The Journal of Open Source Software

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

Thomas Corpetti [1¶], **Pierrick Matelot**[2], **Augustin de la Brosse**[1], and **Candide Lissak** [2]

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

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Working with remote sensing data often involves managing large, multi-band georeferenced rasters with varying spatial resolutions, extents, and coordinate reference systems (Mamatov et al., 2024). Established libraries such as `rasterio`, `Raster Forge`, `PODPAC`, `EarthPy` or `GDAL` (Garrard, 2016; Gillies et al., 2013; Oliveira et al., 2024; Ueckermann et al., 2020; Wasser et al., 2019) provide extensive functionality for these tasks, but they can be verbose and require a solid understanding of geospatial concepts such as projections, geotransforms, and metadata management. While efficient, many of these libraries are often specialized in a specific sub-task (e.g., visualization, array manipulation, or graphical interfaces) and may not be fully suited to users whose primary expertise lies outside GIS—such as data scientists, ecologists, agronomists, or climate researchers. This steep learning curve can slow down the development of operational workflows.

rastereasy is a Python library designed to bridge this gap by providing a high-level, human-readable interface for common geospatial raster and vector operations (e.g., *.tif*, .jp2, *.shp) (Mamatov et al., 2024; Ritter & Ruth, 1997). Built on well-established libraries including `rasterio`, `numpy`, `shapely`, `geopandas`, and `scikit-learn` (Gillies et al., 2013; Gillies & others, 2013; Harris et al., 2020; Jordahl et al., 2021; Kramer, 2016), it enables users to perform typical GIS tasks—such as resampling, cropping, reprojection, stacking, clipping rasters with shapefiles, or rasterizing vector layers—in just a few lines of code. Some basic Machine Learning functionalities (clustering, fusion) are also implemented.

By abstracting away much of the underlying technical complexity, **rastereasy** makes geospatial processing directly accessible within Python scripts. It is particularly suited for analysts and machine learning practitioners who need to integrate geospatial data handling into their workflows without deep GIS expertise, while also helping experienced geographers prototype more quickly. Beyond core raster operations, it includes utilities for harmonizing multi-source imagery, performing clustering and domain adaptation, and preparing datasets for downstream analysis.

With its current implementation, **rastereasy** provides a solid foundation for further development and integration into the Python geospatial ecosystem. 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 low-level functionalities for reading, writing, and processing geospatial raster data. However, these tools often require extensive knowledge of geospatial data structures, coordinate reference systems, and metadata handling, which can represent a steep learning curve for users whose primary expertise lies outside GIS.

rastereasy addresses this gap by offering a high-level, human-readable interface that abstracts away much of the underlying complexity while retaining the flexibility of the core libraries. Rather than replacing efficient lower-level libraries, **rastereasy** builds upon them, most notably `rasterio`, `shapely`, `geopandas` and abstracts away repetitive or technical boilerplate code. This design makes it possible to perform in a few lines of Python what would otherwise require many more lines in a raw `rasterio` or `GDAL` workflow. It provides streamlined access to common geospatial operations, including:

- **Band manipulation**: select, reorder, or remove spectral bands by index or by name.
- **Tiling and stitching**: split large rasters into smaller tiles for processing or machine learning workflows, and reconstruct them when needed.
- **Harmonization**: align rasters with different resolutions, projections, and extents, optionally adapting spectral values via domain adaptation (Courty et al., 2016).
- **Visualization tools**: quickly generate color composites, histograms, and spectral plots for georeferenced images.
- **Basics of machine learning**: clustering (Ikotun et al., 2023) and classification fusion using the Dempster–Shafer framework (Shafer, 1992).

rastereasy is intended for researchers and practitioners who need to integrate geospatial raster processing into broader data analysis or machine learning pipelines, without having to become GIS specialists. At the same time, it can also benefit geographers and remote sensing experts by offering a concise syntax for prototyping and testing ideas quickly.

Example of use

The core class of **rastereasy** is **GeoImage**, which wraps a raster as a `numpy` array while preserving all georeferencing metadata. This allows direct numerical operations while maintaining spatial consistency. For example users can easily manipulate spectral bands using high-level functions and compute indices (Xue & Su, 2017).

Example:

```python
import rastereasy

# Load an entire image
img = rastereasy.Geoimage("example.tif")

# Print metadata
img.info()

# Load a smal window of the image
deb_row=35
end_row=712
deb_col=40
end_col=450
area_pixel=((deb_row,end_row),(deb_col,end_col))
image=rastereasy.Geoimage("example",area=area_pixel)

# Print metadata
img.info()
```

```python
# Resample to 2m resolution
img_resampled = img.resample(2)

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

# Reproject to EPSG:4326
img_reprojected = img.reproject("EPSG:4326")

# Compute NDVI
r=img.select_bands(['4'])
nir=img.select_bands(['8'])
ndvi = (nir - r) / (nir + r)

# Change the name of the bands
ndvi.change_names({'ndvi':1})

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

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

```python
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)

# Apply a simple transformation: remove specific spectral bands
img_removed = img.remove_bands(["SWIR1", "NIR"])
```

all these functions have an `inplace` option to modify directly the images. These minimal examples illustrate how common geospatial tasks can be executed in just a few lines.

Visualization

One can visualize histogrmas, color composites, spectra, …

```python
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)
```

This gives the following images:

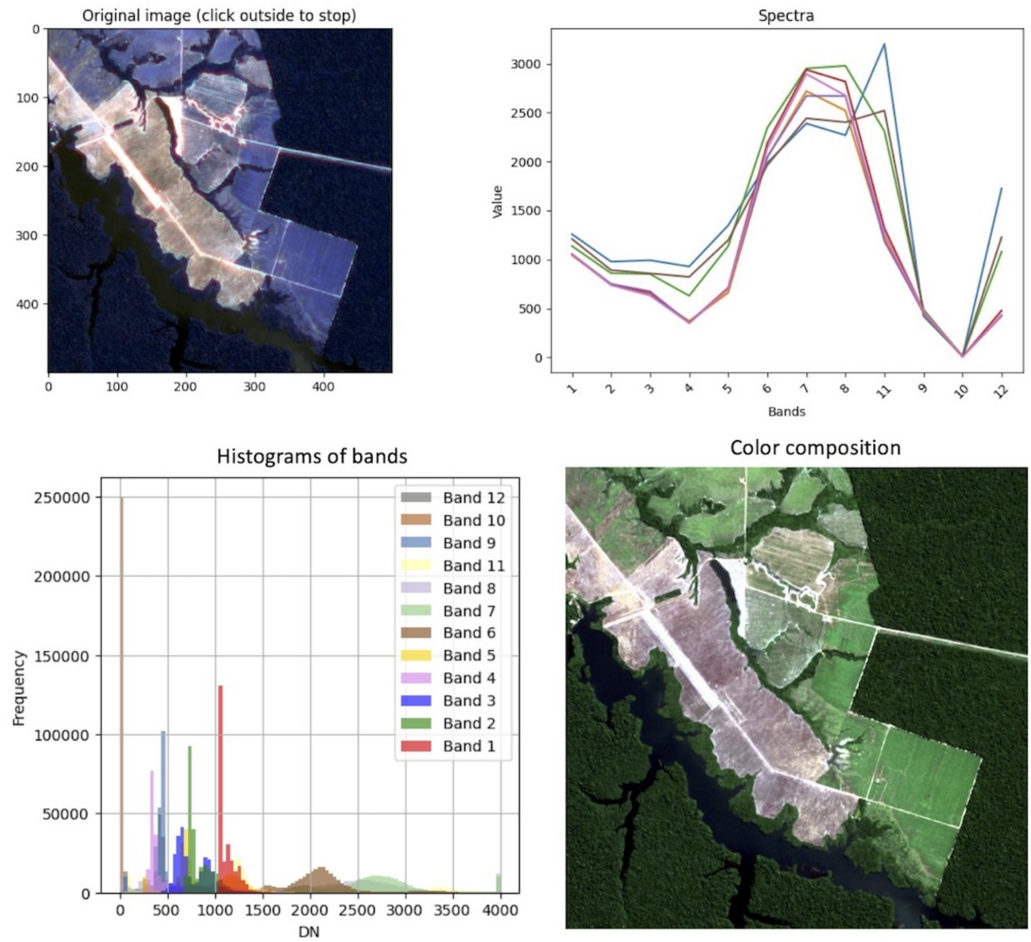

Figure 1: Examples of visualizations provided by rastereasy. Complete examples can be seen on the rastereasy package documentation : https://rastereasy.github.io/

Harmonization of bands

Here is an example of adapting the histogram of a source image to a target image (domain adaptation), which is useful, for instance, when applying a machine learning algorithm trained on the target domain to the source domain.

```python
import rastereasy

# read images
ims = rastereasy.Geoimage("source.tif")
imt = rastereasy.Geoimage("target.tif")

# plotting colorcomp and spectra
ims.colorcomp(extent='pixel', title='source data')
imt.colorcomp(extent='pixel', title='target data')
ims.hist(superpose=True,title='Histogram source data')
imt.hist(superpose=True,title='Histogram target data')

# Performing adaptation with earth mover distance
ims_to_imt = ims.adapt(imt,mapping='emd')

# plotting colorcomp and spectra of the adapted image
```

```python
ims_to_imt.colorcomp(extent='pixel', title='transported source data')
ims_to_imt.hist(superpose=True,title='Histogram transported source data')
```

Here are the generated images:

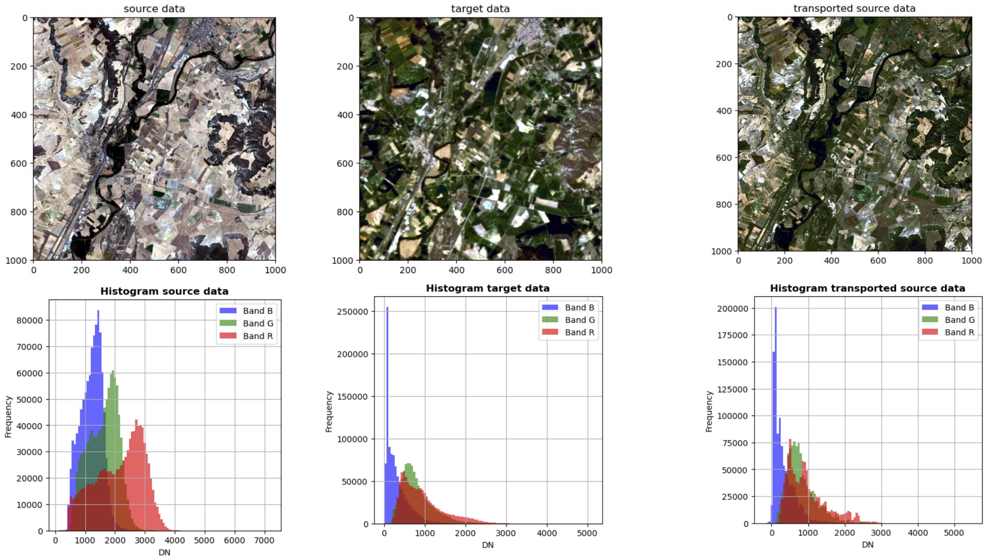

Figure 2: Examples of band harmonization with rastereasy

Filters

Most classical filters (gaussian, laplacian, sobel, median) as well as user-defined generic filters can be performed. Here are some examples.

```python
import rastereasy

name_im='image.tif'
image=rastereasy.Geoimage(name_im)

# Gaussian filter
image_filtered_gaussian = image.filter("gaussian",sigma=8)

# Generic filter
import numpy as np
blur_kernel = np.ones((9, 9)) / (81)
image_filtered_generic = image.filter(method="generic", kernel=blur_kernel)
```

For additional functionalities such as spectral plots, rasterization, harmonization, clustering, or classification fusion, see the [rastereasy documentation](#).

Performance and Scalability

`rastereasy` is designed as a high-level wrapper around efficient geospatial libraries such as `rasterio`, `numpy`, and `geopandas`. In its current implementation, the default behavior is either to load full rasters into memory and it also supports windowed reading via the underlying `rasterio` API, allowing users to read and process only subsets of rasters without loading entire files into memory.

While this is convenient for small to medium-sized datasets, it can become a limiting factor when working with very large georeferenced images (e.g., > 10 GB).

Currently, most operations are single-threaded and executed in memory; planned enhancements include lazy loading (processing data on demand) and parallel processing (e.g., for tiling, reprojection, or large mosaics) to improve scalability.

Documentation and community guidelines

Full documentation, including numerous Jupyter Notebook tutorials, is available at: https://rastereasy.github.io/

Contribution guidelines and issue reporting instructions are provided in the repository to encourage community-driven development. We welcome contributions of all types, including:

- Bug reports and feature requests: please use the GitHub Issues section, providing clear descriptions, example data, and reproducible steps when possible
- Code contributions: fork the repository, create a feature branch, and submit a pull request with detailed explanations and tests for new functionality
- Documentation improvements: suggestions to improve tutorials, add examples, or clarify function descriptions are highly valued
- Community support: engage in discussions, answer questions from other users, and help maintain a collaborative and respectful environment

All contributors are expected to adhere to the Contributor Covenant Code of Conduct, version 1.4, ensuring a welcoming and inclusive community.

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. https://doi.org/10.1109/IGARSS.2016.7729925

Garrard, C. (2016). *Geoprocessing with python*. Simon; Schuster.

Gillies, S., & others. (2013). The shapely user manual. *URL Https://Pypi. Org/Project/Shapely*.

Gillies, S., Ward, B., Petersen, A., & others. (2013). Rasterio: Geospatial raster i/o for python programmers. *URL Https://Github. Com/Mapbox/Rasterio*.

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

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. https://doi.org/10.1016/j.ins.2022.11.139

Jordahl, K., Van den Bossche, J., Wasserman, J., McBride, J., Fleischmann, M., Gerard, J., Tratner, J., Perry, M., Farmer, C., Hjelle, G. A., & others. (2021). Geopandas/geopandas: v0. 7.0. *Zenodo*. https://doi.org/10.5281/zenodo.3669853

Kramer, O. (2016). Scikit-learn. In *Machine learning for evolution strategies* (pp. 45–53). Springer.

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 Scientific Publishing. https://doi.org/10.4018/979-8-3693-6381-2.ch007

Oliveira, A., Fachada, N., & Matos-Carvalho, J. P. (2024). Raster forge: Interactive raster manipulation library and GUI for python. *Software Impacts*, *20*, 100657. https://doi.org/10.1016/j.simpa.2024.100657

Ritter, N., & Ruth, M. (1997). The GeoTiff data interchange standard for raster geographic images. *International Journal of Remote Sensing*, *18*(7), 1637–1647. https://doi.org/10.1080/014311697218340

Shafer, G. (1992). Dempster-shafer theory. *Encyclopedia of Artificial Intelligence*, *1*, 330–331.

Ueckermann, M. P., Bieszczad, J., Entekhabi, D., Shapiro, M. L., Callendar, D. R., Sullivan, D., & Milloy, J. (2020). PODPAC: Open-source python software for enabling harmonized, plug-and-play processing of disparate earth observation data sets and seamless transition onto the serverless cloud by earth scientists. *Earth Science Informatics*, *13*(4), 1507–1521. https://doi.org/10.1007/s12145-020-00506-0

Wasser, L., Joseph, M., McGlinchy, J., Palomino, J., Korinek, N., Holdgraf, C., & Head, T. (2019). EarthPy: A python package that makes it easier to explore and plot raster and vector data using open source python tools. *Journal of Open Source Software*, *4*(43), 1886. https://doi.org/10.21105/joss.01886

Xue, J., & Su, B. (2017). Significant remote sensing vegetation indices: A review of developments and applications. *Journal of Sensors*, *2017*(1), 1353691. https://doi.org/10.1155/2017/1353691