





Rastereasy: A Python package for an easy manipulation of remote sensing images

Thomas Corpetti¹, Pierrick Matelot², Augustin de la Brosse¹, and Candide Lissak²

¹ CNRS, UMR 6554 LETG, Univ. Rennes 2, Place du Recteur Henri Le Moal, 35043 Rennes Cedex, France ² Université de Rennes, Inserm, Irset, UMR_S 1085  Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

The analysis and processing of remote sensing images have many important applications in various fields such as environmental monitoring, urban planning, or even agriculture. However, handling large georeferenced raster datasets can be challenging due to their complexity and size.

rastereasy is a Python library for simple manipulation of georeferenced images (*.tif, *.jp2, *.shp, ...) [Ritter & Ruth (1997)](Mamatov et al., 2024). The goal is to simplify geospatial workflows by offering tools for reading and processing raster and vector files, resampling, cropping, reprojection, stacking, etc of raster images, easy visualizations such as color composites and spectral plots, use (train / apply) some classical Machine Learning algorithms on images ...

Compared to traditional RGB image manipulation, satellite images are highly specific due to their specific notions of spatial resolution, geographic extent, projection system, and they embed multiple spectral bands which prevents from an easy visualization. Dedicated software such as QGIS exists to handle these images, as well as specialized libraries for tasks like tiling, resampling, and reprojection. However, these tools require expertise in metadata management and geospatial systems, which can be a barrier for users unfamiliar with geographic data handling.

rastereasy is designed to simplify these processes, providing an easy-to-use interface for standard operations on multispectral and georeferenced images. It is particularly aimed at users who are experienced in data processing but not necessarily in geospatial analysis, while also streamlining workflows for geographers by leveraging rasterio and other geospatial libraries. It is particularly useful, among other things, for preparing sample data for deep neural networks.

The source code is available at <https://github.com/pythonraster/rastereasy> and a documentation <https://rastereasy.github.io/>.

Statement of need

Many existing remote sensing libraries, such as rasterio and gdal [Garrard (2016)](Gillies et al., 2013), provide powerful functionalities but often require a deep understanding of geospatial data structures. **rastereasy** abstracts these complexities by offering a high-level interface for:

- Band manipulation:** Extract, reorder, and remove spectral bands easily.
- Tiling and stitching:** Split large raster images into smaller tiles and reconstruct them.
- Harmonization:** Align rasters with different spatial resolutions and extents.

- 38 ▪ **Visualization tools:** Quick and interactive display of georeferenced images and spectral
39 signatures.
 - 40 ▪ **Basics of machine learning:** Clustering of images (Ikotun et al., 2023), adaptation of
41 spectral bands (domain adaptation) (Courty et al., 2016)
 - 42 ▪ **Fusion of classifications:** Fusion of mass function under the Dempster-Shafer framework
43 (Shafer, 1992)
 - 44 ▪ ...
- 45 The package is designed for researchers and practitioners in remote sensing who need efficient
46 tools for image preprocessing and analysis. It integrates seamlessly with rasterio and numpy,
47 making it compatible with existing geospatial workflows.

48 Example of use

49 In rastereasy, the core class of the library is **Geoimage**. This class allows users to manipulate
50 a satellite image as a numpy array while preserving essential geospatial information, such
51 as georeferencing, spectral bands, and projection system. This makes it easy to perform
52 calculations on the data while maintaining its spatial consistency.

53 For example, applying a simple transformation, extracting spectral bands, performing operations
54 or modifying an image is straightforward: Here's a quick example of what you can do with
55 rastereasy:

```
import rastereasy

# Load a georeferenced image
image = rastereasy.Geoimage("example.tif")

# Get image information
image.info()

# Print value of pixel [100,200]
print(image[100,200])

# Create a color composite
image.colorcomp(['4', '3', '2'])

# Resample and reproject
image_resampled = image.resampling(2)
image_reproject = image.reproject("EPSG:4326")

# This can also be done in inplace mode
image.resampling(2, inplace=True)

# Save the processed image
image.save("processed_image.tif")
```

56 all these functions have an inplace option to modify directly the images

57 Band Operations and feature computation

58 Users can easily manipulate spectral bands using high-level functions and compute indices
59 (Xue & Su, 2017):

```
import rastereasy
```

```
# Load a georeferenced image
```

```
img = rastereasy.Geoimage("example.tif")
```

```
# select red and near-infrared bands, positined in 4th and 8th positions
```

```
r=img.select_bands(4)
```

```
nir=img.select_bands(8)
```

```
# Compute NDVI (Normalized Difference Vegetation Index )
```

```
NDVI = (nir-r)/(nir+r)
```

```
# Apply a simple transformation: remove specific spectral bands
```

```
img = img.remove_bands([10, 8])
```

```
# Perform a reprojection
```

```
img_reproj = img.reproject(target_crs="EPSG:4326")
```

60 In one prefers to deal with explicit names for spectral bands, this is easily done by specifying
61 names

```
import rastereasy
```

```
# Load a satellite image and give specific names
```

```
name_bands = {"NIR":8,"G":3,"CO" : 1,"SWIR2":11,"B": 2,  
              "R":4,"RE1":5,"RE2":6,"RE3":7,"WA":9,  
              "SWIR1":10,"SWIR3":12}
```

```
img = Geoimage("satellite_image.tif",names=name_bands)
```

```
# select red and near-infrared bands
```

```
r=img.select_bands('R')
```

```
nir=img.select_bands('NIR')
```

```
# Compute NDVI (Normalized Difference Vegetation Index )
```

```
NDVI = (nir-r)/(nir+r)
```

```
# Apply a simple transformation: remove specific spectral bands
```

```
img_removed = img.remove_bands(["SWIR1", "NIR"])
```

```
# Perform a reprojection
```

```
img_reproj = img.reproject(target_crs="EPSG:4326")
```

```
# see also get_bands, switch_bands, ...
```

62 Image Tiling

63 Splitting a large image into smaller tiles with optional overlap (useful for data preparation):

```
from rastereasy import im2tiles
```

```
im2tiles("satellite_image.tif", "output_folder", nb_lig=512, nb_col=512, overlap=50)
```

64 Visualization

65 One can visualize histogrmass, color composites, spectra, ...

```
import rastereasy
image = rastereasy.Geoimage("example.tif")
# plotting spectra
image.plot_spectra()
# Making a color composition
image.colorcomp([4,3,2])
# Visualization of histograms
image.hist(superpose=True)
```

66 This gives the following images:

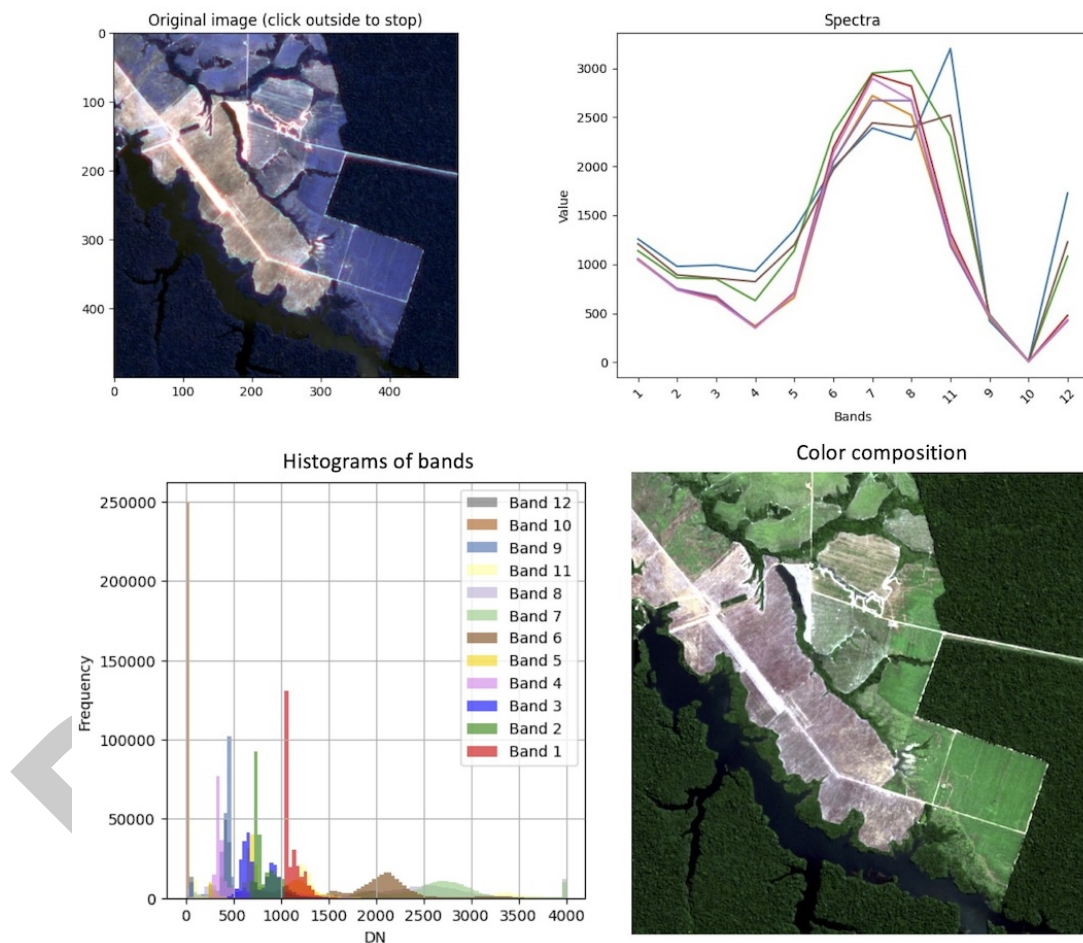


Figure 1: Examples of visualizations provided by rastereasy. Complete examples can be seen on the rastereasy package documentation : <https://rastereasy.github.io/>

67 Harmonization

68 Aligning images with different extents and resolutions:

```
from rastereasy import extract_common_areas
im1_common, im2_common = extract_common_areas(im1, im2)
```

69 Adapt the spectral values of to images with optimal transport

```
import rastereasy
```

```
image1 = rastereasy.Geoimage("im1.tif")
image2 = rastereasy.Geoimage("im2.tif")
# Change image 1 to adapt it to image 2

image1.adapt(image2, mapping='sinkhorn', inplace=True)
```

Performance and Scalability

rastereasy leverages numpy for efficient numerical operations and rasterio for optimized I/O operations, ensuring scalability for large datasets. Parallel processing capabilities are planned for future releases.

A complete documentation is available at <https://rastereasy.github.io/> with many notebooks for examples.

The source code is available here : <https://github.com/pythonraster/rastereasy>

Acknowledgments

This library is partly supported by the [ANR MONI-TREE](#) project (ANR-23-CE04-0017)

References

- Courty, N., Flamary, R., Tuia, D., & Corpetti, T. (2016). Optimal transport for data fusion in remote sensing. *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 3571–3574.
- Garrard, C. (2016). *Geoprocessing with python*. Simon; Schuster.
- Gillies, S., Ward, B., Petersen, A., & others. (2013). Rasterio: Geospatial raster i/o for python programmers. URL <https://Github.Com/Mapbox/Rasterio>.
- Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., & Heming, J. (2023). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622, 178–210.
- Mamatov, I., Galety, M. G., Alimov, R., Sriharsha, A., Rofoo, F. F. H., & Sunitha, G. (2024). Geospatial data storage and management. In *Ethics, machine learning, and python in geospatial analysis* (pp. 150–167). IGI Global.
- Ritter, N., & Ruth, M. (1997). The GeoTiff data interchange standard for raster geographic images. *International Journal of Remote Sensing*, 18(7), 1637–1647.
- Shafer, G. (1992). Dempster-shafer theory. *Encyclopedia of Artificial Intelligence*, 1, 330–331.
- Xue, J., & Su, B. (2017). Significant remote sensing vegetation indices: A review of developments and applications. *Journal of Sensors*, 2017(1), 1353691.