", "yolo26n-seg", "stardist", "grounded-sam", "sam2.1",
    "persam", "rmbg-1.4", "rmbg-2.0", "withoutbg", "birefnet-lite", "sam3.1",
    "isnet-general-use", "silueta", "u2net"),
  threshold = 0.5,
  box_threshold = 0.25,
  text_threshold = 0.25,
  iou_threshold = 0.5,
  exemplar = FALSE,
  sim_threshold = 0.5,
  min_dist = 16,
  feat_res = 256,
  max_objects = NULL,
  superres_map = FALSE,
  return_features = FALSE,
  feature_model = c("yolo", "clip-vit-b32", "dinov2"),
  plot_features = FALSE,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  fill_hull = TRUE,
  filter = 0,
  erode = 0,
  dilate = 0,
  opening = 0,
  closing = 0,
  min_area = 0,
  invert = FALSE,
  pick_object = FALSE,
  prompt = NULL,
  show_mask = FALSE,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `image` object, 2D grayscale matrix, 3D color array (RGB/RGBA), or a `list` of images/arrays. When a list is passed, each element is automatically processed in batch, returning a corresponding list of binary masks.
`type`	Character specifying the visual and output modality:

- `"segment"` (default): Returns an `image` object where detected foreground is preserved and background is replaced by a solid color specified by `col_background`.
- `"mask"`: Returns the binary mask (for semantic models) or single multi-label integer instance map (for YOLO-seg, StarDist, Grounded-SAM, and PerSAM) as an `image` object (`colormode = "Grayscale"`).
- `"highlight"`: Overlays the original image with semi-transparent colored polygons covering each detected instance, ideal for visual inspection, quality control, and publication figures. When `mask = FALSE`, overlays bounding boxes directly.
- `"boxes"`: Overlays bounding boxes on the original image and returns a data frame of detected boxes.

`col_background`
: Character string or hex code specifying the solid background color when `type = "segment"`. Defaults to `"white"`. Accepts any valid R color name (e.g., `"black"`, `"transparent"`) or hexadecimal code (e.g., `"#FFFFFF"`).

`col_highlight`
: Fill color for the semi-transparent overlay polygon when `type = "highlight"`. Defaults to `"salmon"`. When multiple instances are detected, distinct categorical colors are automatically generated from a rainbow palette.

`alpha`
: Numeric transparency value in `[0, 1]` for the highlight polygon overlay when `type = "highlight"`. Default is `0.4` (40% opacity).

`border`
: Color for the border stroke of the highlight polygon when `type = "highlight"`. Default is `NA` (no border stroke).

`lwd`
: Numeric line width for the bounding boxes and highlight polygon contours. Default is `1`.

`bbox`
: Logical. If `TRUE`, draws YOLO-style bounding boxes with class labels and confidence scores around detected instances. Defaults to `FALSE`.

`mask`
: Logical. If `TRUE` (default), computes pixel-accurate instance or semantic masks. When `mask = FALSE` and `bbox = TRUE`, skips the mask decoder step and returns only the detected bounding box coordinates (fast object detection). Defaults to `TRUE`.

`cex`
: Size factor for bounding box label text (default `1.0`). Use smaller values (e.g., `0.2` or `0.5`) to reduce text size for dense scenes or many objects.

`pad`
: Padding multiplier controlling the background badge size around label text (default `1.0`). Can be a single value or a length-2 vector `c(pad_x, pad_y)`. Use `pad = 0` for a badge fitting tightly to the text.

`show_text`
: Logical. Whether to display text labels on bounding boxes (default `TRUE`).

`show_conf`
: Logical. Whether to display confidence scores in labels (default `TRUE`).

`show_class`
: Logical. Whether to display class names in labels (default `TRUE`).

`show_id`
: Logical. If `TRUE`, overlays object instance IDs in bounding box labels or at their center of mass (computed via `poly_mass()`) when `type = "highlight"` and `bbox = FALSE`. Defaults to `FALSE`.

`badge`
: Logical. Whether to draw a solid background badge behind label text (default `TRUE`). If `FALSE`, only the text is drawn directly.

`rainbow` Logical. If `TRUE`, assigns a distinct, unique color to each segmented object. If `FALSE` (the default), objects of the same class share the exact same color. Defaults to `FALSE`.

`model` Model architecture or path to a custom ONNX file. Built-in choices:

- `"yolo26n-seg"` (or `"yolo26s-seg"`, `"yolo26m-seg"`, `"yolo26l-seg"`, `"yolo26x-seg"`): YOLO26 Instance Segmentation (real-time polygon mask & box detection, 80 COCO categories).
- `"stardist"` (or `"stardist-dsb2018"`): StarDist star-convex radial polygon segmentation for overlapping grains/cells.
- `"grounded-sam"`: Zero-shot open-vocabulary instance segmentation prompted by text.
- `"persam"`: One-shot visual exemplar segmentation guided by reference click points.
- `"u2netp"`, `"u2net"`: U2-Net salient foreground cutout (fast CPU-friendly default).
- `"ben2"`: Boundary-aware Enhanced Network (BEN2) for razor-sharp leaf margins, serrations, and roots.
- `"rmbg-1.4"`, `"rmbg-2.0"`: BRIA state-of-the-art commercial-grade background removal.
- `"withoutbg"`, `"birefnet-lite"`, `"isnet-general-use"`, `"silueta"`: Specialized salient cutout backends.

`threshold` Numeric value in `[0, 1]` specifying the cutoff threshold applied to the model's continuous probability/logit output map. Default is `0.5`. Increasing this threshold (e.g., `0.7`) makes segmentation more conservative (avoiding background noise), whereas decreasing it (e.g., `0.3`) includes faint, thin, or translucent object edges.

`box_threshold` Numeric confidence threshold in `[0, 1]` for Grounded-SAM and YOLO detection bounding box proposals. Default is `0.25`.

`text_threshold`

Numeric token logit confidence threshold in `[0, 1]` for Grounded-SAM text-to-box association. Default is `0.25`.

`iou_threshold` Numeric Non-Maximum Suppression (NMS) Intersection over Union (IoU) threshold in `[0, 1]` for deduplicating overlapping bounding box proposals. Default is `0.5`.

`exemplar` Logical. If `TRUE` (or if `model = "persam"`), activates one-shot visual exemplar segmentation (PerSAM) using SAM 2.1 embeddings. The user can click on 1 or more representative exemplar objects in the plot window (or pass coordinates via `prompt`), and the model will automatically compute visual feature embeddings, search the entire image for all similar instances via cosine similarity, and segment all matching objects. **Multi-Prototype Max-Pooling:** When multiple exemplars are clicked (e.g., one bright log, one dark log, one with moss), each click forms an independent 256-D prototype vector. The similarity map calculates the maximum cosine similarity against *any* prototype, preventing vector dilution and capturing wide visual diversity without prototype cancellation. Default is `FALSE`.

`sim_threshold` Numeric cosine similarity threshold in `[0, 1]` for PerSAM visual exemplar matching. Higher values (e.g. `0.55 - 0.65`) require closer visual resemblance, while lower values (e.g. `0.40 - 0.50`) capture more diverse, shadowed, or smaller instances. Default is `0.5`.

`min_dist` Numeric minimum spatial distance in pixels (within the 1024x1024 embedding space) between candidate peaks to prevent duplicate instance detections of the same object. Default is `16`.

`feat_res` Spatial resolution / algorithm for exemplar matching. Options:

- `64` (or `"fast"`, `"vit"`, `"low"`): Fast 64x64 ViT patch token matching (stride 16). Extremely fast and ideal for large objects (e.g. people, leaves, fruits, animals).
- `256` (or `"medium"`): 256x256 multi-scale CNN feature fusion (stride 4).
- `1024` (or `"dense"`, `"conv"`, `"high"`, `"grain"`): Dense Stride-1 Convolutional Exemplar matching at full 1024x1024 pixel resolution with single-pass Zero-Mean Normalized Cross-Correlation (ZNCC) and ViT semantic guidance. Essential for resolving small touching objects, seeds, and grains with sharp pixel boundaries. Defaults to `256`.

`max_objects` Maximum number of instances to detect and segment. Default is `300`.

`superres_map` Logical. If `TRUE`, applies Real-ESRGAN neural super-resolution to upscale the PerSAM similarity map for finer object boundaries and visualization. Defaults to `FALSE`.

`return_features` Logical. If `TRUE`, extracts deep feature representations for detected or segmented objects and the global image. Attaches `"features"` (an $N \times D$ matrix of per-object deep embeddings, 512-D for CLIP or 384-D for DINOv2) and `"feature_map"` (dense spatial 3-component PCA false-color `Image`) as attributes to the returned object. Defaults to `FALSE`.

`feature_model` Foundation model used for feature representation when `return_features = TRUE`. Options are `"clip-vit-b32"` (default, OpenAI CLIP ViT-B/32 producing 512-D semantic embeddings) or `"dinov2"` (Meta DINOv2-S ViT producing 384-D self-supervised patch embeddings).

`plot_features` Logical. If `TRUE` and `return_features = TRUE`, visualizes the dense spatial PCA feature map. Defaults to `FALSE`.

`threads` Integer specifying the number of threads for parallel ONNX Runtime execution. Default is `0`, which automatically detects and utilizes all available logical CPU cores on the system.

`engine` Character specifying the backend execution engine: `"cpu"` (default) or `"gpu"`. On Windows, `"gpu"` utilizes Microsoft DirectML over DirectX 12 hardware (compatible with NVIDIA, AMD, and Intel GPUs without requiring CUDA toolkit installation).

`device_id` Integer GPU device adapter index. Default is `-1`, which automatically selects the high-performance discrete GPU adapter with the maximum dedicated video memory (VRAM).

`fill_hull` Logical. If `TRUE` (default), automatically fills internal holes and voids (such as specular glare or reflective spots) within the segmented foreground object using morphological hole reconstruction (`fill_holes_cpp()`).

`filter` Integer specifying the window radius for binary median filtering (`median_filter_binary_cpp()`). Default is `0` (disabled). A positive value (e.g., `2` or `3`) eliminates salt-and-pepper noise and smoothes jagged boundaries without blurring sharp object corners.

`erode` Integer specifying the radius for morphological erosion (`erode_cpp()`). Default is `0` (disabled). Shrinks foreground boundaries inward by the given pixel radius, useful for severing thin touching bridges between adjacent objects or removing outer halo artifacts.

`dilate` Integer specifying the radius for morphological dilation (`dilate_cpp()`). Default is `0` (disabled). Expands foreground boundaries outward by the given pixel radius, useful for restoring peripheral details or slightly undersized masks.

`opening` Integer specifying the radius for morphological opening (erosion followed by dilation). Default is `0` (disabled). Removes small stray islands and thin protrusions while preserving overall object geometry.

`closing` Integer specifying the radius for morphological closing (dilation followed by erosion). Default is `0` (disabled). Fuses narrow internal cracks, gaps, and small indentations along the perimeter.

`min_area` Integer minimum object size in pixels evaluated via connected component labeling (`bwlabel_cpp()`). If positive, discards all foreground connected components smaller than this area. If `0` (default), automatically isolates and retains only the single largest connected component in the image.

`invert` Logical. If `TRUE`, inverts the output binary mask such that foreground becomes `0` and background becomes `1`. Default is `FALSE`.

`pick_object` Logical. If `TRUE`, activates an interactive graphics prompt via `graphics::locator()`, allowing the user to directly click on the object(s) of interest in the plot window to guide SAM models. Press `<Esc>` or right-click when finished. Default is `FALSE`.

`prompt` Flexible prompt specification for SAM, Grounded-SAM, and PerSAM models:

- `NULL` (default): Defaults to the center of the image `c(width / 2, height / 2)` for SAM models.
- Character string (e.g., `"leaf"`, `"fruit"`, `"weed"`): Triggers Grounded-SAM open-vocabulary object detection.
- Numeric vector `c(x, y)`: Coordinates of a single foreground point prompt or single visual exemplar.
- Two-column numeric matrix or data frame `cbind(x, y)`: Specifies multiple positive point prompts or multiple exemplar prototypes.
- Numeric vector of length 4 `c(xmin, ymin, xmax, ymax)`: Specifies a bounding box prompt in pixel coordinates.
- `"center"`: Uses the image center coordinates `c(width / 2, height / 2)`.

- `"box"`: Uses a bounding box spanning the full image canvas.

`show_mask`	Deprecated. If `TRUE`, equivalent to `type = "mask"`.
`verbose`	Logical. If `TRUE` (default), displays step-by-step progress feedback and timing indicators via `cli::cli_progress_step()`. Set to `FALSE` for silent execution in automated pipelines.
`plot`	Logical. If `TRUE` (default), renders the resulting binary mask to the active graphics device.
`dir`	Character string specifying the local directory where ONNX model files are stored and cached. Defaults to `pliman_model_dir()`.
`...`	Additional arguments passed down to internal plotting methods.

Details

This unified function supports five major computer vision workflows:

1. **Salient Foreground Segmentation & Background Removal:** Semantic cutout using architectures such as U2-Net, BEN2, BRIA RMBG 1.4/2.0, IS-Net, withoutBG, BiRefNet, and Silueta.
2. **Real-Time YOLO26 Instance Segmentation (`yolo26n-seg`):** Real-time instance segmentation combining bounding box detection with 32 prototype mask channels and continuous bilinear sub-pixel interpolation. Returns a single multi-label mask image (`type = "mask"`), smooth polygon overlays (`type = "highlight"`), or background cutouts (`type = "segment"`), accompanied by per-instance morphological measurements (area, perimeter, circularity, etc.).
3. **Star-Convex Object Detection & Segmentation (`stardist`):** Predicts 48 radial star-convex distance rays and object probability, specifically tailored for densely packed, touching, or convex structures such as seeds, grains, nuclei, and cells.
4. **Zero-Shot Text-Prompted Detection & Instance Segmentation (Grounded-SAM):** Combines Grounding DINO with SAM 2.1 to locate, outline, and count arbitrary concepts described in natural language (e.g., `prompt = "leaf, fruit, insect"`).
5. **One-Shot Visual Exemplar Segmentation & Counting (PerSAM):** Personalize Segment Anything via SAM 2.1. The user clicks on 1 or more exemplar objects (or provides coordinates), and the algorithm automatically searches, segments, and counts all visually similar objects across the entire image using Multi-Prototype Max-Pooling.

Value

- If `mask = FALSE` and `bbox = TRUE`: A data frame containing the bounding box coordinates (`id`, `xmin`, `ymin`, `xmax`, `ymax`, `score`, `label`).
- If `type = "segment"`: An `image` object containing the segmented image.
- If `type = "mask"`: A single multi-label or binary `image` object (`colormode = "Grayscale"`).
- If `type = "highlight"`: Invisibly returns a list with:
  - `"boxes"`: Data frame of detected bounding boxes and scores.
  - `"labels"`: Single multi-label integer mask `image` where each pixel value corresponds to instance `id`.

  - `"summary"`: Text summary of detected objects and counts.
  - `"counts"`: Data frame of instance counts per class label.
  - `"contours"`: List of polygon contour coordinate matrices.
  - `"features"`: If `return_features = TRUE`, numeric matrix of per-object deep feature embeddings.
  - `"feature_map"`: If `return_features = TRUE`, dense spatial 3-component PCA false-color `Image`.
  - `"similarity_map"`: Continuous cosine similarity `image` object (when `exemplar = TRUE`).
- If `img` is a list: A list of the corresponding outputs.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("leaf.png")

  # 1. Basic salient foreground segmentation with white background (CPU-friendly)
  seg <- image_segment_dl(img, model = "u2netp")

  # 2. Razor-sharp plant boundary extraction (leaf margins, serrations, roots)
  seg_ben <- image_segment_dl(img, model = "ben2")

  # 3. Semi-transparent polygon overlay for visual inspection
  image_segment_dl(img, model = "u2netp", type = "highlight")

  # 4. Zero-shot text-prompted instance segmentation (Grounded-SAM)
  # Segments leaves and lesions, displaying bounding boxes and summary
  res_gs <- image_segment_dl(
    img,
    model = "grounded-sam",
    prompt = "plant leaf, lesion",
    type = "highlight",
    bbox = TRUE
  )
  # Inspect counts and morphological measures
  print(attr(res_gs, "counts"))
  print(head(attr(res_gs, "features")))

  # 5. Ultra-fast pure object detection without mask generation
  # (Skips SAM 2.1 completely for 10x faster execution)
  boxes <- image_segment_dl(
    img,
    model = "grounded-sam",
    prompt = "insect, leaf",
    mask = FALSE,
    bbox = TRUE
  )

  # 6. One-shot visual exemplar segmentation (PerSAM):
  # Click on 1 or more representative items (e.g. seeds, logs, grains, cells)
```

```
  # to automatically find, outline, and count all matching instances:
  res_persam <- image_segment_dl(
    img,
    exemplar = TRUE,
    sim_threshold = 0.48,
    type = "highlight",
    bbox = TRUE
  )

  # 7. Multi-prototype visual exemplar segmentation with predefined coordinates:
  # Passing coordinates of distinct log types (light, dark, weathered)
  pts <- rbind(c(65, 380), c(425, 380), c(630, 380))
  res_logs <- image_segment_dl(
    img,
    exemplar = TRUE,
    prompt = pts,
    sim_threshold = 0.45,
    type = "highlight",
    engine = "gpu"
  )
  # 8. YOLO26 instance segmentation (real-time polygon mask & box detection)
  res_yolo <- image_segment_dl(
    img,
    model = "yolo26n-seg",
    type = "highlight",
    bbox = TRUE
  )

## End(Not run)
```

`image_segment_kmeans` *Image segmentation using k-means clustering*

Description

Segments image objects using clustering by the k-means clustering algorithm

Usage

```
image_segment_kmeans(
  img,
  bands = 1:3,
  nclasses = 2,
  invert = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
```

```
  fill_hull = FALSE,
  plot = TRUE
)
```

Arguments

`img`	An `image` object.
`bands`	A numeric integer/vector indicating the RGB band used in the segmentation. Defaults to `1:3`, i.e., all the RGB bands are used.
`nclasses`	The number of desired classes after image segmentation.
`invert`	Invert the segmentation? Defaults to `FALSE`. If `TRUE` the binary matrix is inverted.
`erode`, `dilate`, `opening`, `closing`, `filter`	**Morphological operations (brush size)** • `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground. • `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background. • `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects. • `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects. • `filter` performs median filtering in the binary image. Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges. Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.
`fill_hull`	Fill holes in the objects? Defaults to `FALSE`.
`plot`	Plot the segmented image?

Value

A list with the following values:

- `image` The segmented image considering only two classes (foreground and background)
- `clusters` The class of each pixel. For example, if `ncluster = 3`, `clusters` will be a two-way matrix with values ranging from 1 to 3. `masks` A list with the binary matrices showing the segmentation.

References

Hartigan, J. A. and Wong, M. A. (1979). Algorithm AS 136: A K-means clustering algorithm. Applied Statistics, 28, 100-108. doi:10.2307/2346830

Examples

```
if (interactive()) {
img <- image_pliman("la_leaves.jpg", plot = TRUE)
seg <- image_segment_kmeans(img)
seg <- image_segment_kmeans(img, fill_hull = TRUE, invert = TRUE, filter = 10)
}
```

---

image_segment_manual *Image segmentation by hand*

Description

This R code is a function that allows the user to manually segment an image based on the parameters provided. This only works in an interactive section.

Usage

```
image_segment_manual(
  img,
  shape = c("free", "circle", "rectangle"),
  type = c("select", "remove"),
  viewer = get_pliman_viewer(),
  resize = TRUE,
  edge = 5,
  plot = TRUE
)
```

Arguments

`img`	An `image` object.
`shape`	The type of shape to use. Defaults to ”free”. Other possible values are ”circle” and ”rectangle”. Partial matching is allowed.
`type`	The type of segmentation. By default (`type = "select"`) objects are selected. Use `type = "remove"` to remove the selected area from the image.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are ”base” and ”mapview”. If set to ”base”, the base R graphics system is used for interactive plotting. If set to ”mapview”, the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to ”mapview” for all functions.
`resize`	By default, the segmented object is resized to fill the original image size. Use `resize = FALSE` to keep the segmented object in the original scale.
`edge`	Number of pixels to add in the edge of the segmented object when `resize = TRUE`. Defaults to 5.
`plot`	Plot the segmented object? Defaults to `TRUE`.

Details

If the shape is "free", it allows the user to draw a perimeter to select/remove objects. If the shape is "circle", it allows the user to click on the center and edge of the circle to define the desired area. If the shape is "rectangle", it allows the user to select two points to define the area.

Value

A list with the segmented image and the mask used for segmentation.

Examples

```
if (interactive()) {
img <- image_pliman("la_leaves.jpg")
seg <- image_segment_manual(img)
plot(seg$mask)

}
```

---

image_segment_mask *Segment an* image *object using a brush mask*

Description

It combines make_mask() and make_brush() to segment an image object using a brush of desired size, shape, and position.

Usage

```
image_segment_mask(
  img,
  size,
  shape = "disc",
  rel_pos_x = 0.5,
  rel_pos_y = 0.5,
  type = c("binary", "shadow"),
  col_background = "white",
  plot = TRUE,
  ...
)
```

Arguments

img	A image object
size	A numeric containing the size of the brush in pixels. This should be an odd number; even numbers are rounded to the next odd one.
shape	A character vector indicating the shape of the brush. Can be "box", "disc", "diamond", "Gaussian" or "line" Defaults to "disc".

`rel_pos_x`, `rel_pos_y`
: A relative position to include the brush in the image. Defaults to 0.5. This means that the brush will be centered in the original image. Smaller values move the brush toward the left and top, respectively.

`type`
: Defines the type of the mask. By default, a binary mask is applied. This results in white pixels in the original image that matches the 0s pixels in the brush. If `type = "shadow"` is used, a shadow mask is produced

`col_background`
: Background color after image segmentation. Defaults to `"white"`.

`plot`
: Plots the generated mask? Defaults to `TRUE`.

`...`
: Additional arguments passed to `make_brush()`.

Value

A color `image` object

Examples

```
if (interactive()) {
img <- image_pliman("soybean_touch.jpg")
plot(img)
image_segment_mask(img, size = 601)
image_segment_mask(img,
                   size = 401,
                   shape = "diamond",
                   rel_pos_x = 0,
                   rel_pos_y = 0,
                   type = "shadow")
}
```

`image_segment_sahi` *Slicing Aided Hyper Inference (SAHI) for Instance Segmentation*

Description

Performs high-resolution sliding-window hyper inference for instance segmentation using `image_segment_dl()`. Slices large canvases, predicts masks per tile, shifts polygon contours to global coordinates, and applies cross-slice NMS.

Usage

```
image_segment_sahi(
  img,
  model = "yolo26n-seg",
  slice_height = 640,
  slice_width = 640,
  overlap_height_ratio = 0.2,
  overlap_width_ratio = 0.2,
```

```
  conf_threshold = 0.25,
  iou_threshold = 0.45,
  full_image_inference = TRUE,
  type = c("highlight", "segment", "mask"),
  col_background = "white",
  col_highlight = "salmon",
  alpha = 0.4,
  border = NA,
  lwd = 1,
  show_id = TRUE,
  rainbow = TRUE,
  plot = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An `image` object, file path, or 3D array.
`model`	YOLO ONNX model name or file path (e.g., `"yolo26n"`, `"yolov8s.onnx"`).
`slice_height`	Integer. Height of sliding window slices. Defaults to `640`.
`slice_width`	Integer. Width of sliding window slices. Defaults to `640`.
`overlap_height_ratio`	Numeric scalar between 0 and 0.5. Height overlap fraction. Defaults to `0.20`.
`overlap_width_ratio`	Numeric scalar between 0 and 0.5. Width overlap fraction. Defaults to `0.20`.
`conf_threshold`	Numeric scalar between 0 and 1. Minimum detection confidence. Defaults to `0.25`.
`iou_threshold`	Numeric scalar between 0 and 1. IoU threshold for cross-slice NMS. Defaults to `0.45`.
`full_image_inference`	Logical. If `TRUE` (default), runs an additional detection pass on the resized full canvas to capture objects larger than individual slices.
`type`	Type of presentation: `"highlight"` (default, overlays translucent colored masks on original image), `"segment"` (draws polygon borders), or `"mask"` (binary/label mask).
`col_background`	Color of background when `type = "segment"`. Defaults to `"white"`.
`col_highlight`	Base color for highlighted instances when `rainbow = FALSE`. Defaults to `"salmon"`.
`alpha`	Numeric transparency factor (0 = fully transparent, 1 = fully opaque) for highlight fill. Defaults to `0.40`.
`border`	Color for instance polygon borders. Defaults to `NA` (no border).

lwd	Numeric border line width. Defaults to 1.
show_id	Logical. Display instance index number at centroid. Defaults to TRUE.
rainbow	Logical. If TRUE (default), bounding boxes are rendered in rainbow colors.
plot	Logical. Display global detections overlaid on the full image. Defaults to TRUE.
verbose	Logical. Show progress bar and tile metrics. Defaults to TRUE.
...	Additional arguments passed to image_detect_dl().

Value

A data.frame of detections with attached global contours, counts, and summaries.

---

image_shp *Construct a shape file from an image*

Description

Creates a list of object coordinates given the desired number of nrow and columns. It starts by selecting 4 points at the corners of objects of interest in the plot space. Then, given nrow and ncol, a grid is drawn and the objects' coordinates are returned.

Usage

```
image_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  interactive = FALSE,
  viewer = get_pliman_viewer(),
  col_line = "red",
  size_line = 2,
  col_text = "red",
  size_text = 1,
  plot = TRUE
)
```

Arguments

img	An object of class image
nrow	The number of desired rows in the grid. Defaults to 1.
ncol	The number of desired columns in the grid. Defaults to 1.

`buffer_x`, `buffer_y`
: Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.

`interactive`
: If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.

`viewer`
: The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are `"base"` and `"mapview"`. If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

`col_line`, `col_text`
: The color of the line/text in the grid. Defaults to `"red"`.

`size_line`, `size_text`
: The size of the line/text in the grid. Defaults to `2.5`.

`plot`
: Plots the grid on the image? Defaults to `TRUE`.

Value

A list with `row * col` objects containing the plot coordinates.

Examples

```
if (interactive()) {
library(pliman)
flax <- image_pliman("flax_leaves.jpg")
shape <- image_shp(flax, nrow = 3, ncol = 5)
}
```

`image_skeleton_mat` *Medial Axis Tree (MAT) Topological Skeletonization*

Description

`image_skeleton_mat()` performs noise-resistant, topological Medial Axis Tree (MAT) skeletonization. Unlike conventional morphological erosion or iterative thinning algorithms (e.g., Guo-Hall, Zhang-Suen) which are prone to internal "ladder rungs", loops, and surface hair spurs on thick or textured objects, this algorithm:

1. Solidifies the binary mask by filling small cavities and applying morphological closing.

2. Computes the Exact Euclidean Distance Transform (EDT) for sub-pixel centerline tracing.
3. Applies fast thinning followed by radius-adaptive spur pruning (filtering boundary hairs proportional to local thickness).
4. Converts the skeleton to a topological graph and applies Maximum Spanning Forest (Kruskal's algorithm) to dissolve 100% of internal loops and ladder rungs.
5. Smooths centerlines along the medial ridge.

Usage

```
image_skeleton_mat(
  img,
  index = "GRAY",
  closing = 1,
  fill_size = 200,
  prune_length = 8,
  radius_prune = 1.2,
  dissolve_cycles = TRUE,
  smooth = TRUE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE,
  ...
)
```

Arguments

`img`	An `Image` object, a 2D binary matrix, or a list of images.
`index`	The index to use to binarize the image if `img` is not already binary. Default is `"GRAY"`.
`closing`	Integer radius (in pixels) for morphological closing before skeletonization (default: 1). Smooths bark roughness and bridges boundary fissures.
`fill_size`	Integer maximum area (in pixels) of enclosed cavities to fill (default: 200).
`prune_length`	Minimum branch length (in pixels) to retain (default: 8).
`radius_prune`	Numeric factor multiplied by local radius for adaptive spur pruning (default: 1.2).
`dissolve_cycles`	Logical. If `TRUE` (default), dissolves all loops and ladder rungs into an acyclic tree.
`smooth`	Logical. If `TRUE` (default), applies moving-average smoothing to skeleton paths.
`parallel`	Logical. If `TRUE`, processes multiple images in parallel.
`workers`	Number of workers for parallel execution.
`verbose`	Logical. If `TRUE`, displays progress.
`plot`	Logical. If `TRUE`, plots the resulting skeleton.
`...`	Additional arguments passed to `help_binary()` if `img` is not binary.

Value

A binary `Image` object (or list of `Image` objects) with values 0 and 1.

Examples

```
## Not run:
library(pliman)
img <- image_import(image_pliman("root.jpg"))
skel <- image_skeleton_mat(img, plot = TRUE)

## End(Not run)
```

---

image_square *Squares an image*

Description

Converts a rectangular image into a square image by expanding the rows/columns using `image_expand()`.

Usage

```
image_square(img, plot = TRUE, ...)
```

Arguments

`img`	An `image` object.
`plot`	Plots the extended image? defaults to `FALSE`.
`...`	Further arguments passed on to `image_expand()`.

Value

The modified `image` object.

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
dim(img)
square <- image_square(img)
dim(square)
}
```

`image_stardist_dl` *Star-Convex Object Detection with StarDist*

Description

Detects round, convex, and overlapping objects (e.g. seeds, cells, nuclei, spores) by predicting radial star-convex polygons and object probability using StarDist.

Usage

```
image_stardist_dl(
  img,
  model = "stardist",
  prob_threshold = 0.5,
  nms_threshold = 0.3,
  type = c("segment", "mask", "polygons"),
  col_highlight = "salmon",
  border = "white",
  lwd = 2,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`model`	Model name or path to a custom ONNX file. Defaults to `"stardist"`.
`prob_threshold`	Probability threshold for detecting object centers (default 0.5).
`nms_threshold`	Polygon IoU threshold for Non-Maximum Suppression (default 0.3).
`type`	Output type: `"segment"` (highlight overlay), `"mask"` (integer labels), or `"polygons"`.
`col_highlight`	Fill color for overlay polygons (default `"salmon"`).
`border`	Border color for polygon contours (default `"white"`).
`lwd`	Line width for polygon contours (default 2).
`threads`	Number of CPU threads (default 0 for auto-tuning).
`engine`	Execution engine: `"cpu"` or `"gpu"` (DirectML).
`device_id`	GPU device ID (default -1 for auto).

verbose	Logical. Whether to show progress messages.
plot	Logical. Whether to plot the results.
dir	Model directory (default pliman_model_dir()).
...	Additional arguments passed to graphics::plot().

Value

A list containing:

- boxes: Bounding boxes data frame with centers and scores.
- polygons_x, polygons_y: Coordinates of radial polygon vertices for each object.
- labels: Integer matrix with instance IDs.
- mask: Logical foreground mask.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("grains.png")
  res <- image_stardist_dl(img)

## End(Not run)
```

---

image_stitch *Native Planar Image Stitching and Panorama*

Description

image_stitch() merges multiple overlapping planar images (e.g. root scanner tiles, oversized plant leaves, bench scans, or drone orthomosaic patches) into a single seamless composite without requiring OpenCV, Python, or external software.

The native pipeline performs:

1. **Multi-Scale Normalized Cross-Correlation (NCC)**: Coarse-to-fine registration finding the exact sub-pixel horizontal and vertical shifts between adjacent tiles.
2. **Cosine-Smooth Seam Feathering**: Eliminates visible seam lines and exposure steps by applying non-linear cosine weighted transitions across the overlap zone.
3. **Dynamic Canvas Expansion**: Recursively chains two or more images into a single composite image array.

Usage

```
image_stitch(
  images,
  direction = c("horizontal", "vertical", "auto"),
  blend = TRUE,
  overlap_hint = 0.2,
  plot = FALSE,
  return_details = FALSE,
  verbose = TRUE
)
```

Arguments

images	A list of Image objects or a character vector of file paths.
direction	Stitching direction: "horizontal" (left-to-right), "vertical" (top-to-bottom), or "auto" (automatically chosen based on best correlation peak).
blend	Logical. If TRUE (default), applies smooth seam feathering across overlapping image bands. If FALSE, places images directly with a sharp cut.
overlap_hint	Expected fractional overlap between adjacent images (default: 0.2 = 20%).
plot	Logical. If TRUE, plots the stitched result.
return_details	Logical. If TRUE, returns a named list with the stitched image and registration diagnostics (shifts, NCC scores, overlap dimensions). If FALSE (default), returns the stitched Image directly.
verbose	Logical. If TRUE (default), displays progress messages.

Value

An Image object (or a named list if return_details = TRUE).

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
## Not run:
library(pliman)

# Stitch two overlapping leaf scans
imgs <- list(
  image_read("leaf_left.jpg"),
  image_read("leaf_right.jpg")
)
stitched <- image_stitch(imgs, direction = "horizontal")
plot(stitched)

## End(Not run)
```

`image_superres_dl` *4x Super-Resolution with Real-ESRGAN Compact*

Description

Upscales images by 4x using generative convolutional super-resolution with intelligent tiled overlapping to prevent GPU/CPU memory exhaustion.

Usage

```
image_superres_dl(
  img,
  model = "realesrgan-compact",
  scale = 4,
  tile_size = 256,
  tile_pad = 16,
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = TRUE,
  plot = TRUE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

`img`	An `Image` object, a 2D/3D numeric array, a path to an image file, or a list of images.
`model`	Model name or path to a custom ONNX file. Defaults to `"realesrgan-compact"`.
`scale`	Upscaling factor (default 4).
`tile_size`	Processing tile size in pixels (default 256).
`tile_pad`	Overlap margin in pixels for seamless blending (default 16).
`threads`	Number of CPU threads (default 0 for auto-tuning).
`engine`	Execution engine: `"cpu"` or `"gpu"` (DirectML).
`device_id`	GPU device ID (default -1 for auto).
`verbose`	Logical. Whether to show progress messages.
`plot`	Logical. Whether to plot the super-resolved image.
`dir`	Model directory (default `pliman_model_dir()`).
`...`	Additional arguments passed to `graphics::plot()`.

Value

An `Image` object with dimensions `(orig_w * 4) x (orig_h * 4) x 3`.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("low_res.png")
  img_hr <- image_superres_dl(img)

## End(Not run)
```

`image_thinning_guo_hall` *Perform Guo-Hall thinning on a binary image or list of binary images*

Description

This function performs the Guo-Hall thinning algorithm (Guo and Hall, 1989) on a binary image or a list of binary images.

Usage

```
image_thinning_guo_hall(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE,
  ...
)
```

Arguments

`img`	The binary image or a list of binary images to be thinned. It can be either a single binary image of class 'Image' or a list of binary images.
`parallel`	Logical, whether to perform thinning using multiple cores (parallel processing). If TRUE, the function will use multiple cores for processing if available. Default is FALSE.
`workers`	Integer, the number of workers (cores) to use for parallel processing. If NULL (default), it will use 40% of available cores.
`verbose`	Logical, whether to display progress messages during parallel processing. Default is TRUE.
`plot`	Logical, whether to plot the thinned images. Default is FALSE.
`...`	Additional arguments to be passed to `image_binary()` if `img` is not a binary image.

Value

If `img` is a single binary image, the function returns the thinned binary image. If `img` is a list of binary images, the function returns a list containing the thinned binary images.

References

Guo, Z., and R.W. Hall. 1989. Parallel thinning with two-subiteration algorithms. Commun. ACM 32(3): 359-373. doi:10.1145/62065.62074

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("potato_leaves.jpg", plot = TRUE)
image_thinning_guo_hall(img, index = "R", plot = TRUE)
}
```

---

image_to_mat *Convert an image to a data.frame*

Description

Given an object image, converts it into a data frame where each row corresponds to the intensity values of each pixel in the image.

Usage

```
image_to_mat(img, parallel = FALSE, workers = NULL, verbose = TRUE)
```

Arguments

`img`	An image object.
`parallel`	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 70% of available cores.
`workers`	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
`verbose`	If `TRUE` (default) a summary is shown in the console.

Value

A list containing three matrices (R, G, and B), and a data frame containing four columns: the name of the image in `image` and the R, G, B values.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf.jpg")
dim(img)
mat <- image_to_mat(img)
dim(mat[[1]])
}
```

---

image_transpose *Transpose an Image*

Description

Transposes the spatial dimensions (width and height) of an image while preserving multi-channel spatial structure, color modes, and metadata attributes.

Usage

```
image_transpose(img)

## S3 method for class 'image'
t(x)
```

Arguments

img, x	An image object or 2D/3D array.

Value

A transposed image object with swapped row/column dimensions.

---

image_view *Create an interactive map view of an image*

Description

This function allows users to interactively edit and analyze an image using mapview and mapedit packages.

Usage

```
image_view(
  img,
  object = NULL,
  r = 1,
  g = 2,
  b = 3,
  edit = FALSE,
  alpha = 0.7,
  attribute = "area",
  title = "Edit the image",
  show = c("rgb", "index"),
  index = "B",
  max_pixels = 1e+06,
  downsample = NULL,
  color_regions = custom_palette(),
  quantiles = c(0, 1),
  ...
)
```

Arguments

`img`	An `image` object.
`object`	(Optional). An object computed with `analyze_objects()`. If an object is informed, an additional layer is added to the plot, showing the contour of the analyzed objects, with a color gradient defined by `attribute`.
`r`, `g`, `b`	The layer for the Red, Green and Blue band, respectively. Defaults to `1`, `2`, and `3`.
`edit`	If `TRUE` enable editing options using `mapedit::editMap()`.
`alpha`	The transparency level of the rectangles' color (between 0 and 1).
`attribute`	The name of the quantitative variable in the `object_index` to be used for coloring the rectangles.
`title`	The title of the map view. Use to provide short orientations to the user.
`show`	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
`index`	The index to use for the index view. Defaults to "B".
`max_pixels`	integer > 0. Maximum number of cells to use for the plot. If `max_pixels < npixels(img)`, regular sampling is used before plotting.
`downsample`	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to `NULL`, which will find the best downsampling factor to approximate the `max_pixels` value.
`color_regions`	The color palette for displaying index values. Default is `custom_palette()`.
`quantiles`	the upper and lower quantiles used for color stretching. Set to `c(0, 1)`
`...`	Additional arguments to be passed to `downsample_fun`.

Value

An `sf` object, the same object returned by `mapedit::editMap()`.

Examples

```
if (interactive()) {
# Example usage:
img <- image_pliman("sev_leaf.jpg")
image_view(img)
}
```

---

image_watershed *Intelligent Morphological Watershed (Spanning Forest + Union-Find)*

Description

This function performs a morphological watershed segmentation on a binary image. It implements a highly optimized Spanning Forest algorithm coupled with a Disjoint-Set (Union-Find) data structure. This approach eliminates the need for slow iterative scans ($O(N^2)$ complexity), allowing the algorithm to execute in milliseconds even when resolving tens of thousands of distinct objects. It also features a mechanism to merge highly connected objects, effectively preventing over-segmentation.

Usage

```
image_watershed(
  img,
  sensitivity = "medium",
  tolerance = NULL,
  ext = 1,
  min_size = NULL,
  rel_size = NULL,
  fill_hull = FALSE,
  max_size = NULL
)
```

Arguments

`img`	A binary `image` object or a matrix/array.
`sensitivity`	Text ("low", "medium", "high", "extreme") or Number (e.g. 2.5). Automatically defines the best parameters for optimal precision. (Overrides manual adjustment of tolerance).

`tolerance`	Manual Fine-Tuning. If provided, overrides `sensitivity` and directly controls the merging threshold for connected components. Values < 1.0 are interpreted as relative tolerance (percentage of the peak height). Values $>= 1.0$ are interpreted as absolute tolerance in pixels (linear Euclidean distance).
`ext`	Pixel neighborhood radius (default = 1). Specifies the extension radius used during the regional maxima calculation and watershed flooding.
`min_size`	Minimum area (in pixels) for an object segment retained after watershed.
`rel_size`	Relative threshold as a fraction of the mean segment area (e.g. `0.1` for 10% of mean area).
`fill_hull`	Logical. Fill holes in the binary mask before watershed segmentation. Default is `FALSE`.
`max_size`	Maximum area (in pixels) for an object segment retained after watershed. Default is `NULL`.

Value

A grayscale `image` object containing the labels.

Examples

```
## Not run:
  library(pliman)
  bin <- image_pliman("soybean_touch.jpg") |> image_binary(index = "B")
  w_med <- image_watershed(bin$B)

## End(Not run)
```

`landmarks` *Create image landmarks*

Description

An interactive section where the user will be able to click on the image to select landmarks manually is open. With each mouse click, a point is drawn and an upward counter is shown in the console. After `n` counts or after the user press Esc, the interactive process is interrupted and a `data.frame` with the `x` and `y` coordinates for the landmarks is returned.

Usage

```
landmarks(
  img,
  n = Inf,
  viewer = get_pliman_viewer(),
  scale = NULL,
  calibrate = FALSE
)
```

Arguments

`img`	An `image` object.
`n`	The number of landmarks to produce. Defaults to `Inf`. In this case, landmarks are chosen up to the user press Esc.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`scale`	A known scale of the coordinate values. If `NULL` (default) `scale = 1` is used.
`calibrate`	A logical argument indicating whether a calibration step must be performed before picking up the landmarks. If so, `calibrate()` is called internally. Users must then select two points and indicate a known distance. A scale value will internally be computed and used in the correction of the coordinates (from pixels to the unit of the known distance).

Value

A `data.frame` with the `x` and `y`-coordinates from the landmarks.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
img <- image_pliman("potato_leaves.jpg")
x <- landmarks(img)
}
```

`landmarks_add` *Artificially inflates the number of landmarks*

Description

Interpolates supplementary landmarks that correspond to the mean coordinates of two adjacent landmarks.

Usage

```
landmarks_add(x, n = 3, smooth_iter = 0, plot = TRUE, nrow = NULL, ncol = NULL)
```

Arguments

x	A matrix, a data.frame a list of perimeter coordinates, often produced with object_contour(), landmarks(), or landmarks_regradi().
n	The number of iterations. Defaults to 3.
smooth_iter	The number of smoothing iterations to perform. This will smooth the perimeter of the interpolated landmarks using poly_smooth().
plot	Creates a plot? Defaults to TRUE.
ncol, nrow	The number of rows or columns in the plot grid when a list is used in x. Defaults to NULL, i.e., a square grid is produced.

Value

A Matrix of interpolated coordinates.

Examples

```
library(pliman)

# equally spaced landmarks
plot_polygon(contours[[4]])
ldm <- landmarks_regradi(contours[[4]], plot = FALSE)
points(ldm$coords, pch = 16)
segments(mean(ldm$coords[,1]),
         mean(ldm$coords[,2]),
         ldm$coords[,1],
         ldm$coords[,2])

ldm_add <- landmarks_add(ldm, plot = FALSE)
points(ldm_add, col = "red")
points(ldm$coords, pch = 16)

# smoothed version
ldm_add_smo <- landmarks_add(ldm, plot = FALSE, smooth_iter = 10)
lines(ldm_add_smo, col = "blue", lwd = 3)
```

---

landmarks_angle *Angles between landmarks*

Description

Computes the angle from two interlandmark vectors using the difference of their arguments using complex vectors (Claude, 2008).

Usage

```
landmarks_angle(x, unit = c("rad", "deg"))
```

Arguments

`x`	An object computed with `landmarks()`.
`unit`	The unit of the angle. Defaults to radian (rad). Use `unit = "deg"` to return the angles in degrees.

Value

A matrix with the angles for each landmark combination.

Note

Borrowed from Claude (2008), pp. 50

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
img <- image_pliman("potato_leaves.jpg")
x <- landmarks(img)
landmarks_angle(x)
}
```

`landmarks_dist` *Distances between landmarks*

Description

Computes the distance between two landmarks as the square root of the sum of the squared differences between each coordinate (Claude, 2008).

Usage

```
landmarks_dist(x)
```

Arguments

`x`	An object computed with `landmarks()`.

Value

A matrix with the distances for each landmark combination.

Note

Borrowed from Claude (2008), pp. 49

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
if(isTRUE(interactive())){
library(pliman)
img <- image_pliman("potato_leaves.jpg")
x <- landmarks(img)
landmarks_dist(x)
}
```

`landmarks_regradi` *Pseudolandmarks with equally spaced angles*

Description

Select `n` landmarks that are spaced with a regular sequence of angles taken between the outline coordinates and the centroid.

Usage

```
landmarks_regradi(
  x,
  n = 50,
  close = TRUE,
  plot = TRUE,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

`x`	A `matrix`, a `data.frame` a `list` of perimeter coordinates, often produced with `object_contour()`.
`n`	Number of points to be sampled. Defaults to 50.
`close`	Return a closed polygon? Defaults to `TRUE`.
`plot`	Create a plot? Defaults to `TRUE`.
`ncol`, `nrow`	The number of rows or columns in the plot grid when a `list` is used in `x`. Defaults to `NULL`, i.e., a square grid is produced.

Value

A list with the following objects:

- `pixindices`: Vector of radius indices.
- `radii`: Vector of sampled radii lengths.
- `Xc`: The centroid coordinate of `x` axis.
- `Yc`: The centroid coordinate of `y` axis.
- `coords`: Coordinates of sampled points arranged in a two-column matrix.

If `x` is a list, a list of objects described above is returned.

Note

Borrowed from Claude (2008), pp. 53

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Examples

```
library(pliman)
plot_polygon(contours[[1]])
ldm <- landmarks_regradi(contours)
```

`leading_zeros` *Add leading zeros to a numeric sequence*

Description

Add `n` leading zeros to a numeric sequence. This is useful to create a character vector to rename files in a folder.

Usage

```
leading_zeros(x, n = 3)
```

Arguments

`x`	A numeric vector or a list of numeric vectors.
`n`	The number of leading zeros to add. Defaults to `3`.

Value

A character vector or a list of character vectors.

Examples

```
library(pliman)
leading_zeros(1:5)
leading_zeros(list(a = 1:3,
                   b = 1:5),
              n = 2)
```

`line_on_halfplot` *Extract mid-lines from half-plots*

Description

For each polygon in an `sf` object, computes the line segment joining the midpoints of the longer pair of opposite edges (the "half-plot line").

Usage

```
line_on_halfplot(shapefile)
```

Arguments

`shapefile`	An `sf` object of polygons. Each geometry must be closed (first and last coordinate coincide) so that `st_coordinates(...)` yields a repeating start point.

Value

A `SpatVector` (from the **terra** package) of line geometries representing the half-plot midlines.

Examples

```
if(interactive()){
library(pliman)
shp <- shapefile_input( paste0(image_pliman(), "/soy_shape.rds"))
mosaic <- mosaic_input( paste0(image_pliman(), "/soy_dsm.tif"))
mosaic_plot(mosaic)
half <- line_on_halfplot(shp)
shapefile_plot(half, add = TRUE, col = "blue")

}
```

list_cameras *List Connected Video Cameras*

Description

Scans and lists all video capture devices (webcams, USB cameras, capture cards) available on the system, indicating device IDs and native camera names.

Usage

```
list_cameras(details = FALSE)
```

Arguments

details	Logical. If TRUE, also displays supported resolutions, framerates, and pixel formats for each device. Default is FALSE.

Value

A data frame containing camera id and friendly name, invisibly.

Examples

```
## Not run:
  # List available cameras
  list_cameras()

  # View detailed resolutions and formats
  list_cameras(details = TRUE)

## End(Not run)
```

make_brush *Makes a brush*

Description

Generates brushes of various sizes and shapes that can be used as structuring elements.

Usage

```
make_brush(size, shape = "disc", step = TRUE, sigma = 0.3, angle = 45)
```

Arguments

`size`	A numeric containing the size of the brush in pixels. This should be an odd number; even numbers are rounded to the next odd one.
`shape`	A character vector indicating the shape of the brush. Can be `"box"`, `"disc"`, `"diamond"`, `"Gaussian"` or `"line"` Defaults to `"disc"`.
`step`	Logical. Creates a binary brush if `TRUE`.
`sigma`	Standard deviation for Gaussian brush shape.
`angle`	Angle (in degrees) for line brush shape.

Value

A 2D matrix of 0s and 1s containing the desired brush.

Examples

```
if (interactive()) {
make_brush(size = 51) |> image()
make_brush(size = 51, shape = "diamond") |> image()
}
```

`make_mask` *Makes a mask in an image*

Description

Make a mask using an `image` object and a brush.

Usage

```
make_mask(img, brush, rel_pos_x = 0.5, rel_pos_y = 0.5, plot = TRUE)
```

Arguments

`img`	A `image` object
`brush`	An object created with `make_brush()`
`rel_pos_x`, `rel_pos_y`	A relative position to include the brush in the image. Defaults to 0.5. This means that the brush will be centered in the original image. Smaller values move the brush toward the left and top, respectively.
`plot`	Plots the generated mask? Defaults to `TRUE`.

Details

It applies a brush to an Image, selecting the `image` pixels that match the brush values equal to 1. The position of the brush in the original image is controlled by the relative positions x (`rel_pos_x`) and y (`rel_pos_y`) arguments. The size of the brush must be smaller or equal to the smaller dimension of `image`.

Value

A binary image with 0s and 1s.

Examples

```
if (interactive()) {
img <- image_pliman("soybean_touch.jpg")
make_mask(img, brush = make_brush(size = 201))
make_mask(img,
          brush = make_brush(size = 401, shape = "diamond"),
          rel_pos_x = 0.1,
          rel_pos_y = 0.8)
}
```

---

mean.image *Mean method for* image *objects*

Description

Mean method for image objects

Usage

```
## S3 method for class 'image'
mean(x, ..., na.rm = FALSE)
```

Arguments

x	An image object.
...	Additional arguments.
na.rm	Logical; should NAs be removed?

---

measure_disease *Performs plant disease measurements*

Description

- measure_disease() computes the percentage of symptomatic leaf area and (optionally) counts and compute shapes (area, perimeter, radius, etc.) of lesions in a sample or entire leaf using color palettes. See more at **Details**.
- measure_disease_iter() provides an iterative section for measure_disease(), where the user picks up samples in the image to create the needed color palettes.

Usage

```
measure_disease(
  img,
  img_healthy = NULL,
  img_symptoms = NULL,
  img_background = NULL,
  pattern = NULL,
  opening = c(10, 0),
  closing = c(0, 0),
  filter = c(0, 0),
  erode = c(0, 0),
  dilate = c(0, 0),
  parallel = FALSE,
  workers = NULL,
  resize = FALSE,
  fill_hull = TRUE,
  index_lb = NULL,
  index_dh = "GLI",
  has_white_bg = FALSE,
  threshold = NULL,
  invert = FALSE,
  lower_noise = 0.1,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  randomize = TRUE,
  nsample = 3000,
  watershed = FALSE,
  lesion_size = "medium",
  tolerance = NULL,
  extension = NULL,
  show_features = FALSE,
  show_segmentation = FALSE,
  plot = TRUE,
  show_original = TRUE,
  show_background = TRUE,
  show_contour = TRUE,
  contour_col = "white",
  contour_size = 1,
  col_leaf = NULL,
  col_lesions = NULL,
  col_background = NULL,
  marker = FALSE,
  marker_col = NULL,
  marker_size = NULL,
  save_image = FALSE,
  prefix = "proc_",
```

```
  name = NULL,
  dir_original = NULL,
  dir_processed = NULL,
  return_masks = FALSE,
  verbose = TRUE
)

measure_disease_iter(
  img,
  has_background = TRUE,
  r = 3,
  by_leaf = FALSE,
  viewer = get_pliman_viewer(),
  opening = c(10, 0),
  closing = c(0, 0),
  filter = c(0, 0),
  erode = c(0, 0),
  dilate = c(0, 0),
  show = "rgb",
  index = "NGRDI",
  ...
)
```

Arguments

`img`	The image to be analyzed.
`img_healthy`	A color palette of healthy tissues.
`img_symptoms`	A color palette of lesioned tissues.
`img_background`	A color palette of the background (if exists). These arguments can be either an `image` object stored in the global environment or a character value. If a chacarceter is used (eg., `img_healthy = "leaf"`), the function will search in the current working directory a valid image that contains "`leaf`" in the name. Note that if two images matches this pattern, an error will occour.
`pattern`	A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be analyzed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on.
`erode`, `dilate`, `opening`, `closing`, `filter`	**Morphological operations (brush size)** • `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground. • `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.

- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`parallel` Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. The number of sections is set up to 30% of available cores.

`workers` A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.

`resize` Resize the image before processing? Defaults to `FALSE`. Use a numeric value of range 0-100 (proportion of the size of the original image).

`fill_hull` Fill holes in the image? Defaults to `TRUE`. This is useful to fill holes in leaves, e.g., those caused by insect attack, ensuring the hole area will be accounted for the leaf, not background.

`index_lb` The index used to segment the foreground (e.g., leaf) from the background. If not declared, the entire image area (pixels) will be considered in the computation of the severity.

`index_dh` The index used to segment diseased from healthy tissues when `img_healthy` and `img_symptoms` are not declared. Defaults to `"GLI"`. See `image_index()` for more details.

`has_white_bg` Logical indicating whether a white background is present. If `TRUE`, pixels that have R, G, and B values equals to 1 will be considered as `NA`. This may be useful to compute an image index for objects that have, for example, a white background. In such cases, the background will not be considered for the threshold computation.

`threshold` By default (`threshold = NULL`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. Inform any non-numeric value different than "Otsu" to iteratively choose the threshold based on a raster plot showing pixel intensity of the index. Must be a vector of length 2 to indicate the threshold for `index_lb` and `index_dh`, respectively.

`invert` Inverts the binary image if desired. This is useful to process images with black background. Defaults to `FALSE`.

`lower_noise` By default, lesions with lesser than 10% of the mean area of all lesions are removed (`lower_noise = 0.1`). Increasing this value will remove larger

lesions. To define an explicit lower or upper size (in pixel unit), use the `lower_size` and `upper_size` arguments.

`lower_size` Lower limit for size for the image analysis. Leaf images often contain dirt and dust. To prevent dust from affecting the image analysis, the lower limit of analyzed size is set to 0.1, i.e., objects with lesser than 10% of the mean of all objects are removed. One can set a known area or use `lower_limit = 0` to select all objects (not advised).

`upper_size` Upper limit for size for the image analysis. Defaults to `NULL`, i.e., no upper limit used.

`topn_lower`, `topn_upper`
Select the top `n` lesions based on its area. `topn_lower` selects the `n` lesions with the smallest area whereas `topn_upper` selects the `n` lesions with the largest area.

`randomize` Randomize the lines before training the model? Defaults to `TRUE`.

`nsample` The number of sample pixels to be used in training step. Defaults to `3000`.

`watershed` If `TRUE` (Default) implements the Watershed Algorithm to segment lesions connected by a fairly few pixels that could be considered as two distinct lesions. If `FALSE`, lesions that are connected by any pixel are considered unique lesions. For more details see `image_watershed()`.

`lesion_size` The size of the lesion. Used to automatically tune `tolerance` and `extension` parameters. One of the following. `"small"` (2-5 mm in diameter, e.g, rust pustules), `"medium"` (0.5-1.0 cm in diameter, e.g, wheat leaf spot), `"large"` (1-2 cm in diameter, and `"elarge"` (2-3 cm in diameter, e.g, target spot of soybean).

`tolerance` The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest. Defaults to `NULL`, i.e., starting values are set up according to the argument `lesion_size`. Values < 1.0 are interpreted as relative tolerance (percentage of the peak height). Values $>= 1.0$ are interpreted as absolute tolerance in pixels (linear Euclidean distance).

`extension` Radius of the neighborhood in pixels for the detection of neighboring objects. Defaults to 20. Higher value smooths out small objects.

`show_features` If `TRUE` returnS the lesion features such as number, area, perimeter, and radius. Defaults to `FALSE`.

`show_segmentation`
Shows the object segmentation colored with random permutations. Defaults to `TRUE`.

`plot` Show image after processing? Defaults to `TRUE`.

`show_original` Show the symptoms in the original image?

`show_background`
Show the background? Defaults to `TRUE`. A white background is shown by default when `show_original = FALSE`.

`show_contour` Show a contour line around the lesions? Defaults to `TRUE`.

`contour_col`, `contour_size`
: The color and size for the contour line around objects. Defaults to `contour_col = "white"` and `contour_size = 1`.

`col_leaf` Leaf color after image processing. Defaults to `"green"`

`col_lesions` Symptoms color after image processing. Defaults to `"red"`.

`col_background`
: Background color after image processing. Defaults to `"NULL"`.

`marker`, `marker_col`, `marker_size`
: The type, color and size of the object marker. Defaults to `NULL`, which shows nothing. Use `marker = "point"` to show a point in each lesion or `marker = "*"` where `"*"` is any variable name of the `shape` data frame returned by the function.

`save_image` Save the image after processing? The image is saved in the current working directory named as `proc_*` where `*` is the image name given in `img`.

`prefix` The prefix to be included in the processed images. Defaults to `"proc_"`.

`name` The name of the image to save. Use this to overwrite the name of the image in `img`.

`dir_original`, `dir_processed`
: The directory containing the original and processed images. Defaults to `NULL`. In this case, the function will search for the image `img` in the current working directory. After processing, when `save_image = TRUE`, the processed image will be also saved in such a directory. It can be either a full path, e.g., `"C:/Desktop/imgs"`, or a subfolder within the current working directory, e.g., `"/imgs"`.

`return_masks` Returns the leaf/disease masks? Defaults to `FALSE`.

`verbose` If `TRUE` (default) a summary is shown in the console.

`has_background`
: A logical indicating if the image has a background to be segmented before processing.

`r` The radius of neighborhood pixels. Defaults to `2`. A square is drawn indicating the selected pixels.

`by_leaf` Compute the severity by leaf? If `TRUE`, `measure_disease_byl()` is called internallty and the severity is computed for each object (leaf) in the image. The background segmentation is then controlled by the argument `index`.

`viewer` The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

`show` The show option for the mapview viewer, either `"rgb"` or `"index"`.

`index` The index to be shown when `show = "rgb"`.

`...` Further parameters passed on to `measure_disease()`.

Details

In `measure_disease()`, a general linear model (binomial family) fitted to the RGB values is used to segment the lesions from the healthy leaf. If a pallet of background is provided, the function takes care of the details to isolate it before computing the number and area of lesions. By using `pattern` it is possible to process several images with common pattern names that are stored in the current working directory or in the subdirectory informed in `dir_original`.

If `img_healthy` and `img_symptoms` are not declared, RGB-based phenotyping of foliar disease severity is performed using the index informed in `index_lb` to first segment leaf from background and `index_dh` to segment diseased from healthy tissues.

`measure_disease_iter()` only run in an interactive section. In this function, users will be able to pick up samples of images to iteratively create the needed color palettes. This process calls `pick_palette()` internally. If `has_background` is TRUE (default) the color palette for the background is first created. The sample of colors is performed in each left-button mouse click and continues until the user press Esc. Then, a new sampling process is performed to sample the color of healthy tissues and then diseased tissues. The generated palettes are then passed on to measure_disease(). All the arguments of such function can be passed using the ... (three dots).

When `show_features = TRUE`, the function computes a total of 36 lesion features (23 shape features and 13 texture features). The Haralick texture features for each object based on a gray-level co-occurrence matrix (Haralick et al. 1979). See more details in `analyze_objects()`.

Value

- `measure_disease()` returns a list with the following objects:
  - `severity` A data frame with the percentage of healthy and symptomatic areas.
  - `shape`,`statistics` If `show_features = TRUE` is used, returns the shape (area, perimeter, etc.) for each lesion and a summary statistic of the results.
- `measure_disease_iter()` returns a list with the following objects:
  - `results` A list with the objects returned by `measure_disease()`.
  - `leaf` The color palettes for the healthy leaf.
  - `disease` The color palettes for the diseased leaf.
  - `background` The color palettes for the background.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf_nb.jpg")
healthy <- image_pliman("sev_healthy.jpg")
lesions <- image_pliman("sev_sympt.jpg")
image_combine(img, healthy, lesions, ncol = 3)
```

```
sev <-
 measure_disease(img = img,
                 img_healthy = healthy,
                 img_symptoms = lesions,
                 plot = TRUE)

# an interactive section
measure_disease_iter(img)
}
```

`measure_disease_byl` *Performs plant disease measurements by leaf*

Description

Computes the percentage of symptomatic leaf area using color palettes or RGB indexes by each leaf of an image. This allows, for example, processing replicates of the same treatment and obtaining the results for each replication with a single image. To do that, leaf samples are first splitten with `object_split()` and then, `measure_disease()` is applied to the list of leaves.

Usage

```
measure_disease_byl(
  img,
  index = "B",
  index_lb = "B",
  index_dh = "NGRDI",
  lower_size = NULL,
  watershed = TRUE,
  invert = FALSE,
  fill_hull = FALSE,
  opening = c(10, 0),
  closing = c(0, 0),
  filter = c(0, 0),
  erode = c(0, 0),
  dilate = c(0, 0),
  threshold = "Otsu",
  extension = NULL,
  tolerance = NULL,
  object_size = "large",
  img_healthy = NULL,
  img_symptoms = NULL,
  plot = TRUE,
  save_image = FALSE,
  dir_original = NULL,
```

```
  dir_processed = NULL,
  pattern = NULL,
  parallel = FALSE,
  workers = NULL,
  show_features = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	The image to be analyzed.
`index`	A character value specifying the target mode for conversion to binary to segment the leaves from background. Defaults to "B" (blue). See `image_index()` for more details. Personalized indexes can be informed as, e.g., `index = "R*G/B`.
`index_lb`	The index used to segment the foreground (e.g., leaf) from the background. If not declared, the entire image area (pixels) will be considered in the computation of the severity.
`index_dh`	The index used to segment diseased from healthy tissues when `img_healthy` and `img_symptoms` are not declared. Defaults to `"GLI"`. See `image_index()` for more details.
`lower_size`	To prevent dust from affecting object segmentation, objects with lesser than `10%` of the mean of all objects are removed. . One can set a known area or use `lower_limit = 0` to select all objects (not advised).
`watershed`	If `TRUE` (default) performs watershed-based object detection.
`invert`	Inverts the binary image if desired.
`fill_hull`	Fill holes in the binary image? Defaults to `FALSE`.
`opening`, `closing`, `filter`, `erode`, `dilate`	**Morphological operations (brush size)**, applied in hierarchical order.
`threshold`	The threshold method (`"Otsu"`, `"Adaptive"`, or numeric).
`extension`	Neighborhood radius for watershed segmentation.
`tolerance`	Minimum height for watershed segmentation.
`object_size`	The size of the object (`"small"`, `"medium"`, `"large"`, `"elarge"`). Used to automatically set up `tolerance` and `extension` parameters.
`img_healthy`	A color palette of healthy tissues.
`img_symptoms`	A color palette of lesioned tissues.
`plot`	Show image after processing?
`save_image`	Save the image after processing? The image is saved in the current working directory named as `proc_*` where `*` is the image name given in `img`.
`dir_original`, `dir_processed`	The directory containing the original and processed images. Defaults to `NULL`. In this case, the function will search for the image `img` in the current working directory. After processing, when `save_image = TRUE`,

	the processed image will be also saved in such a directory. It can be either a full path, e.g., `"C:/Desktop/imgs"`, or a subfolder within the current working directory, e.g., `"/imgs"`.
`pattern`	A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be analyzed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on.
`parallel`	Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. The number of sections is set up to 30% of available cores.
`workers`	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
`show_features`	If `TRUE` returnS the lesion features such as number, area, perimeter, and radius. Defaults to `FALSE`.
`verbose`	If `TRUE` (default) shows progress summary in console.
`...`	Additional arguments passed on to `measure_disease()`.

Value

- A list with the following objects:
  - `severity` A data frame with the percentage of healthy and symptomatic areas for each leaf in the image(s).
  - `shape,statistics` If `show_features = TRUE` is used, returns the shape (area, perimeter, etc.) for each lesion and a summary statistic of the results.

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("mult_leaves.jpg", plot = TRUE)
sev <-
 measure_disease_byl(img = img,
                     index_lb = "B",
                     index_dh = "NGRDI",
                     workers = 2)
sev$severity
}
```

`measure_disease_shp` *Measure disease using shapefiles*

Description

This function calls `measure_disease()` in each image polygon of a shapefile object generated with `image_shp()` and bind the results into read-ready data frames.

Usage

```
measure_disease_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  prepare = FALSE,
  viewer = "mapview",
  index_lb = "HUE2",
  index_dh = "NGRDI",
  pattern = NULL,
  threshold = NULL,
  invert = FALSE,
  dir_original = NULL,
  show_features = FALSE,
  interactive = FALSE,
  plot = TRUE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	The image to be analyzed. Either an image of class `image` or a character string containing the image name. In the last, the image will be searched in the root directory. Declare dir_original to inform a subfolder that contains the images to be processed.
`nrow`, `ncol`	The number of rows and columns to generate the shapefile. Defaults to `1`.
`buffer_x`, `buffer_y`	Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.

`prepare` Logical value indicating whether to prepare the image for analysis using `image_prepare()` function. This allows to align and crop the image before processing. Defaults to `FALSE`.

`viewer` The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

`index_lb` The index used to segment the foreground (e.g., leaf) from the background. If not declared, the entire image area (pixels) will be considered in the computation of the severity.

`index_dh` The index used to segment diseased from healthy tissues when `img_healthy` and `img_symptoms` are not declared. Defaults to `"GLI"`. See `image_index()` for more details.

`pattern` A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be analyzed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on.

`threshold` By default (`threshold = NULL`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. Inform any non-numeric value different than "Otsu" to iteratively choose the threshold based on a raster plot showing pixel intensity of the index. Must be a vector of length 2 to indicate the threshold for `index_lb` and `index_dh`, respectively.

`invert` Inverts the binary image if desired. This is useful to process images with black background. Defaults to `FALSE`.

`dir_original` The directory containing the original and processed images. Defaults to `NULL`. In this case, the function will search for the image `img` in the current working directory.

`show_features` If `TRUE` returnS the lesion features such as number, area, perimeter, and radius. Defaults to `FALSE`.

`interactive` If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.

`plot` Show image after processing? Defaults to `TRUE`.

`parallel` Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time, especially when `pattern` is used is informed. The number of sections is set up to 30% of available cores.

workers	A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.
verbose	If TRUE (default) a summary is shown in the console.
...	Aditional arguments passed on to measure_disease.

Value

An object of class plm_disease_byl. See more details in the Value section of measure_disease().

Examples

```
if (interactive()) {
# severity for the three leaflets (from left to right)
img <- image_pliman("mult_leaves.jpg", plot = TRUE)
sev <-
 measure_disease_shp(img = img,
                     nrow = 1,
                     ncol = 3,
                     index_lb = "B",
                     index_dh = "NGRDI")
sev$severity
}
```

---

measure_injury *Measures Injury in Images*

Description

The measures_injury function calculates the percentage of injury in images by performing binary segmentation and identifying lesions. It processes either a single image or a batch of images specified by a pattern in a directory.

Usage

```
measure_injury(
  img = NULL,
  pattern = NULL,
  index = "GRAY",
  threshold = "Otsu",
  invert = FALSE,
  opening = 5,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  filter_order = c("erode", "dilate", "opening", "closing", "filter", "fill_hull"),
  plot = TRUE,
  dir_original = NULL,
```

```
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)
```

Arguments

`img`	The image to be analyzed.
`pattern`	A pattern of file name used to identify images to be imported.
`index`	A character value specifying the index for conversion to binary image.
`threshold`	The threshold method (`"Otsu"`, `"Adaptive"`, or numeric).
`invert`	Inverts the binary image if desired.
`opening`, `closing`, `filter`, `erode`, `dilate`	**Morphological operations (brush size)**, applied in hierarchical order.
`filter_order`	Hierarchical order of morphological operations.
`plot`	Show image after processing?
`dir_original`	The directory containing the original and processed images. Defaults to NULL. In this case, the function will search for the image img in the current working directory.
`parallel`	If `TRUE` processes the images asynchronously in parallel.
`workers`	Number of parallel processes.
`verbose`	If `TRUE` (default) shows progress summary in console.

Details

The function processes each image by reading it, applying binary segmentation to detect lesions, filling the segmented areas, calculating the injury percentage, and optionally saving the processed image with highlighted lesions. In batch mode, it uses the provided pattern to identify images in the specified directory and can utilize parallel processing for efficiency.

Value

A numeric value representing the injury percentage for a single image, or a data frame with injury percentages for batch processing.

`model_to_int8` *Convert Installed ONNX Models to 8-Bit Precision (INT8)*

Description

`model_to_int8()` converts installed or custom ONNX neural network models to 8-bit integer (INT8) precision. Dynamic quantization reduces model disk size by up to 75% and accelerates CPU inference by 2–3x with virtually no loss in detection or segmentation accuracy.

When `model = NULL` (default), `model_to_int8()` scans the pliman model directory (`pliman_model_dir()`) and converts all installed FP32 `.onnx` models into `_int8.onnx` models. Existing INT8 models are skipped unless `overwrite = TRUE`.

Usage

```
model_to_int8(
  model = NULL,
  dir = pliman_model_dir(),
  weight_type = c("QUInt8", "QInt8"),
  overwrite = FALSE,
  verbose = TRUE
)

models_to_int8(
  model = NULL,
  dir = pliman_model_dir(),
  weight_type = c("QUInt8", "QInt8"),
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

`model`	Optional character vector of model name(s) or file path(s) to quantize. If `NULL` (default), all unquantized `.onnx` models in `dir` are converted.
`dir`	Directory where installed models are located. Defaults to `pliman_model_dir()`.
`weight_type`	Precision type: `"QUInt8"` (unsigned 8-bit, default) or `"QInt8"` (signed 8-bit).
`overwrite`	Logical. If `TRUE`, re-quantizes models even if an `_int8.onnx` version already exists. Defaults to `FALSE`.
`verbose`	Logical. Show progress messages and compression summary. Defaults to `TRUE`.

Value

A `data.frame` of class `pliman_quantize_summary` listing each model, original size, INT8 size, reduction percentage, and conversion status.

Examples

```
## Not run:
library(pliman)

# Convert a specific installed model
model_to_int8("capsulas_det.onnx")

# Convert all installed models in pliman_model_dir()
summary <- model_to_int8()
print(summary)

## End(Not run)
```

`mosaic_aggregate` *SpatRaster aggregation*

Description

Aggregate a SpatRaster to create a new SpatRaster with a lower resolution (larger cells), using the GDAL's gdal_translate utility https://gdal.org/programs/gdal_translate.html

Usage

```
mosaic_aggregate(
  mosaic,
  pct = 50,
  fun = "nearest",
  file = NULL,
  overwrite = FALSE,
  in_memory = TRUE
)
```

Arguments

`mosaic`	SpatRaster
`pct`	The size as a fraction (percentage) of the input image size. Either a scalar (eg., 50), or a length-two numeric vector. In the last, different percentage reduction/expansion can be used for columns, and rows, respectively.
`fun`	The resampling function. Defaults to `nearest`, which applies the nearest neighbor (simple sampling) resampler. Other accepted values are: 'average', 'rms', 'bilinear', 'cubic', 'cubicspline', 'lanczos', and 'mode'. See Details for a detailed explanation.
`file`	The file path (including filename and extension) where the aggregated raster should be saved. Defaults to `NULL`.
`overwrite`	Logical. If `TRUE`, overwrites `file` if it exists, or replaces the original raster file if `file = NULL` and `mosaic` is an in-disk `SpatRaster`. Defaults to `FALSE`.
`in_memory`	Wheter to return an 'in-memory' `SpatRaster`. If `FALSE`, the aggregated raster will be returned as an 'in-disk' object.

Value

SpatRaster

Examples

```
if (interactive()) {
library(pliman)
library(terra)
```

```
r <- rast()
values(r) <- 1:ncell(r)
r2 <- mosaic_aggregate(r, pct = 10)
opar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
mosaic_plot(r)
mosaic_plot(r2)
par(opar)
}
```

`mosaic_analyze` *Analyze a mosaic of remote sensing data*

Description

This function analyzes a mosaic of remote sensing data (UVAs or satellite imagery), extracting information from specified regions of interest (ROIs) defined in a shapefile or interactively drawn on the mosaic. It allows counting and measuring individuals (eg., plants), computing canopy coverage, and statistical summaries (eg., mean, coefficient of variation) for vegetation indices (eg, NDVI) at a block, plot, individual levels or even extract the raw results at pixel level.

Usage

```
mosaic_analyze(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
  crop_to_shape_ext = TRUE,
  grid = TRUE,
  nrow = 1,
  ncol = 1,
  plot_width = NULL,
  plot_height = NULL,
  layout = "lrtb",
  indexes = NULL,
  shapefile = NULL,
  basemap = NULL,
  build_shapefile = TRUE,
  check_shapefile = TRUE,
  buffer_edge = 1,
  buffer_col = 0,
  buffer_row = 0,
```

```
  segment_plot = FALSE,
  segment_individuals = FALSE,
  segment_pick = FALSE,
  mask = NULL,
  dsm = NULL,
  dsm_lower = 0.2,
  dsm_upper = NULL,
  dsm_window_size = c(5, 5),
  simplify = FALSE,
  map_individuals = FALSE,
  map_direction = c("horizontal", "vertical"),
  watershed = TRUE,
  tolerance = 1,
  extension = 1,
  include_if = "centroid",
  plot_index = "GLI",
  segment_index = NULL,
  in_memory = TRUE,
  threshold = "Otsu",
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  lower_noise = 0.15,
  lower_size = NULL,
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  summarize_fun = "mean",
  summarize_quantiles = NULL,
  attribute = NULL,
  invert = FALSE,
  color_regions = rev(grDevices::terrain.colors(50)),
  alpha = 1,
  max_pixels = 2e+06,
  downsample = NULL,
  quantiles = c(0, 1),
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

`mosaic` A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.

`r`, `g`, `b`, `re`, `nir`, `swir`, `tir`
The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image

is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.

`crop_to_shape_ext`
: Crop the mosaic to the extension of shapefile? Defaults to `TRUE`. This allows for a faster index computation when the region of the built shapefile is much smaller than the entire mosaic extension.

`grid`
: Logical, indicating whether to use a grid for segmentation (default: TRUE).

`nrow`
: Number of rows for the grid (default: 1).

`ncol`
: Number of columns for the grid (default: 1).

`plot_width`, `plot_height`
: The width and height of the plot shape (in the mosaic unit). It is mutually exclusiv with `buffer_col` and `buffer_row`.

`layout`
: Character: one of

- `'tblr'` for top/bottom left/right orientation
- `'tbrl'` for top/bottom right/left orientation
- `'btlr'` for bottom/top left/right orientation
- `'btrl'` for bottom/top right/left orientation
- `'lrtb'` for left/right top/bottom orientation
- `'lrbt'` for left/right bottom/top orientation
- `'rltb'` for right/left top/bottom orientation
- `'rlbt'` for right/left bottom/top orientation

`indexes`
: An optional `SpatRaster` object with the image indexes, computed with `mosaic_index()`.

`shapefile`
: An optional shapefile containing regions of interest (ROIs) for analysis.

`basemap`
: An optional basemap generated with `mosaic_view()`.

`build_shapefile`
: Logical, indicating whether to interactively draw ROIs if the shapefile is `NULL` (default: TRUE).

`check_shapefile`
: Logical, indicating whether to validate the shapefile with an interactive map view (default: TRUE). This enables live editing of the drawn shapefile by deleting or changing the drawn grids.

`buffer_edge`
: Width of the buffer around the shapefile (default: 5).

`buffer_col`, `buffer_row`
: Buffering factor for the columns and rows, respectively, of each individual plot's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the plot by 25% on each side.

`segment_plot`
: Logical, indicating whether to segment plots (default: FALSE). If `TRUE`, the `segment_index` will be computed, and pixels with values below the `threshold` will be selected.

segment_individuals
: Logical, indicating whether to segment individuals within plots (default: FALSE). If TRUE, the segment_index will be computed, and pixels with values below the threshold will be selected, and a watershed-based segmentation will be performed.

segment_pick
: When segment_plot or segment_individuals are TRUE, segment_pick allows segmenting background (eg., soil) and foreground (eg., plants) interactively by picking samples from background and foreground using mosaic_segment_pick()

mask
: An optional mask (SpatRaster) to mask the mosaic.

dsm
: A SpatRaster object representing the digital surface model. Must be a single-layer raster. If a DSM is informed, a mask will be derived from it using mosaic_chm_mask().

dsm_lower
: A numeric value specifying the lower height threshold. All heights greater than this value are retained.

dsm_upper
: An optional numeric value specifying the upper height threshold. If provided, only heights between lower and upper are retained.

dsm_window_size
: An integer (meters) specifying the window size (rows and columns, respectively) for creating a DTM using a moving window. Default is c(5, 5).

simplify
: Removes vertices in polygons to form simpler shapes. The function implementation uses the Douglas-Peucker algorithm using sf::st_simplify() for simplification.

map_individuals
: If TRUE, the distance between objects within plots is computed. The distance can be mapped either in the horizontal or vertical direction. The distances, coefficient of variation (CV), and mean of distances are then returned.

map_direction
: The direction for mapping individuals within plots. Should be one of "horizontal" or "vertical" (default).

watershed
: If TRUE (default), performs watershed-based object detection. This will detect objects even when they are touching one another. If FALSE, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.

tolerance
: The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest. Values < 1.0 are interpreted as relative tolerance (percentage of the peak height). Values $>= 1.0$ are interpreted as absolute tolerance in pixels (linear Euclidean distance).

extension
: Radius of the neighborhood in pixels for the detection of neighboring objects. A higher value smooths out small objects.

include_if Character vector specifying the type of intersection. Defaults to "centroid" (individuals in which the centroid is included within the drawn plot will be included in that plot). Other possible values include "covered", "overlap", and "intersect". See Details for a detailed explanation of these intersecting controls.

plot_index The index(es) to be computed for the drawn plots. Either a single vegetation index (e.g., "GLAI"), a vector of indexes (e.g., c("GLAI", "NGRDI", "HUE")), or a custom index based on the available bands (e.g., "(R-B)/(R+B)"). See pliman_indexes() and image_index() for more details.

segment_index The index used for segmentation. The same rule as plot_index. Defaults to NULL

in_memory Logical, indicating whether the indexes should be computed in memory. Defaults to TRUE. In most cases, this is 2-3 times faster, but errors can occur if mosaic is a large SpatRaster. If FALSE, raster algebra operations are performed on temporary files.

threshold By default (threshold = "Otsu"), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is provided, this value will be used as a threshold.

opening, closing, filter, erode, dilate
Morphological operations (brush size), applied in hierarchical order.

lower_noise Objects with area less than lower_noise * mean area are removed.

lower_size, upper_size
Lower and upper size limits for object filtering.

topn_lower, topn_upper
Select top n objects based on area.

summarize_fun The function to compute summaries for the pixel values. Defaults to "mean," i.e., the mean value of the pixels (either at a plot- or individual-level) is returned.

summarize_quantiles
quantiles to be computed when 'quantile' is on summarize_fun.

attribute The attribute to be shown at the plot when plot is TRUE. Defaults to the first summary_fun and first segment_index.

invert Logical, indicating whether to invert the mask. Defaults to FALSE, i.e., pixels with intensity greater than the threshold values are selected.

color_regions The color palette for regions (default: rev(grDevices::terrain.colors(50))).

alpha opacity of the fill color of the raster layer(s).

max_pixels Maximum number of pixels to render in the map or plot (default: 500000).

downsample Downsampling factor to reduce the number of pixels (default: NULL). In this case, if the number of pixels in the image (width x height) is greater than max_pixels a downsampling factor will be automatically chosen so that the number of plotted pixels approximates the max_pixels.

quantiles the upper and lower quantiles used for color stretching.

plot Logical, indicating whether to generate plots (default: TRUE).

verbose Logical, indicating whether to display verbose output (default: TRUE).

Details

Since multiple blocks can be analyzed, the length of arguments `grid`, `nrow`, `ncol`, `buffer_edge`, , `buffer_col`, `buffer_row`, `segment_plot`, `segment_i, ndividuals`, `includ_if`, `threshold`, `segment_index`, `invert`, `filter`, `threshold`, `lower_size`, `upper_size`, `watershed`, and `lower_noise`, can be either an scalar (the same argument applied to all the drawn blocks), or a vector with the same length as the number of drawn. In the last, each block can be analyzed with different arguments.

When `segment_individuals = TRUE` is enabled, individuals are included within each plot based on the `include_if` argument. The default value (`'centroid'`) includes an object in a given plot if the centroid of that object is within the plot. This makes the inclusion mutually exclusive (i.e., an individual is included in only one plot). If `'covered'` is selected, objects are included only if their entire area is covered by the plot. On the other hand, selecting `overlap` is the complement of `covered`; in other words, objects that overlap the plot boundary are included. Finally, when `intersect` is chosen, objects that intersect the plot boundary are included. This makes the inclusion ambiguous (i.e., an object can be included in more than one plot).

Value

A list containing the following objects:

- `result_plot`: The results at a plot level.
- `result_plot_summ`: The summary of results at a plot level. When `segment_individuals = TRUE`, the number of individuals, canopy coverage, and mean values of some shape statistics such as perimeter, length, width, and diameter are computed.
- `result_individ`: The results at an individual level.
- `map_plot`: An object of class `mapview` showing the plot-level results.
- `map_individual`: An object of class `mapview` showing the individual-level results.
- `shapefile`: The generated shapefile, with the drawn grids/blocks.

Examples

```
if (interactive()) {
library(pliman)
url <- "https://github.com/TiagoOlivoto/images/raw/master/pliman/rice_field/rice_ex.tif"
mosaic <- mosaic_input(url)
# Draw a polygon (top left, top right, bottom right, bottom left, top left)
# include 8 rice lines and one column
res <-
 mosaic_analyze(mosaic,
                r = 1, g = 2, b = 3,
                segment_individuals = TRUE,     # segment the individuals
                segment_index = "(G-B)/(G+B-R)",# index for segmentation
                filter = 4,
                nrow = 8,
                map_individuals = TRUE)
# map with individual results
res$map_indiv
}
```

`mosaic_analyze_iter` *Analyze mosaics iteratively*

Description

High-resolution mosaics can take a significant amount of time to analyze, especially when `segment_individuals = TRUE` is used in mosaic_analyze(). This is because the function needs to create in-memory arrays to segment individual using the watershed algorithm. This process utilizes a for-loop approach, iteratively analyzing each shape within the mosaic one at a time. To speed up processing, the function crops the original mosaic to the extent of the current shape before analyzing it. This reduces the resolution for that specific analysis, sacrificing some detail for faster processing.

Usage

```
mosaic_analyze_iter(
  mosaic,
  shapefile,
  basemap = NULL,
  r = 3,
  g = 2,
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
  plot = TRUE,
  verbose = TRUE,
  max_pixels = 3e+06,
  attribute = NULL,
  summarize_fun = "mean",
  segment_plot = FALSE,
  segment_individuals = FALSE,
  segment_index = "VARI",
  plot_index = "VARI",
  color_regions = rev(grDevices::terrain.colors(50)),
  alpha = 0.75,
  quantiles = c(0, 1),
  parallel = FALSE,
  workers = NULL,
  ...
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`shapefile`	An optional shapefile containing regions of interest (ROIs) for analysis.

basemap	An optional basemap generated with mosaic_view().
r, g, b, re, nir, swir, tir	The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
plot	Logical, indicating whether to generate plots (default: TRUE).
verbose	Logical, indicating whether to display verbose output (default: TRUE).
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).
attribute	The attribute to be shown at the plot when plot is TRUE. Defaults to the first summary_fun and first segment_index.
summarize_fun	The function to compute summaries for the pixel values. Defaults to "mean," i.e., the mean value of the pixels (either at a plot- or individual-level) is returned.
segment_plot	Logical, indicating whether to segment plots (default: FALSE). If TRUE, the segment_index will be computed, and pixels with values below the threshold will be selected.
segment_individuals	Logical, indicating whether to segment individuals within plots (default: FALSE). If TRUE, the segment_index will be computed, and pixels with values below the threshold will be selected, and a watershed-based segmentation will be performed.
segment_index	The index used for segmentation. The same rule as plot_index. Defaults to NULL
plot_index	The index(es) to be computed for the drawn plots. Either a single vegetation index (e.g., "GLAI"), a vector of indexes (e.g., c("GLAI", "NGRDI", "HUE")), or a custom index based on the available bands (e.g., "(R-B)/(R+B)"). See pliman_indexes() and image_index() for more details.
color_regions	The color palette for regions (default: rev(grDevices::terrain.colors(50))).
alpha	opacity of the fill color of the raster layer(s).
quantiles	the upper and lower quantiles used for color stretching.
parallel	If TRUE processes the images asynchronously in parallel.
workers	Number of parallel processes.
...	Further arguments passed on to mosaic_analyze()

Value

A list containing the following objects:

- result_plot: The results at a plot level.
- result_plot_summ: The summary of results at a plot level. When segment_individuals = TRUE, the number of individuals, canopy coverage, and mean values of some shape statistics such as perimeter, length, width, and diameter are computed.

- `result_individ`: The results at an individual level.
- `map_plot`: An object of class `mapview` showing the plot-level results.
- `map_individual`: An object of class `mapview` showing the individual-level results.

`mosaic_chm` *Calculate Canopy Height Model and Volume*

Description

This function calculates the canopy height model (CHM) and the volume for a given digital surface model (DSM) raster layer. Optionally, a digital terrain model (DTM) can be provided or interpolated using a set of points or a moving window.

Usage

```
mosaic_chm(
  dsm,
  dtm = NULL,
  points = NULL,
  interpolation = c("Tps", "Kriging"),
  window_size = c(5, 5),
  ground_quantile = 0,
  mask = NULL,
  mask_soil = TRUE,
  verbose = TRUE
)
```

Arguments

`dsm`	A `SpatRaster` object representing the digital surface model. Must be a single-layer raster.
`dtm`	(optional) A `SpatRaster` object representing the digital terrain model. Must be a single-layer raster. If not provided, it can be interpolated from points or created using a moving window.
`points`	(optional) An `sf` object representing sample points for DTM interpolation. If provided, `dtm` will be interpolated using these points.
`interpolation`	(optional) A character string specifying the interpolation method to use when `points` are provided. Options are "Kriging" (default) or "Tps" (Thin Plate Spline).
`window_size`	An integer (meters) specifying the window size (rows and columns, respectively) for creating a DTM using a moving window. Default is c(10, 10).

`ground_quantile`
: Numeric value between `0` and `1` indicating the quantile threshold for ground point selection in the CHM computation. Lower values (e.g., `0`) retain the lowest ground points, while higher values (e.g., `1`) consider higher ground elevations. Default is `0`, which uses the lowest points within each window.

`mask`
: (optional) A `SpatRaster` object used to mask the CHM and volume results. Default is NULL.

`mask_soil`
: Is `mask` representing a soil mask (eg., removing plants)? Default is TRUE.

`verbose`
: Return the progress messages. Default is TRUE.

Details

The function first checks if the input `dsm` is a valid single-layer `SpatRaster` object. If `dtm` is not provided, The function generates a Digital Terrain Model (DTM) from a Digital Surface Model (DSM) by downsampling and smoothing the input raster data. It iterates over the DSM matrix in windows of specified size, finds the minimum value within each window, and assigns these values to a downsampled matrix. After downsampling, the function applies a mean filter to smooth the matrix, enhancing the visual and analytical quality of the DTM. Afterwards, DTM is resampled with the original DSM.

If both `dsm` and `dtm` are provided, the function ensures they have the same extent and number of cells, resampling `dtm` if necessary. The CHM is then calculated as the difference between `dsm` and `dtm`, and the volume is calculated by multiplying the CHM by the pixel size. The results are optionally masked using the provided `mask`.

Value

A `SpatRaster` object with three layers: `dtm` (digital terrain model), `height` (canopy height model), and `volume`.

`mosaic_chm_extract` *Extracts height metrics and plot quality from a Canopy Height Model (CHM)*

Description

This function extracts height-related summary statistics from a CHM using a given shapefile.

Usage

```
mosaic_chm_extract(
  chm,
  shapefile,
  chm_threshold = NULL,
  quantiles = c(0, 0.05, 0.5, 0.95, 1)
)
```

Arguments

`chm`	An object computed with `mosaic_chm()`.
`shapefile`	An `sf` object containing the polygons over which height metrics are extracted.
`chm_threshold`	A numeric value representing the height threshold for calculating coverage. If `NULL`, coverage is not computed.
`quantiles`	A numeric vector specifying the quantiles to be extracted. Defaults to `c(0, 0.05, 0.5, 0.95, 1)`.

Value

An `sf` object containing height summary statistics for each plot, including:

- `min`: Minimum height value.
- `q05`: 5th percentile height value.
- `q50`: Median height value.
- `q95`: 95th percentile height value.
- `max`: Maximum height value.
- `mean`: Mean height value.
- `volume`: Total sum of heights multiplied by CHM resolution.
- `coverage`: If a mask is used in `mosaic_chm()` or `chm_threshold` is informed, returns the proportion of pixels covered within the plot. Otherwise, returns 1.

`mosaic_chm_mask` *Apply a height mask to CHM data*

Description

This function applies a height-based mask to a Canopy Height Model (CHM), focusing on areas with heights above a specified `lower` threshold and, optionally, below an `upper` threshold.

Usage

```
mosaic_chm_mask(
  dsm,
  lower,
  upper = NULL,
  window_size = c(5, 5),
  interpolation = "Tps"
)
```

Arguments

`dsm`	A `SpatRaster` object representing the digital surface model. Must be a single-layer raster.
`lower`	A numeric value specifying the lower height threshold. All heights greater than this value are retained.
`upper`	An optional numeric value specifying the upper height threshold. If provided, only heights between `lower` and `upper` are retained.
`window_size`	An integer (meters) specifying the window size (rows and columns, respectively) for creating a DTM using a moving window. Default is c(10, 10).
`interpolation`	(optional) A character string specifying the interpolation method to use when `points` are provided. Options are "Kriging" (default) or "Tps" (Thin Plate Spline).

Details

The `mosaic_chm` function, used internally, generates the DTM from the DSM by downsampling and smoothing raster data, applying a moving window to extract minimum values and then interpolating the results. The CHM is computed as the height difference between the DSM and DTM. This function calculates and applies a mask based on height thresholds.

Value

An `SpatRaster` object representing the masked CHM.

`mosaic_classify` *Classify a Mosaic Based on Index Breaks*

Description

This function classifies a given raster mosaic based on user-defined breaks. It provides an option to calculate the frequency and area of each class, as well as plot the classified mosaic.

Usage

```
mosaic_classify(mosaic, breaks, frequency = TRUE, plot = TRUE)
```

Arguments

`mosaic`	A `SpatRaster` object representing the mosaic to be classified.
`breaks`	A numeric vector specifying the breakpoints for classification.
`frequency`	Logical. If `TRUE`, computes the class frequency and area (in hectares).
`plot`	Logical. If `TRUE`, plots the classified mosaic.

Value

A list with two elements:

- `classified`: A `SpatRaster` object containing the classified mosaic.
- `class_freq`: A data frame containing class frequencies, areas (ha), and percentages (if `frequency = TRUE`).

Examples

```
if(interactive()){
library(pliman)
library(terra)

# Create an example raster
r <- terra::rast(matrix(runif(100, min = 0, max = 1), nrow=10, ncol=10))

# Classify the raster
result <- mosaic_classify(r, breaks = c(0.3, 0.6))

# View results
result$classified
result$class_freq
}
```

---

mosaic_clip *Clip Raster Mosaic by Polygons*

Description

Quickly partition a large raster mosaic into individual tiles using a polygon layer. Each tile is clipped by either the polygon's bounding box or (optionally) the exact feature geometry, and written to disk as a separate GeoTIFF named by the feature's `unique_id`.

Usage

```
mosaic_clip(
  mosaic,
  shapefile,
  unique_id = "unique_id",
  out_dir = NULL,
  overwrite = TRUE,
  verbose = TRUE,
  exact = FALSE,
  parallel = FALSE,
  workers = NULL
)
```

Arguments

`mosaic`	A terra::SpatRaster object or a file path pointing to a raster. In-memory rasters are first written to a temporary GeoTIFF.
`shapefile`	An sf::sf, terra::SpatVector, or path to a vector file. Must contain a column named `unique_id` for naming each output tile.
`unique_id`	A column present in `shapefile` that uniquely identifies the plots to be clipped.
`out_dir`	Directory where clipped rasters will be saved. Defaults to the current working directory. Created recursively if it does not exist.
`overwrite`	Logical; if `TRUE` (the default), existing files in `out_dir` with the same name will be overwritten.
`verbose`	Logical; if `TRUE` (default), progress bars and status messages will be shown.
`exact`	Logical; if `FALSE` (default), tiles are cropped by each feature's bounding box. If `TRUE`, the function extracts each polygon as a cutline for an exact crop (slower, but shape-accurate).
`parallel`	Logical; if `TRUE` (default), processing is parallelized using `mirai`. Set to `FALSE` for purely sequential execution.
`workers`	Integer; number of parallel daemons to launch when `parallel = TRUE`. Defaults to 70% of available cores.

Details

Clip a Raster Mosaic by Polygons

This function wraps GDAL's `warp` utility for efficient raster clipping. When `parallel = TRUE`, it will spawn multiple workers via `mirai` and process tiles in batches. Use `exact = TRUE` to clip to the true polygon shape (at some extra cost), or leave `exact = FALSE` for a faster bounding-box crop.

Value

Invisibly returns a character vector of file paths to all clipped GeoTIFFs.

`mosaic_crop` *Crop or Mask a Mosaic Raster*

Description

This function allows cropping of a raster mosaic interactively or programmatically:

- **Interactive Mode**: If neither `shapefile` nor `mosaic2` is provided, an interactive map is shown via `mosaic_view()`, allowing users to draw a rectangle to define the cropping area.

- **Shapefile Mode**: If a `SpatVector` is provided in `shapefile`, cropping or masking is performed based on its extent or exact shape, optionally with a buffer.
- **Raster Mode**: If `mosaic2` is provided, `mosaic` will be cropped to match the extent of `mosaic2`.

For disk-based mosaics, cropping with shapefiles uses GDAL (`sf::gdal_utils()`) to improve efficiency.

Usage

```
mosaic_crop(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  shapefile = NULL,
  in_memory = FALSE,
  mosaic2 = NULL,
  buffer = 0,
  show = c("rgb", "index"),
  index = "R",
  max_pixels = 5e+05,
  downsample = NULL,
  type = c("crop", "mask"),
  ...
)
```

Arguments

`mosaic`	A `SpatRaster` object to be cropped.
`r`, `g`, `b`, `re`, `nir`	Integer indices representing the red, green, blue, red-edge, and near-infrared bands of the input mosaic. Default assumes BGR format (b = 1, g = 2, r = 3).
`shapefile`	An optional `SpatVector` (or `sf` object) to use as cropping/masking geometry. Can be created interactively with `shapefile_input()`.
`in_memory`	Logical. If `TRUE`, raster processing will occur entirely in memory using `terra`. If `FALSE` (default), disk-based processing with GDAL will be used when appropriate.
`mosaic2`	A second `SpatRaster` whose extent will be used to crop `mosaic`.
`buffer`	A numeric value indicating a buffer (in CRS units) to apply around the shapefile geometry.
`show`	A character value indicating what to display in the interactive viewer. Either `"rgb"` or `"index"`.
`index`	The index to show if `show = "index"`. Default is `"R"`.
`max_pixels`	Maximum number of pixels to render in the interactive viewer.

`downsample`	Optional downsampling factor for display purposes.
`type`	Either `"crop"` (default) or `"mask"`: • `"crop"` crops the mosaic to the bounding box of the shapefile. • `"mask"` sets pixels outside the shapefile geometry to `NA` (recommended when using exact shapes).
`...`	Additional arguments passed to `mosaic_view()`.

Details

Crop or mask a `SpatRaster` object (`mosaic`) based on user input from an interactive map or by using a provided shapefile or another raster.

Value

A cropped or masked `SpatRaster` object.

Examples

```
if (interactive()) {
  library(pliman)
  # Load a sample raster
  mosaic <- mosaic_input(system.file("ex/elev.tif", package = "terra"))

  # Interactive cropping with drawn rectangle
  cropped <- mosaic_crop(mosaic)

  # View result
  mosaic_view(cropped)
}
```

---

mosaic_draw *Drawing Lines or Polygons with Raster Information*

Description

Drawing Lines or Polygons with Raster Information

Usage

```
mosaic_draw(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  index = "NGRDI",
  show = "rgb",
```

```
  segment = FALSE,
  viewer = c("mapview", "base"),
  threshold = "Otsu",
  invert = FALSE,
  summarize_fun = NULL,
  buffer = 2,
  color_regions = rev(grDevices::terrain.colors(50)),
  alpha = 1,
  max_pixels = 1e+06,
  downsample = NULL,
  quantiles = c(0, 1),
  plot = TRUE,
  plot_layout = c(1, 2, 3, 3)
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`r`, `g`, `b`, `re`, `nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
`index`	The index to use for the index view. Defaults to "B".
`show`	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
`segment`	Should the raster object be segmented? If set to `TRUE`, pixels within each polygon/rectangle will be segmented based on the `threshold` argument.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`threshold`	By default (threshold = "Otsu"), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold.
`invert`	Inverts the mask if desired. Defaults to `FALSE`.
`summarize_fun`	An optional function or character vector. When `summarize_fun = "mean"`, the mean values of `index` are calculated within each object. For more details on available functions, refer to `exactextractr::exact_extract()`.
`buffer`	Adds a buffer around the geometries of the SpatVector created. Note that the distance unit of `buffer` will vary according to the CRS of `mosaic`.
`color_regions`	The color palette for displaying index values. Defaults to `rev(grDevices::terrain.colors(50)`

`alpha`	opacity of the fill color of the raster layer(s).
`max_pixels`	Maximum number of pixels to render in the map or plot (default: 500000).
`downsample`	Downsampling factor to reduce the number of pixels (default: NULL). In this case, if the number of pixels in the image (width x height) is greater than `max_pixels` a downsampling factor will be automatically chosen so that the number of plotted pixels approximates the `max_pixels`.
`quantiles`	the upper and lower quantiles used for color stretching.
`plot`	Plots the draw line/rectangle? Defaults to `TRUE`.
`plot_layout`	The de plot layout. Defaults to `plot_layout = c(1, 2, 3, 3)`. Ie., the first row has two plots, and the second row has one plot.

Details

The `mosaic_draw` function enables you to create mosaic drawings from remote sensing data and compute vegetation indices.

- If a line is drawn using the "Draw Polyline" tool, the profile of `index` is displayed on the y-axis along the line's distance, represented in meter units. It is important to ensure that the Coordinate Reference System (CRS) of `mosaic` has latitude/longitude units for accurate distance representation.
- If a rectangle or polygon is drawn using the "Draw Rectangle" or "Draw Polygon" tools, the `index` values are calculated for each object. By default, the raw data is returned. You can set the `summarize_fun` to compute a summary statistic for each object.

Value

An invisible list containing the mosaic, draw_data, distance, distance_profile, geometry, and map.

Examples

```
if (interactive()) {
library(pliman)
# Load a raster showing the elevation of Luxembourg
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))

# draw a polyline to see the elevation profile along the line
mosaic_draw(mosaic, buffer = 1500)
}
```

`mosaic_epsg` *Determine EPSG Code for a Mosaic*

Description

This function calculates the EPSG code for a given mosaic based on its geographic extent.

Usage

```
mosaic_epsg(mosaic)
```

Arguments

`mosaic`	A raster object representing the mosaic for which the EPSG code is to be determined.

Details

The function calculates the centroid of the mosaic's extent, determines the UTM zone based on the centroid's longitude, and identifies the hemisphere based on the centroid's latitude. The EPSG code is then constructed accordingly.

Value

A character string representing the EPSG code corresponding to the UTM zone and hemisphere of the mosaic's centroid. If the mosaic is not in the lon/lat coordinate system, a warning is issued.

Examples

```
if (interactive()) {
library(pliman)
library(terra)

# Create a sample mosaic
mosaic <- rast(nrow=10, ncol=10, xmin=-120, xmax=-60, ymin=30, ymax=60)

# Get the EPSG code for the mosaic
mosaic_epsg(mosaic)
}
```

`mosaic_extract` *Extract Values from a Raster Mosaic Using a Shapefile*

Description

`mosaic_extract()` (and its alias `extract_cpp()`) extracts pixel values or summary statistics from a `terra::SpatRaster` object using an `sf` or `SpatVector` shapefile as spatial reference. It runs on a highly optimized C++ engine with OpenMP multi-threading and exact coverage calculations, completely independent of external GDAL libraries or exactextractr. For rasters on disk, it dynamically monitors available system RAM via native C++ API (`get_free_ram()`) and automatically selects between full in-memory processing and memory-safe spatial block processing (when raster size exceeds 80% of available RAM).

Usage

```
mosaic_extract(
  mosaic,
  shapefile,
  fun = NULL,
  return = "df",
  exact = FALSE,
  coverage_area = FALSE,
  subdiv = 5L,
  summarize_quantiles = c(0.05, 0.975),
  verbose = FALSE,
  max_cells_in_memory = NULL,
  ...
)

extract_cpp(
  mosaic,
  shapefile,
  fun = NULL,
  return = "df",
  exact = FALSE,
  coverage_area = FALSE,
  subdiv = 5L,
  summarize_quantiles = c(0.05, 0.975),
  verbose = FALSE,
  max_cells_in_memory = NULL,
  ...
)
```

Arguments

`mosaic`	A `SpatRaster` object (in memory or on disk).

shapefile	An sf data frame, sfc geometry list, or SpatVector object defining regions.
fun	Optional summary function(s) to compute for each polygon: NULL (default, returns raw pixel values list), "mean", "median", "sum", "min", "max", "sd", "count", "quantiles" (or "quantile"), or a vector of functions such as c("mean", "quantiles").
return	Character string specifying the return object type when statistics are computed: "df" (default, returns summary data frame) or "shapefile" (returns original shapefile object with extracted columns bound to it).
exact	Logical. If TRUE, computes exact pixel coverage area fractions. Defaults to FALSE.
coverage_area	Logical. Include polygon coverage area metrics (covered_area, plot_area, coverage) when computing summary statistics, or include raw coverage_area column when fun = NULL? Defaults to FALSE.
subdiv	Integer subdivision grid size per dimension when exact = TRUE. Defaults to 5.
summarize_quantiles	Numeric vector of quantile probabilities (values between 0 and 1) to compute when "quantiles" (or "quantile") is included in fun. Defaults to c(0.05, 0.975).
verbose	Logical. Displays progress steps and status messages? Defaults to FALSE.
max_cells_in_memory	Optional override for maximum cell count. Defaults to NULL (dynamically computed based on 80% of available system RAM).
...	Additional arguments.

Value

- If fun = NULL, a list of data.frames (one for each polygon) containing columns cell, row, col, and raster layer values (plus coverage_area if coverage_area = TRUE).
- If fun is specified and return = "shapefile", returns the **original shapefile** (sf or SpatVector) with the extracted summary columns attached (including covered_area, plot_area, coverage if coverage_area = TRUE).
- If fun is specified and return = "df" (default), returns a data.frame of summary statistics (including covered_area, plot_area, coverage if coverage_area = TRUE).

Examples

```
## Not run:
library(pliman)
shp <- shapefile_input( paste0(image_pliman(), "/soy_shape.rds"))
mosaic <- mosaic_input( paste0(image_pliman(), "/soy_ortho.tif"))

# Extract summary statistics attached to shapefile
mosaic_extract(mosaic, shp, fun = "mean", return = "shapefile")

# Extract mean and quantiles with plot area coverage metrics
```

```
mosaic_extract(mosaic, shp, fun = c("mean", "quantiles"), coverage_area = TRUE)

## End(Not run)
```

`mosaic_hist` *A wrapper around terra::hist()*

Description

Create a histogram of the values of a `SpatRaster`.

Usage

```
mosaic_hist(mosaic, layer, ...)
```

Arguments

`mosaic`	SpatRaster
`layer`	positive integer or character to indicate layer numbers (or names). If missing, all layers are used
`...`	Further arguments passed on to `terra::hist()`.

Value

A `NULL` object

Examples

```
if (interactive()) {
library(pliman)
r <- mosaic_input(system.file("ex/elev.tif", package="terra"))
mosaic_hist(r)
}
```

`mosaic_index` *Mosaic Index*

Description

Compute or extract an index layer from a multi-band mosaic raster.

Usage

```
mosaic_index(
  mosaic,
  index = "NGRDI",
  r = 3,
  g = 2,
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
  plot = TRUE,
  in_memory = TRUE,
  output = c("memory", "disk"),
  workers = 1,
  verbose = TRUE
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`index`	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use `pliman_indexes_rgb()` and `pliman_indexes_me()` to see the available RGB and multispectral indexes, respectively. Users can also calculate their own index using `R, G, B, RE, NIR, SWIR, and TIR` bands (eg., `index = "R+B/G"`) or using the names of the mosaic's layers (ex., "(band_1 + band_2) / 2").
`r`, `g`, `b`, `re`, `nir`, `swir`, `tir`	The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
`plot`	Plot the computed index? Defaults to `TRUE`.
`in_memory`	Logical, indicating whether the indexes should be computed in memory. Defaults to `TRUE`. In most cases, this is 2-3 times faster, but errors can occur if `mosaic` is a large `SpatRaster`. If `FALSE`, raster algebra operations are performed on temporary files.
`output`	Character(1), either `"memory"` or `"disk"`. If `"memory"`, the function returns a `terra::SpatRaster` object assembled in memory. If `"disk"`, each index layer is written out to a temporary GeoTIFF and the function returns a `terra::SpatRaster` object that points to those rasters. Default is `"memory"`.
`workers`	numeric. The number of workers you want to use for parallel processing when computing multiple indexes.
`verbose`	Whether to display progress messages.

Details

This function computes or extracts an index layer from the input mosaic raster based on the specified index name. If the index is not found in the package's predefined index list (see `image_index()` for more details), it attempts to compute the index using the specified band indices. The resulting index layer is returned as an `SpatRaster` object.

Value

An index layer extracted/computed from the mosaic raster.

Examples

```
if (interactive()) {
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
names(mosaic)
elev2 <- mosaic_index(mosaic, "elevation * 5", plot = FALSE)
oldpar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))

mosaic_plot(mosaic)
mosaic_plot(elev2)

# return the original parameters
par(oldpar)
}
```

---

mosaic_index2 *Mosaic Index with GDAL*

Description

Compute or extract an index layer from a multi-band mosaic raster using gdal_calc.py (https://gdal.org/programs/gdal_calc.html). This requires a Python and GDAL installation.

Usage

```
mosaic_index2(
  mosaic,
  index = "B",
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  plot = TRUE,
```

```
  python = Sys.which("python.exe"),
  gdal = Sys.which("gdal_calc.py")
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
index	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use pliman_indexes_rgb() and pliman_indexes_me() to see the available RGB and multispectral indexes, respectively. Users can also calculate their own index using R, G, B, RE, NIR, SWIR, and TIR bands (eg., index = "R+B/G") or using the names of the mosaic's layers (ex., "(band_1 + band_2) / 2").
r, g, b, re, nir	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
plot	Plot the computed index? Defaults to TRUE.
python	The PATH for python.exe
gdal	The PATH for gdal_calc.py

Value

An index layer extracted/computed from the mosaic raster.

Examples

```
if(interactive() & (Sys.which('python.exe') != '' ) & (Sys.which('gdal_calc.py') != '' )){
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
names(mosaic) <- "R"
elev2 <- mosaic_index2(mosaic, "R * 5", plot = FALSE)
oldpar <- par(no.readonly=TRUE)
mosaic_plot(mosaic)
mosaic_plot(elev2)
par(mfrow=c(1,2))
}
```

---

mosaic_input *Create and Export mosaics*

Description

Create and Export mosaics

Usage

```
mosaic_input(
  mosaic,
  mosaic_pattern = NULL,
  info = TRUE,
  check_16bits = FALSE,
  check_datatype = FALSE,
  ...
)

mosaic_export(mosaic, filename, datatype = NULL, overwrite = FALSE, ...)
```

Arguments

mosaic	• For `mosaic_input()`, a file path to the raster to imported, a matrix, array or a list of `SpatRaster` objects. • For `mosaic_export()`, an `SpatRaster` object.
mosaic_pattern	A pattern name to import multiple mosaics into a list.
info	Print the mosaic informations (eg., CRS, extent). Defaults to `TRUE`
check_16bits	Checks if mosaic has maximum value in the 16-bits format (65535), and replaces it by NA. Defaults to `FALSE`.
check_datatype	Logical. If `TRUE`, checks and suggests the appropriate data type based on the raster values.
...	Additional arguments passed to `terra::rast()` (`mosaic_input()`) or `terra::writeRaster()` (`mosaic_output()`)
filename	character. The Output filename.
datatype	The datatype. By default, the function will try to guess the data type that saves more memory usage and file size. See `terra::writeRaster()` and `terra::datatype()` for more details.
overwrite	logical. If `TRUE`, filename is overwritten.

Details

- `mosaic_input()` is a simply wrapper around `terra::rast()`. It creates a `SpatRaster` object from scratch, from a filename, or from another object.
- `mosaic_export()` is a simply wrapper around `terra::writeRaster()`. It write a `SpatRaster` object to a file.

Value

- `mosaic_input()` returns an `SpatRaster` object.
- `mosaic_export()` do not return an object.

Examples

```
if (interactive()) {
library(pliman)

# create an SpatRaster object based on a matrix
x <- system.file("ex/logo.tif", package="terra")
rast <- mosaic_input(x)
mosaic_plot(rast)

# create a temporary filename for the example
f <- file.path(tempdir(), "test.tif")
mosaic_export(rast, f, overwrite=TRUE)
list.files(tempdir())
}
```

---

mosaic_interpolate *Mosaic interpolation*

Description

Performs the interpolation of points from a raster object.

Usage

```
mosaic_interpolate(mosaic, points, method = c("bilinear", "loess", "idw"))
```

Arguments

`mosaic`	An `SpatRaster` object
`points`	An `sf` object with the points for x and y coordinates, usually obtained with `shapefile_build()`. Alternatively, an external shapefile imported with `shapefile_input()` containing the x and y coordinates can be used. The function will handle most used shapefile formats (eg., .shp, .rds) and convert the imported shapefile to an sf object.
`method`	One of "bilinear" (default), "loess" (local regression) or "idw" (Inverse Distance Weighting).

Value

An `SpatRaster` object with the same extent and crs from `mosaic`

mosaic_lonlat2epsg *Project a Mosaic from Lon/Lat to EPSG-based CRS*

Description

This function projects a given mosaic from the lon/lat coordinate system to an EPSG-based CRS determined by the mosaic's extent.

Usage

```
mosaic_lonlat2epsg(mosaic)
```

Arguments

mosaic	A raster object representing the mosaic to be projected. The mosaic must be in the lon/lat coordinate system.

Value

A raster object representing the projected mosaic. If the mosaic is not in the lon/lat coordinate system, a warning is issued.

Examples

```
if (interactive()) {
library(terra)
library(pliman)

# Create a sample mosaic
mosaic <- rast(nrow=10, ncol=10, xmin=-120, xmax=-60, ymin=30, ymax=60)

# Project the mosaic to the appropriate UTM zone
mosaic_lonlat2epsg(mosaic)
}
```

mosaic_plot *A wrapper around terra::plot()*

Description

Plot the values of a SpatRaster

Usage

```
mosaic_plot(
  mosaic,
  col = custom_palette(c("red", "yellow", "forestgreen"), n = 200),
  smooth = TRUE,
  ...
)
```

Arguments

mosaic	SpatRaster
col	character vector to specify the colors to use. Defaults to `custom_palette(c("red", "yellow", "forestgreen"))`.
smooth	logical. If TRUE (default) the cell values are smoothed (only if a continuous legend is used).
...	Further arguments passed on to `terra::plot()`.

Value

A `NULL` object

Examples

```
if (interactive()) {
library(pliman)
r <- mosaic_input(system.file("ex/elev.tif", package="terra"))
mosaic_plot(r)
}
```

---

mosaic_plot_rgb *A wrapper around terra::plotRGB()*

Description

Plot the RGB of a SpatRaster

Usage

```
mosaic_plot_rgb(mosaic, ...)
```

Arguments

mosaic	SpatRaster
...	Further arguments passed on to `terra::plotRGB()`.

Value

A `NULL` object

`mosaic_prepare` *Prepare a mosaic*

Description

Prepare an `SpatRaster` object to be analyzed in pliman. This includes cropping the original mosaic, aligning it, and cropping the aligned object. The resulting object is an object of class `image` that can be further analyzed.

Usage

```
mosaic_prepare(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  crop_mosaic = TRUE,
  align = TRUE,
  crop_aligned = TRUE,
  rescale = TRUE,
  coef = 0,
  viewer = "mapview",
  max_pixels = 5e+05,
  show = "rgb",
  index = "R"
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`r`, `g`, `b`, `re`, `nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
`crop_mosaic`	Logical, whether to crop the mosaic interactively before aligning it (default: FALSE).
`align`	Logical, whether to align the mosaic interactively (default: TRUE).
`crop_aligned`	Logical, whether to crop the aligned mosaic interactively (default: TRUE).
`rescale`	Rescale the final values? If `TRUE` the final values are rescaled so that the maximum value is 1.
`coef`	An addition coefficient applied to the resulting object. This is useful to adjust the brightness of the final image. Defaults to 0.

viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).
show	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
index	The index to use for the index view. Defaults to "B".

Value

A prepared object of class image.

Examples

```
if (interactive()) {
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
mosaic_prepare(mosaic)
}
```

---

mosaic_project *Project a Mosaic to a New Coordinate Reference System (CRS)*

Description

This function projects a given mosaic to a specified CRS.

Usage

```
mosaic_project(mosaic, y, ...)
```

Arguments

mosaic	A raster object representing the mosaic to be projected.
y	The target CRS to which the mosaic should be projected. This can be specified in various formats accepted by the terra::project() function.
...	Additional arguments passed to the terra::project() function.

Value

A raster object representing the projected mosaic.

Examples

```
if (interactive()) {
library(terra)
library(pliman)

# Create a sample mosaic
mosaic <- rast(nrow=10, ncol=10, xmin=-120, xmax=-60, ymin=30, ymax=60)
mosaic
# Define target CRS (EPSG code for WGS 84 / UTM zone 33N)
target_crs <- "EPSG:32633"

# Project the mosaic
projected_mosaic <- mosaic_project(mosaic, "EPSG:32633")
projected_mosaic
}
```

---

mosaic_resample *A wrapper around terra::resample()*

Description

Transfers values between SpatRaster objects that do not align (have a different origin and/or resolution). See `terra::resample()` for more details

Usage

```
mosaic_resample(mosaic, y, ...)
```

Arguments

`mosaic`	SpatRaster to be resampled
`y`	SpatRaster with the geometry that x should be resampled to
`...`	Further arguments passed on to `terra::resample()`.

Value

SpatRaster

Examples

```
if (interactive()) {
library(pliman)
library(terra)
r <- rast(nrows=3, ncols=3, xmin=0, xmax=10, ymin=0, ymax=10)
values(r) <- 1:ncell(r)
s <- rast(nrows=25, ncols=30, xmin=1, xmax=11, ymin=-1, ymax=11)
x <- mosaic_resample(r, s, method="bilinear")
opar <- par(no.readonly =TRUE)
```

```
par(mfrow=c(1,2))
plot(r)
plot(x)
par(opar)
}
```

`mosaic_rotate` *Rotate a mosaic image by specified angles*

Description

This function rotates a mosaic image by 90, 180, or 270 degrees.

Usage

```
mosaic_rotate(mosaic, angle, direction = "clockwise")
```

Arguments

`mosaic`	A `SpatRaster` object representing the mosaic image.
`angle`	An integer specifying the rotation angle. Must be one of 90, 180, or 270.
`direction`	A string specifying the rotation direction. Must be either ”clockwise” or ”anticlockwise”.

Value

A `SpatRaster` object with the rotated mosaic image.

Examples

```
if (interactive()) {
library(pliman)
# Convert a mosaic raster to an Image object
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
r90 <- mosaic_rotate(mosaic, 90)
r180 <- mosaic_rotate(mosaic, 180)
r270 <- mosaic_rotate(mosaic, 270)
# Plot all rotations side by side
par(mfrow = c(2, 2))
mosaic_plot(mosaic, main = "Original")
mosaic_plot(r90, main = "90 Degrees")
mosaic_plot(r180, main = "180 Degrees")
mosaic_plot(r270, main = "270 Degrees")
par(mfrow = c(1, 1))

}
```

mosaic_segment *Segment a mosaic*

Description

Segment a `SpatRaster` using a computed image index. By default, values greater than `threshold` are kept in the mask.

Usage

```
mosaic_segment(
  mosaic,
  index = "R",
  r = 3,
  g = 2,
  b = 1,
  re = NA,
  nir = NA,
  swir = NA,
  tir = NA,
  threshold = "Otsu",
  invert = FALSE,
  return = c("mosaic", "mask")
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`index`	A character value (or a vector of characters) specifying the target mode for conversion to a binary image. Use `pliman_indexes_rgb()` and `pliman_indexes_me()` to see the available RGB and multispectral indexes, respectively. Users can also calculate their own index using `R, G, B, RE, NIR, SWIR, and TIR` bands (eg., `index = "R+B/G"`) or using the names of the mosaic's layers (ex., "(band_1 + band_2) / 2").
`r`, `g`, `b`, `re`, `nir`, `swir`, `tir`	The red, green, blue, red-edge, near-infrared, shortwave Infrared, and thermal infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.
`threshold`	By default (threshold = "Otsu"), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is provided, this value will be used as a threshold.
`invert`	Logical, indicating whether to invert the mask. Defaults to `FALSE`, i.e., pixels with intensity greater than the threshold values are selected.

return	The output of the function. Either 'mosaic' (the segmented mosaic), or 'mask' (the binary mask).

Value

The segmented mosaic (SpatRaster object)

Examples

```
if (interactive()) {
library(pliman)
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
seg <-
mosaic_segment(mosaic,
               index = "elevation",
               threshold = 350)
mosaic_plot(seg)
}
```

---

mosaic_segment_pick *Segments a mosaic interactively*

Description

The function segments a mosaic using an interative process where the user picks samples from background (eg., soil) and foreground (eg., plants).

Usage

```
mosaic_segment_pick(
  mosaic,
  basemap = NULL,
  g = 2,
  r = 3,
  b = 1,
  max_pixels = 2e+06,
  downsample = NULL,
  quantiles = c(0, 1),
  return = c("mosaic", "mask")
)
```

Arguments

mosaic	A mosaic of class SpatRaster, generally imported with mosaic_input().
basemap	An optional mapview object.
r, g, b	The layer for the Red, Green and Blue band, respectively. Defaults to 1, 2, and 3.
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).

`downsample`	Downsampling factor to reduce the number of pixels (default: NULL). In this case, if the number of pixels in the image (width x height) is greater than `max_pixels` a downsampling factor will be automatically chosen so that the number of plotted pixels approximates the `max_pixels`.
`quantiles`	the upper and lower quantiles used for color stretching.
`return`	The output of the function. Either 'mosaic' (the segmented mosaic), or 'mask' (the binary mask).

Value

An `SpatRaster` object with the segmented `mosaic` (if `return = 'mosaic'`) or a mask (if `return = 'mask'`).

Examples

```
if (interactive()) {
 mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
 seg <- mosaic_segment_pick(mosaic)
 mosaic_plot(seg)
}
```

`mosaic_to_pliman` *Mosaic to pliman*

Description

Convert an `SpatRaster` object to a `image` object with optional scaling.

Usage

```
mosaic_to_pliman(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  re = 4,
  nir = 5,
  rescale = TRUE,
  coef = 0
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`r`, `g`, `b`, `re`, `nir`	The red, green, blue, red-edge, and near-infrared bands of the image, respectively. By default, the function assumes a BGR as input (b = 1, g = 2, r = 3). If a multispectral image is provided up to seven bands can be used to compute built-in indexes. There are no limitation of band numbers if the index is computed using the band name.

`rescale`	Rescale the final values? If `TRUE` the final values are rescaled so that the maximum value is 1.
`coef`	An addition coefficient applied to the resulting object. This is useful to adjust the brightness of the final image. Defaults to 0.

Details

This function converts `SpatRaster` into an `image` object, which can be used for image analysis in `pliman`. Note that if a large `SpatRaster` is loaded, the resulting object may increase considerably the memory usage.

Value

An `image` object with the same number of layers as `mosaic`.

Examples

```
if (interactive()) {
library(pliman)
# Convert a mosaic raster to an Image object
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))
pliman_image <- mosaic_to_pliman(mosaic)
plot(pliman_image)
}
```

`mosaic_to_rgb` *Mosaic to RGB*

Description

Convert an `SpatRaster` to a three-band RGB image of class `image`.

Usage

```
mosaic_to_rgb(mosaic, r = 3, g = 2, b = 1, coef = 0, plot = TRUE)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`r`, `g`, `b`	The red, green, blue bands.
`coef`	An addition coefficient applied to the resulting object. This is useful to adjust the brightness of the final image. Defaults to 0.
`plot`	Logical, whether to display the resulting RGB image (default: TRUE).

Details

This function converts `SpatRaster` that contains the RGB bands into a three-band RGB. It allows you to specify the band indices for the red, green, and blue channels, as well as apply a scaling coefficient to the final image. By default, the resulting RGB image is displayed, but this behavior can be controlled using the `plot` parameter.

Value

An object of class `image` object.

Examples

```
if (interactive()) {

library(pliman)
# Convert a mosaic raster to an RGB image and display it
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))

# Convert a mosaic raster to an RGB image without displaying it
rgb_image <- mosaic_to_rgb(c(mosaic * 2, mosaic - 0.3, mosaic * 0.8))
plot(rgb_image)
}
```

---

mosaic_vectorize *Vectorize a* `SpatRaster` *mask to an* `sf` *object*

Description

Converts a raster mask into a vectorized `sf` object, with various options for morphological operations and filtering.

Usage

```
mosaic_vectorize(
  mask,
  aggregate = NULL,
  watershed = TRUE,
  tolerance = 1,
  extension = 1,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  fill_hull = FALSE,
  lower_size = NULL,
```

```
  upper_size = NULL,
  topn_lower = NULL,
  topn_upper = NULL,
  smooth = FALSE
)
```

Arguments

`mask`	An optional mask (SpatRaster) to mask the mosaic.
`aggregate`	The size as a fraction (percentage) of the input image size. Either a scalar (eg., 50), or a length-two numeric vector. In the last, different percentage reduction/expansion can be used for columns, and rows, respectively.
`watershed`	If `TRUE` (default), performs watershed-based object detection. This will detect objects even when they are touching one another. If FALSE, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.
`tolerance`	The minimum height of the object in the units of image intensity between its highest point (seed) and the point where it contacts another object (checked for every contact pixel). If the height is smaller than the tolerance, the object will be combined with one of its neighbors, which is the highest. Values < 1.0 are interpreted as relative tolerance (percentage of the peak height). Values >= 1.0 are interpreted as absolute tolerance in pixels (linear Euclidean distance).
`extension`	Radius of the neighborhood in pixels for the detection of neighboring objects. A higher value smooths out small objects.
`opening`, `closing`, `filter`, `erode`, `dilate`	**Morphological operations (brush size)**, applied in hierarchical order.
`fill_hull`	Fill holes in the binary image? Defaults to `FALSE`.
`lower_size`, `upper_size`	Lower and upper size limits for object filtering.
`topn_lower`, `topn_upper`	Select top `n` objects based on area.
`smooth`	Smoothes the contours using a moving average filter. Default is `FALSE`.

Value

An `sf` object containing vectorized features from the raster mask, with added area measurements.

Examples

```
if (interactive()) {
library(pliman)
mask <- image_pliman("mask.tif")
shp <- mosaic_vectorize(mask, watershed = FALSE)
mosaic_plot(mask)
shapefile_plot(shp, add = TRUE, lwd = 3)
}
```

`mosaic_view` *Mosaic View*

Description

Mosaic View

Usage

```
mosaic_view(
  mosaic,
  r = 3,
  g = 2,
  b = 1,
  edit = FALSE,
  title = "",
  shapefile = NULL,
  attribute = NULL,
  viewer = c("mapview", "base"),
  show = c("rgb", "index"),
  index = "B",
  max_pixels = 1e+06,
  downsample = NULL,
  downsample_fun = "nearest",
  alpha = 1,
  quantiles = c(0, 1),
  color_regions = custom_palette(c("red", "yellow", "forestgreen")),
  axes = FALSE,
  ...
)
```

Arguments

`mosaic`	A mosaic of class `SpatRaster`, generally imported with `mosaic_input()`.
`r`, `g`, `b`	The layer for the Red, Green and Blue band, respectively. Defaults to `1`, `2`, and `3`.
`edit`	If `TRUE` enable editing options using `mapedit::editMap()`.
`title`	A title for the generated map or plot (default: "").
`shapefile`	An optional shapefile of class `sf` to be plotted over the mosaic. It can be, for example, a plot-level result returned by `mosaic_analyze()`.
`attribute`	The attribute name(s) or column number(s) in shapefile table of the column(s) to be rendered.

viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
show	The display option for the map view. Options are "rgb" for RGB view and "index" for index view.
index	The index to use for the index view. Defaults to "B".
max_pixels	Maximum number of pixels to render in the map or plot (default: 500000).
downsample	Downsampling factor to reduce the number of pixels (default: NULL). In this case, if the number of pixels in the image (width x height) is greater than max_pixels a downsampling factor will be automatically chosen so that the number of plotted pixels approximates the max_pixels.
downsample_fun	The resampling function. Defaults to nearest. See further details in mosaic_aggregate().
alpha	opacity of the fill color of the raster layer(s).
quantiles	the upper and lower quantiles used for color stretching.
color_regions	The color palette for displaying index values. Default is custom_palette().
axes	logical. Draw axes? Defaults to FALSE.
...	Additional arguments passed on to terra::plot() when viewer = "base".

Details

The function can generate either an interactive map using the 'mapview' package or a static plot using the 'base' package, depending on the viewer and show parameters. If show = "index" is used, the function first computes an image index that can be either an RGB-based index or a multispectral index, if a multispectral mosaic is provided.

Value

An sf object, the same object returned by mapedit::editMap().

Examples

```
if (interactive()) {
library(pliman)
# Load a raster showing the elevation of Luxembourg
mosaic <- mosaic_input(system.file("ex/elev.tif", package="terra"))

# Generate an interactive map using 'mapview'
mosaic_view(mosaic)

# Generate a static plot using 'base'
```

```
mosaic_view(mosaic, viewer = "base")
}
```

`object_bbox` *Compute Bounding Boxes from Contours*

Description

This function calculates the bounding boxes for a given list of contours.

Usage

```
object_bbox(contours)
```

Arguments

`contours`	A list of matrices, where each matrix contains two columns representing (x, y) coordinates of a contour.

Value

A list of bounding boxes, where each bounding box is represented as a list with `x_min`, `y_min`, `x_max`, and `y_max` values.

Examples

```
if(interactive()){
contours <- list(
  matrix(c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
           110, 120, 130, 140, 150, 160, 170, 180, 190, 200),
         ncol = 2, byrow = FALSE)
)
bbox_list <- object_bbox(contours)
print(bbox_list)
}
```

`object_edge` *Object edges*

Description

Applies the Sobel-Feldman Operator to detect edges. The operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions.

Usage

```
object_edge(img, sigma = 1, threshold = "Otsu", thinning = FALSE, plot = TRUE)
```

Arguments

`img`	An image or a list of images of class `image`.
`sigma`	Gaussian kernel standard deviation used in the gaussian blur.
`threshold`	The theshold method to be used. If `threshold = "Otsu"` (default), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If any non-numeric value different than `"Otsu"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index. Alternatively, provide a numeric value to be used as the threshold value.
`thinning`	Logical value indicating whether a thinning procedure should be applied to the detected edges. See `image_skeleton()`
`plot`	Logical value indicating whether a plot should be created

Value

A binary version of `image`.

References

Sobel, I., and G. Feldman. 1973. A 3×3 isotropic gradient operator for image processing. Pattern Classification and Scene Analysis: 271–272.

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf_nb.jpg", plot = TRUE)
object_edge(img)
}
```

`object_export` *Export multiple objects from an image to multiple images*

Description

Givin an image with multiple objects, `object_export()` will split the objects into a list of objects using `object_split()` and then export them to multiple images into the current working directory (or a subfolder). Batch processing is performed by declaring a file name pattern that matches the images within the working directory.

Usage

```
object_export(
  img,
  pattern = NULL,
  dir_original = NULL,
  dir_processed = NULL,
  format = ".jpg",
  squarize = FALSE,
  augment = FALSE,
  times = 12,
  index = "NB",
  lower_size = NULL,
  watershed = FALSE,
  invert = FALSE,
  fill_hull = FALSE,
  opening = 3,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  threshold = "Otsu",
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  edge = 20,
  remove_bg = FALSE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)
```

Arguments

`img`	The image to be analyzed.
`pattern`	A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images in the current working directory that

the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported and processed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on. An error will be returned if the pattern matches any file that is not supported (e.g., img1.pdf).

`dir_original` The directory containing the original images. Defaults to `NULL`. It can be either a full path, e.g., `"C:/Desktop/imgs"`, or a subfolder within the current working directory, e.g., `"/imgs"`.

`dir_processed` Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.

`format` The format of image to be exported.

`squarize` Squarizes the image before the exportation? If `TRUE`, `image_square()` will be called internally.

`augment` A logical indicating if exported objects should be augmented using `image_augment()`. Defaults to `FALSE`.

`times` The number of times to rotate the image.

`index` A character value specifying the index for conversion to binary image.

`lower_size` Plant images often contain dirt and dust. To prevent dust from affecting the image analysis, objects with lesser than 10% of the mean of all objects are removed. Set `lower_limit = 0` to keep all the objects.

`watershed` If `TRUE` (default) performs watershed-based object detection.

`invert` Inverts the binary image if desired.

`fill_hull` Fill holes in the binary image? Defaults to `FALSE`.

`opening`, `closing`, `filter`, `erode`, `dilate`
Morphological operations (brush size), applied in hierarchical order.

`threshold` The threshold method (`"Otsu"`, `"Adaptive"`, or numeric).

`extension` Neighborhood radius for watershed segmentation.

`tolerance` Minimum height for watershed segmentation.

`object_size` The size of the object (`"small"`, `"medium"`, `"large"`, `"elarge"`). Used to automatically set up `tolerance` and `extension` parameters.

`edge` The number of pixels to be added in the edge of the segmented object. Defaults to 5.

`remove_bg` If `TRUE`, the pixels that are not part of objects are converted to white.

`parallel` If `TRUE` processes the images asynchronously in parallel.

`workers` Number of parallel processes.

`verbose` If `TRUE` (default) shows progress summary in console.

Value

A `NULL` object.

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("potato_leaves.jpg")
object_export(img,
              remove_bg = TRUE)
}
```

---

object_export_shp *Export multiple objects from an image to multiple images*

Description

Givin an image with multiple objects, `object_export_shp()` will split the objects into a list of objects using `object_split_shp()` and then export them to multiple images into the current working directory (or a subfolder). Batch processing is performed by declaring a file name pattern that matches the images within the working directory.

Usage

```
object_export_shp(
  img,
  pattern = NULL,
  dir_original = NULL,
  dir_processed = NULL,
  format = ".jpg",
  subfolder = NULL,
  squarize = FALSE,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  interactive = FALSE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  viewer = pliman::get_pliman_viewer()
)
```

Arguments

`img`	An object of class `image`
`pattern`	A pattern of file name used to identify images to be processed. For example, if `pattern = "im"` all images in the current working directory that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported and processed. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on. An error will

be returned if the pattern matches any file that is not supported (e.g., img1.pdf).

`dir_original` The directory containing the original images. Defaults to `NULL`. It can be either a full path, e.g., `"C:/Desktop/imgs"`, or a subfolder within the current working directory, e.g., `"/imgs"`.

`dir_processed` Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.

`format` The format of image to be exported.

`subfolder` Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.

`squarize` Squarizes the image before the exportation? If `TRUE`, `image_square()` will be called internally.

`nrow` The number of desired rows in the grid. Defaults to `1`.

`ncol` The number of desired columns in the grid. Defaults to `1`.

`buffer_x`, `buffer_y`
Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.

`interactive` If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.

`parallel` If `TRUE` processes the images asynchronously in parallel.

`workers` Number of parallel processes.

`verbose` If `TRUE` (default) shows progress summary in console.

`viewer` The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are `"base"` and `"mapview"`. If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.

Value

A `NULL` object.

Examples

```
if (interactive()) {
library(pliman)
```

```
flax <- image_pliman("flax_leaves.jpg", plot = TRUE)
object_export_shp(flax)

}
```

`object_haralick` *Compute Haralick texture features*

Description

Computes 13 Haralick texture features for segmented objects in an image based on a Gray-Level Co-occurrence Matrix (GLCM). The GLCM is computed using 4 directions (0, 45, 90, and 135 degrees) and is made symmetric before calculating the texture metrics.

Usage

```
object_haralick(
  mask,
  ref,
  haralick.nbins = 32,
  haralick.scales = 1,
  haralick.band = "GRAY"
)
```

Arguments

`mask`	A mask (binary image or labeled matrix) identifying the objects.
`ref`	A grayscale or RGB reference image (matrix or Image object) containing the intensities.
`haralick.nbins`	The number of bins used to discretize intensity values before computing the GLCM (default is 32).
`haralick.scales`	A numeric vector of scales at which texture is measured (default is 1).
`haralick.band`	The band to use for computing texture features if `ref` is an RGB image. Defaults to `"GRAY"`.

Details

The Haralick texture features are calculated from the normalized GLCM, $p(i, j)$, where $i, j = 1, \ldots, N_g$ and N_g is the number of gray levels (`haralick.nbins`). The features returned are:

- **asm** (Angular Second Moment/Energy): Measures texture uniformity. High values indicate a highly uniform, homogeneous, and orderly texture (e.g., a flat color or a perfectly repeating pattern).

$$f_1 = \sum_i \sum_j p(i, j)^2$$

- **con** (Contrast): Measures local variations in gray levels. High values indicate sharp local variations (e.g., sharp edges, coarse textures, or noise).

$$f_2 = \sum_{n=0}^{N_g-1} n^2 \sum_{|i-j|=n} p(i,j)$$

- **cor** (Correlation): Measures linear gray-level dependencies. High values (close to 1) indicate structured, predictable patterns (e.g., linear stripes).

$$f_3 = \frac{\sum_i \sum_j (i-\mu_x)(j-\mu_y)p(i,j)}{\sigma_x \sigma_y}$$

- **var** (Sum of Squares/Variance): Measures dispersion of gray levels from the mean, suggesting a high variety of tones and shades within the object.

$$f_4 = \sum_i \sum_j (i+1-\mu)^2 p(i,j)$$

.

- **idm** (Inverse Difference Moment/Homogeneity): Measures texture homogeneity. High values indicate a very homogeneous texture where adjacent pixels have very similar values.

$$f_5 = \sum_i \sum_j \frac{p(i,j)}{1+(i-j)^2}$$

- **sav** (Sum Average): Measures overall brightness. High values suggest the texture is overall light/bright.

$$f_6 = \sum_{i=2}^{2N_g} i p_{x+y}(i)$$

- **sva** (Sum Variance): Measures the complexity and variance in overall brightness between neighboring pixel sums.

$$f_7 = \sum_{i=2}^{2N_g} (i-f_8)^2 p_{x+y}(i)$$

(calculated using Sum Entropy f_8).

- **sen** (Sum Entropy): Measures disorder/randomness in the sum of neighboring pixel levels. Higher values mean more chaotic brightness sums.

$$f_8 = -\sum_{i=2}^{2N_g} p_{x+y}(i) \log_{10}(p_{x+y}(i))$$

- **ent** (Entropy): Measures overall complexity and randomness. High values indicate highly detailed, unpredictable, and random patterns.

$$f_9 = -\sum_i \sum_j p(i,j) \log_{10}(p(i,j))$$

- **dva** (Difference Variance): Matches Contrast (`con`), denoting variation in gray level differences (roughness).
- **den** (Difference Entropy): Measures disorder/randomness of intensity differences between adjacent pixels.

$$f_{11} = -\sum_{i=0}^{N_g-1} p_{x-y}(i) \log_{10}(p_{x-y}(i))$$

- **f12** (Information Measure of Correlation 1): Quantifies the structural coupling and mutual information between neighboring pixels relative to their individual distributions (1).

$$f_{12} = \frac{HXY - HXY_1}{\max(HX, HY)}$$

- **f13** (Information Measure of Correlation 2): Quantifies the structural coupling and mutual information between neighboring pixels relative to their individual distributions (2).

$$f_{13} = \sqrt{1 - \exp(-2(HXY_2 - HXY))}$$

(using base 10 values in the exponent).

Here, p_{x+y} is the sum distribution:

$$p_{x+y}(k) = \sum_i \sum_{j,i+j=k} p(i,j)$$

and p_{x-y} is the difference distribution:

$$p_{x-y}(k) = \sum_i \sum_{j,|i-j|=k} p(i,j)$$

Value

A matrix containing the 13 Haralick texture features for each object.

Examples

```
library(pliman)
img <- image_pliman("soy_green.jpg")
wat <-
  image_binary(img, "B", opening = 5)[[1]] |>
  image_watershed()
cont <- contour(wat)
centroids <- poly_mass(cont)
hara <- object_haralick(wat, img)
plot(img)
# Contrast
text(centroids[, 1], centroids[,2], labels = round(hara[, "con.s1"], 4))
```

`object_label` *Labels objects*

Description

All pixels for each connected set of foreground (non-zero) pixels in x are set to an unique increasing integer, starting from 1. Hence, max(x) gives the number of connected objects in `x`.

Usage

```
object_label(
  img,
  index = "B",
  invert = FALSE,
  fill_hull = FALSE,
  threshold = "Otsu",
  k = 0.1,
  windowsize = NULL,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  filter_order = c("erode", "dilate", "opening", "closing", "filter", "fill_hull"),
  watershed = FALSE,
  tolerance = NULL,
  extension = NULL,
  object_size = "medium",
  plot = TRUE,
  ncol = NULL,
  nrow = NULL,
  verbose = TRUE
)
```

Arguments

`img`	An image object.
`index`	A character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with `pliman_indexes()` and `image_index()` for more details.
`invert`	Inverts the binary image, if desired.
`fill_hull`	Fill holes in the objects? Defaults to `FALSE`.
`threshold`	The theshold method to be used.

- By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold.
- If `threshold = "adaptive"`, adaptive thresholding (Shafait et al. 2008) is used, and will depend on the `k` and `windowsize` arguments.
- If any non-numeric value different than `"Otsu"` and `"adaptive"` is used, an iterative section will allow you to choose the threshold based on a raster plot showing pixel intensity of the index.

`k` a numeric in the range 0-1. when `k` is high, local threshold values tend to be lower. when `k` is low, local threshold value tend to be higher.

`windowsize` windowsize controls the number of local neighborhood in adaptive thresholding. By default it is set to `1/3 * minxy`, where `minxy` is the minimum dimension of the image (in pixels).

`erode`, `dilate`, `opening`, `closing`, `filter`

Morphological operations (brush size)

- `dilate` puts the mask over every background pixel, and sets it to foreground if any of the pixels covered by the mask is from the foreground.
- `erode` puts the mask over every foreground pixel, and sets it to background if any of the pixels covered by the mask is from the background.
- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.

Hierarchically, the operations are performed as opening $>$ closing $>$ filter. The value declared in each argument will define the brush size.

`filter_order` A character vector indicating the order in which morphological operations are applied. Defaults to `c("erode", "dilate", "opening", "closing", "filter", "fill_hull")`.

`watershed` If `TRUE` (default) performs watershed-based object detection.

`tolerance` Minimum height for watershed segmentation.

`extension` Neighborhood radius for watershed segmentation.

`object_size` The size of the object (`"small"`, `"medium"`, `"large"`, `"elarge"`). Used to automatically set up `tolerance` and `extension` parameters.

`plot` Show image after processing?

`nrow`, `ncol` The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.

`verbose` If `TRUE` (default) a summary is shown in the console.

Value

A list with the same length of `img` containing the labeled objects.

Examples

```
if (interactive()) {
img <- image_pliman("soybean_touch.jpg")
# segment the objects using the "B" (blue) band.
object_label(img, index = "B")
object_label(img, index = "B", watershed = TRUE)
}
```

`object_map` *Map Object Distances*

Description

Computes distances between objects in an `anal_obj` object and returns a list of distances, coefficient of variation (CV), and means.

Usage

```
object_map(object, by_column = "img", direction = c("horizontal", "vertical"))
```

Arguments

`object`	An `anal_obj` object computed with `analyze_objects_shp()`.
`by_column`	The column name in the object's results data frame to group objects by. Default is "img".
`direction`	The direction of mapping. Should be one of "horizontal" or "vertical". Default is "horizontal".

Value

A list with the following components:

`distances`	A list of distances between objects grouped by unique values in the specified column/row.
`cvs`	A vector of coefficient of variation (CV) values for each column/row.
`means`	A vector of mean distances for each column/row.

See Also

`analyze_objects_shp`

Examples

```
if (interactive()) {
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects_shp(flax,
                       nrow = 3,
                       ncol = 1,
                       watershed = FALSE,
                       index = "R/(G/B)",
                       plot = FALSE)
plot(res$final_image_mask)
plot(res$shapefiles)

# distance from each leave within each row
result <- object_map(res)
result$distances
result$cvs
result$means
}
```

---

object_mark *Mark Object Points*

Description

Marks the coordinates of objects in an `anal_obj` object on a plot.

Usage

```
object_mark(object, col = "white")
```

Arguments

`object`	An `anal_obj` object computed with `analyze_objects_shp()` or `analyze_objects_shp()`.
`col`	The color of the marked points. Default is "white".

See Also

analyze_objects_shp

Examples

```
if (interactive()) {
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects(flax,
                       watershed = FALSE,
```

```
                        index = "R/(G/B)",
                        plot = FALSE)
object_mark(res)
}
```

`object_rgb` *Extract red, green and blue values from objects*

Description

Given an image and a matrix of labels that identify each object, the function extracts the red, green, and blue values from each object.

Usage

```
object_rgb(img, labels)
```

Arguments

`img`	An `image` object
`labels`	A mask containing the labels for each object. This can be obtained with `image_bwlabel()` or `image_watershed()`.

Value

A data.frame with `n` rows (number of pixels for all the objects) and the following columns:

- `id`: the object id;
- `R`: the value for the red band;
- `G`: the value for the blue band;
- `B`: the value for the green band;

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")
# segment the objects using the "B" (blue) band (default)

labs <- object_label(img, watershed = TRUE)
rgb <- object_rgb(img, labs[[1]])
head(rgb)
}
```

`object_scatter` *Plot object thumbnails at (x, y) coordinates derived from image features*

Description

Extracts connected objects from an image, computes their features, crops each object, converts the crop to a raster with alpha, and draws each thumbnail centered at its corresponding `(x, y)` feature location in a ggplot. Optionally overlays object IDs. Caching can be used to avoid recomputing object features on repeated calls.

Usage

```
object_scatter(
  img,
  x,
  y,
  scale = 0.1,
  xy_ratio = 1,
  xlab = x,
  ylab = y,
  erosion = 2,
  dilatation = FALSE,
  show_id = FALSE,
  color_id = "black",
  size_id = 3,
  cache = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An image of class `image` (or compatible) from which objects will be segmented and measured.
`x`	Character scalar. Name of the feature column (returned by `get_measures(res)`) to use on the x-axis.
`y`	Character scalar. Name of the feature column to use on the y-axis.
`scale`	Numeric in (0, 1]. Relative thumbnail size as a fraction of the data range along each axis (larger values draw larger thumbnails).
`xy_ratio`	Numeric scalar. Factor applied to the vertical scaling (y-axis) of thumbnails. Use values other than 1 to stretch or compress thumbnails vertically.
`xlab`, `ylab`	Character scalars used as x- and y-axis labels. Defaults to `x` and `y`, respectively.

`erosion`, `dilatation`
: Integer (non-negative). Size of the structuring element for morphological erosion/dilatation of the segmented objects.

`show_id`
: Logical. If `TRUE`, overlays object IDs at their `x, y` locations.

`color_id`
: Character. Color used for the ID labels when `show_id = TRUE`.

`size_id`
: Numeric. Text size for the ID labels when `show_id = TRUE`.

`cache`
: Logical. If `TRUE` (default), caches results of object extraction using a simple key based on image dimensions and parameters.

`verbose`
: If `TRUE` (default), shows the progress of analysis.

`...`
: Additional arguments forwarded to `analyze_objects()`.

Value

A list with two elements:

- `features` — a data frame (or tibble) with object-level features returned by `get_measures(res)`. Must contain columns named `x` and `y`.
- `plot` — a `ggplot` object. The thumbnail scatter plot.

Scaling behavior

Thumbnails are sized relative to the observed ranges in `x` and `y`. If the two axes differ substantially in range, perceived thumbnail aspect on the plotting device may vary. Use `xy_ratio` to adjust vertical scaling.

Examples

```
if(interactive()){
img <- image_pliman("potato_leaves.jpg")
plot(img)
res <- object_scatter(
 img = img,
 index = "B",
 x = "area",
 y = "solidity",
 watershed = FALSE,
 scale = 0.5
)
res$plot

# remove cached data
clear_pliman_cache()
}
```

`object_split` *Splits objects from an image into multiple images*

Description

Using threshold-based segmentation, objects are first isolated from background. Then, a new image is created for each single object. A list of images is returned.

Usage

```
object_split(
  img,
  index = "NB",
  lower_size = NULL,
  watershed = TRUE,
  invert = FALSE,
  fill_hull = FALSE,
  opening = 3,
  closing = FALSE,
  filter = FALSE,
  erode = FALSE,
  dilate = FALSE,
  threshold = "Otsu",
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  edge = 3,
  remove_bg = FALSE,
  plot = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	The image to be analyzed.
`index`	A character value specifying the index for conversion to binary image.
`lower_size`	Plant images often contain dirt and dust. To prevent dust from affecting the image analysis, objects with lesser than 10% of the mean of all objects are removed. Set `lower_limit = 0` to keep all the objects.
`watershed`	If `TRUE` (default) performs watershed-based object detection.
`invert`	Inverts the binary image if desired.
`fill_hull`	Fill holes in the binary image? Defaults to `FALSE`.
`opening`, `closing`, `filter`, `erode`, `dilate`	**Morphological operations (brush size)**, applied in hierarchical order.

`threshold`	The threshold method (`"Otsu"`, `"Adaptive"`, or numeric).
`extension`	Neighborhood radius for watershed segmentation.
`tolerance`	Minimum height for watershed segmentation.
`object_size`	The size of the object (`"small"`, `"medium"`, `"large"`, `"elarge"`). Used to automatically set up `tolerance` and `extension` parameters.
`edge`	The number of pixels to be added in the edge of the segmented object. Defaults to 5.
`remove_bg`	If `TRUE`, the pixels that are not part of objects are converted to white.
`plot`	Show image after processing?
`verbose`	If `TRUE` (default) shows progress summary in console.
`...`	Additional arguments passed on to `image_combine()`

Value

A list of objects of class `image`.

See Also

`analyze_objects()`, `image_binary()`

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("la_leaves.jpg", plot = TRUE)
imgs <- object_split(img) # set to NULL to use 50% of the cores
}
```

`object_split_shp` *Splits image objects based on a shape file*

Description

Here, `image_shp()` is used to create a shape file based on the desired number of rows and columns. Then, using the object coordinates, a list of `image` objects is created.

Usage

```
object_split_shp(
  img,
  nrow = 1,
  ncol = 1,
  buffer_x = 0,
  buffer_y = 0,
  interactive = FALSE,
```

```
  viewer = get_pliman_viewer(),
  only_shp = FALSE,
  ...
)
```

Arguments

`img`	An object of class `image`
`nrow`	The number of desired rows in the grid. Defaults to `1`.
`ncol`	The number of desired columns in the grid. Defaults to `1`.
`buffer_x`, `buffer_y`	Buffering factor for the width and height, respectively, of each individual shape's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the shape by 25% on each side.
`interactive`	If `FALSE` (default) the grid is created automatically based on the image dimension and number of rows/columns. If `interactive = TRUE`, users must draw points at the diagonal of the desired bounding box that will contain the grid.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are `"base"` and `"mapview"`. If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`only_shp`	If `TRUE` returns only the shapefiles with the coordinates for each image. If `FALSE` (default) returns the splitted image according to `nrow` and `ncol` arguments.
`...`	Other arguments passed on to `image_shp()`

Value

A list of `image` objects

Examples

```
if (interactive()) {
library(pliman)
flax <- image_pliman("flax_leaves.jpg", plot = TRUE)
objects <- object_split_shp(flax, nrow = 3, ncol = 5)
image_combine(objects$imgs)
}
```

`object_to_color` *Apply color to image objects*

Description

The function applies the color informed in the argument `color` to segmented objects in the image. The segmentation is performed using image indexes. Use `image_index()` to identify the better candidate index to segment objects.

Usage

```
object_to_color(
  img,
  pick_palettes = FALSE,
  background = NULL,
  foreground = NULL,
  index = "NB",
  color = "blue",
  plot = TRUE,
  ...
)
```

Arguments

`img`	An image object.
`pick_palettes`	Logical argument indicating wheater the user needs to pick up the color palettes for foreground and background for the image. If `TRUE` `pick_palette()` will be called internally so that the user can sample color points representing foreground and background.
`foreground`, `background`	A color palette for the foregrond and background, respectively (optional).
`index`	A character value (or a vector of characters) specifying the target mode for conversion to binary image. See the available indexes with `pliman_indexes()` and `image_index()` for more details.
`color`	The color to apply in the image objects. Defaults to `"blue"`.
`plot`	Plots the modified image? Defaults to `TRUE`.
`...`	Additional arguments passed on to `image_binary()`.

Value

An object of class `image`

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("la_leaves.jpg")
img2 <- object_to_color(img, index = "G-R")
image_combine(img, img2)
}
```

`object_veins` *Object veins detection and quantification*

Description

Detects leaf veins using Difference of Gaussians (DoG) bandpass filtering, dynamic resolution-independent boundary erosion, and per-object Otsu adaptive thresholding. Optionally applies Guo-Hall skeleton thinning.

Usage

```
object_veins(
  img,
  labels = NULL,
  index = "GRAY",
  watershed = TRUE,
  sigma1 = 0.75,
  sigma2 = sigma1 * 5,
  threshold = -1,
  channel = 0,
  erode_size = -1,
  rel_erode = 0.005,
  thinning = FALSE,
  overlay = TRUE,
  col = "red",
  plot = TRUE
)
```

Arguments

`img`	An image or a list of images of class `image`.
`labels`	Optional matrix or `image` object containing object labels (e.g., obtained from `object_label()`). If `NULL` (default), labels are automatically computed using `object_label()`.
`index`	The image index used for object segmentation when `labels = NULL` (default `"GRAY"`). See `image_index()`.
`watershed`	Logical argument indicating whether watershed segmentation should be performed when `labels = NULL` (default `TRUE`).

`sigma1`	Fine Gaussian scale parameter for vein detection (default `0.75`).
`sigma2`	Coarse Gaussian scale parameter for vein detection. Defaults to `sigma1 * 5`.
`threshold`	Fixed vein threshold value in `[0, 1]`. If `< 0` (default `-1`), adaptive per-object Otsu thresholding is performed.
`channel`	Channel to extract for vein detection: `0` for green (default, best for leaves) or `1` for perceptual grayscale.
`erode_size`	Fixed boundary erosion radius in pixels. If `< 0` (default `-1`), the erosion radius is dynamically calculated using `rel_erode`.
`rel_erode`	Relative boundary erosion fraction of `min(width, height)` (default `0.005` = 0.5%). Trims boundary pixels to eliminate outer edge artifacts independently of image resolution.
`thinning`	Logical value indicating whether Guo-Hall skeleton thinning should be applied to detected veins (default `FALSE`). If `TRUE`, measures vein length skeleton density instead of vein area fraction.
`overlay`	Logical argument indicating whether detected veins should be overlaid in color onto the original image (default `TRUE`).
`col`	Color to use for overlaying detected veins (default `"red"`).
`plot`	Logical value indicating whether a plot should be created (default `TRUE`). If `overlay = TRUE`, plots the original image with overlaid veins.

Value

A list containing:

- `proportion`: A numeric vector with vein proportions for each object ID.
- `vein_map`: An `image` object representing the binary vein map (or thinned skeleton).
- `overlay`: An `image` object showing the original image with veins highlighted in `col` (if `overlay = TRUE`).

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf_nb.jpg", plot = TRUE)
veins <- object_veins(img, overlay = TRUE, col = "red")
veins$proportion
}
```

onnx_install *Proprietary ONNX Runtime Installer for pliman*

Description

Downloads and configures the official Microsoft ONNX Runtime C++ shared library directly into pliman's local bin directory, requiring **zero external packages** and **zero Python**.

Usage

```
onnx_install(version = "1.20.1", force = FALSE, engine = c("gpu", "cpu"))

pliman_install_onnx(
  version = "1.20.1",
  force = FALSE,
  engine = c("gpu", "cpu")
)
```

Arguments

`version`	Character string with the ONNX Runtime release version (default `"1.20.1"`).
`force`	Logical. If `TRUE`, forces re-download even if already present (default `FALSE`).
`engine`	Execution engine: `"cpu"` (default) or `"gpu"`.

Value

A character string with the path to the installed shared library.

Examples

```
## Not run:
  onnx_install()
  onnx_install(engine = "gpu")

## End(Not run)
```

onnx_quantize *Quantize ONNX Models to 8-Bit Precision (INT8)*

Description

`onnx_quantize()` compresses 32-bit floating point (FP32) ONNX neural network models to 8-bit integer (INT8) precision. Dynamic quantization reduces model file sizes by up to 75% (e.g. from 24 MB to 6 MB) and accelerates CPU inference throughput by 2–3x with negligible accuracy trade-off.

Usage

```
onnx_quantize(
  model_path,
  output_path = NULL,
  weight_type = c("QUInt8", "QInt8"),
  verbose = TRUE
)
```

Arguments

model_path	Character string. File path to the input FP32 .onnx model.
output_path	Optional file path for the quantized INT8 model. If NULL (default), appends "_int8.onnx" to the original model name.
weight_type	Precision type: "QUInt8" (unsigned 8-bit, default) or "QInt8" (signed 8-bit).
verbose	Logical. Display compression metrics and file sizes. Defaults to TRUE.

Value

The normalized file path to the quantized .onnx model.

Examples

```
## Not run:
library(pliman)

# Quantize classification model
q_model <- onnx_quantize("dataset_flores_yolo_cls.onnx")

# Run inference with 3x speedup on CPU
res <- image_classify_dl("rose.jpg", model = q_model)

## End(Not run)
```

Ops.image *Group generic Ops method for* image *objects*

Description

Group generic Ops method for image objects

Usage

```
## S3 method for class 'image'
Ops(e1, e2)
```

Arguments

`e1`	Left operand.
`e2`	Right operand.

`order_quad_corners` *Order Quadrilateral Corners Topologically*

Description

Sorts 4 quadrilateral corner vertices into standard Top-Left (TL), Top-Right (TR), Bottom-Right (BR), and Bottom-Left (BL) ordering.

Usage

```
order_quad_corners(pts)
```

Arguments

`pts`	A 4x2 matrix, data frame with 2 columns, list with `TL, TR, BR, BL`, or an 8-element numeric vector.

Value

A named list with `$TL`, `$TR`, `$BR`, `$BL` coordinates (numeric pairs `c(x, y)`).

`otsu` *Calculate Otsu's threshold*

Description

Given a numeric vector with the pixel's intensities, returns the threshold value based on Otsu's method, which minimizes the combined intra-class variance

Usage

```
otsu(values)
```

Arguments

`values`	A numeric vector with the pixel values.

Value

A double (threshold value).

References

Otsu, N. 1979. Threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern SMC-9(1): 62–66. doi: doi:10.1109/tsmc.1979.4310076

Examples

```
if (interactive()) {
img <- image_pliman("soybean_touch.jpg")
thresh <- otsu(image_data(img[,,3]))
plot(img[,,3] < thresh)
}
```

---

palettes *Create image palettes*

Description

`image_palette()` creates image palettes by applying the k-means algorithm to the RGB values.

Usage

```
image_palette(
  img,
  pattern = NULL,
  npal = 5,
  proportional = TRUE,
  colorspace = c("rgb", "hsb"),
  remove_bg = FALSE,
  index = "B",
  filter_order = c("erode", "dilate", "opening", "closing", "filter", "fill_hull"),
  plot = TRUE,
  save_image = FALSE,
  prefix = "proc_",
  dir_original = NULL,
  dir_processed = NULL,
  return_pal = FALSE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE
)
```

Arguments

`img`	An image object.
`pattern`	A pattern of file name used to identify images to be imported.

`npal`	The number of color palettes.
`proportional`	Creates a joint palette with proportional size equal to the number of pixels in the image? Defaults to `TRUE`.
`colorspace`	The color space to produce the clusters. Defaults to `rgb`. If `hsb`, the color space is first converted from RGB > HSB before k-means algorithm be applied.
`remove_bg`	Remove background from the color palette? Defaults to `FALSE`.
`index`	An image index used to remove the background, passed to `image_binary()`.
`filter_order`	Hierarchical order of morphological operations.
`plot`	Plot the generated palette? Defaults to `TRUE`.
`save_image`	Save the image after processing?
`prefix`	Prefix for processed images.
`dir_original, dir_processed`	Directories for original and processed images.
`return_pal`	Return the color palette image? Defaults to `FALSE`.
`parallel`	If TRUE processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine.
`workers`	Number of parallel processes.
`verbose`	If `TRUE` (default) shows progress summary in console.

Value

`image_palette()` returns a list with two elements:

- `palette_list` A list with `npal` color palettes of class `image`.
- `joint` An object of class `image` with the color palettes
- `proportions` The proportion of the entire image corresponding to each color in the palette
- `rgbs` The average RGB value for each palette

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf.jpg")
pal <- image_palette(img, npal = 5)
}
```

`pipe` *Forward-pipe operator*

Description

Pipe an object forward into a function or call expression.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	The result you are piping.
`rhs`	Where you are piping the result to.

Author(s)

Nathan Eastwood <nathan.eastwood@icloud.com> and Antoine Fabri <antoine.fabri@gmail.com>. The code was obtained from poorman package at https://github.com/nathaneastwood/poorman/blob/master/R/pipe.R

Examples

```
if (interactive()) {
library(pliman)

# Basic use:
 iris %>% head()

# use to apply several functions to an image
img <- image_pliman("la_leaves.jpg")

img %>%
 image_resize(50) %>%        # resize to 50% of the original size
 object_isolate(id = 1) %>%  # isolate object 1
 image_filter() %>%          # apply a median filter
 plot()                      # plot
}
```

`pixel_index` *Get the pixel indices for a given row of a binary image*

Description

This function finds the first row in the bin matrix that has a value greater than 0 (TRUE). It then calculates the minimum, median, and maximum values for the pixels in that row and creates an array containing the row index, the minimum pixel index, the median pixel index, and the maximum pixel index.

Usage

```
pixel_index(bin, row = NULL, direction = "updown")
```

Arguments

`bin`	A logical matrix representing a binary image
`row`	An optional row index. If not provided, the function selects the first non-zero row.
`direction`	The direction for row selection when row is not provided. If set to `"updown"`, the function starts scanning from the top of the image towards the bottom. If set to `"downup"`, the function starts scanning from the bottom towards the top.

Value

A numeric vector containing the row index, the minimum pixel index, the median pixel index, and the maximum pixel index.

Examples

```
if (interactive()) {
library(pliman)
leaf <- image_pliman("sev_leaf.jpg")
bin <- image_binary(leaf, "NB")[[1]]

# first row with leaf (17)
pixel_index(bin)

# index at the row 100
pixel_index(bin, row = 100)

plot(leaf)
points(x = 248, y = 17, pch = 16, col = "red", cex = 2)
points(x = 163, y = 100, pch = 16, col = "red", cex = 2)
points(x = 333, y = 100, pch = 16, col = "red", cex = 2)
}
```

`pliman_available_models` *List Available Pre-Trained Neural Models*

Description

Lists all supported background removal, object detection, segmentation, and classification models, their disk sizes, input resolutions, descriptions, local download status, and models dynamically discovered from GitHub release assets.

Usage

```
pliman_available_models(dir = pliman_model_dir(), refresh = FALSE)
```

Arguments

`dir`	Directory where models are stored (default: `pliman_model_dir()`).
`refresh`	Logical. If `TRUE`, forces re-fetching the asset catalog from GitHub releases (default `FALSE`).

Value

A data frame containing model metadata and local availability.

Examples

```
## Not run:
pliman_available_models()
pliman_available_models(refresh = TRUE)

## End(Not run)
```

`pliman_clear_sessions` *Clear Cached ONNX Runtime Sessions*

Description

Releases all in-memory compiled neural network sessions (e.g., Grounded-SAM, SAM 2.1, U2-Net) and frees allocated RAM.

Usage

```
pliman_clear_sessions()
```

Value

Invisible `NULL`.

`pliman_configure_dl` *Configure Complete Deep Learning Environment for pliman (100% Native C++)*

Description

Sets up the entire zero-Python, zero-external-package Deep Learning environment in a single call with interactive CLI feedback.

Usage

```
pliman_configure_dl(
  models = "u2netp",
  dir = pliman_model_dir(),
  force = FALSE,
  engine = c("gpu", "cpu")
)
```

Arguments

`models`	Character vector of models to pre-download. Defaults to `"u2netp"`. Pass `"all"` to download all 5 models, or `NULL` to only configure the engine without downloading models.
`dir`	Directory to store models (default: `pliman_model_dir()`).
`force`	Logical. Force re-download and re-configuration even if already present (default `FALSE`).
`engine`	Execution engine: `"cpu"` (default) or `"gpu"`.

Details

This function:

1. Creates and verifies the local model directory (`tools::R_user_dir("pliman", "data")/models`).
2. Downloads and verifies the official Microsoft ONNX Runtime C++ shared library directly.
3. Downloads the specified pre-trained models (`u2netp` and `isnet-general-use` by default).
4. Runs a self-test inference using the native C++ engine to confirm 100% operational status.

Value

Logical `TRUE` invisibly on success.

Examples

```
## Not run:
  pliman_configure_dl()
  pliman_configure_dl(engine = "gpu")

## End(Not run)
```

`pliman_download_model` *Download Pre-Trained Neural Models*

Description

Downloads one or more pre-trained neural models into the user's model directory. Supports downloading by model name, model asset name, dynamic GitHub release asset, or directly from a custom URL (e.g. `https://.../model.onnx`).

Usage

```
pliman_download_model(
  model = "u2netp",
  dir = pliman_model_dir(),
  force = FALSE,
  refresh = FALSE
)
```

Arguments

`model`	Character vector naming the model(s) or direct URL(s) to download. Can be a known model name (e.g. `"yolo26n"`, `"rmbg-2.0"`), an asset filename (e.g. `"yolo26x-cls.onnx"`, `"yolo26s-obb.pt"`), `"all"`, or a direct HTTP/HTTPS URL.
`dir`	Directory to save the model files (default: `pliman_model_dir()`).
`force`	Logical. If `TRUE`, forces re-download even if the file exists (default `FALSE`).
`refresh`	Logical. If `TRUE`, forces re-fetching the GitHub release asset catalog (default `FALSE`).

Value

A character vector with the absolute path(s) to the downloaded model file(s).

Examples

```
## Not run:
  # Download a single model
  pliman_download_model("u2netp")
```

```
  # Download directly from a custom URL
  pliman_download_model("https://example.com/custom_yolo.onnx")

  # Download all models at once
  pliman_download_model("all")

## End(Not run)
```

`pliman_download_onnx` *Download Microsoft ONNX Runtime C++ Library Directly*

Description

Downloads and installs the official Microsoft ONNX Runtime C++ shared library directly from Microsoft's GitHub releases, with **zero external package dependencies** and **zero Python**.

Usage

```
pliman_download_onnx(
  version = "1.20.1",
  dir = pliman_onnx_dir(),
  force = FALSE,
  engine = c("gpu", "cpu")
)
```

Arguments

`version`	Character string with the ONNX Runtime release version (default `"1.20.1"`).
`dir`	Directory to store the library (default: `pliman_onnx_dir()`).
`force`	Logical. If `TRUE`, forces re-download even if already present (default `FALSE`).
`engine`	Execution engine: `"cpu"` (default) or `"gpu"`. When `"gpu"` on Windows, downloads and configures `onnxruntime-directml.dll` and `DirectML.dll` from Microsoft's NuGet CDN.

Details

Automatically detects the host operating system (Windows, Linux, or macOS), downloads the pre-compiled CPU binary (~15-60 MB), extracts the shared library (`onnxruntime.dll` on Windows, `libonnxruntime.so` on Linux, `libonnxruntime.dylib` on macOS), places it in `pliman`'s binary directory, and configures the environment.

Value

A character string with the path to the installed shared library.

Examples

```
## Not run:
  pliman_download_onnx()
  pliman_download_onnx(engine = "gpu")

## End(Not run)
```

---

pliman_gpu_info *GPU Information and DirectML Status in pliman*

Description

Queries the system's graphics hardware (via DirectX DXGI on Windows) to list available GPU adapters, dedicated video memory (VRAM), and DirectML status.

Usage

```
pliman_gpu_info()
```

Value

A data frame containing information on detected GPU adapters, or a message if no GPU is available.

Examples

```
## Not run:
  pliman_gpu_info()

## End(Not run)
```

---

pliman_images *Sample images*

Description

Sample images installed with the **pliman** package

Format

`*.jpg` format

- `colorcheck.jpg` An image with RGB colorcheck, available at https://github.com/HarryCWright/PlantSizeCl
- `flax_leaves.jpg` Flax leaves in a white background
- `flax_grains.jpg` Flax grains with background light.

- `la_back.jpg` A cyan palette representing the background of images la_pattern, la_leaves, and soybean_touch.
- `la_leaf.jpg` A sample of the leaves in `la_leaves`
- `la_leaves.jpg` Tree leaves with a sample of known area.
- `mult_leaves.jpg` Three soybean leaflets with soybean rust symptoms.
- `objects_300dpi.jpg` An image with 300 dpi resolution.
- `potato_leaves.jpg` Three potato leaves, which were gathered from Gupta et al. (2020).
- `sev_leaf.jpg` A soybean leaf with a blue background.
- `sev_leaf_nb.jpg` A soybean leaf without background.
- `sev_back.jpg` A blue palette representing the background of `sev_leaf`.
- `sev_healthy.jpg` Healthy area of `sev_leaf`.
- `sev_sympt.jpg` The symptomatic area `sev_leaf`.
- `shadow.jpg` A shaded leaf, useful to test adaptive thresholding
- `soy_green.jpg` Soybean grains with a white background.
- `soybean_grain.jpg` A sample palette of the grains in `soy_green`.
- `soybean_touch.jpg` Soybean grains with a cyan background touching one each other.
- `field_mosaic.jpg` An UVA image from a soybean field.

`*.tif` format

The following `.tif` files are provided as sample data, representing a slice from a large orthomosaic with soybean plots in the vegetative stage. These files were kindly provided by Arthur Bernardeli.

- `ortho.tif`: An orthomosaic with soybean plots (5 rows and 3 columns).
- `dsm.tif`: A digital surface model (DSM) for the soybean plots.
- `dtm.tif`: A digital terrain model (DTM) for the area.
- `mask.tif`: A mask that represents the soybean plants.

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Source

Personal data, Gupta et al. (2020).

References

Gupta, S., Rosenthal, D. M., Stinchcombe, J. R., & Baucom, R. S. (2020). The remarkable morphological diversity of leaf shape in sweet potato (Ipomoea batatas): the influence of genetics, environment, and G×E. New Phytologist, 225(5), 2183–2195. doi:10.1111/NPH.16286

pliman_indexes_ican_compute *List Computable Indexes Based on Available Bands*

Description

This function reads index equations from a CSV file included in the `pliman` package, determines which bands are used in each index equation, and checks which indexes can be computed based on the provided available bands.

Usage

```
pliman_indexes_ican_compute(available)
```

Arguments

`available`	A character vector of available bands (e.g., `c("R", "G")`).

Value

A data frame of indexes that can be computed with the available bands.

Examples

```
if (interactive()) {
library(pliman)
available_bands <- c("R", "G")
computable_indexes <- pliman_indexes_ican_compute(available_bands)
print(computable_indexes)
}
```

pliman_model_dir *Model Directory Management for pliman Deep Learning Models*

Description

Gets or sets the local directory where pre-trained ONNX models are stored. Defaults to the standard R user data directory: `tools::R_user_dir("pliman", which = "data")/models`.

Usage

```
pliman_model_dir(dir = NULL)
```

Arguments

`dir`	Optional character string specifying a custom directory path. If provided, creates the directory if it does not exist.

Value

A character string with the normalized absolute path to the model directory.

Examples

```
pliman_model_dir()
```

---

pliman_models_base_url *Get Base URL for Pre-Trained Neural Models*

Description

Returns the base URL for downloading pre-trained ONNX models from GitHub Releases. Can be customized globally via `options(pliman.models_base_url = "...")`.

Usage

```
pliman_models_base_url()
```

Value

Character URL string ending in a slash.

---

pliman_onnx_dir *Directory for ONNX Runtime C++ Shared Libraries*

Description

Gets or sets the local directory where the official Microsoft ONNX Runtime C++ shared libraries (`onnxruntime.dll`, `libonnxruntime.so`, or `libonnxruntime.dylib`) are saved.

Usage

```
pliman_onnx_dir(dir = NULL)
```

Arguments

`dir`	Optional custom directory path.

Value

A character string with the path to the bin directory.

Examples

```
pliman_onnx_dir()
```

pliman_onnx_lib_path *Locate ONNX Runtime C++ Shared Library*

Description

Finds the local path to the ONNX Runtime dynamic library (`onnxruntime.dll` on Windows, `libonnxruntime.so` on Linux, `libonnxruntime.dylib` on macOS), or `onnxruntime-directml.dll` when `engine = "gpu"` on Windows.

Usage

```
pliman_onnx_lib_path(engine = c("gpu", "cpu"))
```

Arguments

`engine`	Execution engine: `"cpu"` (default) or `"gpu"`.

Value

Character path to the dynamic library or `NULL` if not found.

pliman_viewer *Global option for controlling the viewer in pliman package*

Description

Users can set the value of this option using `options("pliman_viewer", value)`. The default value is "base". Use "mapview" to allow image to be plotted/edited using the R packages mapview and mapedit

plot.image *Plot method for* `image` *objects*

Description

Render an `image` object using native base R graphics, with support for title, custom bounding limits (`xlim`, `ylim`), and axes.

Usage

```
## S3 method for class 'image'
plot(
  x,
  y = NULL,
  colormode = NULL,
  main = NULL,
  axes = FALSE,
  xlab = "",
  ylab = "",
  xlim = NULL,
  ylim = NULL,
  asp = 1,
  interpolate = TRUE,
  max_pixels = 2e+06,
  add = FALSE,
  ...
)
```

Arguments

`x`	An `image` object.
`y`	Unused.
`colormode`	Unused; reserved for compatibility.
`main`	Title for the plot.
`axes`	Logical; should axes be drawn? Default `FALSE`.
`xlab`	X-axis label.
`ylab`	Y-axis label.
`xlim`	X-axis limits.
`ylim`	Y-axis limits.
`asp`	Aspect ratio. Default `1`.
`interpolate`	Logical; should raster image be interpolated? Default `TRUE`.
`max_pixels`	Maximum number of pixels to render on screen plot. Default `2e6` (2 MP) for fast rendering. Set to `NULL` for full resolution.
`add`	Logical; add to existing plot window? Default `FALSE`.
`...`	Additional graphical arguments passed to `graphics::rasterImage()`.

Value

Invisibly returns the plotted `image` object.

Examples

```
## Not run:
img <- image(array(as.raw(sample(0:255, 30000)), dim = c(100, 100, 3)))
plot(img, main = "Sample Image")
plot(img, xlim = c(20, 80), ylim = c(20, 80), main = "Zoomed Region")

## End(Not run)
```

---

plot.image_shp *S3 method* plot *for* image_shp *objects*

Description

Draws the bounding boxes for each object computed with image_shp().

Usage

```
## S3 method for class 'image_shp'
plot(
  x,
  img = NULL,
  col_line = "black",
  size_line = 2,
  col_text = "black",
  size_text = 0.75,
  ...
)
```

Arguments

x	An object computed with image_shp().
img	The image that was used to compute the shapefile (optional)
col_line, col_text	The color of the line/text in the grid. Defaults to "red".
size_line, size_text	The size of the line/text in the grid. Defaults to 2.5.
...	Currently not used.

Value

A NULL object

Examples

```
if (interactive()) {
library(pliman)
flax <- image_pliman("flax_leaves.jpg")
shape <- image_shp(flax, nrow = 3, ncol = 5)

# grid on the existing image
plot(flax)
plot(shape)
}
```

---

plot.root_analysis *Plot method for root_analysis objects*

Description

Plot method for root_analysis objects

Usage

```
## S3 method for class 'root_analysis'
plot(
  x,
  type = c("all", "skeleton", "hierarchy", "classification", "overlay", "diameter",
    "depth", "angles", "classes", "profile", "envelope", "shape", "topology",
    "angles_depth", "tropism", "diameter_dist", "density", "orders", "order_metrics",
    "binary"),
  which = NULL,
  background = if (!is.null(x$original_image)) "image" else "white",
  nodes = TRUE,
  alpha = 0.85,
  col = NULL,
  lwd = 2,
  cex_nodes = 1,
  legend_pos = "auto",
  cex_legend = NULL,
  axes = TRUE,
  grid = TRUE,
  ...
)
```

Arguments

`x`	An object of class `root_analysis`.
`type`	Type of plot: • `"all"` (default): 4-panel diagnostic phenotyping dashboard.

- "skeleton": Topological skeleton map with primary/lateral curved root paths and nodes.
- "overlay": Original image with translucent diameter heatmap and topological nodes overlaid.
- "diameter": Continuous diameter heatmap (with optional background and nodes).
- "depth": Stratified depth layering barplot with root crossings.
- "angles": Branching angles distribution histogram.
- "classes": Diameter classes breakdown.

which	Optional alias for type.
background	Background for skeleton, diameter, or overlay plot: "image" (original image if available), "white", or "black". Defaults to "image" if original image exists, otherwise "white".
nodes	Logical. If TRUE, overlays the topological nodes (Crown, Branch points, and Root tips).
alpha	Numeric transparency in [0, 1] for the diameter heatmap overlay (default: 0.65).
col	Optional vector of colors for the diameter heatmap palette.
lwd	Line width for drawing root skeleton segments (default: 2.0).
legend_pos	Position of the legend: "auto" (default, automatically places legend in the corner with fewest/no root obstacles), "topleft", "topright", "bottomleft", "bottomright", or FALSE to omit legends.
cex_legend	Character expansion factor for the legend (default: NULL, which automatically uses 0.65 for multi-panel dashboards and 0.75 for individual plots).
axes	Logical. If TRUE (default), displays coordinate axes. If FALSE, axes are suppressed.
grid	Logical. If TRUE (default), displays standardized subtle background grid lines on analytical and distribution charts. If FALSE, grid lines are suppressed.
...	Additional graphical arguments passed to plotting functions.

---

plot.yolo_fewshot_model

Plot a Few-Shot Exemplar Model

Description

Plots reference image(s) with bounding boxes, segmentation polygons, and clicked exemplar points. When the model contains multiple reference images, they are arranged automatically in a multi-panel grid. Setting rainbow = TRUE renders each detected instance in distinct vibrant colors.

Usage

```
## S3 method for class 'yolo_fewshot_model'
plot(
  x,
  task = x$task,
  rainbow = TRUE,
  show_boxes = NULL,
  show_polygons = NULL,
  show_points = TRUE,
  col_border = "#00FFCC",
  col_poly = "#00FFCC55",
  col_point = "#FFFF00",
  lwd = 2,
  mfrow = NULL,
  which = NULL,
  ...
)
```

Arguments

`x`	A `yolo_fewshot_model` object.
`task`	Target task: `"segment"` or `"detect"`. Defaults to `x$task`.
`rainbow`	Logical. If `TRUE`, colors each detected instance with distinct vibrant hues. Defaults to `FALSE`.
`show_boxes`	Logical. Show bounding boxes. Defaults to `TRUE` for detection tasks.
`show_polygons`	Logical. Show segmentation polygons. Defaults to `TRUE` for segmentation tasks.
`show_points`	Logical. Show clicked exemplar points. Defaults to `TRUE`.
`col_border`	Border color when `rainbow = FALSE`. Defaults to `"#00FFCC"`.
`col_poly`	Fill color for polygons when `rainbow = FALSE`. Defaults to `"#00FFCC55"`.
`col_point`	Point color for exemplars. Defaults to `"#FFFF00"`.
`lwd`	Line width for borders. Defaults to `2`.
`mfrow`	Numeric vector of length 2 for grid layout (e.g. `c(2, 2)`). If `NULL`, calculated automatically.
`which`	Indices or names of reference images to plot. If `NULL` (default), plots all reference images.
`...`	Additional arguments passed to graphics methods.

Value

Invisible `x`.

`plot.yolo_train` *Plot Diagnostic Evaluations and Performance Curves for YOLO Models*

Description

Generates high-resolution, publication-ready base R vector graphics for evaluating trained YOLO models. Supports multi-panel overview dashboards, confusion matrices (raw counts and normalized rates), precision-recall (PR) curves, confidence vs. F1-score curves, and epoch-wise loss/metric progressions.

Usage

```
## S3 method for class 'yolo_train'
plot(
  x,
  which = c("all", "metrics", "confusion_matrix_norm", "confusion_matrix", "pr_curve",
    "f1_curve", "results", "labels"),
  native = TRUE,
  bg = "#fafafa",
  mar = 0.5,
  ...
)
```

Arguments

`x`	An object of class `yolo_train` or character path/name of a model.
`which`	Which plot to display. One of: • `"all"` (default): A 2x2 high-resolution diagnostic dashboard with Loss curves, Validation mAP progression, Confusion Matrix, and Precision-Recall curve. • `"metrics"`: High-resolution native R vector curves plotting loss, mAP, precision, and recall progression from `results.csv`. • `"confusion_matrix_norm"`: Normalized confusion matrix heatmap (counts + rates). • `"confusion_matrix"`: Raw count confusion matrix heatmap. • `"pr_curve"`: Precision-Recall (PR) curve with shaded area under the curve and mAP50. • `"f1_curve"`: F1 score vs. Confidence curve. • `"results"`: Alias for `"metrics"`. • `"labels"`: Dataset label distribution and bounding box correlogram (if available).
`native`	Logical. When `TRUE` (default), renders crisp, resizable native R base graphic vector plots directly from metrics and CSV evaluation tables rather than displaying raster image files.

`bg`	Background color for native plots. Defaults to `"#fafafa"`.
`mar`	Plot margins passed to base graphics. Defaults to `0.5`.
`...`	Additional arguments passed to base graphics.

Details

Diagnostic Evaluation Statistics and Mathematical Formulas:

1. Spatial Overlap (Intersection over Union, IoU): Given a predicted bounding box or segmentation mask $\mathcal{B}_{\text{pred}}$ and a ground truth annotation $\mathcal{B}_{\text{true}}$, the spatial overlap is quantified by the Jaccard index:

$$\text{IoU} = \frac{|\mathcal{B}_{\text{pred}} \cap \mathcal{B}_{\text{true}}|}{|\mathcal{B}_{\text{pred}} \cup \mathcal{B}_{\text{true}}|}$$

A detection is designated as a True Positive (TP) if IoU $\geq$ threshold (typically 0.50) and the predicted class matches ground truth.

2. Confusion Matrix with Background Class: Object detection models must detect both the presence and location of objects. Hence, detections can fail by predicting an incorrect class, by hallucinating an object on the background, or by failing to detect a real object. The evaluation matrix M of dimensions $(N_c+1) \times (N_c+1)$ includes the background class:

- **Cell** (i, j) **for** $i, j \leq N_c$**:** Ground truth object of class i predicted as class j. Diagonal entries $M(i, i)$ represent **True Positives (TP)**.
- **Last column** $(i, N_c + 1)$**:** Ground truth object of class i completely missed by the model (misclassified as background, i.e., **False Negatives, FN**).
- **Last row** $(N_c + 1, j)$**:** Background region falsely predicted as an object of class j (i.e., **False Positives, FP**).

The row-normalized confusion matrix computes the empirical conditional probability $P(\hat{Y} = j \mid Y = i)$:

$$M_{\text{norm}}(i, j) = P(\hat{Y} = j \mid Y = i) = \frac{M(i, j)}{\sum_{k=1}^{N_c+1} M(i, k)}$$

The diagonal entry $M_{\text{norm}}(i, i)$ corresponds directly to the per-class Sensitivity (Recall). Conversely, the column-normalized entry represents the empirical precision $P(Y = i \mid \hat{Y} = j)$:

$$M_{\text{prec}}(i, j) = P(Y = i \mid \hat{Y} = j) = \frac{M(i, j)}{\sum_{k=1}^{N_c+1} M(k, j)}$$

3. Precision and Recall: At any given confidence threshold $\tau \in [0, 1]$:

$$\text{Precision}(\tau) = \frac{\text{TP}(\tau)}{\text{TP}(\tau) + \text{FP}(\tau)}$$

$$\text{Recall}(\tau) = \frac{\text{TP}(\tau)}{\text{TP}(\tau) + \text{FN}(\tau)}$$

Precision measures the accuracy of positive predictions (minimizing false alarms), while Recall measures the completeness of coverage (minimizing missed objects).

4. F1-Score vs. Confidence Curve: The F1-score is the harmonic mean of Precision and Recall as a function of the confidence threshold τ:

$$F_1(\tau) = 2 \times \frac{\text{Precision}(\tau) \times \text{Recall}(\tau)}{\text{Precision}(\tau) + \text{Recall}(\tau)} = \frac{2\,\text{TP}(\tau)}{2\,\text{TP}(\tau) + \text{FP}(\tau) + \text{FN}(\tau)}$$

The peak of this curve identifies the optimal operating threshold τ^* that maximizes the harmonic trade-off:

$$\tau^* = \arg\max_{\tau} F_1(\tau)$$

5. Precision-Recall (PR) Curve and Average Precision (AP): As the confidence threshold τ ranges from 1 down to 0, recall monotonically increases from 0 to 1, while precision generally declines. The interpolated precision $p_{\text{interp}}(r)$ at recall level r is the maximum precision achieved at any recall $\tilde{r} \geq r$:

$$p_{\text{interp}}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$$

Average Precision (AP) is the integral area under this interpolated curve, numerically approximated via the 101-point COCO evaluation grid:

$$\text{AP} = \int_0^1 p_{\text{interp}}(r)\, dr \approx \frac{1}{101} \sum_{k=0}^{100} p_{\text{interp}}\left(\frac{k}{100}\right)$$

6. Mean Average Precision (mAP):

- **mAP@50 (mAP50):** Mean AP across all N_c target classes evaluated at an IoU threshold of 0.50:
$$\text{mAP50} = \frac{1}{N_c} \sum_{c=1}^{N_c} \text{AP}_c^{(0.50)}$$
- **mAP@50-95 (mAP50-95):** The primary metric of the COCO benchmark, computed as the unweighted mean of mAP across 10 IoU thresholds from 0.50 to 0.95 in steps of 0.05:
$$\text{mAP50-95} = \frac{1}{10 \cdot N_c} \sum_{k=0}^{9} \sum_{c=1}^{N_c} \text{AP}_c^{(0.50+0.05k)}$$

Interpretation Guide for Diagnostic Plots:

- `which = "all"`: Displays a 2x2 publication-ready dashboard combining: (1) Training & Validation Loss curves (detecting divergence, underfitting, or overfitting), (2) Validation mAP progression across epochs (with the best epoch highlighted with a star), (3) Normalized Confusion Matrix (identifying frequent misclassifications and background leakage), and (4) Precision-Recall curve with shaded area under the curve (quantifying overall detector precision).
- `which = "confusion_matrix"`: Displays raw detection counts across all classes and background. High counts in the bottom row indicate excessive false positive background detections; high counts in the rightmost column indicate missed objects.
- `which = "confusion_matrix_norm"`: Displays normalized conditional probabilities. The diagonal values represent per-class sensitivity (percentage of true objects correctly detected).

- `which = "pr_curve"`: A steep curve maintaining precision close to 1.0 across high recall indicates a highly robust model with minimal false positives.
- `which = "f1_curve"`: Used to choose the ideal confidence threshold for downstream inference in `image_detect_dl` or `image_segment_dl`.
- `which = "metrics"`: Displays a 2x3 panel of detailed epoch-by-epoch loss curves (box, class, mask), precision, recall, and learning rate schedules.

Value

Invisibly returns `x` or plotted data frame.

See Also

`yolo_plot()`, `yolo_results()`, `yolo_train()`

Examples

```
## Not run:
library(pliman)

# Load diagnostics from a previously trained model
res <- yolo_results("capsulas_det")

# Inspect training summary and plot 2x2 diagnostic dashboard
print(res)
plot(res)

# Plot individual evaluation curves in high-resolution vector format:
plot(res, which = "confusion_matrix")
plot(res, which = "confusion_matrix_norm")
plot(res, which = "pr_curve")
plot(res, which = "f1_curve")
plot(res, which = "metrics")

## End(Not run)
```

---

plot_bbox *Add Bounding Boxes to an Existing Plot*

Description

This function overlays bounding boxes onto an existing plot.

Usage

```
plot_bbox(bbox_list, col = "red")
```

Arguments

bbox_list	A list of bounding boxes, as returned by `object_bbox()`.
col	The color for the bounding boxes. Defaults to `"red"`.

Value

None (adds bounding boxes to an existing plot).

Examples

```
if(interactive()){
plot(NA,
    xlim = c(0, 200),
    ylim = c(0, 200),
    asp = 1)
contours <- list(
  matrix(c(10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
           110, 120, 130, 140, 150, 160, 170, 180, 190, 200),
         ncol = 2, byrow = FALSE)
)
bbox_list <- object_bbox(contours)
plot_bbox(bbox_list)
}
```

---

plot_id *Generate plot IDs with different layouts*

Description

Based on a shapefile, number of columns and rows, generate plot IDs with different layouts.

Usage

```
plot_id(
  shapefile = NULL,
  nrow,
  ncol,
  layout = c("tblr", "tbrl", "btlr", "btrl", "lrtb", "lrbt", "rltb", "rlbt"),
  plot_prefix = "P",
  serpentine = FALSE
)
```

Arguments

shapefile	An object computed with `shapefile_build()`
nrow	The number of columns
ncol	The number of rows

`layout` Character: one of

- `'tblr'` for top/bottom left/right orientation
- `'tbrl'` for top/bottom right/left orientation
- `'btlr'` for bottom/top left/right orientation
- `'btrl'` for bottom/top right/left orientation
- `'lrtb'` for left/right top/bottom orientation
- `'lrbt'` for left/right bottom/top orientation
- `'rltb'` for right/left top/bottom orientation
- `'rlbt'` for right/left bottom/top orientation

`plot_prefix` The plot_id prefix. Defaults to `'P'`.

`serpentine` Create a serpentine-based layout? Defaults to `FALSE`.

Value

A list of plot IDs with specified layout and updated rows/columns.

`plot_index` *Plot an image index*

Description

Plot an image index

Usage

```
plot_index(
  img = NULL,
  object = NULL,
  index = NULL,
  remove_bg = TRUE,
  viewer = get_pliman_viewer(),
  all_layers = TRUE,
  layer = 1,
  max_pixels = 1e+06,
  downsample = NULL,
  downsample_fun = NULL,
  color_regions = custom_palette(n = 100),
  ncol = NULL,
  nrow = NULL,
  aspect_ratio = NA
)
```

Arguments

`img`	An optional `image` object or an object computed with `image_index()`. If `object` is provided, then the input image is obtained internally.
`object`	An object computed with `analyze_objects()` using the argument `return_mask = TRUE`.
`index`	The index to plot. Defaults to the index computed from the `object` if provided. Otherwise, the `B` index is computed. See `image_index()` for more details.
`remove_bg`	Logical value indicating whether to remove the background when `object` is provided. Defaults to `TRUE`.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`all_layers`	Render all layers when `img` is an object computed with `image_index()` and `viewer = "mapview"`?.
`layer`	The layer to plot when `img` is an object computed with `image_index()` and `viewer = "mapview"`. Defaults to the first layer (first index computed).
`max_pixels`	integer > 0. Maximum number of cells to plot the index. If `max_pixels < npixels(img)`, downsampling is performed before plotting the index. Using a large number of pixels may slow down the plotting time.
`downsample`	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to `NULL`, which will find the best downsampling factor to approximate the `max_pixels` value.
`downsample_fun`	function; if given, downsampling will apply `downsample_fun``` to each of the the subtiles.
`color_regions`	The color palette for displaying index values. Default is `custom_palette()`.
`nrow`, `ncol`	The number of rows or columns in the plot grid. Defaults to `NULL`, i.e., a square grid is produced.
`aspect_ratio`	Numeric, giving the aspect ratio y/x. Defaults to `NA`. See `graphics::plot.window()` for more details.

Value

None

Examples

```
if (interactive()) {
# Example usage:
library(pliman)
img <- image_pliman("sev_leaf.jpg")
plot_index(img, index = c("R", "G"))
}
```

`plot_index_shp` *Plot rectangles colored by a quantitative attribute and overlay on an RGB image*

Description

This function plots rectangles on top of an RGB image, where each rectangle is colored based on a quantitative variable. The quantitative variable is specified in the `attribute` argument and should be present in the `object_index` of the `object` computed using `analyze_objects_shp()`. The rectangles are colored using a color scale.

Usage

```
plot_index_shp(
  object,
  attribute = "coverage",
  r = 1,
  g = 2,
  b = 3,
  color = c("red", "yellow", "darkgreen"),
  viewer = c("mapview", "base"),
  max_pixels = 5e+05,
  downsample = NULL,
  downsample_fun = NULL,
  alpha = 0.7,
  legend.position = "bottom",
  na.color = "gray",
  classes = 6,
  round = 3,
  horiz = TRUE
)
```

Arguments

`object`	An object computed with `analyze_objects_shp()`.
`attribute`	The name of the quantitative variable in the `object_index` to be used for coloring the rectangles.

r, g, b	The layer for the Red, Green and Blue band, respectively. Defaults to 1, 2, and 3.
color	A vector of two colors to be used for the color scale.
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
max_pixels	integer > 0. Maximum number of cells to plot the index. If max_pixels < npixels(img), downsampling is performed before plotting the index. Using a large number of pixels may slow down the plotting time.
downsample	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to NULL, which will find the best downsampling factor to approximate the max_pixels value.
downsample_fun	function; if given, downsampling will apply downsample_fun`` to each of the the subtiles.
alpha	The transparency level of the rectangles' color (between 0 and 1).
legend.position	The position of the color legend, either "bottom" or "right".
na.color	The color to be used for rectangles with missing values in the quantitative variable.
classes	The number of classes in the color scale.
round	The number of decimal places to round the legend values.
horiz	Logical, whether the legend should be horizontal (TRUE) or vertical (FALSE).

Value

The function plots rectangles colored by the specified quantitative variable on top of the RGB image and shows the continuous color legend outside the plot.

Examples

```
if (interactive()) {
library(pliman)

# Computes the DGCI index for each flax leaf
flax <- image_pliman("flax_leaves.jpg", plot =TRUE)
res <-
   analyze_objects_shp(flax,
                       buffer_x = 0.1,
                       buffer_y = 0.02,
                       nrow = 3,
                       ncol = 5,
```

```
                        plot = FALSE,
                        object_index = "DGCI")
plot_index_shp(res, attribute = "DGCI")
}
```

`plot_line_segment` *Plot Detected Line Segments*

Description

Plots the detected line segments from the output of `image_line_segment()`. Each segment is drawn as a red line on the existing plot.

Usage

```
plot_line_segment(x, col = "red", lwd = 1)
```

Arguments

`x`	A list returned by `image_line_segment()`, containing detected line segments.
`col`	The color of lines
`lwd`	The width of lines. Defaults to 1

Value

No return value. The function adds line segments to an existing plot.

Examples

```
library(pliman)
```

`plot_lw` *Plot length and width lines on objects*

Description

This function plots the length and width lines given an `object` computed with `analyze_objects()`. The function does not call `plot.new`, so it must be called after an image is plotted. This can be done either using, e.g., `plot(img)`, or `analyze_objects(..., plot = TRUE)`.

Usage

```
plot_lw(
  object,
  col_length = "red",
  col_width = "green",
  lwd_length = 2,
  lwd_width = 2
)
```

Arguments

`object`	An object computed with `analyze_objects()`.
`col_length`	The color of the length line. Default is `"red"`.
`col_width`	The color of the width line. Default is `"green"`.
`lwd_length`	The line width of the length line. Default is 2.
`lwd_width`	The line width of the width line. Default is 2.

Details

This function takes an object computed with `analyze_objects()` and plots the length and width lines of each object onto an image. The length and width lines are calculated based on the position and orientation of the object, and are plotted using the specified colors and line widths.

Examples

```
if (interactive()) {
img <- image_pliman("flax_leaves.jpg")
res <- analyze_objects(img, watershed = FALSE, show_contour = FALSE)
plot_lw(res)
}
```

---

plot_yolo_heatmap *Plot YOLO Feature Activation Heatmap*

Description

Visualizes the 2D continuous activation heatmap or prototype feature energy extracted during YOLO inference, either standalone or overlaid with alpha transparency onto the original image.

Usage

```
plot_yolo_heatmap(
  img,
  heatmap = NULL,
  col_palette = "inferno",
  alpha = 0.55,
  overlay = TRUE,
  title = NULL,
  plot = TRUE,
  ...
)
```

Arguments

`img`	An `Image` object, matrix, array, or an object returned by `image_segment_dl()` or `image_detect_dl()` with `return_features = TRUE`.
`heatmap`	An `Image` or 2D matrix representing the activation heatmap. If `NULL` and `img` contains a `$heatmap` or `attr(img, "heatmap")`, it is extracted automatically.
`col_palette`	Color palette name (`"inferno"`, `"magma"`, `"viridis"`, `"plasma"`, `"spectral"`), or a custom vector of colors. Default is `"inferno"`.
`alpha`	Numeric transparency of the heatmap overlay on the original image (0 = only original image, 1 = only heatmap). Default is `0.55`.
`overlay`	Logical. If `TRUE` (default), blends the colored heatmap over the original image. If `FALSE`, plots the standalone colorized heatmap.
`title`	Optional title for the plot.
`plot`	Logical. Whether to display the plot immediately. Default is `TRUE`.
`...`	Additional arguments passed to `graphics::plot()`.

Value

An `Image` object containing the colorized / blended heatmap visualization, invisibly.

Examples

```
## Not run:
  library(pliman)
  img <- image_import("tomatoes.jpg")
  res <- image_segment_dl(img, model = "yolo26n-seg", return_features = TRUE)
  plot_yolo_heatmap(res)

## End(Not run)
```

`poly_apex_base_angle` *Calculate the apex and base angles of an object*

Description

This function calculates the apex and base angles of an object. It takes as input a matrix of coordinates and returns the apex angle, base angle, and the coordinates of the apex and base as a list. The angles are computed after the object is aligned in the vertical axis with `poly_align()`.

Usage

```
poly_apex_base_angle(
  x,
  percentiles = c(0.25, 0.75),
  invert = FALSE,
  plot = TRUE
)
```

Arguments

`x`	A matrix of coordinates representing the contour of the object, often obtained with `object_contour()`.
`percentiles`	A numeric vector of two percentiles between 0 and 1 indicating the height of the points from the top to the bottom. The function calculates the apex angle between the two percentiles and the base angle between the lowest point and the highest point.
`invert`	If `TRUE`, aligns the object along the horizontal axis.
`plot`	Plots the polygon with the points? Defaults to `TRUE`.

Value

A list containing the apex angle, base angle, apex coordinates, and base coordinates.

Examples

```
if (interactive()) {
library(pliman)
# a matrix of coordinates
angls <- poly_apex_base_angle(contours[[2]])
angls

# or a list of coordinates
poly_apex_base_angle(contours)
}
```

`poly_pcv` *Compute Perimeter Complexity Value (PCV)*

Description

This function calculates the Perimeter Complexity Value (PCV) for a given set of coordinates representing a contour. The PCV measures the variation of distances between the original coordinates and the smoothed coordinates relative to the perimeter length of the original contour. See more in `details` section.

Usage

```
poly_pcv(x, niter = 100)
```

Arguments

`x`	A matrix or a list of matrices representing the coordinates of the polygon(s).
`niter`	An integer specifying the number of smoothing iterations. See `poly_smooth()` for more details.

Details

The PCV is computed using the following formula:

$$PCV = \frac{sum(dists) \times sd(dists)}{perim}$$

where *dists* represents the distances between corresponding points in the original and smoothed coordinates, and *perim* is the perimeter length of the smoothed contour.

The PCV is computed by first smoothing the input contour using a specified number of iterations. The smoothed contour is then used to compute the distances between corresponding points in the original and smoothed coordinates. These distances reflect the variations in the contour shape after smoothing. The sum of these distances represents the overall magnitude of the variations. Next, the sum of distances is multiplied by the standard deviation of the distances to capture the dispersion or spread of the variations. Finally, this value is divided by the perimeter length of the original contour to provide a relative measure of complexity. Therefore, the PCV provides a relative measure of complexity by considering both the magnitude and spread of the variations in the contour shape after smoothing.

Value

The PCV value(s) computed for the contour(s).

If `x` is a matrix, returns the complexity value of the polygon's perimeter. If `x` is a list of matrices, returns a numeric vector of complexity values for each polygon.

Examples

```
if (interactive()) {
library(pliman)
set.seed(20)
shp <- efourier_shape(npoints = 1000)
poly_pcv(shp)

# increase the complexity of the outline
shp2 <- poly_jitter(shp, noise_x = 20, noise_y = 250, plot = TRUE)

smo <- poly_smooth(shp2, niter = 100, plot = FALSE)
plot_contour(smo, col = "red")
poly_pcv(shp2)
}
```

---

poly_width_at *Width at a given height*

Description

The function computes the polygonal convex hull of the points in x and then returns the number of points that lie below a specified set of heights along the vertical axis of the convex hull.

Usage

```
poly_width_at(
  x,
  at = c(0.05, 0.25, 0.5, 0.75, 0.95),
  unify = FALSE,
  plot = FALSE
)
```

Arguments

x	A vector containing two-dimensional data points (often produced with object_contour).
at	A vector of heights along the vertical axis of the convex hull at which to count the number of points below. The default value is `c(0.05, 0.25, 0.5, 0.75, 0.95)`, which means the function will return the number of points below the 5th, 25th, 50th, 75th, and 95th percentiles of the convex hull. If `at = "heights"` is used, the function returns the width for each point of the object length.
unify	A logical value indicating whether to use the unified convex hull calculation method. If unify = TRUE, coordinates in x will be first bound before computing the convex hull.
plot	A logical value that specifies whether the widths should be plotted.

Details

The convex hull computed from x is aligned along the major axis and then converted to a binary image. For each height in the at vector, the function computes the corresponding row number in the binary image (i.e., the row number that corresponds to the specified height along the vertical axis of the convex hull) and sums the values in that row to obtain the number of points that lie below the specified height. If the convex hull contains multiple polygons and unify = FALSE, the function loops over each polygon and returns a list of the number of points below the specified heights for each polygon. If the convex hull contains only one polygon or multiple polygons and unify = TRUE, the function returns a vector of the number of points below the specified heights for that single polygon.

Value

A vector with the widths of the convex hull at the specified heights or a list of vectors with the widths of each component.

Examples

```
if (interactive()) {
cont <- contours[[2]]
plot_polygon(cont |> conv_hull() |> poly_align())
#  width below 5th, 25th, 50th, 75th, and 95th percentiles of the length
wd <- poly_width_at(cont)
wd

# width along the height
poly_width_at(cont, at = "height", plot = TRUE)

}
```

---

predict_binary_glm *Fast Binary Image Prediction from a Logistic Regression Model*

Description

Evaluates a fitted `glm` logistic regression model directly on image channel matrices without creating massive data frames or running slow `predict.glm` loops over millions of pixels.

Usage

```
predict_binary_glm(model, img)
```

Arguments

`model`	A fitted `glm` logistic regression model object.
`img`	An `image` object.

Value

An integer matrix containing 1 (foreground) and 0 (background).

---

prepare_to_shp *Prepare images to analyze_objects_shp()*

Description

It is a simple wrapper around `image_align()` and `image_crop()`. In this case, only the option `viewer = "base"` is used. To use `viewer = "mapview"`, please, use such functions separately.

Usage

```
prepare_to_shp(img, align = "vertical")
```

Arguments

`img`	A `image` object
`align`	The desired alignment. Either `"vertical"` (default) or `"horizontal"`.

Value

An aligned and cropped `image` object.

Examples

```
if (interactive()) {
img <- image_pliman("flax_leaves.jpg")
prepare_to_shp(img)
}
```

---

print.image *Print method for* `image` *objects*

Description

Print method for `image` objects

Usage

```
## S3 method for class 'image'
print(x, ...)
```

Arguments

`x`	An `image` object.
`...`	Unused.

`print.pliman_video_frames` *Print method for pliman_video_frames*

Description

Print method for pliman_video_frames

Usage

```
## S3 method for class 'pliman_video_frames'
print(x, ...)
```

Arguments

`x`	An object of class `pliman_video_frames`.
`...`	Additional arguments (not used).

`print.root_analysis` *Print method for root_analysis objects*

Description

Print method for root_analysis objects

Usage

```
## S3 method for class 'root_analysis'
print(x, ...)
```

Arguments

`x`	A `root_analysis` object.
`...`	Ignored.

print.yolo_train *Print YOLO Training Summary*

Description

Displays a concise summary of YOLO model training results, performance, and saved artifacts.

Usage

```
## S3 method for class 'yolo_train'
print(x, ...)
```

Arguments

x	An object of class yolo_train.
...	Additional arguments (currently unused).

Value

Invisibly returns x.

random_color *Random built-in color names*

Description

Randomly chooses single or multiple built-in color names which R knows about. See more at grDevices::colors()

Usage

```
random_color(n = 1, distinct = FALSE)
```

Arguments

n	The number of color names. Defaults to 1.
distinct	Logical indicating if the colors returned should all be distinct. Defaults to FALSE.

Value

A character vector of color names

Examples

```
library(pliman)
random_color(n = 3)
```

remove_reflection_cpp

Grayscale Reflection Removal via Masked Median Inpainting

Description

Removes specular reflections from a grayscale index matrix before thresholding. Algorithm:

1. Auto-detects which class is background by sampling border pixels after Otsu.
2. BFS from image borders marks all background pixels reachable from the exterior.
3. Internal holes = background-class pixels NOT reached by BFS (enclosed by grain).
4. Each internal hole pixel is replaced by the MEDIAN of its foreground (grain) neighbours within `radius`. This is orientation-agnostic and never touches the grain boundary or the external background.

Usage

```
remove_reflection_cpp(img, radius = 5L, has_white_bg = TRUE)
```

Arguments

`img`	Grayscale index matrix.
`radius`	Neighbourhood radius for median sampling (default = 5).
`has_white_bg`	Unused — background class is detected automatically.

Value

Corrected grayscale matrix.

rgb_image

Create an RGB Image from Red, Green, and Blue Channels

Description

Combines single-channel matrices, arrays, or `image` objects representing Red, Green, and Blue channels into a 3D color RGB `image` object.

Usage

```
rgb_image(red = NULL, green = NULL, blue = NULL)

rgbImage(red = NULL, green = NULL, blue = NULL)
```

Arguments

`red`, `green`, `blue`

Single-channel matrices, arrays, or `image` objects. Any channel can be `NULL` (will be filled with zeros).

Value

A 3D `image` object of color mode `"Color"`.

`root_analyze` *Advanced Root System Phenotyping Architecture*

Description

`root_analyze()` performs comprehensive, high-throughput phenotyping of plant root systems from scans, camera images, or binary masks, surpassing the analytical capabilities of standalone tools like RhizoVision Explorer and WinRHIZO:

- **Exact Euclidean Distance Transform (EDT)**: Computes continuous local diameters for every skeletal root pixel with sub-pixel precision.
- **Topological Skeletonization & Pruning**: Guo-Hall thinning followed by spurious spur pruning eliminates surface roughness noise without shortening genuine lateral roots.
- **Root Hierarchy Extraction**: Automatically distinguishes the Primary Root Axis (order 1 taproot / seminal root) from Secondary / Lateral roots (order 2 and higher).
- **True Insertion / Branching Angles**: Calculates angular divergence of lateral roots relative to parent root growth vectors in degrees (0-180 deg).
- **3D Root Geometry**: Computes surface area ($\sum \pi \cdot d \cdot l$) and volume ($\sum \frac{\pi}{4} \cdot d^2 \cdot l$) using cylindrical integration.
- **Stratified Depth Profile**: Dissects the root system into horizontal depth horizons, computing root length, surface area, volume, and RhizoVision-style root crossings per layer.
- **Global Network Shape**: Calculates network width, depth, width-to-depth ratio, convex hull area, network solidity, and bushiness.
- **Custom Diameter Size Classes**: Stratifies length, area, and volume across fine, medium, coarse, and user-defined root diameter bins.

Usage

```
root_analyze(
  img,
  index = "GRAY",
  threshold = "otsu",
  otsu_adj = 1.2,
  fill_hull = FALSE,
  max_order = 3,
```

```
  background = "auto",
  pixel_size = 1,
  dpi = NULL,
  unit = "mm",
  num_classes = 4,
  depth_slices = 10,
  crown_point = NULL,
  cluster_radius = 3,
  prune_length = NULL,
  gravitropic_alpha = 3,
  closing = 1,
  fill_size = 200,
  radius_prune = 1.2,
  dissolve_cycles = TRUE,
  plot = FALSE,
  plot_type = "all",
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An `Image` object, a 2D binary matrix, a file path to an image, or a list of images.
`index`	Color index used to segment roots if `img` is an RGB image (default: `"GRAY"`). Other valid options include `"R"`, `"G"`, `"B"`, `"VARI"`, `"B-G"`, etc.
`threshold`	Segmentation threshold for binarization. Defaults to `"otsu"` (which applies Otsu thresholding scaled by `otsu_adj`, by default +20%). Can also be a numeric threshold in `[0, 1]` (or `[0, 255]`), `"adaptive"`, or `"otsu_strict"`.
`otsu_adj`	Numeric multiplier applied to the Otsu threshold (default: 1.2, i.e., +20% adjustment). Shifts the threshold towards the background to prevent losing fine, faint lateral roots. Set to `1.0` or use `threshold = "otsu_strict"` for standard unadjusted Otsu.
`fill_hull`	Logical. If `TRUE`, fills internal holes/cavities within the segmented binary root mask in R before passing to the C++ engine (default: `FALSE`). Useful to prevent hollow rings in thick root axes caused by glare, but should remain `FALSE` for dense root systems to avoid bridging narrow gaps between adjacent roots.
`max_order`	Integer specifying the maximum root branching order to classify based on node hierarchy (default: 3). `1` = Primary only; `2` = Primary and Lateral; `3` = Primary, Secondary, and Tertiary+ roots.
`background`	Background color: `"auto"` (default, senses automatically from image corners), `"white"`, or `"black"`.
`pixel_size`	Physical size of one pixel in the chosen `unit` (default: 1.0).

`dpi` Optional integer scanner/camera resolution in dots per inch. If provided, overrides `pixel_size` using `pixel_size = 25.4 / dpi` (when `unit = "mm"`).

`unit` Unit of measurement for lengths and coordinates (default: `"mm"`).

`num_classes` Number of diameter size classes to compute based on quantiles (default: 4). Can also be a numeric vector of custom diameter cut points (e.g., `c(0.5, 2.0)`).

`depth_slices` Integer specifying the number of horizontal depth strata to compute vertical distribution profiles (default: 10).

`crown_point` Optional numeric vector of length 2 `c(x, y)` specifying the coordinates (in pixels) of the root crown / stem base. If `NULL` (default), the crown is automatically detected at the topmost foreground pixel of the largest connected root component. Manually providing `crown_point = c(x, y)` is especially recommended for cuttings or flat-cut stem stumps, where a wide horizontal cut surface can otherwise split into a false "Y" bifurcation at the top.

`cluster_radius` Numeric radius (in pixels) within which adjacent branching junctions (degree >= 3) are clustered and merged into a single consensus node (default: 3 pixels). Thick roots often generate multi-pixel clusters of junction pixels during skeletonization; increasing `cluster_radius` (e.g., 5 to 8 pixels) merges these micro-junctions into clean, single branching nodes.

`prune_length` Optional integer minimum length in pixels for terminal spur branches (default: `NULL`, which defaults to 10 pixels in C++). Tips with path length shorter than this are pruned away as noise. If fine lateral roots or short emerging tips are being lost, reduce this to a smaller value (e.g., `3` to `5`).

`gravitropic_alpha` Numeric weight (default: 3) controlling how strongly the primary root detection favours downward-growing edges over purely diameter-based selection. Higher values (e.g., 3.0–5.0) guide the taproot identification along the downward vertical axis, even when horizontal lateral roots are thicker than the primary axis. Set to `0` to use diameter alone.

`closing` Integer morphological closing radius (in pixels) applied to the binary mask in C++ (default: 1). Performs morphological dilation followed by erosion. Smooths bark roughness and bridges micro-fissures in segmented roots. Set to `0` when analyzing dense root systems with closely spaced or overlapping roots to prevent artificial "webbing" (connecting membranes) between adjacent roots in the diameter map and skeleton.

`fill_size` Integer maximum area (in pixels) of enclosed cavities to fill internally in C++ (default: 200). Converts enclosed background regions (holes) with area <= `fill_size` pixels into root foreground, preventing false loops and hollow centers in thick root axes caused by glare or bark reflectance. Operates independently of `fill_hull`. Set to `0` to completely disable internal hole filling, avoiding the fusion of narrow gaps between roots.

`radius_prune` Numeric multiplier applied to local parent root radius for adaptive spur pruning (default: 1.2). Terminal branches with length shorter than `max(prune_length, parent_radius * radius_prune)` are pruned away. This eliminates surface roughness, bark bumps, and root hairs along thick roots without shortening genuine lateral roots. Lower values (e.g., 0.8 to 1.0) preserve more fine lateral roots.

`dissolve_cycles` Logical. If `TRUE` (default), uses Kruskal's algorithm on a Maximum Spanning Forest (MSF) to dissolve all internal loops and ladder rungs, guaranteeing an acyclic tree graph. In 2D projections, overlapping or touching roots create closed loops; the algorithm removes the edge with the lowest weight (weighted by $\text{diameter}^{3.5} \cdot \text{vertical\_ratio}^{1.5} / \sqrt{\text{length}}$), preferentially preserving thick, downward-growing parent roots and cutting horizontal or looping cross-connections. Set to `FALSE` if you wish to preserve all bridging or intersecting root segments in the skeleton and total length calculations.

`plot` Logical. If `TRUE` (default: `FALSE`), plots the analyzed root system.

`plot_type` Type of visualization: `"all"` (default 4-panel dashboard), `"skeleton"`, `"hierarchy"` (or `"classification"`), `"overlay"`, `"diameter"`, `"depth"`, `"angles"`, `"classes"`, `"profile"` (or `"envelope"`), `"topology"`, `"diameter_dist"`, or `"orders"`.

`verbose` Logical. If `TRUE` (default), displays progress messages.

`...` Additional arguments passed to `image_binary()`.

Value

An object of class `root_analysis` (or `root_analysis_list` for multiple images) containing:

- `summary`: Single-row `data.frame` with all phenotypic traits.
- `roots`: Detailed `data.frame` of all individual root segments (order, length, avg diameter, surface area, volume, branching angle, insertion depth).
- `nodes`: Coordinates, degrees, and types (`crown`, `junction`, `tip`) of all network nodes.
- `paths`: List of matrices containing (x, y) coordinates for each curved root segment.
- `diameter_classes`: Distribution of length, surface area, and volume across diameter classes.
- `depth_profile`: Stratified depth distribution with lengths, volumes, and root crossings.
- `skeleton`: 2D binary matrix of the pruned skeleton.
- `diameter_map`: 2D matrix of local root diameters.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

References

Seethepalli, A., Guo, H., Liu, X., Ruff, T., & York, L. M. (2021). RhizoVision Explorer: open-source software for generalized root phenotyping using image analysis. Plant Physiology, 187(2), 739-757.

Examples

```
## Not run:
library(pliman)

# Analyze a root image scanned at 600 DPI
res <- root_analyze("root_sample.jpg", dpi = 600)

# View global phenotypic summary
res$summary

# Plot 4-panel diagnostic dashboard
plot(res)

# Plot continuous diameter heatmap
plot(res, type = "diameter")

## End(Not run)
```

`sad` *Produces Santandard Area Diagrams*

Description

Given an object computed with `measure_disease()` or `measure_disease_byl()` a Standard Area Diagram (SAD) with `n` images are returned with the respective severity values.

Usage

```
sad(
  object,
  n,
  show_original = FALSE,
  show_contour = FALSE,
  nrow = NULL,
  ncol = NULL,
  ...
)
```

Arguments

`object`	An object computed with `measure_disease()` or `measure_disease_byl()`.
`n`	The number of leaves in the Standard Area Diagram.

show_original	Show original images? Defaults to FALSE, i.e., a mask is returned.
show_contour	Show original images? Defaults to FALSE, i.e., a mask is returned.
nrow, ncol	The number of rows and columns in the plot. See [image_combine())] [image_combine())]: R:image_combine())
...	Other arguments passed on to measure_disease().

Details

The leaves with the smallest and highest severity will always be in the SAD. If n = 1, the leaf with the smallest severity will be returned. The others are sampled sequentially to achieve the n images after severity has been ordered in an ascending order. For example, if there are 30 leaves and n is set to 3, the leaves sampled will be the 1st, 15th, and 30th with the smallest severity values.

The SAD can be only computed if an image pattern name is used in argument pattern of measure_disease(). If the images are saved, the n images will be retrevied from dir_processed directory. Otherwise, the severity will be computed again to generate the images.

Value

A data frame with the severity values for the n sampled leaves. A plot with the standard area diagram can be saved by wrapping sad() with png().

References

Del Ponte EM, Pethybridge SJ, Bock CH, et al (2017) Standard area diagrams for aiding severity estimation: Scientometrics, pathosystems, and methodological trends in the last 25 years. Phytopathology 107:1161–1174. doi:10.1094/PHYTO02170069FI

Examples

```
if (interactive()) {
library(pliman)
sev <-
measure_disease(pattern = "sev_leaf",
                img_healthy = "sev_healthy",
                img_symptoms = "sev_sympt",
                img_background  = "sev_back",
                plot = FALSE,
                save_image = TRUE,
                show_original = FALSE,
                dir_original = image_pliman(),
                dir_processed = tempdir())
sad(sev, n = 2)
}
```

sentinel_to_tif *Convert Sentinel data to GeoTIFF format*

Description

This function converts Sentinel satellite data files to GeoTIFF format.

Usage

```
sentinel_to_tif(layers = NULL, path = ".", destination, spat_res = 10)
```

Arguments

layers	(character) Vector of file paths to Sentinel data files. If NULL, the function searches for files in the specified path with names containing "B".
path	(character) Directory path where Sentinel data files are located. Default is the current directory.
destination	(character) File path for the output GeoTIFF file.
spat_res	(numeric) Spatial resolution of the output GeoTIFF file. Default is 10 meters.

Details

The function converts Sentinel satellite data files to GeoTIFF format using GDAL utilities. It builds a virtual raster file (VRT) from the input files and then translates it to GeoTIFF format. Compression is applied to the output GeoTIFF file using DEFLATE method.

separate_col *Turns a single character column into multiple columns.*

Description

Given either a regular expression or a vector of character positions, `separate_col()` turns a single character column into multiple columns.

Usage

```
separate_col(.data, col, into, sep = "[^[:alnum:]]+")
```

Arguments

.data	A data frame
col	Column name
into	Names of new variables to create as character vector
sep	The separator between columns. By default, a regular expression that matches any sequence of non-alphanumeric values.

Value

A mutated `.data`

Examples

```
library(pliman)
df <- data.frame(x = paste0("TRAT_", 1:5),
                 y = 1:5)
df
separate_col(df, x, into = c("TRAT", "REP"))
```

---

set_pliman_viewer *Set the value of the pliman_viewer option*

Description

Sets the value of the pliman_viewer option used in the package.

Usage

```
set_pliman_viewer(value)
```

Arguments

`value`	The value to be set for the pliman_viewer option.

---

shapefile_build *Build a shapefile from a mosaic raster*

Description

This function takes a mosaic raster to create a shapefile containing polygons for the specified regions. Users can drawn Areas of Interest (AOIs) that can be either a polygon with `n` sides or a rectangular/curvilinear grid, defined by `nrow`, and `ncol` arguments.

Usage

```
shapefile_build(
  mosaic,
  basemap = NULL,
  controlpoints = NULL,
  r = 3,
  g = 2,
  b = 1,
  crop_to_shape_ext = FALSE,
  grid = TRUE,
```

```
  method = c("rectangular", "radial", "landmark"),
  engine = c("cpp", "cpp_old"),
  nrow = 1,
  ncol = 1,
  curved = TRUE,
  density = 30,
  plot_width = NULL,
  plot_height = NULL,
  layout = "lrtb",
  serpentine = TRUE,
  build_shapefile = TRUE,
  check_shapefile = FALSE,
  sf_to_polygon = FALSE,
  buffer = FALSE,
  buffer_edge = 1,
  buffer_col = 0,
  buffer_row = 0,
  as_sf = TRUE,
  verbose = TRUE,
  max_pixels = 1e+06,
  downsample = NULL,
  quantiles = c(0, 1)
)
```

Arguments

`mosaic`	A `SpatRaster` object, typically imported using `mosaic_input()`. If not provided, a latitude/longitude basemap will be generated in the "EPSG:4326" coordinate reference system.
`basemap`	An optional `mapview` object.
`controlpoints`	An object created with `mosaic_view()` using the argument `edit = TRUE`. This must contain the points that defines the region of interest to be analyzed.
`r, g, b`	The layer for the Red, Green and Blue band, respectively. Defaults to `1`, `2`, and `3`.
`crop_to_shape_ext`	Crop the mosaic to the extension of shapefile? Defaults to `TRUE`. This allows for a faster index computation when the region of the built shapefile is much smaller than the entire mosaic extension.
`grid`	Logical, indicating whether to use a grid for segmentation (default: TRUE).
`method`	Character. The generation method: `"rectangular"` (default), `"radial"`, or `"landmark"`.
`engine`	Character string specifying the computation engine to use: `"cpp"` (default) or `"cpp_old"`.
`nrow`	Number of rows for the grid (default: 1).
`ncol`	Number of columns for the grid (default: 1).

`curved` Logical. If `TRUE` (default), the function uses vertex densification logic to draw the longitudinal borders of the plots as smooth arcs, following the rail's curvature (recommended for radial and landmark grids). If `FALSE`, the plots are generated using only the four corners (simple/faceted polygons), which is faster but geometrically less accurate on curves.

`density` Integer. The number of intermediate points used to draw the curved edges of each polygon. This argument is ignored if `curved` is `FALSE` (as the border will be drawn with only 1 step). Typical values are 20 to 50 for visual smoothness.

`plot_width`, `plot_height`
The width and height of the plot shape (in the mosaic unit). It is mutually exclusiv with `buffer_col` and `buffer_row`.

`layout` Character: one of

- `'tblr'` for top/bottom left/right orientation
- `'tbrl'` for top/bottom right/left orientation
- `'btlr'` for bottom/top left/right orientation
- `'btrl'` for bottom/top right/left orientation
- `'lrtb'` for left/right top/bottom orientation
- `'lrbt'` for left/right bottom/top orientation
- `'rltb'` for right/left top/bottom orientation
- `'rlbt'` for right/left bottom/top orientation

`serpentine` Create a serpentine-based layout? Defaults to `FALSE`.

`build_shapefile`
Logical, indicating whether to interactively draw ROIs if the shapefile is `NULL` (default: TRUE).

`check_shapefile`
Logical, indicating whether to validate the shapefile with an interactive map view (default: TRUE). This enables live editing of the drawn shapefile by deleting or changing the drawn grids.

`sf_to_polygon` Convert sf geometry like POINTS and LINES to POLYGONS? Defaults to `FALSE`. Using `TRUE` allows using POINTS to extract values from a raster using `mosaic_extract()`.

`buffer` Numeric. A buffer distance (in map units) to be applied uniformly to the final generated plots. If negative, the plots will be shrunk; if positive, the plots will be expanded. Default is `FALSE` (no buffer).

`buffer_edge` Width of the buffer around the shapefile (default: 5).

`buffer_col`, `buffer_row`
Buffering factor for the columns and rows, respectively, of each individual plot's side. A value between 0 and 0.5 where 0 means no buffering and 0.5 means complete buffering (default: 0). A value of 0.25 will buffer the plot by 25% on each side.

`as_sf` Logical value indicating whether to convert the imported shapefile to an `sf` object (default is `TRUE`).

`verbose` Logical, indicating whether to display verbose output (default: TRUE).

`max_pixels`	Maximum number of pixels to render in the map or plot (default: 500000).
`downsample`	Downsampling factor to reduce the number of pixels (default: NULL). In this case, if the number of pixels in the image (width x height) is greater than `max_pixels` a downsampling factor will be automatically chosen so that the number of plotted pixels approximates the `max_pixels`.
`quantiles`	the upper and lower quantiles used for color stretching.

Details

The function supports three generation methods:

- **Rectangular:** Uses standard interpolation, ideal for regular grids. Supports buffers and fixed plot sizes.
- **Radial:** Uses an orthogonal projection from a centerline, ideal for curvilinear grids (e.g., contour farming).
- **Landmark:** Uses the specific control points defined by the user as anchors for the plot boundaries.

Since multiple blocks can be created, the length of arguments `grid`, `nrow`, `ncol`, `buffer_edge`, `buffer_col`, and `buffer_row` can be either an scalar (the same argument applied to all the drawn blocks), or a vector with the same length as the number of drawn blocks. In the last, shapefiles in each block can be created with different dimensions.

Curvilinear Grid Handling: When defining an AOI using the **Polygon** or **Rectangle** tools, the corners drawn by the user establish the **path and orientation** of the grid. The function automatically checks the angle formed by these drawn control points, creates the grid using user-defined `nrow` and `ncol` and uses a rotation-adjusted `layout` to ensure the plot ID sequence follows the intended serpentine or linear pattern along the long axis of the drawn boundary.

Value

A list with the built shapefile. Each element is an `sf` object with the coordinates of the drawn polygons.

Examples

```
if (interactive()) {
library(pliman)
mosaic <- image_pliman("soy_ortho.tif")

# Draw a regular polygon with 7 rows and 3 columns
shps <-
      shapefile_build(mosaic,
                      nrow = 7,
                      ncol = 3,
                      build_shapefile = FALSE)
mosaic_plot(mosaic)
shapefile_plot(shps[[1]], add = TRUE)

# Here, you'll need to draw the four control points that defines the grid
```

```
# top left, top right, bottom right, bottom left
shps <-
      shapefile_build(mosaic,
                      nrow = 5,
                      ncol = 3)
mosaic_plot(mosaic)
shapefile_plot(shps[[1]], add = TRUE)
}
```

`shapefile_edit` *Edit Features in a Shapefile*

Description

This function allows you to interactively edit features in a shapefile using the mapedit package.

Usage

```
shapefile_edit(
  shapefile,
  mosaic = NULL,
  basemap = NULL,
  r = 3,
  g = 2,
  b = 1,
  max_pixels = 3e+06
)
```

Arguments

`shapefile`	A shapefile (`sf` object) that can be created with `shapefile_input()`.
`mosaic`	Optionally, a mosaic (SpatRaster) to be displayed as a background.
`basemap`	An optional `mapview` object.
`r`	Red band index for RGB display (default is 3).
`g`	Green band index for RGB display (default is 2).
`b`	Blue band index for RGB display (default is 1).
`max_pixels`	Maximum number of pixels for down-sampling the mosaic (default is 3e6).

Value

A modified shapefile with user-edited features.

Examples

```
if (interactive()) {
library(pliman)
shp <- shapefile_input(system.file("ex/lux.shp", package="terra"))
edited <- shapefile_edit(shp)
}
```

`shapefile_interpolate` *Interpolate values at specific points based on coordinates and a target variable*

Description

This function interpolates values at specified points using x, y coordinates and a target variable from a shapefile. It supports "Kriging" and "Tps" interpolation methods.

Usage

```
shapefile_interpolate(
  shapefile,
  z,
  x = "x",
  y = "y",
  interpolation = c("Kriging", "Tps"),
  verbose = FALSE
)
```

Arguments

`shapefile`	An sf object containing the x, y, and target variable (z) columns. It is highly recommended to use `shapefile_measures()` to obtain this data.
`z`	A string specifying the name of the column in the shapefile that contains the target variable to be interpolated.
`x`	A string specifying the name of the column containing x-coordinates. Default is 'x'.
`y`	A string specifying the name of the column containing y-coordinates. Default is 'y'.
`interpolation`	A character vector specifying the interpolation method. Options are "Kriging" or "Tps".
`verbose`	Logical; if TRUE, progress messages will be displayed.

Value

A vector of interpolated values at the specified points.

`shapefile_measures` *Extract geometric measures from a shapefile object*

Description

`shapefile_measures()` calculates key geometric measures such as the number of points, area, perimeter, width, height, and centroid coordinates for a given shapefile (polygon) object.

Usage

```
shapefile_measures(shapefile, n = NULL, verbose = TRUE)
```

Arguments

`shapefile`	An `sf` object representing the shapefile. It should contain polygonal geometries for which the measures will be calculated.
`n`	An integer specifying the number of polygons to process. If `NULL`, all polygons are considered.
`verbose`	Trigger an alert when complex geometries are used?

Details

This function processes a single or multi-polygon `sf` object and computes geometric properties. It calculates distances between points, extracts the centroid coordinates, and computes the area and perimeter of the polygons. The width and height are derived from sequential distances between points.

Value

A modified `sf` object with added columns for:

- `xcoord`: The x-coordinate of the centroid.
- `ycoord`: The y-coordinate of the centroid.
- `area`: The area of the polygon (in square units).
- `perimeter`: The perimeter of the polygon (in linear units).
- `width`: The calculated width based on sequential distances between points. The result will only be accurate if the polygon is rectangular.
- `height`: The calculated height based on sequential distances between points. The result will only be accurate if the polygon is rectangular.

Examples

```
if (interactive()) {
library(pliman)

path_shp <- paste0(image_pliman(), "/soy_shape.rds")
shp <- shapefile_input(path_shp)
shapefile_measures(shp)
}
```

---

shapefile_operations *Spatial Operations on Shapefiles*

Description

These functions perform various spatial operations on two shapefiles, including determining which geometries fall within, outside, touch, cross, overlap, or intersect another geometry. They also include functions for geometric operations such as intersection, difference, and union.

Usage

```
shapefile_within(shp1, shp2)

shapefile_outside(shp1, shp2)

shapefile_overlaps(shp1, shp2)

shapefile_touches(shp1, shp2)

shapefile_crosses(shp1, shp2)

shapefile_intersection(shp1, shp2)

shapefile_difference(shp1, shp2)

shapefile_union(shp1, shp2)
```

Arguments

shp1	An sf object representing the first shapefile.
shp2	An sf object representing the second shapefile.

Details

All functions ensure that the coordinate reference systems (CRS) of both shapefiles are the same before performing operations. If the CRSs are different, `shp2` will be transformed to match the CRS of `shp1`.

- `shapefile_within()`: Filters features in `shp1` that are fully within `shp2`.
- `shapefile_outside()`: Filters features in `shp1` that are outside or do not overlap `shp2`.
- `shapefile_overlaps()`: Filters features in `shp1` that overlap with `shp2`.
- `shapefile_touches()`: Filters features in `shp1` that touch the boundary of `shp2`.
- `shapefile_crosses()`: Filters features in `shp1` that cross through `shp2`.
- `shapefile_intersection()`: Computes the geometric intersection of `shp1` and `shp2`.
- `shapefile_difference()`: Computes the geometric difference of `shp1` minus `shp2`.
- `shapefile_union()`: Computes the geometric union of `shp1` and `shp2`.

Value

A filtered `sf` object or the result of the geometric operation.

Examples

```
if (interactive()) {
library(pliman)

shp1 <- shapefile_input(paste0(image_pliman(), "/shp1.rds"))
shp2 <- shapefile_input(paste0(image_pliman(), "/shp2.rds"))
shapefile_view(shp1) + shapefile_view(shp1)

# Apply operations
shapefile_within(shp1, shp2)
shapefile_outside(shp1, shp2)
shapefile_overlaps(shp1, shp2)
shapefile_touches(shp1, shp2)
shapefile_crosses(shp1, shp2)
shapefile_intersection(shp1, shp2)
shapefile_difference(shp1, shp2)
shapefile_union(shp1, shp2)
}
```

`shapefile_plot` *A wrapper around terra::plot()*

Description

Plot the values of a SpatVector

Usage

```
shapefile_plot(shapefile, ...)
```

Arguments

`shapefile`	An SpatVector of sf object.
`...`	Further arguments passed on to `terra::plot()`.

Value

A `NULL` object

Examples

```
if(interactive()){
library(pliman)
r <- shapefile_input(system.file("ex/lux.shp", package="terra"))
shapefile_plot(r)
}
```

---

shapefile_surface *Generate a spatial surface plot based on interpolated values*

Description

This function creates a surface plot from an interpolated spatial model, with options to customize plot appearance, grid resolution, and color palette.

Usage

```
shapefile_surface(
  model,
  curve = TRUE,
  nx = 300,
  ny = 300,
  xlab = "Longitude (UTM)",
  ylab = "Latitude (UTM)",
  col = custom_palette(c("darkred", "yellow", "forestgreen"), n = 100),
  ...
)
```

Arguments

`model`	An interpolated spatial object (e.g., from `shapefile_interpolate()`) containing the data for plotting.
`curve`	Logical; if TRUE, a contour plot is generated (`type = "C"`), otherwise an image plot (`type = "I"`). Default is TRUE.

nx	Integer; the number of grid cells in the x-direction. Default is 300.
ny	Integer; the number of grid cells in the y-direction. Default is 300.
xlab	Character; label for the x-axis. Default is "Longitude (UTM)".
ylab	Character; label for the y-axis. Default is "Latitude (UTM)".
col	A color palette function for the surface plot. Default is a custom palette from dark red to yellow to forest green.
...	Additional parameters to pass to `fields::surface`.

Value

A surface plot showing spatially interpolated data.

---

shapefile_transform *Apply Affine Transformations to Spatial Grids*

Description

Rotates, shifts, and scales (anisotropically) an `sf` object containing grid polygons. This function is particularly useful for manually aligning experimental grids over non-georeferenced imagery (e.g., drone flights) by adjusting the grid's pose to match the visual features.

Usage

```
shapefile_transform(
  shapefile,
  shift_x = 0,
  shift_y = 0,
  angle = 0,
  scale_x = 1,
  scale_y = 1
)
```

Arguments

shapefile	An `sf` object containing the polygons to be transformed.
shift_x	Numeric. Translation along the X-axis (in map units). Positive values move right. Default is 0.
shift_y	Numeric. Translation along the Y-axis (in map units). Positive values move up. Default is 0.
angle	Numeric. Rotation angle in degrees. Positive values rotate the grid **clockwise** (compass direction). Default is 0.
scale_x	Numeric. Scaling factor for the X-axis (width). Values > 1 expand, values < 1 compress. Default is 1.0.
scale_y	Numeric. Scaling factor for the Y-axis (height). Values > 1 expand, values < 1 compress. Default is 1.0.

Details

The transformations are applied in the following order for each vertex:

1. **Centering:** Vertices are translated so the grid's global centroid becomes (0,0).
2. **Scaling:** Anisotropic scaling is applied (`scale_x`, `scale_y`).
3. **Rotation:** Rotation is applied around the centroid. Positive angles rotate **clockwise**.
4. **Translation:** The grid is moved back to its original position plus the defined offsets (`shift_x`, `shift_y`).

This logic ensures the grid rotates and expands "in place" rather than flying off relative to the coordinate origin.

Value

An `sf` object with the transformed geometry, preserving all original attributes.

Examples

```
if(interactive()){
library(pliman)
shp <- system.file("ex/lux.shp", package="terra")
shp_file <- shapefile_input(shp)
shapefile_plot(shp_file)
rot <- shapefile_transform(shp_file, angle = 5)
shapefile_plot(rot, add = TRUE, border = "red")

}
```

`Summary.image` *Group generic Summary method for* `image` *objects (min, max, range, sum, prod, any, all)*

Description

Group generic Summary method for `image` objects (min, max, range, sum, prod, any, all)

Usage

```
## S3 method for class 'image'
Summary(..., na.rm = FALSE)
```

Arguments

`...`	`image` objects or numbers.
`na.rm`	Logical; should NAs be removed?

summary_index *Summary an object index*

Description

If more than one index is available, the function performs a Principal Component Analysis and produces a plot showing the contribution of the indexes to the PC1 (see pca()). If an index is declared in index and a cut point in cut_point, the number and proportion of objects with mean value of index bellow and above cut_point are returned. Additionaly, the number and proportion of pixels bellow and above the cutpoint is shown for each object (id).

Usage

```
summary_index(
  object,
  index = NULL,
  cut_point = NULL,
  select_higher = FALSE,
  plot = TRUE,
  type = "var",
  ...
)
```

Arguments

object	An object computed with analyze_objects().
index	The index desired, e.g., "B". Note that these value must match the index(es) used in the argument object_index of analyze_objects().
cut_point	The cut point.
select_higher	If FALSE (default) selects the objects with index smaller than the cut_point. Use select_higher = TRUE to select the objects with index higher than cut_point.
plot	Shows the contribution plot when more than one index is available? Defaults to TRUE.
type	The type of plot to produce. Defaults to "var". See more at get_biplot().
...	Further arguments passed on to get_biplot().

Value

A list with the following elements:

- ids The identification of selected objects.
- between_id A data frame with the following columns
  - n The number of objects.

  - `nsel` The number of selected objects.
  - `prop` The proportion of objects selected.
  - `mean_index_sel`, and `mean_index_nsel` The mean value of `index` for the selected and non-selected objects, respectively.
- `within_id` A data frame with the following columns
  - `id` The object identification
  - `n_less` The number of pixels with values lesser than or equal to `cut_point`.
  - `n_greater` The number of pixels with values greater than `cut_point`.
  - `less_ratio` The proportion of pixels with values lesser than or equal to `cut_point`.
  - `greater_ratio` The proportion of pixels with values greater than `cut_point`.
- `pca_res` An object computed with `pca()`

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Examples

```
if (interactive()) {
library(pliman)
soy <- image_pliman("soy_green.jpg")
anal <- analyze_objects(soy, object_index = "G", pixel_level_index = TRUE)
plot_measures(anal, measure = "G")

summary_index(anal, index = "G", cut_point = 0.5)
}
```

`text_embed_dl` *Extract Text Embeddings with OpenAI CLIP ViT-B/32*

Description

Computes L2-normalized 512-dimensional semantic embeddings for character strings using OpenAI's CLIP (ViT-B/32) text model directly via C++ ONNX Runtime.

Usage

```
text_embed_dl(
  text,
  model = "clip-vit-b32",
  threads = 0,
  engine = c("gpu", "cpu"),
  device_id = -1,
  verbose = FALSE,
  dir = pliman_model_dir(),
  ...
)
```

Arguments

text	A character vector of one or more text prompts.
model	Model name or path to a custom ONNX text model file (default "clip-vit-b32").
threads	Number of CPU threads (default 0 for auto-tuning).
engine	Execution engine: "cpu" or "gpu" (DirectML).
device_id	GPU device ID (default -1 for auto).
verbose	Logical. Whether to show progress messages.
dir	Directory where models are stored (default pliman_model_dir()).
...	Additional arguments.

Value

A numeric matrix of dimensions length(text) x 512 containing the L2-normalized text embeddings.

Examples

```
## Not run:
  library(pliman)
  embs <- text_embed_dl(c("healthy soybean leaf", "asian rust lesion", "pest bite"))
  dim(embs) # 3 x 512

## End(Not run)
```

---

to_numeric *Convert* image *between raw (0..255 uint8) and numeric (0..1 double)*

Description

Convert image between raw (0..255 uint8) and numeric (0..1 double)

Usage

```
to_numeric(x)

to_raw(x)
```

Arguments

x	An image object.

Value

An image object converted to numeric storage.

`utils_colorspace` *Convert between colour spaces*

Description

- `rgb_to_srgb()` Transforms colors from RGB space (red/green/blue) to Standard Red Green Blue (sRGB), using a gamma correction of 2.2. The function performs the conversion by applying a gamma correction to the input RGB values (raising them to the power of 2.2) and then transforming them using a specific transformation matrix. The result is clamped to the range 0-1 to ensure valid sRGB values.
- `rgb_to_hsb()` Transforms colors from RGB space (red/green/blue) to HSB space (hue/saturation/brightness). The HSB values are calculated as follows (see https://www.rapidtables.com/conv to-hsv.html for more details).
  - Hue: The hue is determined based on the maximum value among R, G, and B, and it ranges from 0 to 360 degrees.
  - Saturation: Saturation is calculated as the difference between the maximum and minimum channel values, expressed as a percentage.
  - Brightness: Brightness is equal to the maximum channel value, expressed as a percentage.
- `rgb_to_lab()` Transforms colors from RGB space (red/green/blue) to CIE-LAB space, using the sRGB values. See `grDevices::convertColor()` for more details.

Usage

```
rgb_to_hsb(object)

rgb_to_srgb(object)

rgb_to_lab(object)
```

Arguments

`object`	An `image` object, an object computed with `analyze_objects()` with a valid `object_index` argument, or a `data.frame/matrix`. For the last, a three-column data (R, G, and B, respectively) is required.

Value

A data frame with the columns of the converted color space

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

References

See the detailed formulas here

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf.jpg")
rgb_to_lab(img)

# analyze the object and convert the pixels
anal <- analyze_objects(img, object_index = "B", pixel_level_index = TRUE)
rgb_to_lab(anal)
}
```

---

utils_dpi *Utilities for image resolution*

Description

Provides useful conversions between size (cm), number of pixels (px) and dots per inch (dpi).

- `dpi_to_cm()` converts a known dpi value to centimeters.
- `cm_to_dpi()` converts a known centimeter values to dpi.
- `pixels_to_cm()` converts the number of pixels to centimeters, given a known resolution (dpi).
- `cm_to_pixels()` converts a distance (cm) to number of pixels, given a known resolution (dpi).
- `distance()` Computes the distance between two points in an image based on the Pythagorean theorem.
- `dpi()` An interactive function to compute the image resolution given a known distance informed by the user. See more information in the **Details** section.
- `npixels()` returns the number of pixels of an image.

Usage

```
dpi_to_cm(dpi)

cm_to_dpi(cm)

pixels_to_cm(px, dpi)

cm_to_pixels(cm, dpi)

npixels(img)

dpi(img, viewer = get_pliman_viewer(), downsample = NULL, max_pixels = 1e+06)

distance(
```

```
  img,
  viewer = get_pliman_viewer(),
  downsample = NULL,
  max_pixels = 1e+06
)
```

Arguments

dpi	The image resolution in dots per inch.
cm	The size in centimeters.
px	The number of pixels.
img	An image object.
viewer	The viewer option. If not provided, the value is retrieved using get_pliman_viewer(). This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the set_pliman_viewer() function. For example, you can run set_pliman_viewer("mapview") to set the viewer option to "mapview" for all functions.
downsample	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to NULL, which will find the best downsampling factor to approximate the max_pixels value.
max_pixels	integer > 0. Maximum number of cells to use for the plot. If max_pixels < npixels(img), regular sampling is used before plotting.

Details

dpi() only run in an interactive section. To compute the image resolution (dpi) the user must use the left button mouse to create a line of known distance. This can be done, for example, using a template with known distance in the image (e.g., la_leaves.jpg).

Value

- dpi_to_cm(), cm_to_dpi(), pixels_to_cm(), and cm_to_pixels() return a numeric value or a vector of numeric values if the input data is a vector.
- dpi() returns the computed dpi (dots per inch) given the known distance informed in the plot.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
library(pliman)
# Convert  dots per inch to centimeter
dpi_to_cm(c(1, 2, 3))
```

```
# Convert centimeters to dots per inch
cm_to_dpi(c(1, 2, 3))

# Convert centimeters to number of pixels with resolution of 96 dpi.
cm_to_pixels(c(1, 2, 3), 96)

# Convert number of pixels to cm with resolution of 96 dpi.
pixels_to_cm(c(1, 2, 3), 96)

if(isTRUE(interactive())){
#### compute the dpi (dots per inch) resolution ####
# only works in an interactive section
# objects_300dpi.jpg has a known resolution of 300 dpi
img <- image_pliman("objects_300dpi.jpg")
# Higher square: 10 x 10 cm
# 1) Run the function dpi()
# 2) Use the left mouse button to create a line in the higher square
# 3) Declare a known distance (10 cm)
# 4) See the computed dpi
dpi(img)


img2 <- image_pliman("la_leaves.jpg")
# square leaf sample (2 x 2 cm)
dpi(img2)
}
```

`utils_file` *Utilities for file manipulation*

Description

- `file_extension()` Get the extension of a file.
- `file_name()` Get the name of a file.
- `file_dir()` Get or directory of a file
- `manipulate_files()` Manipulate files in a directory with options to rename (insert prefix or suffix) and save the new files to the same or other provided directory.
- `pliman_indexes()` Get the indexes available in pliman.
- `pliman_indexes_eq()` Get the equation of the indexes available in pliman.

Usage

```
file_extension(file)

file_name(file)

file_dir(file)
```

```
manipulate_files(
  pattern,
  dir = NULL,
  prefix = NULL,
  name = NULL,
  suffix = NULL,
  extension = NULL,
  sep = "",
  save_to = NULL,
  overwrite = FALSE,
  remove_original = FALSE,
  verbose = TRUE
)
```

Arguments

`file`	The file name.
`pattern`	A file name pattern.
`dir`	The working directory containing the files to be manipulated. Defaults to the current working directory.
`prefix`, `suffix`	A prefix or suffix to be added in the new file names. Defaults to `NULL` (no prefix or suffix).
`name`	The name of the new files. Defaults to `NULL` (original names). `name` can be either a single value or a character vector of the same length as the number of files manipulated. If one value is informed, a sequential vector of names will be created as "`name`_1", "`name`_2", and so on.
`extension`	The new extension of the file. If not declared (default), the original extensions will be used.
`sep`	An optional separator. Defaults to `""`.
`save_to`	The directory to save the new files. Defaults to the current working directory. If the file name of a file is not changed, nothing will occur. If `save_to` refers to a subfolder in the current working directory, the files will be saved to the given folder. In case of the folder doesn't exist, it will be created. By default, the files will not be overwritten. Set `overwrite = TRUE` to overwrite the files.
`overwrite`	Overwrite the files? Defaults to `FALSE`.
`remove_original`	Remove original files after manipulation? defaults to `FALSE`. If `TRUE` the files in `pattern` will be removed.
`verbose`	If `FALSE`, the code is run silently.

Value

- `file_extension()`, `file_name()`, and `file_dir()` return a character string.
- `manipulate_files()` No return value. If `verbose == TRUE`, a message is printed indicating which operation succeeded (or not) for each of the files attempted.

Examples

```
if (interactive()) {
library(pliman)
# get file name, directory and extension
file <- "E:/my_folder/my_subfolder/image1.png"
file_dir(file)
file_name(file)
file_extension(file)

# manipulate files
dir <- tempdir()
list.files(dir)
file.create(paste0(dir, "/test.txt"))
list.files(dir)
manipulate_files("test",
                 dir = paste0(dir, "\\"),
                prefix = "chang_",
                save_to = paste0(dir, "\\"),
                overwrite = TRUE)
list.files(dir)
}
```

---

utils_image *Import and export images*

Description

Import images from files and URLs and write images to files, possibly with batch processing.

Usage

```
image_import(
  img,
  ...,
  which = 1,
  pattern = NULL,
  path = NULL,
  resize = FALSE,
  plot = FALSE,
  nrow = NULL,
  ncol = NULL
)

image_export(
  img,
  name = NULL,
  prefix = "",
  extension = NULL,
```

```
  subfolder = NULL,
  ...
)

image_input(img, ...)

image_pliman(img, plot = FALSE)
```

Arguments

`img`	• For `image_import()`, a character vector of file names or URLs. • For `image_input()`, a character vector of file names or URLs or an array containing the pixel intensities of an image. • For `image_export()`, an Image object, an array or a list of images. • For `image_pliman()`, a charactere value specifying the image example. See `?pliman_images` for more details.
`...`	• For `image_import()` alternative arguments passed to the corresponding functions from the `jpeg`, `png`, and `tiff` packages. • For `image_input()` further arguments passed on to `as_image()`.
`which`	logical scalar or integer vector to indicate which image are imported if a TIFF files is informed. Defaults to `1` (the first image is returned).
`pattern`	A pattern of file name used to identify images to be imported. For example, if `pattern = "im"` all images in the current working directory that the name matches the pattern (e.g., img1.-, image1.-, im2.-) will be imported as a list. Providing any number as pattern (e.g., `pattern = "1"`) will select images that are named as 1.-, 2.-, and so on. An error will be returned if the pattern matches any file that is not supported (e.g., img1.pdf).
`path`	A character vector of full path names; the default corresponds to the working directory, `getwd()`. It will overwrite (if given) the path informed in `image` argument.
`resize`	Resize the image after importation? Defaults to `FALSE`. Use a numeric value of range 0-100 (proportion of the size of the original image).
`plot`	Plots the image after importing? Defaults to `FALSE`.
`nrow`, `ncol`	Passed on to `image_combine()`. The number of rows and columns to use in the composite image when `plot = TRUE`.
`name`	An string specifying the name of the image. It can be either a character with the image name (e.g., "img1") or name and extension (e.g., "img1.jpg"). If none file extension is provided, the image will be saved as a *.jpg file.
`prefix`	A prefix to include in the image name when exporting a list of images. Defaults to `""`, i.e., no prefix.
`extension`	When `image` is a list, `extension` can be used to define the extension of exported files. This will overwrite the file extensions given in `image`.
`subfolder`	Optional character string indicating a subfolder within the current working directory to save the image(s). If the folder doesn't exist, it will be created.

Value

- `image_import()` returns a new `image` object.
- `image_export()` returns an return vector of file names.
- `image_pliman()` returns a new `image` object with the example image required. If an empty call is used, the path to the `tmp_images` directory installed with the package is returned.

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Examples

```
if (interactive()) {
library(pliman)
folder <- image_pliman()
full_path <- paste0(folder, "/sev_leaf.jpg")
(path <- file_dir(full_path))
(file <- basename(full_path))
image_import(img = full_path)
image_import(img = file, path = path)
}
```

`utils_indexes` *Utilities for image indexes*

Description

- `pliman_indexes()`: Get all the available indexes in pliman.
- `pliman_indexes_rgb()`: Get all the RGB-based available indexes in pliman.
- `pliman_indexes_me()`: Get all the multispectral available indexes in pliman.
- `pliman_indexes_hs()`: Get all the hyperspectral available indexes in pliman.
- `pliman_indexes_eq()`: Get the equations of the available indexes.

Usage

```
pliman_indexes()

pliman_indexes_eq()

pliman_indexes_rgb()

pliman_indexes_me()

pliman_indexes_hs()
```

`utils_measures` *Utilities for object measures*

Description

- `get_measures()` computes object measures (area, perimeter, radius) by using either a known resolution (dpi) or an object with known measurements.
- `plot_measures()` draws the object measures given in an object to the current plot. The object identification (`"id"`) is drawn by default.

Usage

```
get_measures(
  object,
  measure = NULL,
  id = NULL,
  dpi = NULL,
  sep = "\\_|-",
  verbose = TRUE,
  digits = 5
)

plot_measures(
  object,
  measure = "id",
  prefix = "",
  suffix = "",
  id = NULL,
  hjust = NULL,
  vjust = NULL,
  digits = 2,
  size = 0.9,
  col = "white",
  ...
)
```

Arguments

`object`	An object computed with `analyze_objects()`.
`measure`	For `plot_measures()`, a character string; for `get_measures()`, a two-sided formula, e.g., `measure = area ~ 100` indicating the known value of object `id`. The right-hand side is the known value and the left-hand side can be one of the following. • `area` The known area of the object. • `perimeter` The known perimeter of the object. • `radius_mean` The known radius of the object.

- `radius_min` The known minimum radius of the object. If the object is a square, then the `radius_min` of such object will be `L/2` where `L` is the length of the square side.
- `radius_max` The known maximum radius of the object. If the object is a square, then the `radius_max` of such object according to the Pythagorean theorem will be `L x sqrt(2) / 2` where `L` is the length of the square side.

`id` An object in the image to indicate a known value.

`dpi` A known resolution of the image in DPI (dots per inch).

`sep` Regular expression to manage file names. The function combines in the `merge` object the object measures (sum of area and mean of all the other measures) of all images that share the same filename prefix, defined as the part of the filename preceding the first hyphen (-) or underscore (_) (no hyphen or underscore is required). For example, the measures of images named `L1-1.jpeg`, `L1-2.jpeg`, and `L1-3.jpeg` would be combined into a single image information (L1). This feature allows the user to treat multiple images as belonging to a single sample, if desired. Defaults to `sep = "\\_|-"`.

`verbose` If `FALSE`, runs the code silently.

`digits` The number of significant figures. Defaults to `2`.

`prefix`, `suffix` A prefix/suffix to be added to the plotted measures.

`hjust`, `vjust` A numeric value to adjust the labels horizontally and vertically. Positive values will move labels to right (hjust) and top (vjust). Negative values will move the labels to left and bottom, respectively.

`size` The size of the text. Defaults to `0.9`.

`col` The color of the text. Defaults to `"white"`.

`...` Further arguments passed on to `graphics::text()`.

Value

- For `get_measures()`, if `measure` is informed, the pixel values will be corrected by the value of the known object, given in the unit of the right-hand side of `meae`. If `dpi` is informed, then all the measures will be adjusted to the knosurwn `dpi`.
- If applied to an object of class `anal_obj`, returns a data frame with the object `id` and the (corrected) measures.
  - If applied to an object of class `anal_obj_ls`, returns a list of class `measures_ls`, with two objects: (i) `results`, a data frame containing the identification of each image (img) and object within each image (id); and (ii) `summary` a data frame containing the values for each image. If more than one object is detected in a given image, the number of objects (`n`), total area (`area_sum`), mean area (`area_mean`) and the standard deviation of the area (`area_sd`) will be computed. For the other measures (perimeter and radius), the mean values are presented.
- `plot_measures()` returns a `NULL` object, drawing the text according to the x and y coordinates of the objects in `object`.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("objects_300dpi.jpg")
plot(img)
# Image with four objects with a known resolution of 300 dpi
# Higher square: 10 x 10 cm
# Lower square: 5 x 5 cm
# Rectangle: 4 x 2 cm
# Circle: 3 cm in diameter

# Count the objects using the blue band to segment the image
results <-
   analyze_objects(img,
                   index = "B",
                   lower_noise = 0.1)
plot_measures(results, measure = "id")

# Get object measures by declaring the known resolution in dots per inch
(measures <- get_measures(results, dpi = 300))

# Calculated diagonal of the object 1
# 10 * sqrt(2) = 14.14

# Observed diagonal of the object 1
measures[1, "radius_max"] * 2


# Get object measures by declaring the known area of object 1
get_measures(results,
             id = 1,
             area ~ 100)
}
```

---

utils_neural *Deep Learning Background Removal & Salient Object Segmentation in pliman*

Description

The `pliman` neural module provides high-accuracy, zero-Python foreground segmentation and background removal powered by pre-trained Deep Learning models in **ONNX** (Open Neural Network Exchange) format.

1. Supported Neural Architectures

- `"u2netp"` **(Default for CPU)**:
  - **Size:** ~4.4 MB | **Input Resolution:** 320 x 320
  - **Architecture:** Ultra-lightweight U2-Net with nested U-structure residual blocks.
  - **Best For:** Fast batch processing on standard laptops and CPUs without dedicated GPU.
- `"ben2"` **(Boundary-Aware Extraction Network)**:
  - **Size:** ~212.6 MB | **Input Resolution:** 1024 x 1024
  - **Architecture:** BEN2 Base (Prama LLC) with boundary-focused refinement stream.
  - **Best For:** Crisp sub-pixel object boundaries, fine petioles, leaf margins, roots, and hairs.
- `"grounded-sam"` **(Zero-Shot Text-Prompted Detection & Instance Segmentation)**:
  - **Size:** ~314.8 MB (Grounding DINO ~194.4 MB + SAM 2.1 ~120.2 MB) | **Input Resolution:** 800 x 800 (DINO) & 1024 x 1024 (SAM)
  - **Architecture:** Combines Grounding DINO Tiny for open-vocabulary text-prompted bounding box detection with SAM 2.1 Hiera-Tiny for instance mask extraction.
  - **Best For:** Detecting and segmenting specific objects using natural language prompts (e.g. `"cow"`, `"leaf"`, `"lesion"`), with support for bounding-box-only detection (`mask = FALSE, bbox = TRUE`).
- `"sam2.1"` **(Segment Anything 2.1 by Meta AI, alias** `"sam2"`**)**:
  - **Size:** ~120.2 MB (Encoder + Decoder) | **Input Resolution:** 1024 x 1024
  - **Architecture:** Meta AI Hiera-Tiny foundation vision transformer with promptable mask decoder.
  - **Best For:** Zero-shot foundation segmentation with point prompts (`pick_object = TRUE`), box prompts, or center point defaults.
- `"sam3.1"` **(Segment Anything 3.1 by Meta AI, alias** `"sam3"`**)**:
  - **Size:** ~868.1 MB | **Input Resolution:** 1024 x 1024
  - **Architecture:** Meta AI Segment Anything 3.1 with real-time concept-driven segmentation.
  - **Best For:** Advanced foundation segmentation across diverse biological structures.
- `"birefnet-lite"` **(Bilateral Reference Network Lite)**:
  - **Size:** ~213.6 MB | **Input Resolution:** 1024 x 1024
  - **Architecture:** Bilateral Reference Network for High-Resolution Dichotomous Image Segmentation (BiRefNet Lite).
  - **Best For:** Extremely fine margins, hair-thin leaf serrations, complex petioles, translucent halos, and subtle lesions.
- `"isnet-general-use"` **(High Precision Boundary Matting)**:
  - **Size:** ~170.4 MB | **Input Resolution:** 1024 x 1024
  - **Architecture:** IS-Net (Intermediate Supervision Network) specialized in high-resolution boundary matting.
  - **Best For:** Plant leaves, fine serrated margins, lesion borders, chlorotic halos, veins, and complex natural backgrounds.

- `"rmbg-1.4"` **(BRIA AI Standard, alias `"rmbg"`)**:
  - **Size:** ~168.0 MB | **Input Resolution:** 1024 x 1024
  - **Architecture:** BRIA RMBG 1.4 trained on extensive curated foreground datasets.
  - **Best For:** General background removal under challenging lighting, shadows, reflections, and fine object edges.
- `"rmbg-2.0"` **(BRIA AI Next-Gen Model)**:
  - **Size:** ~976.9 MB | **Input Resolution:** 1024 x 1024
  - **Architecture:** BRIA RMBG 2.0 based on a BiRefNet backbone trained on extensive curated commercial datasets.
  - **Best For:** Highest-fidelity boundary matting, transparent surfaces, fine plant hairs, and complex scenes.
- `"withoutbg"` **(State-of-the-Art DepthAnythingV2 + ConvNeXt Matting)**:
  - **Size:** ~433.4 MB | **Input Resolution:** 448 x 448
  - **Architecture:** withoutBG Open Weights: DINOv3 ConvNeXt-fused U-Net guided by DepthAnythingV2 depth features.
  - **Best For:** Complex 3D depth-separated backgrounds, transparent materials, and fine details.
- `"silueta"`:
  - **Size:** ~42.1 MB | **Input Resolution:** 320 x 320
  - **Architecture:** Compact silhouette extraction network.
  - **Best For:** Balanced speed-accuracy tradeoff on edge/CPU hardware.
- `"u2net"`:
  - **Size:** ~167.8 MB | **Input Resolution:** 320 x 320
  - **Architecture:** Full-depth U2-Net salient object detection model.
  - **Best For:** Robust multi-scale salient object detection on standard resolution images.

2. Zero-Python Architecture (Native C++ Runtime)

Unlike standard Python-dependent deep learning packages, `pliman` executes ONNX models natively in C++ via direct Microsoft ONNX Runtime C API dynamic bindings.

- **Zero Python Requirement:** No Python, Conda, Virtualenv, or pip required.
- **Direct C++ Binding:** Connects directly to Microsoft's official ONNX Runtime C++ engine (`onnxruntime.dll` on Windows / `libonnxruntime.so` on Linux / `libonnxruntime.dylib` on macOS).
- **CPU & GPU Support:** Runs optimized CPU multi-threading (AVX2/AVX512) or CUDA GPU acceleration.

3. Model Weight Storage & Cache

Downloaded `.onnx` model weights are stored permanently in the user's R data directory: `tools::R_user_dir("pliman", which = "data")/models` (e.g., `C:/Users/<User>/AppData/Local/R/data/R/pl` on Windows). This follows CRAN policy and ensures models are preserved across R session restarts and package upgrades.

4. Quick Setup Guide

To configure everything in a single call:

```
library(pliman)

# 1. Set up the environment (downloads ONNX Runtime C++ library & default models)
pliman_configure_dl()

# 2. Extract a binary mask
mask <- image_binary_dl(img, model = "u2netp")

# 3. Remove background with transparent alpha channel (RGBA)
img_trans <- image_remove_bg_dl(img, model = "isnet-general-use")

# 4. General segmentation
seg <- image_segment_dl(img, model = "u2netp")
```

`utils_objects` *Utilities for working with image objects*

Description

- `object_id()` get the object identification in an image.
- `object_coord()` get the object coordinates and (optionally) draw a bounding rectangle around multiple objects in an image.
- `object_contour()` returns the coordinates (`x` and `y`) for the contours of each object in the image.
- `object_isolate()` isolates an object from an image.

Usage

```
object_coord(
  img,
  id = NULL,
  index = "NB",
  watershed = TRUE,
  invert = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  fill_hull = FALSE,
  threshold = "Otsu",
  edge = 2,
  extension = NULL,
  tolerance = NULL,
```

```
  object_size = "medium",
  parallel = FALSE,
  workers = NULL,
  plot = TRUE,
  verbose = TRUE
)

object_contour(
  img,
  pattern = NULL,
  dir_original = NULL,
  center = FALSE,
  index = "NB",
  invert = FALSE,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  fill_hull = FALSE,
  smooth = FALSE,
  threshold = "Otsu",
  watershed = TRUE,
  extension = NULL,
  tolerance = NULL,
  object_size = "medium",
  parallel = FALSE,
  workers = NULL,
  plot = TRUE,
  verbose = TRUE
)

object_isolate(
  img,
  id = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  ...
)

object_id(img, parallel = FALSE, workers = NULL, verbose = TRUE, ...)
```

Arguments

`img`	An image of class `image` or a list of `image` objects.
`id`	• For `object_coord()`, a vector (or scalar) of object `id` to compute the bounding rectangle. Object ids can be obtained with `object_id()`. Set `id = "all"` to compute the coordinates for all objects in the image. If `id = NULL` (default) a bounding rectangle is drawn including all the objects.

- For `object_isolate()`, a scalar that identifies the object to be extracted.

`index` The index to produce a binary image used to compute bounding rectangle coordinates. See `image_binary()` for more details.

`watershed` If `TRUE` (default) performs watershed-based object detection. This will detect objects even when they are touching one other. If `FALSE`, all pixels for each connected set of foreground pixels are set to a unique object. This is faster but is not able to segment touching objects.

`invert` Inverts the binary image, if desired. Defaults to `FALSE`.

`opening`, `closing`, `filter`

Morphological operations (brush size)

- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`fill_hull` Fill holes in the objects? Defaults to `FALSE`.

`threshold` By default (`threshold = "Otsu"`), a threshold value based on Otsu's method is used to reduce the grayscale image to a binary image. If a numeric value is informed, this value will be used as a threshold. Inform any non-numeric value different than "Otsu" to iteratively chosen the threshold based on a raster plot showing pixel intensity of the index.

`edge` The number of pixels in the edge of the bounding rectangle. Defaults to 2.

`extension`, `tolerance`, `object_size`

Controls the watershed segmentation of objects in the image. See `analyze_objects()` for more details.

`parallel` Processes the images asynchronously (in parallel) in separate R sessions running in the background on the same machine. It may speed up the processing time when `image` is a list. The number of sections is set up to 50% of available cores.

`workers` A positive numeric scalar or a function specifying the maximum number of parallel processes that can be active at the same time.

`plot` Shows the image with bounding rectangles? Defaults to `TRUE`.

`verbose` If `TRUE` (default) shows progress summary in console.

`pattern` A pattern of file name used to identify images to be imported.

dir_original	The directory containing the original images. Defaults to NULL, which means that the current working directory will be considered.
center	If TRUE returns the object contours centered on the origin.
smooth	whether the object contours should be smoothed with poly_smooth(). Defaults to FALSE. To smooth use a numeric value indicating the number of interactions used to smooth the contours.
...	• For object_isolate(), further arguments passed on to object_coord(). • For object_id(), further arguments passed on to analyze_objects().

Value

- object_id() An image of class "Image" containing the object's identification.
- object_coord() A list with the coordinates for the bounding rectangles. If id = "all" or a numeric vector, a list with a vector of coordinates is returned.
- object_isolate() An image of class "Image" containing the isolated object.

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("la_leaves.jpg")
# Get the object's (leaves) identification
object_id(img)

# Get the coordinates and draw a bounding rectangle around leaves 1 and 3
object_coord(img, id = c(1, 3))

# Isolate leaf 3
isolated <- object_isolate(img, id = 3)
plot(isolated)

}
```

---

utils_pca *Utilities for Principal Component Axis analysis*

Description

- pca() Computes a Principal Component Analysis. It wrappers stats::prcomp(), but returns more results such as data, scores, contributions and quality of measurements for individuals and variables.
- get_biplot(): Produces a biplot for an object computed with pca().
- plot.pca(): Produces several types of plots, depending on the type and which arguments.
  - type = "var" Produces a barplot with the contribution (which = "contrib"), qualitity of adjustment which = "cos2", and a scatter plot with coordinates (which = "coord") for the variables.

- `type = "ind"` Produces a barplot with the contribution (`which = "contrib"`), qualitity of adjustment `which = "cos2"`, and a scatter plot with coordinates (`which = "coord"`) for the individuals.
- `type = "biplot"` Produces a biplot.

Usage

```
pca(x, scale = TRUE)

get_biplot(
  x,
  axes = c(1, 2),
  show = c("both"),
  show_ind_id = TRUE,
  show_unit_circle = TRUE,
  expand = NULL
)

## S3 method for class 'pca'
plot(x, type = "var", which = "contrib", axis = 1, ...)
```

Arguments

`x`	• For `pca()`, a numeric or complex matrix (or data frame) which provides the data for the principal components analysis. • For `plot.pca()` and `get_biplot()`, an object computed with `pca()`.
`scale`	A logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. Defaults to `TRUE`.
`axes`	The principal component axes to plot. Defaults to `axes = c(1, 2)`, i.e., the first and second interaction principal component axis.
`show`	Which to show in the biplot. Defaults to `"both"` (both variables and individuals). One can also use `"var"`, or `"ind"`.
`show_ind_id`	Shows the labels for individuals? Defaults to `TRUE`.
`show_unit_circle`	Shows the unit variance circle? Defaults to `TRUE`.
`expand`	An expansion factor to apply when plotting the second set of points relative to the first. This can be used to tweak the scaling of the two sets to a physically comparable scale. Setting to `TRUE` will automatically compute the expansion factor. Alternatively, a numeric value can be informed.
`type`	One of `"var"` (to plot variables), `"ind"` (to plot individuals), or `"biplot"` to create a biplot.
`which`	Which measure to plot. Either `which = "contribution"` (default), `which = "cos2"` (quality of representation), or `which = "coord"` (coordinates)
`axis`	The axist to plot the contribution/cos2. Defaults to 1.
`...`	Further arguments passed on to `get_biplot()` when `type = "biplot"`. Otherwise, When `which = "coord"`, further arguments passed on to `get_biplot()`.

When `which = "contrib"`, or `which = "cos2"` further arguments passed on to `graphics::barplot()`.

Value

- `pca()` returns a list including:
  - `data`: The raw data used to compute the PCA.
  - `variances`: Variances (eigenvalues), and proportion of explained variance for each component.
  - `center,scale`: the centering and scaling used.
  - `ind,var` A list with the following objects for individuals/variables, respectively.
  - `coord`: coordinates for the individuals/variables (loadings * the component standard deviations)
  - `cos2`: cos2 for the individuals/variables (coord^2)
  - `contrib`: The contribution (in percentage) of a variable to a given principal component: (cos2 * 100) / (total cos2 of the component)
- `plot.pca()` returns a list with the coordinates used.
- `get_biplot()` returns a `NULL` object

Examples

```
library(pliman)
pc <- pca(mtcars[1:10 ,1:6])
plot(pc)
plot(pc, type = "ind")
plot(pc, type = "var", which = "coord")
plot(pc, type = "ind", which = "coord")
plot(pc, type = "biplot")
```

---

utils_pick *Utilities for picking up points in an image*

Description

- `pick_count()` opens an interactive section where the user will be able to click in the image to count objects (points) manually. In each mouse click, a point is drawn and an upward counter is shown in the console. After `n` counts or after the user press Esc, the interactive process is terminated and the number of counts is returned.
- `pick_coord()` Picks coordinates from the image
- `pick_palette()` creates an image palette by picking up color point(s) from the image.
- `pick_rgb()` Picks up the RGB values from selected point(s) in the image.

Usage

```
pick_count(
  img,
  n = Inf,
  col = "red",
  viewer = get_pliman_viewer(),
  external_device = FALSE,
  size = 0.8,
  plot = TRUE,
  verbose = TRUE
)

pick_coords(
  img,
  n = Inf,
  col = "red",
  viewer = get_pliman_viewer(),
  external_device = FALSE,
  size = 0.8,
  verbose = TRUE
)

pick_rgb(
  img,
  n = Inf,
  col = "red",
  viewer = get_pliman_viewer(),
  external_device = FALSE,
  size = 0.8,
  plot = TRUE,
  verbose = TRUE
)

pick_palette(
  img,
  n = Inf,
  r = 2,
  shape = "box",
  viewer = get_pliman_viewer(),
  external_device = FALSE,
  show = "rgb",
  title = "Pick colors in the image",
  index = "B",
  random = TRUE,
  width = 100,
  height = 100,
  col = "red",
  size = 0.8,
```

```
  plot = TRUE,
  palette = TRUE,
  verbose = TRUE
)

pick_rgb_area(
  img,
  n = Inf,
  r = 15,
  shape = "box",
  viewer = get_pliman_viewer(),
  external_device = FALSE,
  title = "Pick samples in the image",
  col = "red",
  size = 0.8,
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

`img`	An `image` object.
`n`	The number of points of the `pick_*` function. Defaults to `Inf`. This means that picking will run until the user press Esc.
`col`, `size`	The color and size for the marker point.
`viewer`	The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are "base" and "mapview". If set to "base", the base R graphics system is used for interactive plotting. If set to "mapview", the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to "mapview" for all functions.
`external_device`	Logical. If `TRUE` (default), opens an external graphics window when running inside RStudio to ensure accurate point selection using `locator()`. Ignored when not in RStudio or when using `viewer = "mapview"`.
`plot`	Call a new `plot(img)` before processing? Defaults to `TRUE`.
`verbose`	If `TRUE` (default) shows a counter in the console.
`r`	The radius of neighborhood pixels. Defaults to `1`.
`shape`	A character vector indicating the shape of the brush around the selected pixel. It can be `"box"`, `"disc"`, `"diamond"`, `"Gaussian"` or `"line"`. Defaults to `"box"`. In this case, if `'r = 1'`, all the 8 surrounding pixels are sampled. Setting to `"disc"` and increasing the radius (`r`) will select surrounding pixels towards the format of a sphere around the selected pixel.

show	How to plot in mapview viewer, either 'rgb or 'index'.
title	The title of the map view when vieweris used.
index	The index to use for the index view. Defaults to 'B'.
random	Randomize the selected pixels? Defaults to TRUE.
width, height	The width and height of the generated palette. Defaults to 100 for both, i.e., a square image of 100 x 100.
palette	Plot the generated palette? Defaults to TRUE.

Value

- pick_count() returns data.frame with the x and y coordinates of the selected point(x).
- pick_rgb() returns a data.frame with the R, G, and B values of the selected point(s).
- pick_palette() returns an object of class image.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("soybean_touch.jpg")

# start a counting process
pick_count(img)

# get rgb from point(s)
pick_rgb(img)

# create a palette from point(s)
pick_palette(img)
}
```

---

utils_polygon *Utilities for Polygons*

Description

Several useful functions for analyzing polygons. All of them are based on a set of coordinate points that describe the edge of the object(s). If a list of polygons is provided, it loops through the list and computes what is needed for each element of the list.

- Polygon measures
  - conv_hull() Computes the convex hull of a set of points.

  - `conv_hull_unified()` Computes the convex hull of a set of points. Compared to `conv_hull()`, `conv_hull_unified()` binds (unifies) the coordinates when x is a list of coordinates.
  - `poly_area()` Computes the area of a polygon given by the vertices in the vectors x and y using the Shoelace formula, as follows (Lee and Lim, 2017):

$$A = \frac{1}{2}\left|\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)\right|$$

    where `x` and `y` are the coordinates that form the corners of a polygon, and `n` is the number of coordinates.
  - `poly_angles()` Calculates the internal angles of the polygon using the law of cosines.
  - `poly_lw()` Returns the length and width of a polygon based on its alignment to the y-axis (with poly_align()). The length is defined as the range along the x-axis, and the width is defined as the range on the y-axis.

- `poly_mass()` Computes the center of mass (centroid) of a polygon given by the vertices in the vectors x and y using the following formulas:

$$C_x = \frac{1}{6A}\sum_{i=1}^{n}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$C_y = \frac{1}{6A}\sum_{i=1}^{n}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

where `C_x` and `C_y` are the coordinates of the center of mass, `A` is the area of the polygon computed by the Shoelace formula, `x` and `y` are the coordinates that form the corners of the polygon, and `n` is the number of coordinates.

- `poly_solidity()` Computes the solidity of a shape as the ratio of the shape area and the convex hull area.

- Perimeter measures
  - `poly_slide()` Slides the coordinates of a polygon given by the vertices in the vectors x and y so that the id-th point becomes the first one.
  - `poly_distpts()` Computes the Euclidean distance between every point of a polygon given by the vertices in the vectors x and y.
  - `poly_centdist()` Computes the Euclidean distance between every point on the perimeter and the centroid of the object.
  - `poly_centdist_mass()` Computes the Euclidean distance between every point on the perimeter and the center of mass of the object.
  - `poly_perimeter()` Computes the perimeter of a polygon given by the vertices in the vectors x and y.
  - `poly_caliper()` Computes the caliper (also called the Feret's diameter) of a polygon given by the vertices in the vectors x and y.
- Circularity measures (Montero et al. 2009).

  - `poly_circularity()` computes the circularity (C), also called shape compactness or roundness measure, of an object. It is given by `C = P^2 / A`, where P is the perimeter and A is the area of the object.
  - `poly_circularity_norm()` computes the normalized circularity (Cn), which is unity for a circle. This measure is invariant under translation, rotation, scaling transformations, and is dimensionless. It is given by: `Cn = P^2 / 4*pi*A`.
  - `poly_circularity_haralick()` computes Haralick's circularity (CH). The method is based on computing all Euclidean distances from the object centroid to each boundary pixel. With this set of distances, the mean (`m`) and the standard deviation (`sd`) are computed. These statistical parameters are used to calculate the circularity, CH, of a shape as `CH = m/sd`.
  - `poly_convexity()` computes the convexity of a shape using the ratio between the perimeter of the convex hull and the perimeter of the polygon.
  - `poly_eccentricity()` computes the eccentricity of a shape using the ratio of the eigenvalues (inertia axes of coordinates).
  - `poly_elongation()` computes the elongation of a shape as `1 - width / length`.
- Utilities for polygons
  - `poly_check()` Checks a set of coordinate points and returns a matrix with x and y columns.
  - `poly_is_closed()` Returns a logical value indicating if a polygon is closed.
  - `poly_close()` and `poly_unclose()` close and unclose a polygon, respectively.
  - `poly_rotate()` Rotates the polygon coordinates by an angle (0-360 degrees) in the counterclockwise direction.
  - `poly_flip_x()`, `poly_flip_y()` flip shapes along the x-axis and y-axis, respectively.
  - `poly_align()` Aligns the coordinates along their longer axis using the var-cov matrix and eigen values.
  - `poly_center()` Centers the coordinates on the origin.
  - `poly_sample()` Samples n coordinates from existing points. Defaults to 50.
  - `poly_sample_prop()` Samples a proportion of coordinates from existing points. Defaults to 0.1.
  - `poly_spline()` Interpolates the polygon contour.
  - `poly_smooth()` Smooths the polygon contour using a simple moving average.
  - `poly_jitter()` Adds a small amount of noise to a set of point coordinates. See `base::jitter()` for more details.
- `poly_measures()` Is a wrapper around the `poly_*()` functions.

Usage

```
poly_check(x)

poly_is_closed(x)

poly_close(x)
```

```
poly_unclose(x)

poly_angles(x)

poly_limits(x)

conv_hull(x)

conv_hull_unified(x)

poly_area(x)

poly_slide(x, fp = 1)

poly_distpts(x)

poly_centdist(x)

poly_centdist_mass(x)

poly_perimeter(x)

poly_rotate(x, angle, plot = TRUE)

poly_align(x, plot = TRUE)

poly_center(x, plot = TRUE)

poly_lw(x)

poly_eccentricity(x)

poly_convexity(x)

poly_caliper(x)

poly_elongation(x)

poly_solidity(x)

poly_flip_y(x)

poly_flip_x(x)

poly_sample(x, n = 50)

poly_sample_prop(x, prop = 0.1)
```

```
poly_jitter(x, noise_x = 1, noise_y = 1, plot = TRUE)

poly_circularity(x)

poly_circularity_norm(x)

poly_circularity_haralick(x)

poly_mass(x)

poly_spline(x, vertices = 100, k = 2)

poly_smooth(x, niter = 10, n = NULL, prop = NULL, plot = TRUE)

poly_measures(x)
```

Arguments

x	A 2-column matrix with the x and y coordinates. If x is a list of vector coordinates, the function will be applied to each element using base::lapply() or base::sapply().
fp	The ID of the point that will become the new first point. Defaults to 1.
angle	The angle (0-360) to rotate the object.
plot	Should the object be plotted? Defaults to TRUE.
n, prop	The number and proportion of coordinates to sample from the perimeter coordinates. In poly_smooth(), these arguments can be used to sample points from the object's perimeter before smoothing.
noise_x, noise_y	A numeric factor to define the noise added to the x and y axes, respectively. See base::jitter() for more details.
vertices	The number of spline vertices to create.
k	The number of points to wrap around the ends to obtain a smooth periodic spline.
niter	An integer indicating the number of smoothing iterations.

Value

- conv_hull() and poly_spline() returns a matrix with x and y coordinates for the convex hull/smooth line in clockwise order. If x is a list, a list of points is returned.
- poly_area() returns a double, or a numeric vector if x is a list of vector points.
- poly_mass() returns a data.frame containing the coordinates for the center of mass, as well as for the maximum and minimum distance from contour to the center of mass.
- poly_slides(), poly_distpts(), poly_spline(), poly_close(), poly_unclose(), poly_rotate(), poly_jitter(), poly_sample(), poly_sample_prop(), and poly_measures returns a data.frame.

- `poly_perimeter()`, `poly_lw()`, `poly_eccentricity()`, `poly_convexity()`, `poly_caliper()`, `poly_elongation()`, `poly_circularity_norm()`, `poly_circularity_haralick()` returns a `double`.

References

Lee, Y., & Lim, W. (2017). Shoelace Formula: Connecting the Area of a Polygon and the Vector Cross Product. The Mathematics Teacher, 110(8), 631–636. doi:10.5951/mathteacher.110.8.0631

Montero, R. S., Bribiesca, E., Santiago, R., & Bribiesca, E. (2009). State of the Art of Compactness and Circularity Measures. International Mathematical Forum, 4(27), 1305–1335.

Chen, C.H., and P.S.P. Wang. 2005. Handbook of Pattern Recognition and Computer Vision. 3rd ed. World Scientific.

Examples

```
if (interactive()) {
library(pliman)
# A 2 x 2 square
df <- draw_square(side = 2)

# square area
poly_area(df)

# polygon perimeter
poly_perimeter(df)

# center of mass of the square
cm <- poly_mass(df)
plot_mass(cm)

# The convex hull will be the vertices of the square
(conv_square <- conv_hull(df) |> poly_close())
plot_contour(conv_square,
             col = "blue",
             lwd = 6)
poly_area(conv_square)


################### Example with a polygon ##################
x <- c(0, 1,   2, 3,  5, 2, -1, 0, 0)
y <- c(5, 6.5, 7, 3,  1, 1,  0, 2, 5)
df_poly <- cbind(x, y)

# area of the polygon
plot_polygon(df_poly, fill = "red")
poly_area(df_poly)

# perimeter of the polygon
poly_perimeter(df_poly)
```

```
# center of mass of polygon
cm <- poly_mass(df_poly)
plot_mass(cm,  col = "blue")

# vertices of the convex hull
(conv_poly <- conv_hull(df_poly))

# area of the convex hull
poly_area(conv_poly)

plot_polygon(conv_poly,
             fill = "red",
             alpha = 0.2,
             add = TRUE)


############ example of circularity measures #################
tri <- draw_circle(n = 200, plot = FALSE)
plot_polygon(tri, aspect_ratio = 1)
poly_circularity_norm(tri)

set.seed(1)
tri2 <-
  draw_circle(n = 200, plot = FALSE) |>
  poly_jitter(noise_x = 100, noise_y = 100, plot = FALSE)

plot_polygon(tri2, aspect_ratio = 1)
poly_circularity_norm(tri2)
}
```

`utils_polygon_plot` *Utilities for plotting polygons*

Description

- `plot_contour()` Plot contour lines.
- `plot_polygon()` Plots a polygon describing the objects.
- `plot_mass()` Plots the center of mass along with maximum and minimum radius.
- `plot_ellipse()` Plots an ellipse that fits the major and minor axis for each object.

Usage

```
plot_contour(x, id = NULL, col = "black", lwd = 1, ...)

plot_polygon(
  x,
  fill = "gray",
  random_fill = TRUE,
```

```
  points = FALSE,
  merge = TRUE,
  border = "black",
  alpha = 1,
  add = FALSE,
  nrow = NULL,
  ncol = NULL,
  aspect_ratio = 1,
  show_id = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

plot_mass(x, id = NULL, col = "black", cex = 1, lwd = 1)

plot_ellipse(object, id = NULL, col = "black", lwd = 1)
```

Arguments

`x`	A 2-column matrix with the `x` and `y` coordinates.
`id`	The object identification (numeric) to plot the contour/ellipse. By default (`id = NULL`), the contour is plotted to all objects.
`col`, `lwd`, `cex`	The color, width of the lines, and size of point, respectively.
`...`	• For `plot_contour()` and `plot_ellipse()` further arguments passed on to `graphics::lines()`. • For `plot_mass()`, further arguments passed on to `graphics::points()`. • For `plot_polygon()`, further arguments passed on to `graphics::polygon()`.
`fill`, `border`, `alpha`	The color to fill the polygon, the color of the polygon's border, and the alpha transparency (1 opaque, 0 transparent).
`random_fill`	Fill multiple objects with random colors? Defaults to `TRUE`.
`points`	Plot the points? Defaults to `FALSE`.
`merge`	Merge multiple objects into a single plot? Defaults to `TRUE`. If `FALSE`, a single call `plot()` will be used for each objects. Use `nrow` and `ncol` to control the number of rows and columns of the window.
`add`	Add the current plot to a previous one? Defaults to `FALSE`.
`nrow`, `ncol`	The number of rows and columns to use in the composite image. Defaults to `NULL`, i.e., a square grid is produced.
`aspect_ratio`	The x/y aspect ratio. Defaults to `1`. This will set up the window so that one data unit in the y direction is equal to one data unit in the x direction. Set `aspect_ratio = NULL` to fit the object to the window size.
`show_id`	Shows the object id? Defaults to `TRUE`.
`xlim`, `ylim`	A numeric vector of length 2 (min; max) indicating the range of `x` and y-axes.
`object`	An object computed with `analyze_objects()`.

Value

a NULL object.

Examples

```
plot_polygon(contours)
plot_contour(contours[[1]], id = 6, col = "red", lwd = 3)
```

---

utils_rows_cols *Utilities for handling with rows and columns*

Description

- columns_to_rownames(): Move a column of .data to its row names.
- rownames_to_column(): Move the row names of .data to a new column.
- remove_rownames(): Remove the row names of .data.
- round_cols() Rounds the values of all numeric variables to the specified number of decimal places (default 2).

Usage

```
column_to_rownames(.data, var = "rowname")

rownames_to_column(.data, var = "rowname")

remove_rownames(.data)

round_cols(.data, digits = 2)
```

Arguments

.data	A data frame
var	Name of column to use for rownames.
digits	The number of significant figures. Defaults to 2.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
if (interactive()) {
library(pliman)
iris2 <- iris |> rownames_to_column()
head(iris2)
iris2$rowname <- paste0("r", iris2$rowname)
iris2 |> column_to_rownames("rowname") |> head()
}
```

`utils_shapefile` *Import/export shapefiles.*

Description

- `shapefile_input()` creates or imports a shapefile and optionally converts it to an `sf` object. It can also cast `POLYGON` or `MULTIPOLYGON` geometries to `MULTILINESTRING` if required.
- `shapefile_export()` exports an object (`sf` or `SpatVector`) to a file.
- `shapefile_view()` is a simple wrapper around `mapview()` to plot a shapefile.

Usage

```
shapefile_input(
  shapefile,
  info = TRUE,
  as_sf = TRUE,
  multilinestring = FALSE,
  make_valid = FALSE,
  ...
)

shapefile_export(shapefile, filename, ...)

shapefile_view(
  shapefile,
  attribute = NULL,
  type = c("shape", "centroid"),
  color_regions = custom_palette(c("red", "yellow", "forestgreen")),
  ...
)
```

Arguments

`shapefile`	For `shapefile_input()`, character (filename), or an object that can be coerced to a SpatVector, such as an `sf` (simple features) object. See `terra::vect()` for more details. For `shapefile_export()`, a `SpatVector` or an `sf` object to be exported as a shapefile.
`info`	Logical value indicating whether to print information about the imported shapefile (default is `TRUE`).
`as_sf`	Logical value indicating whether to convert the imported shapefile to an `sf` object (default is `TRUE`).
`multilinestring`	Logical value indicating whether to cast polygon geometries to `MULTILINESTRING` geometries (default is `FALSE`).

make_valid	If TRUE makes an invalid geometry valid.
...	Additional arguments to be passed to terra::vect() (shapefile_input()), terra::writeVector() (shapefile_export()) or mapview::mapview() (shapefile_view()).
filename	The path to the output shapefile.
attribute	The attribute to be shown in the color key. It must be a variable present in shapefile.
type	A character string specifying whether to visualize the shapefile as "shape" or as "centroid". Partial matching is allowed. If set to "centroid", the function will convert the shapefile's geometry to centroids before displaying. Defaults to "shape".
color_regions	The color palette to represent attribute.

Value

- shapefile_input() returns an object of class sf (default) representing the imported shapefile.
- shapefile_export() returns a NULL object.
- shapefile_view() returns an object of class mapview.

Examples

```
if(interactive()){
library(pliman)
shp <- system.file("ex/lux.shp", package="terra")
shp_file <- shapefile_input(shp, as_sf = FALSE)
shapefile_view(shp_file)
}
```

---

utils_shapes *Utilities for drawing coordinates of known shapes*

Description

The functions computes the coordinates of common shapes such as squares triangles, rectangles and circles.

- draw_circle() Draws a perfect circle with a desired radius.
- draw_square() Draws a square with a desired side.
- draw_rectangle() Draws a rectangle given two desired sides.
- draw_trian_equi() Draws an equilateral triangle with a desired side.
- draw_trian_rect() Draws a triangle rectangle given two cathetus.
- draw_n_tagon() Draws polygons with n sides

Usage

```
draw_circle(radius = 1, n = 1000, plot = TRUE)

draw_square(side = 2, plot = TRUE)

draw_rectangle(side1 = 2, side2 = 3, plot = TRUE)

draw_trian_equi(side = 2, plot = TRUE)

draw_trian_rect(cat1 = 1, cat2 = 2, plot = TRUE)

draw_n_tagon(n, plot = TRUE)
```

Arguments

`radius`	The radius of the circle. Defaults to `1`.
`n`	The number of sides in the `n`-tagon.
`plot`	Plots the result? Defaults to `TRUE`.
`side`	The side of the square/equilateral triangle. Defaults to `2`.
`side1`, `side2`	The first and second sides of the rectangle. Defaults to `2` and `3`, respectively.
`cat1`, `cat2`	The first and second cathetus of the right triangle. Defaults to `1`, and `2`, respectively.

Value

A data frame with the `x` and `y` coordinates

Examples

```
########## An example of a circle ##########
library(pliman)
radius <- 3
circ <- draw_circle(radius = radius)

# area
pi * radius ^ 2
poly_area(circ)

# perimeter
2 * pi * radius
poly_perimeter(circ)

############ An example of a square ############
side <- 2
(square <- draw_square(side = side))

# area
side ^ 2
```

```
poly_area(square)

# perimeter
side * 4
poly_perimeter(square)

############ An example of a rectangle ############
side1 <- 2
side2 <- 3
(rect <- draw_rectangle())

# area
poly_area(rect)

# perimeter
poly_perimeter(rect)
############  An example of an equilateral triangle #########
side <- 1 # defaults
(trig <- draw_trian_equi(side = side))

### area (b*h / 2)
# height of the triangle
(h <- (side * sqrt(3)) / 2)
side * h / 2

poly_area(trig)

### perimeter (side * 3)
poly_perimeter(trig)

############ An example of a rectangle triangle ##########
cat1 <- 2
cat2 <- 3
(df <- draw_trian_rect(cat1, cat2))
# area
(cat1 * cat2) / 2
poly_area(df)

# perimeter
cat1 + cat2 + sqrt(cat1^2 + cat2^2)
poly_perimeter(df)
############ An creating shapes with n sides ############
side <- 2
(square <- draw_square(side = side))

# area
side ^ 2
poly_area(square)

# perimeter
side * 4
poly_perimeter(square)
```

`utils_stats` *These functions applies common statistics to a list of objects, returning a numeric vector.*

Description

These functions applies common statistics to a list of objects, returning a numeric vector.

Usage

```
mean_list(x, ...)

sd_list(x, ...)

max_list(x, ...)

min_list(x, ...)
```

Arguments

`x`	A data.frame or matrix with numeric values.
`...`	Further arguments passed on to the R base function (e.g, mean(), sd(), etc.)

Value

A numeric vector.

Examples

```
mean_list(list(a = 1:10, b = 2:20))
```

`utils_transform` *Spatial transformations*

Description

Performs image rotation and reflection

- `image autocrop()` Crops automatically an image to the area of objects.
- `image_crop()` Crops an image to the desired area.
- `image_trim()` Remove pixels from the edges of an image (20 by default).
- `image_dimension()` Gives the dimension (width and height) of an image.
- `image_rotate()` Rotates the image clockwise by the given angle.
- `image_horizontal()` Converts (if needed) an image to a horizontal image.

- `image_vertical()` Converts (if needed) an image to a vertical image.
- `image_hreflect()` Performs horizontal reflection of the `image`.
- `image_vreflect()` Performs vertical reflection of the `image`.
- `image_resize()` Resize the `image`. See more at `image_resize()`.
- `image_contrast()` Improve contrast locally by performing adaptive histogram equalization.
- `image_dilate()` Performs image dilatation.
- `image_erode()` Performs image erosion.
- `image_opening()` Performs an erosion followed by a dilation.
- `image_closing()` Performs a dilation followed by an erosion.
- `image_filter()` Performs median filtering in constant time.
- `image_blur()` Performs blurring filter of images.
- `image_skeleton()` Performs image skeletonization.

Performs morphological erosion of a binary image.

Performs morphological dilation of a binary image.

Fills internal holes in a binary image.

Performs morphological opening of a binary image (erosion followed by dilation). When `return_exact = TRUE`, morphological reconstruction via watershed-partitioned seed matching is performed. This isolates noise touching large objects into separate watershed labels, identifies surviving labels using the un-dilated eroded marker, and restores 100% of the exact original boundaries and shapes of surviving objects without rounding corners or eroding tips.

Removes small noise objects from a binary image without altering or eroding the boundaries of larger objects. Filtering can be based on an absolute pixel count (`min_size`) or a relative fraction of the mean object area (`rel_size`, e.g. `0.1` for objects smaller than 10% of average object area).

Performs morphological closing of a binary image. Closing is defined as a dilation followed by an erosion. It is useful for filling small holes or connecting nearby objects.

Usage

```
image_autocrop(
  img,
  index = "NB",
  edge = 5,
  opening = FALSE,
  closing = FALSE,
  filter = FALSE,
  invert = FALSE,
  threshold = "Otsu",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
```

```
  plot = FALSE
)

image_crop(
  img,
  width = NULL,
  height = NULL,
  viewer = get_pliman_viewer(),
  downsample = NULL,
  max_pixels = 1e+06,
  show = "rgb",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_dimension(img, parallel = FALSE, workers = NULL, verbose = TRUE)

image_rotate(
  img,
  angle,
  bg_col = "white",
  bg_color = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = TRUE
)

image_horizontal(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_vertical(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_hreflect(
  img,
```

```
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_vreflect(
  img,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_trim(
  img,
  edge = NULL,
  top = NULL,
  bottom = NULL,
  left = NULL,
  right = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_skeleton(
  img,
  kern = NULL,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE,
  ...
)

image_thinning(
  img,
  niter = 3,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE,
  ...
)

image_filter(
```

```
  img,
  size = 2,
  cache = 512,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_blur(
  img,
  sigma = 3,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_contrast(
  img,
  nx = 8,
  ny = 8,
  clip_limit = 3,
  bins = 256,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_erode(
  img,
  size = NULL,
  shape = "disc",
  external = FALSE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_dilate(
  img,
  size = NULL,
  shape = "disc",
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
```

```
  plot = FALSE
)

image_fill_hull(
  img,
  max_size = NULL,
  min_neck_dist = 0,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_opening(
  img,
  size = NULL,
  shape = "disc",
  external = FALSE,
  return_exact = FALSE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)

image_remove_small(
  img,
  min_size = NULL,
  rel_size = NULL,
  watershed = TRUE,
  plot = FALSE
)

image_closing(
  img,
  size = NULL,
  shape = "disc",
  external = FALSE,
  parallel = FALSE,
  workers = NULL,
  verbose = TRUE,
  plot = FALSE
)
```

Arguments

`img`	An `image` object or a list of `image` objects. Must be binary (`storage.mode = "logical"`).
`index`	The index to segment the image. See `image_index()` for more details.

Defaults to `"NB"` (normalized blue).

`edge`
- for `image_autocrop()` the number of pixels in the edge of the cropped image. If `edge = 0` the image will be cropped to create a bounding rectangle (x and y coordinates) around the image objects.
- for `image_trim()`, the number of pixels removed from the edges. By default, 20 pixels are removed from all the edges.

`opening`, `closing`, `filter`

Morphological operations (brush size)

- `opening` performs an erosion followed by a dilation. This helps to remove small objects while preserving the shape and size of larger objects.
- `closing` performs a dilatation followed by an erosion. This helps to fill small holes while preserving the shape and size of larger objects.
- `filter` performs median filtering in the binary image. Provide a positive integer > 1 to indicate the size of the median filtering. Higher values are more efficient to remove noise in the background but can dramatically impact the perimeter of objects, mainly for irregular perimeters such as leaves with serrated edges.

Hierarchically, the operations are performed as opening > closing > filter. The value declared in each argument will define the brush size.

`invert` Inverts the binary image if desired.

`threshold` The threshold method (`"Otsu"`, `"Adaptive"`, or numeric).

`parallel` Logical. If `TRUE`, processes the list of images in parallel.

`workers` Number of parallel workers. If `NULL`, defaults to 40% of available cores.

`verbose` Logical. If `TRUE`, prints progress messages.

`plot` Logical. If `TRUE`, plots the resulting image(s).

`width`, `height`
- For `image_resize()` the Width and height of the resized image. These arguments can be missing. In this case, the image is resized according to the relative size informed in `rel_size`.
- For `image_crop()` a numeric vector indicating the pixel range (x and y, respectively) that will be maintained in the cropped image, e.g., width = 100:200

`viewer` The viewer option. If not provided, the value is retrieved using `get_pliman_viewer()`. This option controls the type of viewer to use for interactive plotting. The available options are ”base” and ”mapview”. If set to ”base”, the base R graphics system is used for interactive plotting. If set to ”mapview”, the mapview package is used. To set this argument globally for all functions in the package, you can use the `set_pliman_viewer()` function. For example, you can run `set_pliman_viewer("mapview")` to set the viewer option to ”mapview” for all functions.

`downsample` integer; for each dimension the number of pixels/lines/bands etc that will be skipped; Defaults to `NULL`, which will find the best downsampling factor to approximate the `max_pixels` value.

`max_pixels`	integer > 0. Maximum number of cells to use for the plot. If `max_pixels < npixels(img)`, regular sampling is used before plotting.
`show`	How to plot in mapview viewer, either `"rgb"` or `"index"`.
`angle`	The rotation angle in degrees.
`bg_col`, `bg_color`	Color used to fill the background pixels, defaults to `"white"`.
`top`, `bottom`, `left`, `right`	The number of pixels removed from `top`, `bottom`, `left`, and `right` when using `image_trim()`.
`kern`	An `image` object or an array, containing the structuring element. Defaults to a brushe generated with `make_brush()`.
`...`	Additional arguments passed on to `image_binary()`.
`niter`	The number of iterations to perform in the thinning procedure. Defaults to 3. Set to `NULL` to iterate until the binary image is no longer changing.
`size`	The size of the structuring element (radius in pixels). Default is `NULL`, which automatically determines the size based on image dimensions.
`cache`	The the L2 cache size of the system CPU in kB (integer). Defaults to `512`.
`sigma`	A numeric denoting the standard deviation of the Gaussian filter used for blurring. Defaults to `3`.
`nx`	Number of contextual regions in the x-direction (tile grid) for CLAHE contrast enhancement. Defaults to 8.
`ny`	Number of contextual regions in the y-direction (tile grid) for CLAHE contrast enhancement. Defaults to 8.
`clip_limit`	Normalized contrast limit value for CLAHE contrast enhancement. Defaults to 3.
`bins`	Number of histogram bins used for CLAHE contrast enhancement. Defaults to 256.
`shape`	A character string specifying the shape of the structuring element. Either `"disc"` (default) or `"square"`.
`external`	Logical. If `TRUE`, erosion does not consume pixels inside internal holes (passed to `image_erode()`).
`max_size`	Maximum size of holes to fill (in pixels). Default is `NULL` (fills all holes).
`min_neck_dist`	Minimum neck distance threshold for hole filling. Default is `0.0`.
`return_exact`	Logical. If `TRUE`, returns the exact original pixels and boundaries of objects in `img` that survived the opening. Uses internal watershed partitioning to isolate and eliminate noise touching large objects, preserving 100% of sharp corners and geometric features of surviving objects.
`min_size`	Minimum area (in pixels) for an object to be retained.
`rel_size`	Relative threshold as a fraction of the mean object area (e.g. `0.1` for 10% of mean area).
`watershed`	Logical. Default `TRUE`. Applies watershed segmentation before measuring component sizes, ensuring small noise projections touching large objects are split and removed.

Value

- `image_skeleton()` returns a binary `image` object.
- All other functions returns a modified version of `image` depending on the `image_*()` function used.
- If `image` is a list, a list of the same length will be returned.

A binary `image` object or a list of `image` objects.

A binary `image` object or a list of `image` objects.

A binary `image` object or a list of `image` objects.

A binary `image` object or a list of `image` objects.

A binary `image` object or a list of binary `image` objects.

A binary `image` object or a list of `image` objects.

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Examples

```
if (interactive()) {
library(pliman)
img <- image_pliman("sev_leaf.jpg")
plot(img)
img <- image_resize(img, 50)
img1 <- image_rotate(img, 45)
img2 <- image_hreflect(img)
img3 <- image_vreflect(img)
img4 <- image_vertical(img)
image_combine(img1, img2, img3, img4)
}
```

---

utils_wd *Set and get the Working Directory quicky*

Description

- `get_wd_here()` gets the working directory to the path of the current script.
- `set_wd_here()` sets the working directory to the path of the current script.
- `open_wd_here()` Open the File Explorer at the directory path of the current script.
- `open_wd()` Open the File Explorer at the current working directory.

Usage

```
set_wd_here(path = NULL)

get_wd_here(path = NULL)

open_wd_here(path = get_wd_here())

open_wd(path = getwd())
```

Arguments

path	Path components below the project root. Defaults to NULL. This means that the directory will be set to the path of the file. If the path doesn't exist, the user will be asked if he wants to create such a folder.

Value

- get_wd_here() returns a full-path directory name.
- get_wd_here() returns a message showing the current working directory.
- open_wd_here() Opens the File Explorer of the path returned by get_wd_here().

Examples

```
if (interactive()) {
get_wd_here()
set_wd_here()
open_wd_here()
}
```

---

uuid *Generate Version 7 UUIDs or Random UUIDs*

Description

This function generates one or more UUIDs (Universally Unique Identifiers). By default, it generates Version 7 UUIDs, which are time-ordered and suitable for use cases requiring efficient indexing and sorting by creation time. Alternatively, random Version 4 UUIDs can be generated by setting usetime = FALSE.

Usage

```
uuid(n = 1, uppercase = FALSE, usetime = FALSE)
```

Arguments

`n`	Integer. Number of UUIDs to generate. Default is `1`.
`uppercase`	Logical. If `TRUE`, the generated UUIDs are returned in uppercase letters. Default is `FALSE`.
`usetime`	Logical. If `TRUE`, generates Version 7 UUIDs using the current timestamp. If `FALSE`, generates random Version 4 UUIDs. Default is `FALSE`.

Details

- **Version 7 UUIDs**: These are time-ordered UUIDs based on the current timestamp in milliseconds since the Unix epoch (`1970-01-01 00:00:00 UTC`). They are ideal for scenarios requiring chronological sorting or indexing.
- **Version 4 UUIDs**: These are randomly generated UUIDs that do not depend on time, ensuring uniqueness through random hexadecimal values.

Value

A character vector of UUIDs of length `n`.

Examples

```
library(pliman)
# Generate a single UUID
uuid()

# Generate 5 UUIDs in uppercase
uuid(n = 3, uppercase = TRUE)

# Generate two random UUIDs
uuid(n = 2, usetime = FALSE)
```

`video_detect_dl` *Real-Time Video Object Detection with YOLO (ONNX Runtime)*

Description

Performs real-time object detection and tracking on video files (.mp4, .avi, .mov) or live camera feeds (webcam) using YOLO models via native C++ ONNX Runtime with zero Python dependencies. When `return_data = TRUE`, it computes an extensive ecological and temporal community profile (richness, abundance, Shannon diversity, Simpson index, Pielou evenness, species accumulation curves, and class persistence rates).

Usage

```
video_detect_dl(
  video = 0,
  model = "yolo26n",
  conf_threshold = 0.25,
  iou_threshold = 0.45,
  labels = NULL,
  classes = NULL,
  detect = TRUE,
  output = NULL,
  save_video = !is.null(output),
  return_data = TRUE,
  track = FALSE,
  trail = TRUE,
  cex = 1,
  pad = 1,
  badge = TRUE,
  show_text = TRUE,
  show_conf = TRUE,
  show_class = TRUE,
  show_id = FALSE,
  text_size = 1,
  count_line = NULL,
  count_line_label = NULL,
  roi = NULL,
  roi_mode = c("crop", "filter"),
  roi_label = "ROI / MONITORED ZONE",
  hide_outside_roi = TRUE,
  hide_counted = TRUE,
  flash_counted = TRUE,
  fullscreen = FALSE,
  window_size = NULL,
  window_scale = 1,
  resolution = NULL,
  crop = NULL,
  backend = c("native", "auto"),
  skeleton = TRUE,
  kpt_threshold = 0.3,
  kpt_radius = 4,
  bbox = TRUE,
  mask = TRUE,
  alpha = 0.4,
  show = NULL,
  show_fps = TRUE,
  hud_pos = c("top-right", "top-left", "bottom-left", "bottom-right", "top", "bottom"),
  hud_layout = c("vertical", "horizontal"),
  infer_every = 1L,
  max_frames = NULL,
```

```
  fps = NULL,
  col = NULL,
  lwd = 2,
  rainbow = TRUE,
  engine = c("gpu", "cpu"),
  device_id = -1,
  threads = 0,
  dir = pliman_model_dir(),
  verbose = TRUE,
  ...
)

video_detect(
  video = 0,
  model = "yolo26n",
  conf_threshold = 0.25,
  iou_threshold = 0.45,
  labels = NULL,
  classes = NULL,
  detect = TRUE,
  output = NULL,
  save_video = !is.null(output),
  return_data = TRUE,
  track = FALSE,
  trail = TRUE,
  cex = 1,
  pad = 1,
  badge = TRUE,
  show_text = TRUE,
  show_conf = TRUE,
  show_class = TRUE,
  show_id = FALSE,
  text_size = 1,
  count_line = NULL,
  count_line_label = NULL,
  roi = NULL,
  roi_mode = c("crop", "filter"),
  roi_label = "ROI / MONITORED ZONE",
  hide_outside_roi = TRUE,
  hide_counted = TRUE,
  flash_counted = TRUE,
  fullscreen = FALSE,
  window_size = NULL,
  window_scale = 1,
  resolution = NULL,
  crop = NULL,
  backend = c("native", "auto"),
  skeleton = TRUE,
```

```
  kpt_threshold = 0.3,
  kpt_radius = 4,
  bbox = TRUE,
  mask = TRUE,
  alpha = 0.4,
  show = NULL,
  show_fps = TRUE,
  hud_pos = c("top-right", "top-left", "bottom-left", "bottom-right", "top", "bottom"),
  hud_layout = c("vertical", "horizontal"),
  infer_every = 1L,
  max_frames = NULL,
  fps = NULL,
  col = NULL,
  lwd = 2,
  rainbow = TRUE,
  engine = c("gpu", "cpu"),
  device_id = -1,
  threads = 0,
  dir = pliman_model_dir(),
  verbose = TRUE,
  ...
)
```

Arguments

`video`	Input video source. Either a camera index (e.g. `0`, `1`), a camera name or substring (e.g. `"EMEET"`, `"HD User Facing"`), `"camera"` (default camera `0`), or a path to a video file (e.g., `"field_drone.mp4"`).
`model`	Model name or path to a custom ONNX file. Defaults to `"yolo26n"`. Pre-trained options include `"yolo26n"`, `"yolo26s"`, `"yolov8n"`, `"yolov8s"`, `"yolov11n"`, etc., or a full path to a custom user-trained `.onnx` model.
`conf_threshold`	Minimum confidence threshold for candidate detections (default 0.25).
`iou_threshold`	IoU threshold for Non-Maximum Suppression (default 0.45).
`labels`	Optional character vector of class names. If `NULL` (default), labels are automatically detected from model metadata or fallback to standard classes.
`classes`	Optional character vector of specific class names to filter and retain. If specified, only detections matching these classes will be kept.
`detect`	Logical. Whether to run object detection model inference. If `FALSE`, the camera/video stream runs in pass-through mode without neural network inference, allowing fast real-time streaming and recording. Defaults to `TRUE`.
`output`	Optional output file path (e.g. `"annotated_video.mp4"`) to save the processed video.
`save_video`	Logical. Whether to save the processed video. Automatically `TRUE` if `output` is specified.

`return_data` Logical. Whether to compute and return the detection dataset and ecological summary statistics (default `TRUE`). If `FALSE`, returns `invisible(NULL)`.

`track` Logical. Whether to enable multi-object tracking and assign persistent IDs (`#1`, `#2`, etc.) across frames. Defaults to `FALSE`.

`trail` Logical. Whether to draw historical trajectory motion trail lines behind tracked objects when `track = TRUE`. Defaults to `TRUE`. Set to `FALSE` to maintain persistent IDs while hiding the trail.

`cex` Size factor for label text (default `1.0`). Use smaller values (e.g., `cex = 0.2` or `0.5`) to reduce text size for dense scenes or many objects.

`pad` Padding multiplier controlling the area/background badge size around label text (default `1.0`). Can be a single value or a length-2 vector `c(pad_x, pad_y)`. Use `pad = 0` or smaller values for a compact area.

`badge` Logical. Whether to draw a solid background badge behind label text (default `TRUE`). If `FALSE`, only the text is drawn directly.

`show_text` Logical. Whether to display text badges above bounding boxes (default `TRUE`).

`show_conf` Logical. Whether to display confidence scores in badges (default `TRUE`).

`show_class` Logical. Whether to display class label names in badges (default `TRUE`).

`show_id` Logical. Whether to display persistent tracking IDs (`#1`, `#2`) in badges when `track = TRUE` (default `FALSE`).

`text_size` Numeric font scale for bounding box text labels and badges (alias for `cex`, default `1.0`).

`count_line` Optional counting tripwire line specified as `c(x1, y1, x2, y2)` in normalized coordinates (`0` to `1`). Objects crossing this virtual line are counted.

`count_line_label` Optional label displayed along the counting line (default `NULL`, no text).

`roi` Optional Region of Interest coordinates `c(xmin, ymin, xmax, ymax)` in normalized coordinates (`0` to `1`).

`roi_mode` Either `"crop"` (infer only inside ROI) or `"filter"` (detect everywhere, retain only detections inside ROI).

`roi_label` Label displayed on the ROI box (default `"ROI / MONITORED ZONE"`).

`hide_outside_roi` Logical. Whether to remove / suppress bounding boxes and visual annotations for objects outside the ROI zone. Defaults to `TRUE` when `roi` is provided.

`hide_counted` Logical. Whether to remove / suppress bounding boxes for objects that have already crossed the virtual counting line. Defaults to `TRUE` when counting / tracking is active.

`flash_counted` Logical. Whether to show a brief neon green confirmation flash (~12 frames) when an object crosses the counting line before removing its bounding box. Defaults to `TRUE`.

`fullscreen` Logical. Whether to display the video feed in full screen mode (default `FALSE`).

window_size	Optional numeric vector c(width, height) for the display window.
window_scale	Scaling factor for the display window (default 1.0).
resolution	Optional camera capture resolution as a numeric vector c(width, height), e.g. c(1920, 1080) or c(1280, 720). Default is NULL (uses native sensor resolution).
crop	Optional frame cropping. Set to "1:1", "square", or TRUE for an automatic 1:1 centered square crop (e.g. 1080x1080 or 720x720), or c(width, height) for a custom centered rectangular crop. Default is NULL.
backend	Camera and display backend: "native" (default, uses native Windows Media Foundation / Win32 GDI with zero external dependencies).
skeleton	Logical. Whether to render the 17-keypoint anatomical skeleton when running pose estimation models (e.g. "yolo26n-pose"). Defaults to TRUE.
kpt_threshold	Minimum confidence threshold for rendering individual keypoints and skeleton limbs (default 0.3).
kpt_radius	Integer radius of keypoint joint markers in pixels (default 4).
bbox	Logical. Whether to render bounding boxes around detected objects or people (default TRUE).
mask	Logical. Whether to render semi-transparent instance segmentation masks when using segmentation models (default TRUE).
alpha	Opacity/transparency of instance segmentation masks (default 0.4, range 0 to 1).
show	Logical. Whether to display the video feed with bounding boxes in real time. Default is interactive().
show_fps	Logical. Whether to overlay a real-time FPS and detection counter badge (default TRUE).
hud_pos	Screen position of the telemetry HUD card: "top-right" (default, avoids top-left ROI zone), "top-left", "bottom-left", "bottom-right", "top", or "bottom".
hud_layout	Layout style of the telemetry HUD: "vertical" (default modern multi-row glassmorphic card) or "horizontal" (sleek capsule pill bar).
infer_every	Integer stride for neural network inference (default 1L, running inference on every frame). Setting e.g. infer_every = 2L (or using alias stride = 2 / skip_frames = 1) runs the heavy neural network on odd frames and smoothly propagates tracks and detections on intermediate frames, cutting compute time in half and maximizing real-time video display smoothness up to the camera's physical frame rate.
max_frames	Optional maximum number of frames to process. Default is NULL (all frames).
fps	Target framerate for video output. If NULL (default), matches the source video or camera.
col	Optional color or palette for bounding boxes. If NULL, distinct colors are assigned.

`lwd`	Line width for bounding boxes (default 2).
`rainbow`	Logical. If `TRUE`, assigns distinct colors per detected instance.
`engine`	Execution engine: `"gpu"` (DirectML, default) or `"cpu"`.
`device_id`	GPU device ID (default -1 for auto).
`threads`	Number of CPU threads (default 0 for auto-tuning).
`dir`	Directory where pre-trained models are stored (default `pliman_model_dir()`).
`verbose`	Logical. Whether to show progress messages.
`...`	Additional arguments passed to graphics plotting functions or aliases (`camera`, `cam`, `device`, `infer`, `inference`, `save`, `return_df`, `return_detections`).

Details

Ecological and Analytical Framework: In automated phenotyping, precision agriculture, ecological monitoring, and industrial inspection, a video stream represents a continuous temporal transect composed of F sequential frames recorded at a frame rate of FPS frames per second. In each frame $t \in \{1, \ldots, F\}$, deep neural network inference detects candidate bounding boxes belonging to S_t recognized classes.

By aggregating detections over space and time, `video_detect_dl` translates raw computer vision detections into standardized ecological metrics, quantifying instantaneous community state, temporal persistence, class dominance, and cumulative discovery rates.

Abundance Metrics and Density Dynamics: Abundance characterizes the numerical density of detected objects:

- **Instantaneous Abundance (N_t)**: The total count of object instances detected in frame t:
$$N_t = \sum_{c=1}^{S} n_{t,c}$$
where $n_{t,c}$ is the count of detected objects belonging to class c in frame t, and S is the total number of candidate classes.
- **Total Video Abundance (N)**: The cumulative sum of all detections identified across the processed footage:
$$N = \sum_{t=1}^{F} N_t = \sum_{c=1}^{S} n_c$$
where $n_c = \sum_{t=1}^{F} n_{t,c}$ is the total detection count for class c.
- **Average Abundance per Frame ($\bar{N}$)**: Quantifies the baseline object density per frame:
$$\bar{N} = \frac{N}{F}$$
- **Peak Abundance (N_{peak})**: The maximum instantaneous count observed in any single frame:
$$N_{\text{peak}} = \max_{1 \le t \le F} N_t$$
The exact frame index $t_{\text{peak}} = \text{argmax}_t\, N_t$ and corresponding timestamp (in seconds) $T_{\text{peak}} = (t_{\text{peak}} - 1)/\text{FPS}$ identify the moment of maximum congestion or cluster density.

Richness and Species Accumulation Dynamics: Richness quantifies class variety and categorical complexity over time:

- **Instantaneous Richness (S_t)**: The number of distinct classes co-occurring simultaneously in frame t:
$$S_t = \sum_{c=1}^{S} \mathbb{I}(n_{t,c} > 0)$$
where $\mathbb{I}(\cdot)$ is the indicator function equal to 1 if class c has at least one detection in frame t, and 0 otherwise.
- **Total Video Richness (S)**: The total count of unique classes recognized across the entire video:
$$S = |\{c : n_c > 0\}|$$
- **Cumulative Richness / Species Accumulation Curve (Collector's Curve)**:
$$S_{\text{cum}}(t) = |\{c : \exists t' \leq t \text{ such that } n_{t',c} > 0\}|$$
The Collector's Curve plots the progressive discovery of novel classes over time. A plateau indicates that sampling effort (video duration) is sufficient to capture the full assemblage of classes present in the environment.
- **Cumulative Detections ($N_{\text{cum}}(t)$)**:
$$N_{\text{cum}}(t) = \sum_{t'=1}^{t} N_{t'}$$
Tracking the cumulative volume of detections provides insights into detection throughput and arrival rates.

Per-Class Ecological Metrics and Constancy: For each class c, `video_detect_dl` extracts detailed persistence and distribution metrics:

- **Total Detections (n_c)**: Total bounding box instances identified for class c.
- **Relative Abundance (p_c)**: The percentage contribution of class c to total video detections:
$$p_c = \frac{n_c}{N} \times 100\%$$
- **Frames Present (F_c)**: The number of frames in which class c was detected at least once:
$$F_c = \sum_{t=1}^{F} \mathbb{I}(n_{t,c} > 0)$$
- **Occurrence Rate / Constancy (O_c)**: The temporal persistence of class c across the video footage:
$$O_c = \frac{F_c}{F} \times 100\%$$
In ecological literature, species with $O_c \geq 50\%$ are classified as constant, $25\% \leq O_c < 50\%$ as accessory, and $O_c < 25\%$ as accidental.
- **Mean Abundance When Present**: The expected number of individuals per frame given that the class is present:
$$\bar{n}_{c,\text{present}} = \frac{n_c}{\max(1, F_c)}$$

- **Mean Abundance Overall**: The unconditional expected number of individuals per frame across all processed frames:

$$\bar{n}_{c,\text{overall}} = \frac{n_c}{F}$$

- **Maximum per Frame** ($\max_t n_{t,c}$): Peak instantaneous count of class c.
- **Mean Confidence** ($\bar{s}_c$): The arithmetic mean of detection confidence scores for class c:

$$\bar{s}_c = \frac{1}{n_c} \sum_{i=1}^{n_c} \text{score}_i$$

- **Temporal Span**: First observed timestamp ($t_{\text{first},c}$) and last observed timestamp ($t_{\text{last},c}$) in seconds.

Diversity, Dominance, and Evenness Indices: Summary diversity indices synthesize class variety and equitability into scalar coefficients:

- **Shannon-Wiener Diversity Index** (H'):

$$H' = -\sum_{c=1}^{S} p_c \ln(p_c)$$

  where $p_c = n_c/N$. H' quantifies the uncertainty in predicting the class identity of a randomly sampled detection. Higher values indicate higher diversity and a more equitable distribution across classes. In addition to the global video-level index, H'_t is computed dynamically for each frame in the `$temporal` data frame.
- **Simpson's Diversity Index** ($1 - D$) **and Dominance** (D):

$$D = \sum_{c=1}^{S} p_c^2, \qquad 1 - D = 1 - \sum_{c=1}^{S} p_c^2$$

  D represents the probability that two randomly chosen detections belong to the same class (dominance), while $1 - D$ (Gini-Simpson index) represents the probability that they belong to different classes. Values range from 0 (monoculture / single class dominating entirely) to $1 - 1/S$ (maximal diversity).
- **Pielou's Evenness Index** (J'):

$$J' = \frac{H'}{\ln(S)}$$

  for $S > 1$ (with $J' = 1.0$ if $S = 1$, and $J' = 0.0$ if $S = 0$). J' normalizes Shannon diversity against its theoretical maximum ($H'_{\max} = \ln(S)$), providing an equitability index bounded between 0 and 1. A value of 1.0 indicates that all classes are detected with equal frequency.
- **Berger-Parker Dominance Index** (d):

$$d = \max_{1 \le c \le S} (p_c)$$

  Quantifies the proportional dominance of the most abundant single class in the assemblage.

Multi-Object Tracking and Unique Entity Counting: When `track = TRUE`, a spatial-temporal Kalman filter tracking algorithm (SORT/ByteTrack) associates detections across consecutive frames using Intersection over Union (IoU) cost matrices:

- **Unique Objects (U)**: Persistent IDs (`#1`, `#2`, ...) track physical entities over time, preventing double-counting when calculating true entity abundance:

$$U = |\{\text{distinct persistent IDs} > 0\}|$$

  For each class c, U_c records the unique tracked individuals assigned to that class.
- **Cumulative Unique Objects Curve ($U_{\text{cum}}(t)$)**: Tracks the progressive entry of novel unique physical entities into the monitored frame.
- **Virtual Tripwire Counting (`count_line`)**: When a virtual line is provided as `c(x1, y1, x2, y2)` in normalized coordinates $[0, 1]$, the trajectory segments between consecutive centroids $\mathbf{p}_{t-1}$ and $\mathbf{p}_t$ of tracked objects are evaluated for intersection with the virtual line segment. Crossing events are registered precisely when the signed 2D cross-product changes orientation, cataloguing the event frame, timestamp, persistent ID, and class label in `$crossings`.

Visualization and S3 Methods: The returned object has class `c("pliman_video_detect", "list")` and provides built-in S3 methods:

- `plot(x, type = ...)`: Renders clean, publication-ready graphics. Supported `type` options:
  - `"temporal"` (default): Two-panel stacked time series displaying total abundance $N(t)$ and richness $S(t)$.
  - `"abundance"`: Multi-class abundance trajectories over time.
  - `"richness"`: Step-function curve of class richness over time.
  - `"cumulative"`: Two-panel plot of the Species Accumulation Curve (Collector's Curve) and cumulative detection volume.
  - `"class"`: Horizontal barplot displaying total detections and relative abundance percentages for each class.
  - `"all"`: 2x2 multi-panel overview combining abundance, richness, class distribution, and the Collector's Curve.
- `summary(x)`: Formats and displays an overview table with video metadata, ecological metrics, and class distribution.
- `print(x)`: Displays an informative console summary with key video metrics and class rankings.
- `as.data.frame(x)`, `head(x)`, `tail(x)`, `dim(x)`, `nrow(x)`, `ncol(x)`, `x$column`, `x[i, j]`: Provide transparent backward compatibility with data frame operations on raw detections.

Value

If `return_data = TRUE`, an object of class `c("pliman_video_detect", "list")` containing:

- `summary`: A `data.frame` of video-level metadata and ecological diversity metrics: `video_source`, `model`, `engine`, `total_frames`, `fps`, `duration_sec`, `total_detections`

(N), `total_richness` (S), `unique_objects` (U), `avg_abundance_frame` ($\bar{N}$), `max_abundance_frame` (N_{peak}), `peak_frame` (t_{peak}), `peak_time_sec` (T_{peak}), `shannon_index` (H'), `simpson_index` ($1 - D$), `pielou_evenness` (J'), and optionally `total_counted` (when `count_line` is used).

- `by_class`: A `data.frame` sorted by total detections detailing per-class metrics: `class`, `total_detections` (n_c), `relative_abundance` (p_c, \ `unique_objects` (U_c), `n_frames_present` (F_c), `occurrence_rate` (O_c, \ `mean_abundance_when_present` ($\bar{n}_{c,\text{present}}$), `mean_abundance_overall` ($\bar{n}_{c,\text{overall}}$), `max_per_frame`, `mean_confidence` ($\bar{s}_c$), `first_seen_sec` ($t_{\text{first},c}$), `last_seen_sec` ($t_{\text{last},c}$), and optionally `total_counted`.
- `unique_objects`: A `data.frame` aggregating every unique tracked object across all observed frames: `id`, `label`, `n_frames`, `first_frame`, `last_frame`, `duration_sec`, `mean_score`, and (when using instance segmentation models) detailed morphological shape features: `area` (in pixels), `area_sd`, `perimeter`, `radius_mean`, `length`, `width`, `circularity`, `eccentricity`, and `asp_ratio`.
- `temporal`: A `data.frame` of frame-by-frame time series at resolution $t \in \{1, \ldots, F\}$: `frame`, `timestamp` (in seconds), instantaneous `abundance` (N_t), instantaneous `richness` (S_t), instantaneous `shannon` diversity (H'_t), and individual count columns for every detected class.
- `cumulative`: A `data.frame` of cumulative progression: `frame`, `timestamp`, `cumulative_detections` ($N_{\text{cum}}(t)$), `cumulative_richness` (Species Accumulation Curve / Collector's Curve, $S_{\text{cum}}(t)$), and optionally `cumulative_unique_objects` ($U_{\text{cum}}(t)$).
- `diversity`: A `list` containing global ecological diversity indices: `richness_S` (S), `total_abundance_N` (N), `shannon_H` (H'), `simpson_1_minus_D` ($1 - D$), `simpson_D` (D), `pielou_J` (J'), `berger_parker_d` (d), `mean_richness_per_frame`, and `mean_abundance_per_frame`.
- `detections`: A `data.frame` of all raw candidate detections across all frames: `frame`, `timestamp`, `id`, `xmin`, `ymin`, `xmax`, `ymax`, `label`, `score`, and `class_id` (plus shape features when running segmentation models).
- `crossings`: (Optional) A `data.frame` of virtual tripwire crossing events when `count_line` is specified.
- `keypoints`: (Optional) A `data.frame` of 17 keypoint anatomical landmarks when running pose estimation models.
- `roi`, `roi_mode`, `count_line`, `output`: Additional metadata reflecting execution configuration.

If `return_data = FALSE`, returns `invisible(NULL)`.

Examples

```
## Not run:
  library(pliman)

  # 1. Video detection and ecological diversity analysis
  dets <- video_detect_dl(
    video = "field_drone.mp4",
    model = "yolo26n",
    conf_threshold = 0.5,
    return_data = TRUE,
```

```
  output = "annotated_field.mp4"
)

# Summary table of ecological metrics
summary(dets)

# Plot community temporal dynamics (abundance & richness)
plot(dets, type = "temporal")

# Plot Species Accumulation Curve (Collector's Curve)
plot(dets, type = "cumulative")

# Plot multi-class abundance trajectories
plot(dets, type = "abundance")

# Multi-panel 2x2 diagnostic overview
plot(dets, type = "all")

# Access per-class breakdown
dets$by_class

# 2. Multi-object tracking with virtual tripwire line counting
tracked_dets <- video_detect_dl(
  video = "conveyor_belt.mp4",
  model = "seed_weights.onnx",
  track = TRUE,
  count_line = c(0.5, 0, 0.5, 1), # vertical counting tripwire
  return_data = TRUE
)

## End(Not run)
```

`video_split` *Split a video into image frames*

Description

`video_split()` extracts individual frames from a video file and saves them directly to disk as image files (`.jpg` or `.png`). It provides native hardware-accelerated decoding on Windows via Windows Media Foundation and C++, saving images directly to disk without intermediate temporary files or external command-line dependencies.

On Linux and macOS, or when explicitly requested via `backend = "av"`, it seamlessly delegates decoding to the `av` package.

You can control the extraction rate by either:

- Specifying `fps` (e.g., `fps = 1` for 1 frame per second).
- Specifying `n` for a fixed number of uniformly spaced frames sampled across the entire video duration.
- Leaving both `NULL` to extract every frame of the video.

`video_to_frames()` is an alias for `video_split()`.

Usage

```
video_split(
  video,
  dir = "frames",
  fps = NULL,
  n = NULL,
  format = c("jpg", "png"),
  quality = 95,
  prefix = "frame_",
  digits = 6,
  backend = c("auto", "native", "av"),
  verbose = TRUE
)
```

Arguments

`video`	Character. Path to the video file (e.g., `"video.mp4"`, `"clip.avi"`).
`dir`	Character. Directory where the extracted frames will be saved. Defaults to `"frames"`. If the directory does not exist, it will be created.
`fps`	Numeric. Desired frame rate (frames per second) for extraction. For example, `fps = 1` extracts one frame per second, `fps = 0.5` extracts one frame every two seconds. Mutually exclusive with `n`.
`n`	Integer. Desired total number of frames to sample uniformly across the entire video duration. For example, `n = 50` will evenly sample 50 frames from the start to the end of the video. Mutually exclusive with `fps`.
`format`	Character. Output image format: `"jpg"` (or `"jpeg"`) or `"png"`. Defaults to `"jpg"`.
`quality`	Integer (1-100). JPEG compression quality. Defaults to `95`. Only used when `format = "jpg"`.
`prefix`	Character. Prefix for generated frame filenames. Defaults to `"frame_"`.
`digits`	Integer. Number of digits with zero-padding in frame filenames (e.g., `digits = 6` produces `"frame_000001.jpg"`). Defaults to `6`.
`backend`	Character. Decoding engine to use: `"auto"` (default), `"native"` (Windows Media Foundation C++), or `"av"` (using package `av`).
`verbose`	Logical. If `TRUE` (default), displays extraction progress and summary.

Value

A data frame of class `c("pliman_video_frames", "data.frame")` containing:

- `frame`: Frame sequential index.
- `file`: Full path to the saved image file.
- `time`: Timestamp of the frame in seconds.
- `width`: Width of the image in pixels.
- `height`: Height of the image in pixels.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
## Not run:
# Extract 1 frame per second
video_split("my_video.mp4", dir = "frames_fps", fps = 1)

# Sample exactly 20 frames uniformly across the video
video_split("my_video.mp4", dir = "frames_n", n = 20)

# Extract all frames as PNG
video_to_frames("my_video.mp4", dir = "all_frames", format = "png")

## End(Not run)
```

---

view — *Generic function for interactive viewing of images and spatial objects*

Description

Generic function for interactive viewing of images and spatial objects

Usage

```
view(x, ...)
```

Arguments

x	An image or spatial object to view.
...	Additional arguments passed to specific view methods.

---

view.image — *Interactive viewer method for* image *objects with Zoom In and Zoom Out*

Description

Provides an interactive viewer for image objects in R. When using base viewer mode, users can interactively zoom in (selecting a bounding rectangle), zoom out, reset, or finish viewing.

Usage

```
## S3 method for class 'image'
view(x, viewer = get_pliman_viewer(), ...)
```

Arguments

x	An image object.
viewer	Character specifying the viewer back-end: "base" (interactive console plot) or "mapview" (interactive Leaflet map). Default is retrieved via get_pliman_viewer().
...	Additional arguments passed to image_view() or plot.image().

Value

Invisibly returns the current zoomed sub-image or full image object.

Examples

```
## Not run:
img <- image_import(image_pliman("sev_leaf.jpg"))
view(img)

## End(Not run)
```

---

yolo_dataset_add_background *Add Negative Background Images to a YOLO Dataset*

Description

Adds negative background images (images without objects and matching empty .txt label files) to an existing YOLO dataset. This is an official Ultralytics best practice to drastically reduce false positive detections. Supports local image folders, vector of image file paths, an Ultralytics Platform .ndjson export file, or a direct URL.

Usage

```
yolo_dataset_add_background(
  dir = "yolo_dataset",
  source,
  n = 20L,
  train_prop = 0.8,
  prefix = "bg_",
  verbose = TRUE
)
```

Arguments

dir	Character or yolo_dataset. Path to the YOLO dataset directory (e.g. "yolo_toras").
source	Character. Either a path to an Ultralytics .ndjson file, a web URL to an .ndjson, a directory with images, or a vector of image paths.

`n`	Integer. Number of background images to add. Defaults to `20L`.
`train_prop`	Numeric. Proportion of background images assigned to the training split. Defaults to `0.80`.
`prefix`	Character. Filename prefix for added background images. Defaults to `"bg_"`.
`verbose`	Logical. Whether to show progress messages. Defaults to `TRUE`.

Value

Invisible list with summary of added files.

Downloading Datasets from Ultralytics Hub

Ultralytics provides a rich repository of ready-to-use open datasets at https://docs.ultralytics.com/datasets/ and https://platform.ultralytics.com/. To obtain an `.ndjson` file to use here:

1. Visit https://docs.ultralytics.com/datasets/ or log in to the Ultralytics Platform at https://platform.ultralytics.com/.
2. Browse or search for your desired dataset (e.g. African Wildlife, COCO8, SKU-110k, VisDrone, VOC).
3. Open the dataset page and navigate to the **Versions** or **Overview** tab.
4. Click the **Download** (Export) icon in the top header.
5. Select **NDJSON** (Newline Delimited JSON) format and download the file (e.g. `"african-wildlife.ndjson"`).
6. Pass the downloaded file path directly to `yolo_dataset_add_background(source = "african-wildlife.ndjson")`!

Examples

```
## Not run:
# Using a local NDJSON downloaded from Ultralytics Hub:
yolo_dataset_add_background(
  dir = "yolo_toras",
  source = "D:/Downloads/pliman_dl/african-wildlife.ndjson",
  n = 20
)

## End(Not run)
```

`yolo_dataset_audit` *Comprehensive YOLO Dataset Health Audit and Auto-Repair*

Description

`yolo_dataset_audit()` conducts a deep structural, syntactic, and geometric health check on any YOLO dataset (Detection, Instance Segmentation, or Image Classification). It detects duplicate file stems across extensions (stem collisions, e.g. `foo.jpg` and `foo.png` colliding on `foo.txt`), orphaned images/labels, corrupt files, invalid coordinates (< 0 or > 1), negative bounding box dimensions, malformed polygons, class index out-of-bounds, and severe class imbalance.

With `repair = TRUE`, it automatically repairs the dataset in-place:

- Renames colliding images and review previews uniquely (`foo_jpg.jpg`, `foo_png.png`) and links existing `.txt` annotations to the best-matching file based on modification timestamps.
- Clamps out-of-bounds coordinates to `[0, 1]`.
- Prunes zero-area bounding boxes.
- Creates empty `.txt` background labels for unannotated images.
- Removes orphaned `.txt` label files.

The returned object wraps the audited dataset seamlessly so it can be chained directly with the pipe operator `|>` into `yolo_train()` or `yolo_dataset_augment()`.

Usage

```
yolo_dataset_audit(data, repair = FALSE, min_images = 10L, verbose = TRUE)
```

Arguments

`data`	A `yolo_dataset` object, `yolo_dataset_cls` object, or directory path.
`repair`	Logical. If `TRUE`, automatically fixes detected issues in-place. Defaults to `FALSE`.
`min_images`	Integer threshold for warning if a class has few samples. Defaults to `10L`.
`verbose`	Logical. Whether to display progress and diagnostic alerts. Defaults to `TRUE`.

Value

An object of class `c("yolo_dataset_audit", "yolo_dataset", "list")` containing:

- `dir`: Root directory of the dataset.
- `is_healthy`: Logical indicator of dataset health.
- `issues_found`: Total count of issues detected.
- `issues_repaired`: Count of issues successfully fixed (when `repair = TRUE`).
- `details`: `data.frame` with file-by-file diagnostic records.
- `dataset`: Underlying `yolo_dataset` or `yolo_dataset_cls` object.

Author(s)

Tiago Olivoto <tiagoolivoto@gmail.com>

Examples

```
## Not run:
library(pliman)

# Audit a YOLO dataset and automatically fix issues
ds_clean <- yolo_dataset_audit("dataset_cafe", repair = TRUE)

# Pipe directly into training!
ds_clean |> yolo_train(model = "yolo26n-seg", epochs = 50)

## End(Not run)
```

---

yolo_dataset_augment *Offline Data Augmentation with Geometric Bounding Box & Polygon Transformations*

Description

`yolo_dataset_augment()` applies robust offline geometric and photometric augmentations (horizontal/vertical flips, 90/180/270 rotations, brightness, contrast, and noise) to a YOLO dataset. Crucially, all bounding boxes and segmentation polygon vertices are dynamically recomputed with mathematical precision to match the transformed images.

Augmentations are exclusively applied to the `train/` split, strictly keeping `val/` and `test/` pristine for unbiased evaluation.

Usage

```
yolo_dataset_augment(
  data,
  out_dir = NULL,
  copies = 1L,
  flip_h = TRUE,
  flip_v = FALSE,
  rotate = c(0, 90, 180, 270),
  brightness = TRUE,
  contrast = TRUE,
  noise = FALSE,
  seed = 42,
  verbose = TRUE
)
```

Arguments

`data`	A `yolo_dataset` object, `yolo_dataset_cls` object, or directory path.
`out_dir`	Output directory. If `NULL` (default), augments the dataset in-place.
`copies`	Integer. The number of augmented variants generated per original training image (default: `1L`). For instance, if you have 20 original images, `copies = 1L` adds 20 augmented images (total 40), while `copies = 3L` adds 60 augmented images (total 80).
`flip_h`	Logical. Allow horizontal flipping with probability 0.5 (default: `TRUE`).
`flip_v`	Logical. Allow vertical flipping with probability 0.5 (default: `FALSE`).
`rotate`	Integer vector of allowable rotation angles in degrees (default: `c(0, 90, 180, 270)`). For each augmented copy, one angle is sampled randomly from this vector.
`brightness`	Logical. Allow random brightness jitter in `[0.82, 1.18]` (default: `TRUE`).
`contrast`	Logical. Allow random contrast scaling in `[0.85, 1.20]` (default: `TRUE`).
`noise`	Logical. Allow subtle additive Gaussian noise (default: `FALSE`).
`seed`	Optional integer seed for reproducibility (default: 42).
`verbose`	Logical. Whether to show progress. Defaults to `TRUE`.

Details

How `copies` and transformations work (Stochastic Sampling vs. Cartesian Grid):

The parameters (`rotate`, `flip_h`, `flip_v`, `brightness`, `contrast`, `noise`) define the *pool of eligible transformations*.

Rather than generating a full Cartesian grid product of all possible combinations (which would produce 30–60 copies per image and quickly exhaust disk space), the function samples a unique combination of transformations for each copy requested:

- `copies = 1L` (default): Generates **exactly 1 new augmented variant per original training image** (resulting in `original + 1 = 2x` training size).
- `copies = 3L`: Generates **3 distinct augmented variants per original training image** (resulting in `original + 3 = 4x` training size).
- `copies = 5L`: Generates **5 distinct augmented variants per original training image** (resulting in `original + 5 = 6x` training size).

For each copy, one angle is randomly sampled from `rotate`, flipping is evaluated probabilistically, and photometric adjustments are applied randomly within valid bounds.

Already augmented images (named with suffix `_aug[0-9]+`) are automatically skipped, preventing recursive dataset explosion if the function is executed multiple times.

Value

A `yolo_dataset` object pointing to the augmented dataset directory.

Examples

```
## Not run:
library(pliman)

# 1. Default: 1 augmented copy per image (doubles training size)
ds_aug1 <- yolo_dataset_augment("dataset_cafe")

# 2. Generate 3 distinct augmented variants per image (quadruples training size)
ds_aug3 <- yolo_dataset_augment("dataset_cafe", copies = 3)

# 3. Allow vertical flips and 4 augmented variants per image
ds_aug4 <- yolo_dataset_augment("dataset_cafe", copies = 4, flip_v = TRUE)

## End(Not run)
```

---

yolo_dataset_autolabel

Auto-Label an Entire Directory into a Production YOLO Dataset

---

Description

`yolo_dataset_autolabel()` batch-processes an entire folder of unannotated images using a trained `yolo_fewshot_model` (from `yolo_fewshot_fit()`). It predicts bounding boxes or instance segmentation polygons, writes standard YOLO `.txt` annotation files, splits the data into `train`, `val`, and `test`, and generates a `data.yaml` descriptor.

It also creates a `review/` visual inspection folder with rendered boxes/polygons so the user can quickly audit the automated labels.

Usage

```
yolo_dataset_autolabel(
  input_dir,
  model,
  output_dir,
  task = if (!is.null(model$task)) model$task else c("segment", "detect"),
  conf_threshold = NULL,
  max_objects = NULL,
  split_ratio = c(0.7, 0.2, 0.1),
  engine = "gpu",
  device_id = -1,
  review = TRUE,
  rainbow = TRUE,
  seed = 42,
  verbose = TRUE
)
```

Arguments

input_dir	Directory containing raw unannotated images.
model	A yolo_fewshot_model object fitted with yolo_fewshot_fit().
output_dir	Target directory to write the YOLO dataset.
task	Target task: "segment" (polygons) or "detect" (bounding boxes). Defaults to model$task.
conf_threshold	Minimum confidence / similarity threshold for accepting predicted instances (e.g. 0.50). If NULL (default), uses model$conf_threshold.
split_ratio	Numeric vector of length 3 for c(train, val, test) split. Defaults to c(0.70, 0.20, 0.10).
engine	Execution engine override: "cpu" or "gpu". If NULL, uses model$engine.
device_id	GPU device ID if engine = "gpu". Defaults to -1.
review	Logical. If TRUE (default), writes annotated review images to review/ subfolder.
rainbow	Logical. If TRUE, review previews are rendered with distinct rainbow colors per instance. Defaults to FALSE.
seed	Random seed for reproducible splitting. Defaults to 42.
verbose	Logical. Show progress messages. Defaults to TRUE.

Value

A yolo_dataset S3 object pointing to the created dataset.

---

yolo_dataset_balance *Balance Class Distributions via Augmentation or Downsampling*

Description

yolo_dataset_balance() balances class distributions in a YOLO dataset to eliminate majority class bias.

- With strategy = "oversample" (default), it generates augmented variants of minority classes using random flips, rotations, and photometric jitter until all classes reach the target count.
- With strategy = "undersample", it downsamples majority classes to match the count of the smallest class (or a specified target_count).

Usage

```
yolo_dataset_balance(
  data,
  strategy = c("oversample", "undersample"),
  target_count = NULL,
  clean = TRUE,
  seed = 42,
  verbose = TRUE
)
```

Arguments

`data`	A `yolo_dataset` object, `yolo_dataset_cls` object, or directory path.
`strategy`	Strategy to apply: `"oversample"` (default) or `"undersample"`.
`target_count`	Optional integer target count per class in `train/`. If `NULL`, uses the maximum count for `"oversample"` and the minimum count for `"undersample"`.
`clean`	Logical. If `TRUE` (default), removes any previously generated `_bal*.ext` files in the training split before computing counts and balancing.
`seed`	Optional random seed (default: 42).
`verbose`	Logical. Whether to show progress. Defaults to `TRUE`.

Value

A `yolo_dataset` object with balanced class distributions.

Examples

```
## Not run:
library(pliman)

# Oversample minority classes to achieve equal sample sizes
ds_balanced <- yolo_dataset_balance("dataset_flores_yolo", strategy = "oversample")

## End(Not run)
```

---

yolo_dataset_classify

Organize and Split Images into a YOLO Classification Dataset

Description

`yolo_dataset_classify()` organizes raw images structured in class subfolders (or custom lists of files) into a standardized YOLO classification dataset with `train/`, `val/`, and optionally `test/` splits per class. The output directory is 100% compliant with Ultralytics YOLOv8, YOLO11, and YOLO26, and can be passed directly to `yolo_train()` for deep learning training.

Usage

```
yolo_dataset_classify(
  src_dir,
  out_dir = "yolo_cls_dataset",
  split = c(train = 0.7, val = 0.2, test = 0.1),
  classes = NULL,
  ext = c("jpg", "jpeg", "png", "bmp", "tif", "tiff", "webp"),
  copy = TRUE,
  balance = FALSE,
  min_images = 10L,
  seed = 42,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

`src_dir`	Character path to the source root directory containing class subdirectories (e.g., `"flower_photos"` containing `"daisy"`, `"dandelion"`, `"roses"`, `"sunflowers"`, `"tulips"`). Non-directory files (such as `LICENSE.txt`) and hidden folders (starting with `.`) are automatically ignored.
`out_dir`	Output directory path for the YOLO classification dataset. Defaults to `"yolo_cls_dataset"`.
`split`	Numeric vector specifying the train / val / test split ratios. Can be of length 2 (e.g. `c(train = 0.80, val = 0.20)`) or length 3 (e.g. `c(train = 0.70, val = 0.20, test = 0.10)`). Ratios are automatically normalized to sum to 1.
`classes`	Optional character vector of specific class folder names to include. If `NULL` (default), all detected subdirectories in `src_dir` are included.
`ext`	Character vector of valid image file extensions (default: `c("jpg", "jpeg", "png", "bmp", "tif", "tiff", "webp")`).
`copy`	Logical. If `TRUE` (default), image files are copied to `out_dir`. If `FALSE`, files are moved using `file.rename()`.
`balance`	Logical or integer. If `TRUE`, balances the dataset by randomly sampling each class down to the count of the smallest class. If an integer `N`, samples at most `N` images per class. Default is `FALSE`.
`min_images`	Integer threshold for displaying a warning if a class contains very few images (default: `10L`). Ultralytics recommends >= 30-100 images per class for robust generalization.
`seed`	Optional integer for reproducible random shuffling and train/val/test partitioning (default: 42).
`overwrite`	Logical. If `TRUE`, overwrites `out_dir` if it already exists (default: `FALSE`).
`verbose`	Logical. If `TRUE` (default), displays progress and an informative summary table.

Details

The resulting directory structure conforms to the official YOLO classification format:

```
out_dir/
  train/
      class_a/
      class_b/
      ...
  val/
      class_a/
      class_b/
      ...
  test/ (optional)
      class_a/
      class_b/
      ...
```

Value

An object of class `c("yolo_dataset_cls", "yolo_dataset", "list")` containing:

- `dir`: Absolute path to the created dataset directory (ready to pass to `yolo_train(data = ...)`).
- `classes`: Character vector of class names.
- `splits`: Character vector of split names created (`"train"`, `"val"`, and optionally `"test"`).
- `summary`: `data.frame` with the number of images per class in each split and totals.
- `total_images`: Total count of processed images.
- `split_ratio`: Named numeric vector of split ratios used.

Author(s)

Tiago Olivoto `<tiagoolivoto@gmail.com>`

Examples

```
## Not run:
library(pliman)

# Organize flower photos into train (70%), val (20%), test (10%)
ds <- yolo_dataset_classify(
  src_dir = "D:/Downloads/pliman_dl/flower_photos",
  out_dir = "dataset_flores_yolo",
  split = c(train = 0.70, val = 0.20, test = 0.10),
  min_images = 10
)

# Inspect summary table
ds
```

```
# Train a YOLO classification model directly with pliman!
model <- yolo_train(
  data = ds,
  model = "yolo11n-cls",
  epochs = 50
)

## End(Not run)
```

`yolo_dataset_convert` *Convert External Annotations (CSV, COCO, Pascal VOC) to YOLO Format*

Description

`yolo_dataset_convert()` converts annotations from standard computer vision formats into a clean, ready-to-train YOLO dataset. Supports:

- **CSV**: Tabular tables (`image`, `xmin`, `ymin`, `xmax`, `ymax`, `class`) commonly exported by ImageJ, Fiji, or field records.
- **COCO**: Standard COCO JSON (`instances_train.json` with `images`, `annotations`, `categories`).
- **Pascal VOC**: Directory of XML annotation files.

Automatically computes normalized YOLO coordinates and creates `train/` and `val/` splits.

Usage

```
yolo_dataset_convert(
  input,
  format = c("csv", "coco", "voc"),
  img_dir = NULL,
  out_dir = "yolo_converted_dataset",
  split = c(train = 0.8, val = 0.2),
  copy = TRUE,
  seed = 42,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

`input`	Path to the annotation file (`.csv`, `.json`) or directory containing Pascal VOC `.xml` files.
`format`	Format type: `"csv"` (default), `"coco"`, or `"voc"`.
`img_dir`	Directory where the raw image files are stored.

out_dir	Destination directory for the YOLO dataset. Defaults to "yolo_converted_dataset".
split	Proportions for train and validation splits (default: c(train = 0.80, val = 0.20)).
copy	Logical. Whether to copy image files into the dataset. Defaults to TRUE.
seed	Random seed for split partitioning (default: 42).
overwrite	Logical. Overwrite existing out_dir. Defaults to FALSE.
verbose	Logical. Show progress. Defaults to TRUE.

Value

A yolo_dataset object pointing to out_dir.

Examples

```
## Not run:
library(pliman)

# Convert CSV bounding boxes from field phenotyping
ds_yolo <- yolo_dataset_convert(
  input   = "field_labels.csv",
  format  = "csv",
  img_dir = "field_photos",
  out_dir = "dataset_field_yolo"
)

## End(Not run)
```

---

yolo_dataset_export *Export Dataset in YOLO Format for Detection or Segmentation*

Description

Automatically generates a structured YOLO dataset (images/, labels/, and data.yaml) for object detection (task = "detect") or instance segmentation (task = "segment"). Annotations can be produced automatically via deep learning foundation models:

- **One-shot visual exemplar auto-labeling** with model = "persam" (Roboflow-style 1-click annotation).
- **Zero-shot text prompt detection** with model = "grounded-sam" (e.g. prompt = "people").
- **Assisted labeling / pseudo-labeling** with pre-trained or fine-tuned YOLO models (model = "custom.onnx").
- **Classical image analysis** with analyze_objects() (model = NULL).

Usage

```
yolo_dataset_export(
  img,
  model = NULL,
  annotations = "auto",
  dir = "yolo_dataset",
  task = c("detect", "segment"),
  class_names = c("object"),
  class_id = 0L,
  prompt = NULL,
  prompt_mode = c("each", "first"),
  interactive = is.null(prompt) && !is.null(model) && identical(tolower(model), "persam")
    && interactive(),
  sim_threshold = 0.5,
  min_dist = 20,
  box_threshold = 0.25,
  text_threshold = 0.25,
  iou_threshold = 0.5,
  feat_res = 256,
  max_objects = NULL,
  train_prop = 0.8,
  n_points = 40L,
  min_area = 15,
  filter_edge = TRUE,
  show_similarity = FALSE,
  superres_map = FALSE,
  engine = c("gpu", "cpu"),
  device_id = -1,
  overwrite = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`img`	An `image` object, a list of `image` objects, a file path, or a directory path containing images to annotate.
`model`	Deep learning model to use for automated or semi-automated labeling: • `"persam"`: One-shot visual exemplar segmentation and detection (Roboflow-style 1-click auto-labeling). • `"grounded-sam"`: Zero-shot text-prompted instance segmentation & detection. • Path to a trained YOLO `.onnx` model (e.g. `"capsulas_det.onnx"`): Assisted labeling / pseudo-labeling. • `"ben2"`, `"u2netp"`, `"birefnet-lite"`: Salient foreground object auto-segmentation. • `NULL` (default): Uses classical computer vision segmentation via `analyze_objects()`.

`annotations` Annotations source when `model = NULL`. Can be:

- `"auto"` (default): automatically calls `analyze_objects()` on each image.
- An object or list of objects returned by `analyze_objects()`.
- A custom list of contour matrices or bounding boxes.

`dir` Output directory path for the YOLO dataset. Defaults to `"yolo_dataset"`.

`task` Task type: `"detect"` (default) for bounding-box detection or `"segment"` for polygon instance segmentation.

`class_names` Character vector of class names (e.g. `c("capsula")` or `c("grain", "defect")`). Defaults to `c("object")`.

`class_id` Numeric class ID assigned to the annotations. Defaults to `0L`.

`prompt` Prompt specification for deep learning models:

- For `model = "persam"`: Exemplar coordinates (`c(x, y)` or matrix). If `NULL` and `interactive = TRUE`, prompts for clicks on the image.
- For `model = "grounded-sam"`: Character text prompt (e.g. `"people"`, `"grain"`, `"leaf"`). Defaults to `class_names` if not supplied.

`prompt_mode` Exemplar prompting mode when labeling multiple images with PerSAM:

- `"each"` (default for interactive): Prompts for exemplar clicks on each image.
- `"first"`: Prompts for exemplar clicks only on the first image, applying the points across all images.

`interactive` Logical. If `TRUE`, activates the interactive visual labeling workflow in the plot window, allowing the user to click exemplars, review instant detections with confidence tags, and confirm or adjust. Defaults to `TRUE` if `model = "persam"`, `prompt = NULL`, and R is running interactively.

`sim_threshold` Numeric cosine similarity threshold in `[0, 1]` for `model = "persam"`. Defaults to `0.50`.

`min_dist` Minimum distance (in pixels) between detected objects/peaks for PerSAM exemplar segmentation. Defaults to `20`.

`box_threshold` Bounding box confidence threshold for Grounded-SAM or YOLO models. Defaults to `0.25`.

`text_threshold`

Text confidence threshold for Grounded-SAM. Defaults to `0.25`.

`iou_threshold` Non-Maximum Suppression (NMS) IoU threshold. Defaults to `0.50`.

`feat_res` Feature map resolution for PerSAM (`64`, `256` (default), `"sr"`, or `1024`).

`max_objects` Maximum number of objects detected per image. Defaults to `300`.

`train_prop` Proportion of images assigned to the training split (default `0.8`). The remaining images are assigned to the validation split (`val`).

`n_points` Target number of polygon vertices per object for segmentation (default `40`). Points are sampled evenly along the contour to keep label files compact and clean.

`min_area` Minimum object area (in pixels) to include in the dataset. Defaults to `15`.

`filter_edge` Logical. If `TRUE` (default), filters out spurious border-cut artifacts touching image edges.

`show_similarity` Logical. If `TRUE`, displays a side-by-side 2-panel preview showing the detected objects on the left and the PerSAM Cosine Similarity Map ("Matriz de Coincidencia") on the right. Defaults to `FALSE`.

`superres_map` Logical. If `TRUE`, applies Real-ESRGAN 4x super-resolution on the PerSAM similarity map with bilateral semantic background gating. Default is `FALSE`.

`engine` Computing device engine: `"cpu"` (default) or `"gpu"`.

`device_id` GPU device ID (default `-1`).

`overwrite` Logical. If `TRUE` (default), overwrites existing dataset files in `dir`.

`verbose` Logical. If `TRUE` (default), displays live progress information.

`...` Additional arguments passed directly to `analyze_objects()` when `model = NULL` and `annotations = "auto"` (e.g. `index = "B"`, `watershed = TRUE`, `tolerance = 1`).

Details

Automated and Semi-Automated Annotation Modes:

- **PerSAM One-Shot Exemplar Auto-Labeling (`model = "persam"`):** The user clicks on 1 or more exemplar objects in the plot window. PerSAM extracts visual embeddings from SAM 2.1, calculates the spatial cosine feature similarity map ("matriz de coincidencia"), and identifies all matching instances across the image. When `interactive = TRUE`, an interactive confirmation menu allows accepting detections (), retrying clicks (`'r'`), or fine-tuning the similarity threshold (`'t 0.55'`) in real-time.
- **Grounded-SAM Text Prompting (`model = "grounded-sam"`):** Accepts natural language text descriptions (e.g. `prompt = "people"`, `prompt = "coffee grain"`, or `class_names = c("person", "bicycle")`) to locate and segment all matching objects.
- **YOLO Pre-Labeling / Active Learning (`model = "<model>.onnx"`):** Applies an existing trained YOLO model to unannotated images to produce draft labels for rapid dataset expansion.
- **Classical Image Analysis (`model = NULL`):** Extracts contours via `analyze_objects()` using thresholding, color indexes, and watershed algorithms.

The exported directory adheres to the standard structure expected by YOLOv8, YOLO11, and YOLO26:

```
dataset_dir/
|-- data.yaml
|-- images/
|   |-- train/
|   \-- val/
\-- labels/
    |-- train/
    \-- val/
```

Value

An invisible list containing:

- `dir`: Root directory of the dataset.
- `yaml_file`: Full path to the generated `data.yaml`.
- `train_images`: Vector of paths to training images.
- `val_images`: Vector of paths to validation images.
- `n_objects`: Total number of annotated objects across the dataset.

Examples

```
## Not run:
library(pliman)

# 1. Semi-Automated 1-Click Labeling with PerSAM:
# Click 1 capsule on screen -> PerSAM finds all 40+ capsules and writes YOLO boxes!
yolo_dataset_export(
  "capsulas_raw/",
  model = "persam",
  task = "detect",
  class_names = "capsula",
  dir = "dataset_capsulas"
)

# 2. Zero-Shot Text Prompt Labeling with Grounded-SAM:
# Annotate every person in photos automatically:
yolo_dataset_export(
  "crowd_photos/",
  model = "grounded-sam",
  prompt = "people",
  class_names = "person",
  dir = "dataset_people"
)

# 3. Classical Auto-Annotation with analyze_objects:
yolo_dataset_export(
  "cafe_grains/",
  task = "segment",
  class_names = "grao_cafe",
  index = "B",
  watershed = TRUE
)

## End(Not run)
```

`yolo_dataset_filter` *Filter or Subset Classes in a YOLO Dataset*

Description

`yolo_dataset_filter()` isolates specific classes of interest from a multi-class dataset (or drops unwanted categories). It rewrites all `.txt` label files, re-indexes surviving class IDs contiguously from 0 to K-1, updates `data.yaml`, and optionally cleans or preserves images that now have zero annotations (as background negatives).

Usage

```
yolo_dataset_filter(
  data,
  keep_classes = NULL,
  drop_classes = NULL,
  out_dir = "filtered_yolo_dataset",
  drop_empty_images = TRUE,
  copy = TRUE,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

`data`	A `yolo_dataset` object, `yolo_dataset_cls` object, or directory path.
`keep_classes`	Character vector of class names to retain. Mutually exclusive with `drop_classes`.
`drop_classes`	Character vector of class names to exclude.
`out_dir`	Output directory path for the filtered dataset. Defaults to `"filtered_yolo_dataset"`.
`drop_empty_images`	Logical. If `TRUE` (default), images left with zero annotations are deleted. If `FALSE`, they are kept as negative background images with empty `.txt` labels.
`copy`	Logical. Whether to copy files to `out_dir`. Defaults to `TRUE`.
`overwrite`	Logical. Whether to overwrite `out_dir` if it exists. Defaults to `FALSE`.
`verbose`	Logical. Whether to show progress. Defaults to `TRUE`.

Value

A `yolo_dataset` object pointing to the filtered dataset.

Examples

```
## Not run:
library(pliman)

# Focus exclusively on 2 classes from a 10-class dataset
ds_focused <- yolo_dataset_filter(
  "dataset_geral",
  keep_classes = c("ferrugem", "mancha_angular"),
  out_dir      = "dataset_doencas"
)

## End(Not run)
```

---

yolo_dataset_from_ndjson

Import an Ultralytics NDJSON Dataset and Export to YOLO Structure

---

Description

Reads an Ultralytics Hub dataset export (`.ndjson` format), parses dataset metadata, classes, bounding boxes or segmentation polygons, downloads images before their signed URLs expire, and writes a complete standard YOLO dataset folder with `images/`, `labels/`, and `data.yaml`. Optionally packages the entire dataset into a `.zip` archive for permanent offline storage and distribution.

Usage

```
yolo_dataset_from_ndjson(
  file,
  dir = NULL,
  splits = c("train", "val", "test"),
  train_prop = NULL,
  max_images = NULL,
  download_images = TRUE,
  parallel = TRUE,
  zip = FALSE,
  overwrite = FALSE,
  verbose = TRUE
)

yolo_dataset_import_ndjson(
  file,
  dir = NULL,
  splits = c("train", "val", "test"),
  train_prop = NULL,
  max_images = NULL,
  download_images = TRUE,
```

```
  parallel = TRUE,
  zip = FALSE,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

file	Character. Path to the .ndjson file (e.g. "african-wildlife.ndjson"), or a direct URL to an .ndjson file.
dir	Character or NULL. Target directory for the exported YOLO dataset. If NULL (default), creates a directory named yolo_<dataset_name> in the same folder as file.
splits	Character vector. Which splits to process ("train", "val", "test"). Defaults to all available splits in the file.
train_prop	Numeric or NULL. If specified (e.g. 0.80), re-splits the images randomly into train and validation subsets instead of using the original splits from the .ndjson. Defaults to NULL.
max_images	Integer or NULL. Maximum number of images to download. Defaults to NULL (all images).
download_images	Logical. Whether to download image files. Defaults to TRUE.
parallel	Logical. Whether to download images concurrently using curl::multi_download. Defaults to TRUE.
zip	Logical. Whether to create a .zip archive of the exported dataset directory. Defaults to FALSE.
overwrite	Logical. Whether to overwrite existing files. Defaults to FALSE (enables resuming).
verbose	Logical. Whether to show progress bar and informative messages. Defaults to TRUE.

Value

An object of class yolo_dataset with dataset directory paths and classes.

Downloading Datasets from Ultralytics Hub

Ultralytics provides a rich repository of ready-to-use open datasets at https://docs.ultralytics.com/datasets/ and https://platform.ultralytics.com/. To obtain an .ndjson file to use here:

1. Visit https://docs.ultralytics.com/datasets/ or log in to the Ultralytics Platform at https://platform.ultralytics.com/.
2. Browse or search for your desired dataset (e.g. African Wildlife, COCO8, SKU-110k, VisDrone, VOC).
3. Open the dataset page and navigate to the **Versions** or **Overview** tab.

4. Click the **Download** (Export) icon in the top header.
5. Select **NDJSON** (Newline Delimited JSON) format and download the file (e.g. `"african-wildlife.ndjson"`).
6. Run `yolo_dataset_from_ndjson(file = "african-wildlife.ndjson", zip = TRUE)` to download all images and generate the complete YOLO dataset before the cloud URLs expire!

Examples

```
## Not run:
# Download and export the entire dataset permanently before URLs expire:
ds <- yolo_dataset_from_ndjson(
  file = "D:/Downloads/pliman_dl/african-wildlife.ndjson",
  zip = TRUE
)
yolo_dataset_preview(ds)

## End(Not run)
```

`yolo_dataset_merge` *Merge Multiple YOLO Datasets with Class Conflict Resolution*

Description

`yolo_dataset_merge()` combines two or more YOLO datasets (Detection, Segmentation, or Classification) into a single, cohesive, conflict-free master dataset. It handles filename collisions by prepending dataset prefixes, harmonizes class indices across differing `data.yaml` definitions (with optional custom class renaming via `class_map`), updates label files on-the-fly, and generates a unified master `data.yaml`.

Usage

```
yolo_dataset_merge(
  datasets,
  out_dir = "merged_yolo_dataset",
  class_map = NULL,
  copy = TRUE,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

`datasets`	A list or character vector of dataset directory paths or `yolo_dataset` objects.
`out_dir`	Output directory path for the merged dataset. Defaults to `"merged_yolo_dataset"`.
`class_map`	Optional named list or vector mapping old class names to unified names (e.g., `list("cerca" = "fence", "folha" = "leaf")`).

copy	Logical. If TRUE (default), copies files. If FALSE, moves files using file.rename().
overwrite	Logical. If TRUE, overwrites out_dir if it exists. Defaults to FALSE.
verbose	Logical. Whether to show progress. Defaults to TRUE.

Value

A yolo_dataset object pointing to out_dir (ready for yolo_train() or piping |>).

Examples

```
## Not run:
library(pliman)

# Merge two datasets collected from different seasons
ds_total <- yolo_dataset_merge(
  datasets = list("dataset_2024", "dataset_2025"),
  out_dir  = "dataset_unificado"
)

## End(Not run)
```

---

yolo_dataset_preview *Preview Annotations in a YOLO Dataset*

Description

Loads sample images and their matching .txt annotation labels from a YOLO dataset directory and displays them in a multi-panel grid with overlaid segmentation polygons or bounding boxes. By default, randomly samples n images from the requested split.

Usage

```
yolo_dataset_preview(
  dir = "yolo_dataset",
  split = c("train", "val", "test"),
  pattern = NULL,
  n = 4L,
  random = TRUE,
  seed = NULL,
  mfrow = NULL,
  rainbow = TRUE,
  col = NULL,
  lwd = 1.5,
  fill = NULL,
  show_labels = TRUE,
  label_bg = FALSE,
  cex = 0.6,
  title = TRUE
)
```

Arguments

dir	Character or yolo_dataset object. Path to the dataset directory (or list from yolo_dataset_export() / yolo_dataset_split()).
split	Character. The subset to preview: "train" (default), "val", or "test".
pattern	Optional regex pattern to filter specific image names (e.g. "toras5\\.png"). Defaults to NULL.
n	Integer. Number of sample images to preview. Defaults to 4L.
random	Logical. If TRUE (default), samples random images from the split. If FALSE, picks the first n images.
seed	Optional integer seed for reproducible random selection.
rainbow	Logical. If TRUE (default), assigns distinct rainbow colors to each object within the image (matching review previews). If FALSE, colors are assigned by class.
col	Border color for bounding boxes/polygons. If provided, overrides rainbow and default class palette.
lwd	Numeric. Line width for borders. Defaults to 1.5.
fill	Fill color for polygons with transparency (e.g. "#00FF6633"). If NULL, no fill.
show_labels	Logical. If TRUE (default), displays class name labels.
label_bg	Logical. If TRUE, draws a subtle background badge behind each label. If FALSE (default), draws clean, discreet text matching review previews without blocking objects.
cex	Numeric. Text size for class name labels. Defaults to 0.6.
title	Logical or character. If TRUE (default), displays panel titles with filename and object count.

Value

An invisible list of the sampled image file paths.

Examples

```
## Not run:
dataset <- yolo_dataset_export(img, dir = "dataset_cafe", model = "persam")
yolo_dataset_preview(dataset, n = 4, random = TRUE)
plot(dataset, n = 6)

## End(Not run)
```

`yolo_dataset_split` *Partition or Re-stratify an Existing YOLO Dataset*

Description

`yolo_dataset_split()` takes any YOLO dataset (flat or previously partitioned) and redistributes all images and matching `.txt` labels into new `train/`, `val/`, and optionally `test/` splits. It ensures that pairs are preserved and regenerates an updated `data.yaml`.

Usage

```
yolo_dataset_split(
  data,
  out_dir = "split_yolo_dataset",
  split = c(train = 0.7, val = 0.2, test = 0.1),
  seed = 42,
  copy = TRUE,
  overwrite = FALSE,
  verbose = TRUE
)
```

Arguments

`data`	A `yolo_dataset` object, `yolo_dataset_cls` object, or directory path.
`out_dir`	Output directory path for the re-split dataset. Defaults to `"split_yolo_dataset"`.
`split`	Proportions for partitioning (default: `c(train = 0.70, val = 0.20, test = 0.10)`).
`seed`	Optional random seed for reproducible shuffling (default: 42).
`copy`	Logical. Whether to copy files. Defaults to `TRUE`.
`overwrite`	Logical. Overwrite existing `out_dir`. Defaults to `FALSE`.
`verbose`	Logical. Whether to show progress. Defaults to `TRUE`.

Value

A `yolo_dataset` object pointing to the newly split dataset.

Examples

```
## Not run:
library(pliman)

# Re-stratify into 70% train, 20% validation, 10% test
ds_split <- yolo_dataset_split("dataset_cafe", split = c(0.7, 0.2, 0.1))

## End(Not run)
```

yolo_dataset_stats *Compute Deep Statistical Profile for a YOLO Dataset*

Description

`yolo_dataset_stats()` computes granular distribution metrics for any YOLO dataset. It reports sample frequencies per split, class proportion percentages, imbalance ratios, and (for object detection/segmentation) bounding box area distributions.

Usage

```
yolo_dataset_stats(data, verbose = TRUE)
```

Arguments

`data`	A `yolo_dataset` object, `yolo_dataset_cls` object, or directory path.
`verbose`	Logical. Display formatted tables in the console. Defaults to `TRUE`.

Value

A `data.frame` containing summary statistics per class.

Examples

```
## Not run:
library(pliman)

stats <- yolo_dataset_stats("dataset_flores_yolo")

## End(Not run)
```

yolo_export *Export a Trained PyTorch YOLO Checkpoint (.pt) to ONNX*

Description

Converts a trained PyTorch YOLO checkpoint (`best.pt` or any custom `.pt` model) into a standalone ONNX model ready for instant C++ execution in `image_segment_dl()` or `image_detect_dl()`.

Usage

```
yolo_export(
  model,
  output_dir = pliman_model_dir(),
  output_name = NULL,
  imgsz = 640,
  auto_install = TRUE
)
```

Arguments

model	Path to the .pt weights file.
output_dir	Target directory to save the exported .onnx file. Defaults to pliman_model_dir().
output_name	Filename of the exported model. If NULL (default), derived from model.
imgsz	Input image resolution (default 640).
auto_install	Logical. If TRUE (default), automatically installs missing export libraries.

Value

The full path to the exported .onnx model.

Examples

```
## Not run:
library(pliman)
yolo_export("best.pt", output_name = "custom_model.onnx")

## End(Not run)
```

---

yolo_fewshot_fit *Interactive Few-Shot Exemplar Fitting for Object Detection and Segmentation*

Description

yolo_fewshot_fit() implements an interactive few-shot visual learning engine powered by Meta AI's Segment Anything 2.1 (SAM 2.1 / PerSAM). The user selects 3–5 sample instances of an object (e.g. seeds, beans, logs, leaves, lesions, fruits) by clicking on an image.

Using the PerSAM exemplar matching engine, the function computes visual prototypes from the exemplars and segments all other matching instances across the entire image with high precision.

The user can specify task = "segment" to obtain instance masks and polygonal boundaries for YOLO segmentation datasets, or task = "detect" for oriented/axis-aligned bounding boxes.

The resulting model object (yolo_fewshot_model) can be inspected, plotted, and directly propagated to hundreds of unannotated images in a folder using yolo_dataset_autolabel().

Usage

```
yolo_fewshot_fit(
  img,
  points = NULL,
  sample = NULL,
  label = "object",
```

```
  task = c("segment", "detect"),
  conf_threshold = 0.5,
  radius = NULL,
  min_dist = 12,
  min_area = NULL,
  max_area = NULL,
  max_objects = NULL,
  method = c("sam", "persam", "adaptive"),
  engine = c("gpu", "cpu"),
  device_id = -1,
  rainbow = TRUE,
  plot = TRUE,
  verbose = TRUE
)
```

Arguments

`img`	An `image` object, file path, or matrix.
`points`	Optional (x, y) coordinates of exemplar points. If `NULL` (default), an interactive plotting window opens for the user to click on 3–5 representative objects.
`label`	Character vector naming the target class(es). Can be a single class (e.g., `"bean"`, `"leaf"`, `"seed"`, `"tora"`) or a character vector of multiple classes (e.g., `c("leaf", "lesion")`, `c("healthy", "damaged")`). Defaults to `"object"`. When multiple classes are specified, `yolo_fewshot_fit()` interactively collects exemplars for each class in sequence, or uses the respective elements from `points` if a list is provided.
`task`	Target task: `"segment"` (instance segmentation polygons and masks) or `"detect"` (bounding boxes). Defaults to `"segment"`.
`conf_threshold`	Numeric scalar between 0 and 1 specifying detection confidence/similarity threshold. Defaults to `0.50` (or `0.25` for adaptive).
`radius`	Optional estimated radius in pixels for target objects. If `NULL` (default), automatically detected from exemplar edge gradients.
`min_dist`	Minimum distance in pixels between detected instances. Defaults to `16`.
`min_area`	Minimum object area in pixels. If `NULL` (default), automatically calibrated from the exemplar scale (`0.20 * median_exemplar_area`) to filter out background noise.
`max_area`	Maximum object area in pixels. If `NULL` (default), automatically calibrated (`3.80 * median_exemplar_area`).
`method`	Segmentation method: `"sam"` (or `"persam"`, Meta Segment Anything 2.1 via ONNX) or `"adaptive"` (fast color-texture morphology). Defaults to `"sam"`.
`engine`	Execution engine for SAM: `"cpu"` or `"gpu"`. Defaults to `"cpu"`.
`device_id`	GPU device ID if `engine = "gpu"`. Defaults to `-1`.

rainbow	Logical. If TRUE (default), plots instances in distinct rainbow colors..
plot	Logical. Display the detection and segmentation overlay. Defaults to TRUE.
verbose	Logical. Show progress messages. Defaults to TRUE.

Value

An object of class yolo_fewshot_model containing exemplar signatures, detected instances on the reference image, and parameters for batch propagation.

Examples

```
## Not run:
library(pliman)

# 1. Fit exemplar model on reference photo (clicks or supplied coordinates)
modelo_tora <- yolo_fewshot_fit("toras/toras22.png", label = "tora", task = "segment", method = "sam")

# 2. Inspect segmentation overlay on reference photo
plot(modelo_tora)

# 3. Auto-label entire directory of 300 photos into a YOLO dataset
ds <- yolo_dataset_autolabel("toras_raw/", model = modelo_tora, output_dir = "dataset_toras_yolo")

## End(Not run)
```

---

yolo_fewshot_predict *Predict Few-Shot Instances on a New Image*

Description

Applies a trained yolo_fewshot_model to detect and segment instances in a new image.

Usage

```
yolo_fewshot_predict(
  model,
  img,
  task = model$task,
  conf_threshold = NULL,
  max_objects = NULL,
  engine = "gpu",
  device_id = -1,
  plot = FALSE,
  rainbow = TRUE
)
```

Arguments

model	A yolo_fewshot_model fitted via yolo_fewshot_fit().
img	An image object or file path to evaluate.
task	Task override: "segment" (polygons) or "detect" (bounding boxes). Defaults to model$task.
conf_threshold	Numeric scalar override for detection confidence. Defaults to model threshold.
max_objects	Maximum number of instances to detect.
engine	Execution engine: "cpu" or "gpu". If NULL, uses model$engine.
device_id	GPU device ID if engine = "gpu". Defaults to -1.
plot	Logical. Display detection results. Defaults to FALSE.
rainbow	Logical. If TRUE, plots instances in distinct rainbow colors. Defaults to FALSE.

Value

A list containing boxes (bounding boxes data.frame) and polygons (list of normalized coordinates).

---

yolo_plot *Plot Diagnostics for YOLO Training Objects or Model Names*

Description

Convenient wrapper function for plot.yolo_train(). Accepts either an existing yolo_train S3 object, a model name string (e.g. "capsulas_det"), or a direct path to a results folder.

Usage

```
yolo_plot(x, ...)
```

Arguments

x	An object of class yolo_train, or a character string specifying a model name or results directory.
...	Arguments passed directly to plot.yolo_train() (e.g. which, native, bg).

Details

This function simplifies diagnostic inspections by allowing users to directly pass model names (e.g. yolo_plot("my_model")) without first having to call yolo_results().

See plot.yolo_train() for full mathematical formulas, statistical definitions of IoU, Confusion Matrix, Precision, Recall, F1-Score, PR Curve, mAP50, and mAP50-95, and plot interpretation guidelines.

Value

Invisibly returns the `yolo_train` object or plotted data frame.

See Also

`plot.yolo_train()`, `yolo_results()`, `yolo_train()`

Examples

```
## Not run:
library(pliman)

# Plot full 2x2 vector diagnostic dashboard directly by model name
yolo_plot("capsulas_det")

# Plot individual evaluation curves in high-resolution vector format:
yolo_plot("capsulas_det", which = "confusion_matrix")
yolo_plot("capsulas_det", which = "confusion_matrix_norm")
yolo_plot("capsulas_det", which = "pr_curve")
yolo_plot("capsulas_det", which = "f1_curve")
yolo_plot("capsulas_det", which = "metrics")

## End(Not run)
```

---

yolo_results *Load Training Results and Diagnostics for a YOLO Model*

Description

Reconstructs a `yolo_train` S3 object from an existing model name, `.onnx`/`.pt` file, or results directory without needing to retrain. The resulting object can be directly printed with `print.yolo_train()` and visualized with `plot.yolo_train()` or `yolo_plot()`.

Usage

```
yolo_results(model = NULL, results_dir = NULL, output_dir = pliman_model_dir())
```

Arguments

`model`	Name or path to a trained YOLO model (e.g. `"capsulas_det"`, `"capsulas_det.onnx"`).
`results_dir`	Optional direct path to the `<model>_results` folder. If `NULL` (default), the folder is searched in `output_dir`, in `pliman_model_dir()`, or alongside the model file.
`output_dir`	Directory where pliman models and results are stored. Defaults to `pliman_model_dir()`.

Details

When a YOLO model is trained via `yolo_train()`, all evaluation tables, weights, and logs are preserved in `<output_name>_results`. `yolo_results()` reconstructs the full training session state by parsing:

- `results.csv`: Epoch-by-epoch loss progressions, precision, recall, mAP50, and mAP50-95.
- `confusion_matrix.csv`: Complete confusion matrix counts including the background class.
- `confusion_matrix_normalized.csv`: Normalized classification rates.
- `pr_curve.csv`: 1,000-point Precision-Recall evaluation curve per class.
- `f1_curve.csv`: 1,000-point F1 vs. Confidence evaluation curve per class.
- Model weights: Automatically locates exported `.onnx` and `.pt` files.

This enables instant, offline performance inspection, publication-ready vector plotting, and threshold selection for any trained model without needing Python or GPU re-execution.

Value

An object of class `c("yolo_train", "list")`.

See Also

`plot.yolo_train()`, `yolo_plot()`, `yolo_train()`

Examples

```
## Not run:
library(pliman)

# Load diagnostics from a previously trained model
res <- yolo_results("capsulas_det")

# Inspect training metrics and plot diagnostic dashboard
print(res)
plot(res)
plot(res, which = "confusion_matrix")
plot(res, which = "pr_curve")

## End(Not run)
```

`yolo_train` *Train and Fine-Tune YOLO Models for Segmentation or Detection*

Description

High-level training interface for Ultralytics YOLO models (YOLO26, YOLO11, YOLOv8) in `pliman`. Supports instance segmentation (`task = "segment"`) and object detection (`task = "detect"`), live formatted training progress, early stopping, automatic `.pt` and `.onnx` export, and rich S3 diagnostic visualization (`plot()`).

Usage

```
yolo_train(
  data = "yolo_dataset/data.yaml",
  model = "yolo26n-seg",
  epochs = 50,
  imgsz = 640,
  batch = 16,
  device = "auto",
  output_dir = pliman_model_dir(),
  output_name = NULL,
  results_dir = NULL,
  export_pt = TRUE,
  patience = 15,
  lr0 = 0.01,
  lrf = 0.01,
  optimizer = "auto",
  close_mosaic = 10,
  amp = NULL,
  plots = FALSE,
  workers = if (.Platform$OS.type == "windows") 0L else 4L,
  auto_install = TRUE,
  ...
)
```

Arguments

`data`	Path to the `data.yaml` file (e.g. `"dataset_cafe/data.yaml"`).
`model`	Base pre-trained model architecture to fine-tune. Defaults to `"yolo26n-seg"`. Can be `"yolo26n-seg"`, `"yolo26s-seg"`, `"yolo11n-seg"`, `"yolo26n"`, or path to a `.pt` file.
`epochs`	Number of training epochs (default `50`).
`imgsz`	Input image resolution for training (default `640`).
`batch`	Batch size (default `16`).
`device`	Device to train on: `"auto"` (default), `"0"` (GPU 0), `"cpu"`, etc.

output_dir	Target directory to save the final exported .onnx model. Defaults to pliman_model_dir().
output_name	Filename of the exported model. If NULL (default), an informative name is automatically created (e.g. "yolo_custom_seg.onnx").
results_dir	Directory where training plots, confusion matrices, and results.csv are permanently preserved. If NULL (default), creates a folder named <output_name>_results inside output_dir.
export_pt	Logical. If TRUE (default), also exports the PyTorch .pt weights (e.g., model.pt) alongside the .onnx model, allowing future re-training or fine-tuning on more complex datasets.
patience	Early stopping patience in epochs. Defaults to 15.
lr0	Initial learning rate (default 0.01).
lrf	Final learning rate fraction (default 0.01).
optimizer	Optimizer to use ("auto", "SGD", "Adam", "AdamW", "RMSProp"). Default is "auto".
close_mosaic	Disable mosaic augmentation in the last N epochs. Default is 10.
amp	Automatic Mixed Precision (AMP). Set to FALSE to disable AMP if GPU checks fail. Default is NULL (auto-detected).
plots	Logical. If FALSE (default), suppresses generation of low-resolution matplotlib raster image files from YOLO; native vector plots are generated directly in R from exported CSV data.
workers	Number of worker threads for data loading. Defaults to 0 on Windows and 4 on Linux/macOS.
auto_install	Logical. If TRUE, automatically installs missing dependencies via pip. Defaults to TRUE.
...	Additional arguments passed directly to model.train() in Ultralytics (e.g. weight_decay = 0.0005, momentum = 0.937, mosaic = 0.5, mixup = 0.1, degrees = 10, box = 7.5, cls = 0.5).

Details

Training Lifecycle and Pipeline:

1. **Dataset Structure Verification**: Inspects bounding box and polygon coordinates in data.yaml to confirm task configuration, automatically reconciling task mismatches (e.g. bounding box dataset with segmentation model).
2. **Hardware Acceleration**: Auto-detects NVIDIA CUDA GPUs with cuDNN acceleration, falling back gracefully to multi-threaded CPU processing.
3. **Pre-trained Weights**: Automatically verifies or downloads official pre-trained YOLO weights from NEPEM repositories.
4. **Training Execution**: Runs Ultralytics training with real-time CLI feedback, early stopping patience, and automated learning rate scheduling.
5. **Pure Numerical Evaluation Export**: Extracts tabular CSV metrics (confusion_matrix.csv, confusion_matrix_normalized.csv, pr_curve.csv, f1_curve.csv, and results.csv) while suppressing low-resolution matplotlib raster images.

6. **Dual Model Export**: Saves both PyTorch `.pt` weights (enabling future fine-tuning or transfer learning) and optimized `.onnx` models (for pure C++ inference via ONNX Runtime in `pliman` without requiring Python).

Loss Functions and Mathematical Formulations: YOLO models optimize a composite multi-task objective balancing bounding box regression, classification, boundary distribution focal loss, and (for instance segmentation) pixel-wise mask prediction:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{cls}}\mathcal{L}_{\text{cls}} + \lambda_{\text{dfl}}\mathcal{L}_{\text{dfl}} + \lambda_{\text{seg}}\mathcal{L}_{\text{seg}}$$

1. Complete IoU (CIoU) Bounding Box Loss ($\mathcal{L}_{\text{box}}$): Bounding box regression uses the Complete IoU loss, which penalizes bounding box overlap error, normalized distance between center coordinates, and aspect ratio discrepancy:

$$\mathcal{L}_{\text{CIoU}} = 1 - \text{IoU} + \frac{\rho^2(b, b^{\text{gt}})}{c^2} + \alpha v$$

where $\rho(b, b^{\text{gt}})$ is the Euclidean distance between the center points of predicted box b and ground truth box b^{gt}, c is the diagonal length of the smallest enclosing bounding box covering both boxes, and v quantifies aspect ratio consistency:

$$v = \frac{4}{\pi^2}\left(\arctan\frac{w^{\text{gt}}}{h^{\text{gt}}} - \arctan\frac{w}{h}\right)^2$$

$$\alpha = \frac{v}{(1 - \text{IoU}) + v}$$

2. Classification Loss ($\mathcal{L}_{\text{cls}}$): Class probabilities are optimized using Binary Cross-Entropy (BCE) with sigmoid activation across all classes:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{N_c}\sum_{c=1}^{N_c}\left[y_c \log(\hat{y}_c) + (1 - y_c)\log(1 - \hat{y}_c)\right]$$

where $y_c \in \{0, 1\}$ represents ground truth class membership and $\hat{y}_c \in [0, 1]$ is the predicted class probability.

3. Instance Segmentation Mask Loss ($\mathcal{L}_{\text{seg}}$): For segmentation models, a pixel-wise binary cross-entropy loss compares the predicted prototype mask $\hat{M}$ with the ground truth binary mask M across mask domain Ω:

$$\mathcal{L}_{\text{seg}} = -\frac{1}{|\Omega|}\sum_{(x,y)\in\Omega}\left[M(x,y)\log\hat{M}(x,y) + (1 - M(x,y))\log(1 - \hat{M}(x,y))\right]$$

4. Distribution Focal Loss ($\mathcal{L}_{\text{dfl}}$): DFL models continuous bounding box coordinates y as probability distributions over discrete bins $\{y_i, y_{i+1}\}$ around the target edges:

$$\mathcal{L}_{\text{dfl}}(S_i, S_{i+1}) = -\left[(y_{i+1} - y)\log(S_i) + (y - y_i)\log(S_{i+1})\right]$$

where S_i and S_{i+1} are softmax probabilities of adjacent integer anchor bins satisfying $y_i \le y \le y_{i+1}$. This enables flexible sub-pixel boundary localization even under heavy occlusion or blurred object boundaries.

5. Cosine Annealing Learning Rate Schedule: During the warm-up phase ($t < T_{\text{warm}}$), the learning rate increases linearly from 0 to η_0. For remaining epochs ($t \geq T_{\text{warm}}$), the learning rate decays according to a cosine annealing schedule:

$$\eta_t = \eta_{\text{final}} + \frac{1}{2}(\eta_0 - \eta_{\text{final}}) \left(1 + \cos\left(\frac{t - T_{\text{warm}}}{T_{\max} - T_{\text{warm}}}\pi\right)\right)$$

where $\eta_0 = \text{lr0}$ and $\eta_{\text{final}} = \text{lr0} \times \text{lrf}$.

Value

An object of class `yolo_train` containing:

- `model_file`: Path to the exported `.onnx` model ready for `pliman`.
- `pt_file`: Path to the saved `.pt` PyTorch weights (if `export_pt = TRUE`).
- `results_dir`: Directory containing training plots, metrics, and logs.
- `plots`: Named list of file paths to generated diagnostic plots.
- `metrics`: Data frame of per-epoch loss and mAP progressions (`results.csv`).
- `confusion_matrix`: Data frame of the final confusion matrix (counts).
- `confusion_matrix_norm`: Data frame of the normalized confusion matrix.
- `pr_curve`: Data frame of the Precision-Recall curve.
- `f1_curve`: Data frame of the F1 vs Confidence curve.
- `best_metrics`: Data frame row of the best performing epoch.
- `epochs`: Number of epochs trained.
- `elapsed_sec`: Elapsed training time in seconds.
- `elapsed_formatted`: Formatted training time (e.g. `"3h 35m 14s"`).
- `task`: Task type (`"segment"` or `"detect"`).
- `call`: Matched function call.

Examples

```
## Not run:
library(pliman)

# Train a custom coffee grain segmentation model
model_res <- yolo_train(
  data = "dataset_cafe/data.yaml",
  model = "yolo26n-seg",
  epochs = 30,
  output_name = "cafe_seg.onnx"
)

# Inspect training summary and plot diagnostic dashboard
print(model_res)
plot(model_res)

# Plot specific evaluation curves:
```

```
plot(model_res, which = "confusion_matrix")
plot(model_res, which = "pr_curve")
plot(model_res, which = "metrics")

# Run inference with your new model directly in pliman!
img <- image_import("cafe.jpeg")
seg <- image_segment_dl(img, model = model_res$model_file, type = "highlight")

## End(Not run)
```

Index

∗ **data**
    contours, 33
∗ **images**
    pliman_images, 243
[.image, 7
%>% *(pipe)*, 237

align_stack *(focus_stack)*, 51
align_stack_cpp, 8
analyze_objects, 9, *24*
analyze_objects(), *9*, *19*, *24*, *25*, *142*, *159*, *225*, *227*, *259*, *262*, *263*, *292*, *303*, *310*, *311*, *323*, *364–367*
analyze_objects2 *(analyze_objects)*, 9
analyze_objects_iter *(analyze_objects)*, 9
analyze_objects_iter(), *9*, *16*, *17*
analyze_objects_minimal, 19
analyze_objects_shp, 22, *221*, *222*
analyze_objects_shp(), *106*, *260*
apply_fun_to_imgs, 25
as.numeric.image, 27
as.raster.image, 27
as_image *(image)*, 63
as_image(), *301*

base::jitter(), *318*, *320*
base::lapply(), *320*
base::sapply(), *320*

c.yolo_fewshot_model, 28
calibrate, 28
calibrate(), *145*
ccc, 29
clear_pliman_cache, 31
cm_to_dpi *(utils_dpi)*, 296
cm_to_dpi(), *296*, *297*
cm_to_pixels *(utils_dpi)*, 296
cm_to_pixels(), *296*, *297*
colorchecker_ref, 32
column_to_rownames *(utils_rows_cols)*, 324
contour, 32
contours, 33
conv_hull *(utils_polygon)*, 316
conv_hull_unified *(utils_polygon)*, 316
cpp_extract_raster, 34
custom_palette, 35
custom_palette(), *142*, *209*, *259*

dilate_cpp, 35
dist_transform, 36
distance *(utils_dpi)*, 296
distance(), *296*
dl_extract_phenotypes, 37
dpi *(utils_dpi)*, 296
dpi(), *296*, *297*
dpi_to_cm *(utils_dpi)*, 296
dpi_to_cm(), *296*, *297*
draw_circle *(utils_shapes)*, 326
draw_n_tagon *(utils_shapes)*, 326
draw_rectangle *(utils_shapes)*, 326
draw_square *(utils_shapes)*, 326
draw_trian_equi *(utils_shapes)*, 326
draw_trian_rect *(utils_shapes)*, 326

efourier, 38
efourier(), *17*, *33*, *40–44*
efourier_coefs, 40
efourier_error, 41
efourier_inv, 42
efourier_norm, 43
efourier_norm(), *40*, *43*
efourier_power, 44
efourier_shape, 45
ellipse, 47
entropy, 48
erode_cpp, 49
erode_external_cpp, 49

erosao_conservativa, 50
exactextractr::exact_extract(), *185*
extract_cpp *(mosaic_extract)*, 188

file.rename(), *361*, *373*
file_dir *(utils_file)*, 298
file_extension *(utils_file)*, 298
file_name *(utils_file)*, 298
fill_holes_cpp, 50
focus_stack, 51
fuse_laplacian_cpp, 56
fuse_multifocus *(focus_stack)*, 51

get_biplot *(utils_pca)*, 311
get_biplot(), *292*, *312*
get_card_colors, 56
get_free_ram, 59
get_free_ram_cpp, 59
get_measures *(utils_measures)*, 303
get_pliman_viewer, 60
get_pliman_viewer(), *24*, *28*, *64*, *106*, *127*, *132*, *145*, *158*, *164*, *185*, *199*, *209*, *215*, *228*, *259*, *261*, *297*, *315*, *335*
get_uuid, 60
get_wd_here *(utils_wd)*, 337
get_wd_here(), *337*, *338*
getwd(), *301*
ggplot_color, 61
gpa, 61
graphics::barplot(), *77*, *313*
graphics::lines(), *323*
graphics::plot(), *77*, *85*, *88*, *95*, *97*, *100*, *105*, *136*, *138*, *264*
graphics::plot.window(), *259*
graphics::points(), *323*
graphics::polygon(), *323*
graphics::text(), *304*
grDevices::colors(), *83*, *271*
grDevices::convertColor(), *295*

hist.image, 62

image, 63
image_align, 64
image_align(), *64*, *269*
image_align_stack *(focus_stack)*, 51
image_alpha, 65
image_augment, 66
image_augment(), *213*
image_autocrop *(utils_transform)*, 329
image_autocrop(), *335*
image_binary, 67
image_binary(), *16*, *139*, *227*, *229*, *236*, *276*, *310*, *336*
image_binary_dl, 70
image_blur *(utils_transform)*, 329
image_bwlabel, 75
image_bwlabel(), *223*
image_canny_edge, 75
image_classify_dl, 76
image_closing *(utils_transform)*, 329
image_color_labels, 78
image_combine, 78
image_combine(), *227*, *301*
image_contour_line, 79
image_contrast *(utils_transform)*, 329
image_correction, 80
image_correction_pick, 82
image_create, 83
image_crop *(utils_transform)*, 329
image_crop(), *269*
image_data, 84
image_depth_dl, 85
image_detect_dl, 86, *256*
image_detect_dl(), *37*, *89*, *131*, *264*, *376*
image_detect_sahi, 88
image_detect_sahi(), *37*
image_detect_world, 90
image_dilate *(utils_transform)*, 329
image_dimension *(utils_transform)*, 329
image_embed_dl, 92
image_erode *(utils_transform)*, 329
image_expand, 93
image_expand(), *134*
image_export *(utils_image)*, 300
image_features_dl, 94
image_fill_hull *(utils_transform)*, 329
image_filter *(utils_transform)*, 329
image_focus_stack *(focus_stack)*, 51
image_fuse_multifocus *(focus_stack)*, 51
image_generate_dl, 96
image_horizontal *(utils_transform)*, 329

image_hreflect *(utils_transform)*, 329
image_import *(utils_image)*, 300
image_index, 97
image_index(), *68*, *114*, *115*, *156*, *161*, *164*, *173*, *176*, *192*, *219*, *229*, *230*, *259*, *334*
image_inpaint_dl, 100
image_input *(utils_image)*, 300
image_label, 101
image_line_segment, 102
image_line_segment(), *262*
image_opening *(utils_transform)*, 329
image_palette *(palettes)*, 235
image_pliman *(utils_image)*, 300
image_pose_dl, 103
image_prepare, 106
image_prepare(), *24*, *25*, *164*
image_remove_bg *(image_remove_bg_dl)*, 107
image_remove_bg_dl, 107
image_remove_reflection, 112
image_remove_small *(utils_transform)*, 329
image_resize, 112
image_resize(), *330*
image_rotate *(utils_transform)*, 329
image_segment, 114
image_segment_dl, 117, *256*
image_segment_dl(), *37*, *129*, *264*, *376*
image_segment_iter *(image_segment)*, 114
image_segment_kmeans, 125
image_segment_manual, 127
image_segment_mask, 128
image_segment_sahi, 129
image_shp, 131
image_shp(), *24*, *163*, *227*, *228*, *249*
image_skeleton *(utils_transform)*, 329
image_skeleton(), *211*
image_skeleton_mat, 132
image_square, 134
image_square(), *213*, *215*
image_stardist_dl, 135
image_stitch, 136
image_superres_dl, 138
image_superres_dl(), *96*
image_thinning *(utils_transform)*, 329
image_thinning_guo_hall, 139
image_to_mat, 140
image_transpose, 141
image_trim *(utils_transform)*, 329
image_trim(), *335*, *336*
image_vertical *(utils_transform)*, 329
image_view, 141
image_view(), *353*
image_vreflect *(utils_transform)*, 329
image_watershed, 143
image_watershed(), *33*, *157*, *223*
is_image *(image)*, 63

landmarks, 144
landmarks(), *39*, *146*, *147*
landmarks_add, 145
landmarks_angle, 146
landmarks_dist, 147
landmarks_regradi, 148
landmarks_regradi(), *39*, *146*
leading_zeros, 149
line_on_halfplot, 150
list_cameras, 151
locator(), *315*

make_brush, 151
make_brush(), *128*, *129*, *336*
make_mask, 152
make_mask(), *128*
manipulate_files *(utils_file)*, 298
mapedit::editMap(), *142*, *143*, *208*, *209*
mapview::mapview(), *326*
max_list *(utils_stats)*, 329
mean.image, 153
mean_list *(utils_stats)*, 329
measure_disease, 153, *165*
measure_disease(), *160*, *162*, *163*, *165*, *277*, *278*
measure_disease_byl, 160
measure_disease_byl(), *158*, *277*
measure_disease_iter *(measure_disease)*, 153
measure_disease_shp, 163
measure_injury, 165
min_list *(utils_stats)*, 329
model_to_int8, 166
models_to_int8 *(model_to_int8)*, 166
mosaic_aggregate, 168
mosaic_aggregate(), *209*
mosaic_analyze, 169

mosaic_analyze(), *176*, *208*
mosaic_analyze_iter, 175
mosaic_chm, 177
mosaic_chm(), *179*
mosaic_chm_extract, 178
mosaic_chm_mask, 179
mosaic_chm_mask(), *172*
mosaic_classify, 180
mosaic_clip, 181
mosaic_crop, 182
mosaic_draw, 184
mosaic_epsg, 187
mosaic_export *(mosaic_input)*, 193
mosaic_extract, 188
mosaic_hist, 190
mosaic_index, 190
mosaic_index(), *171*
mosaic_index2, 192
mosaic_input, 193
mosaic_input(), *170*, *175*, *185*, *191*, *193*, *198*, *202–205*, *208*, *281*
mosaic_interpolate, 195
mosaic_lonlat2epsg, 196
mosaic_plot, 196
mosaic_plot_rgb, 197
mosaic_prepare, 198
mosaic_project, 199
mosaic_resample, 200
mosaic_rotate, 201
mosaic_segment, 202
mosaic_segment_pick, 203
mosaic_segment_pick(), *172*
mosaic_to_pliman, 204
mosaic_to_rgb, 205
mosaic_vectorize, 206
mosaic_view, 208
mosaic_view(), *171*, *176*, *182*, *184*, *281*

npixels *(utils_dpi)*, 296
npixels(), *296*

object_bbox, 210
object_contour, *267*
object_contour *(utils_objects)*, 308
object_contour(), *39*, *47*, *146*, *148*, *265*
object_coord *(utils_objects)*, 308
object_coord(), *311*
object_edge, 211
object_export, 212
object_export_shp, 214
object_haralick, 216
object_id *(utils_objects)*, 308
object_id(), *309*
object_isolate *(utils_objects)*, 308
object_label, 219
object_label(), *230*
object_map, 221
object_mark, 222
object_rgb, 223
object_scatter, 224
object_scatter(), *31*
object_split, 226
object_split(), *160*, *212*
object_split_shp, 227
object_split_shp(), *25*, *214*
object_to_color, 229
object_veins, 230
onnx_install, 232
onnx_quantize, 232
open_wd *(utils_wd)*, 337
open_wd(), *337*
open_wd_here *(utils_wd)*, 337
open_wd_here(), *337*, *338*
Ops.image, 233
order_quad_corners, 234
otsu, 234

palettes, 235
pca *(utils_pca)*, 311
pca(), *292*, *293*
pick_coords *(utils_pick)*, 313
pick_count *(utils_pick)*, 313
pick_palette *(utils_pick)*, 313
pick_palette(), *159*, *229*
pick_rgb *(utils_pick)*, 313
pick_rgb_area *(utils_pick)*, 313
pipe, 237
pixel_index, 238
pixels_to_cm *(utils_dpi)*, 296
pixels_to_cm(), *296*, *297*
pliman_available_models, 239
pliman_clear_sessions, 239
pliman_configure_dl, 240
pliman_download_model, 241
pliman_download_onnx, 242
pliman_gpu_info, 243
pliman_images, 243
pliman_indexes *(utils_indexes)*, 302

pliman_indexes(), *68*, *98*, *115*, *173*, *176*, *219*, *229*
pliman_indexes_eq *(utils_indexes)*, 302
pliman_indexes_hs *(utils_indexes)*, 302
pliman_indexes_ican_compute, 245
pliman_indexes_me *(utils_indexes)*, 302
pliman_indexes_me(), *191*, *193*, *202*
pliman_indexes_rgb *(utils_indexes)*, 302
pliman_indexes_rgb(), *191*, *193*, *202*
pliman_install_onnx *(onnx_install)*, 232
pliman_model_dir, 245
pliman_model_dir(), *166*, *167*, *377*, *381*, *384*
pliman_models_base_url, 246
pliman_onnx_dir, 246
pliman_onnx_lib_path, 247
pliman_viewer, 247
plot.anal_obj *(analyze_objects)*, 9
plot.anal_obj(), *9*
plot.anal_obj_ls *(analyze_objects)*, 9
plot.anal_obj_ls_minimal *(analyze_objects_minimal)*, 19
plot.anal_obj_minimal *(analyze_objects_minimal)*, 19
plot.image, 247
plot.image(), *353*
plot.image_index *(image_index)*, 97
plot.image_shp, 249
plot.pca *(utils_pca)*, 311
plot.root_analysis, 250
plot.yolo_fewshot_model, 251
plot.yolo_train, 253
plot.yolo_train(), *380–382*
plot_bbox, 256
plot_contour *(utils_polygon_plot)*, 322
plot_ellipse *(utils_polygon_plot)*, 322
plot_id, 257
plot_index, 258
plot_index(), *99*
plot_index_shp, 260
plot_line_segment, 262
plot_lw, 262
plot_mass *(utils_polygon_plot)*, 322
plot_measures *(utils_measures)*, 303
plot_polygon *(utils_polygon_plot)*, 322
plot_yolo_heatmap, 263
png(), *278*
poly_align *(utils_polygon)*, 316
poly_align(), *39*
poly_angles *(utils_polygon)*, 316
poly_apex_base_angle, 265
poly_apex_base_angle(), *17*
poly_area *(utils_polygon)*, 316
poly_caliper *(utils_polygon)*, 316
poly_centdist *(utils_polygon)*, 316
poly_centdist_mass *(utils_polygon)*, 316
poly_center *(utils_polygon)*, 316
poly_center(), *39*
poly_check *(utils_polygon)*, 316
poly_circularity *(utils_polygon)*, 316
poly_circularity_haralick *(utils_polygon)*, 316
poly_circularity_norm *(utils_polygon)*, 316
poly_close *(utils_polygon)*, 316
poly_convexity *(utils_polygon)*, 316
poly_distpts *(utils_polygon)*, 316
poly_eccentricity *(utils_polygon)*, 316
poly_elongation *(utils_polygon)*, 316
poly_flip_x *(utils_polygon)*, 316
poly_flip_y *(utils_polygon)*, 316
poly_is_closed *(utils_polygon)*, 316
poly_jitter *(utils_polygon)*, 316
poly_limits *(utils_polygon)*, 316
poly_lw *(utils_polygon)*, 316
poly_mass *(utils_polygon)*, 316
poly_mass(), *119*
poly_measures *(utils_polygon)*, 316
poly_pcv, 266
poly_perimeter *(utils_polygon)*, 316
poly_rotate *(utils_polygon)*, 316
poly_sample *(utils_polygon)*, 316
poly_sample_prop *(utils_polygon)*, 316
poly_slide *(utils_polygon)*, 316
poly_smooth *(utils_polygon)*, 316
poly_smooth(), *15*, *39*, *146*, *266*, *311*

poly_solidity *(utils_polygon)*, 316
poly_spline *(utils_polygon)*, 316
poly_unclose *(utils_polygon)*, 316
poly_width_at, 267
predict_binary_glm, 268
prepare_to_shp, 269
print.image, 269
print.pliman_video_frames, 270
print.root_analysis, 270
print.yolo_train, 271
print.yolo_train(), *381*
process_focus_stack *(focus_stack)*, 51

random_color, 271
remove_reflection_cpp, 272
remove_rownames *(utils_rows_cols)*, 324
rgb_image, 272
rgb_to_hsb *(utils_colorspace)*, 295
rgb_to_lab *(utils_colorspace)*, 295
rgb_to_srgb *(utils_colorspace)*, 295
rgbImage *(rgb_image)*, 272
root_analyze, 273
round_cols *(utils_rows_cols)*, 324
rownames_to_column *(utils_rows_cols)*, 324

sad, 277
sad(), *278*
sd_list *(utils_stats)*, 329
sentinel_to_tif, 279
separate_col, 279
set_pliman_viewer, 280
set_pliman_viewer(), *24*, *28*, *64*, *106*, *127*, *132*, *145*, *158*, *164*, *185*, *199*, *209*, *215*, *228*, *259*, *261*, *297*, *315*, *335*
set_wd_here *(utils_wd)*, 337
set_wd_here(), *337*
sf::sf, *182*
sf::st_simplify(), *172*
shapefile_build, 280
shapefile_build(), *195*, *257*
shapefile_crosses *(shapefile_operations)*, 287
shapefile_difference *(shapefile_operations)*, 287
shapefile_edit, 284
shapefile_export *(utils_shapefile)*, 325
shapefile_input *(utils_shapefile)*, 325
shapefile_input(), *183*, *195*, *284*
shapefile_interpolate, 285
shapefile_intersection *(shapefile_operations)*, 287
shapefile_measures, 286
shapefile_operations, 287
shapefile_outside *(shapefile_operations)*, 287
shapefile_overlaps *(shapefile_operations)*, 287
shapefile_plot, 288
shapefile_surface, 289
shapefile_touches *(shapefile_operations)*, 287
shapefile_transform, 290
shapefile_union *(shapefile_operations)*, 287
shapefile_view *(utils_shapefile)*, 325
shapefile_within *(shapefile_operations)*, 287
stats::prcomp(), *311*
Summary.image, 291
summary_index, 292

t.image *(image_transpose)*, 141
terra::datatype(), *194*
terra::hist(), *190*
terra::plot(), *197*, *209*, *289*
terra::plotRGB(), *197*
terra::project(), *199*
terra::rast(), *194*
terra::resample(), *200*
terra::SpatRaster, *182*
terra::SpatVector, *182*
terra::vect(), *325*, *326*
terra::writeRaster(), *194*
terra::writeVector(), *326*
text_embed_dl, 293
to_numeric, 294
to_raw *(to_numeric)*, 294

utils_colorspace, 295
utils_dpi, 296
utils_file, 298
utils_image, 300

utils_indexes, 302
utils_measures, 303
utils_neural, 305
utils_objects, 308
utils_pca, 311
utils_pick, 313
utils_polygon, 316
utils_polygon_plot, 322
utils_rows_cols, 324
utils_shapefile, 325
utils_shapes, 326
utils_stats, 329
utils_transform, 329
utils_wd, 337
uuid, 338

video_detect *(video_detect_dl)*, 339
video_detect_dl, 339
video_split, 350
video_to_frames *(video_split)*, 350
view, 352
view.image, 352

yolo_dataset_add_background, 353
yolo_dataset_audit, 355
yolo_dataset_aug
(yolo_dataset_augment), 356
yolo_dataset_augment, 356
yolo_dataset_autolabel, 358
yolo_dataset_autolabel(), *377*
yolo_dataset_balance, 359
yolo_dataset_classify, 360
yolo_dataset_cls
(yolo_dataset_classify), 360
yolo_dataset_combine
(yolo_dataset_merge), 372
yolo_dataset_convert, 363
yolo_dataset_export, 364
yolo_dataset_export(), *374*
yolo_dataset_filter, 369
yolo_dataset_from_ndjson, 370
yolo_dataset_health
(yolo_dataset_audit), 355
yolo_dataset_import_ndjson
(yolo_dataset_from_ndjson), 370
yolo_dataset_merge, 372
yolo_dataset_preview, 373
yolo_dataset_split, 375
yolo_dataset_split(), *374*
yolo_dataset_stats, 376
yolo_dataset_subset
(yolo_dataset_filter), 369
yolo_export, 376
yolo_fewshot_fit, 377
yolo_fewshot_fit(), *358*, *359*, *380*
yolo_fewshot_predict, 379
yolo_plot, 380
yolo_plot(), *256*, *381*, *382*
yolo_results, 381
yolo_results(), *256*, *380–382*
yolo_train, 383
yolo_train(), *256*, *360*, *381*, *382*