INTEROPERABILITY AND REPRODUCIBILITY CHALLENGES OF BIG EO DATA: LESSONS FROM THE TRENCHES

Gilberto Camara, Rolf Simoes, Felipe Souza, Karine Ferreira, Gilberto Queiroz, Pedro R. Andrade

National Institute for Space Research (INPE), Brazil

ABSTRACT

This paper examines challenges to achieve interoperability and reproducibility in big Earth observation (EO) data applications. We consider the problems involved in accessing image collections in cloud providers, sharing training data, defining EO data cubes, designing a robust software architecture, and describing and implementing machine learning methods. The paper argues that, due to the inherent complexity of spatiotemporal data, achieving full interoperability is unlikely. Instead, the authors propose an increased emphasis on collaboration between the various on-going efforts for developing open source EO data analytical software.

Index Terms— Big EO analytics, EO data cubes, interoperability, software reuse.

1. INTRODUCTION

The current standard approach for big Earth observation (EO) data analytics is to use cloud computing services. These include commercial providers like Google Earth Engine (GEE), Amazon Web Services (AWS), and Microsoft Planetary Computer (MPC). There are also public services such as Digital Earth Africa (DEAfrica), FAO's SEPAL, the Brazil Data Cube (BDC), and the Copernicus Data Space Ecosystem [1]. Since most data available in these providers comes from the same sources, one might presume that it would be straightforward to design multi-provider applications for EO data analytics. Users would be able to run the same code in different services; they would reuse software developed in one platform in another with minimal effort. Such is not the case today. Instead, there are significant differences between those services, which create a customer lock-in effect. Thus, one might ask: *can the barriers to interoperability and reproducibility in big EO data analysis be removed?*

To answer this question, we consider the main issues faced when designing applications that run in multiple EO cloud computing services. Our arguments are derived from the experience of developing the **R** open source package `sits`[2]. This package provides an end-to-end environment for big EO data analytics based on a user-focused API. It supports land classification with machine learning and deep learning methods using a time-first, space-latter approach. In what follows, we present the main design choices faced by the authors, considering inevitable compromises between options available. We hope that by sharing our experience, other developers may find helpful insights to support their decision-making process.

2. CHALLENGES FOR BIG EO ANALYTICS SOFTWARE DESIGN

2.1. How to access cloud collections?

Most EO cloud providers offer an endpoint that employs the SpatioTemporal Asset Catalog (STAC) specification to access their collections. STAC adoption has been a major advance for big EO data applications. There are working R and Python APIs that implement STAC, available in `https://stacspec.org`. Ideally, all providers would implement STAC consistently, which should enable interoperability of data access methods. In reality, STAC implementations have major variations.

Cloud services use different criteria for geographical area selection and access control. Collection and band names are also not standardised. Such lack of conformance is a challenge for software developers. Each new provider requires a customisation effort. Achieving interoperability when accessing cloud collections requires a better alignment between providers and software developers than the current situation.

doi:10.2760/46796

2.2. How can training data be shared?

Selecting good training samples for machine learning classification of satellite images is critical to achieve accurate results. Experience with machine learning methods has shown that the number and quality of training samples are crucial factors in obtaining good outcomes. Nevertheless, there are currently no standards nor community-driven formats to share training samples for big EO data.

Defining standards for sharing training data is further complicated by the variety of associated formats. The most common data structure is a set of labels associated with points or polygons in a single image. However, advanced machine learning methods require other formats. Deep-learning methods in 2D use labels associated to image chips, while image time series analysis use points or polygons associated to EO data cubes. Given the lack of well-accepted protocols, software developers have to spend much effort to design APIs capable of reading different training data formats. Thus, there is a need for consistent efforts by standards bodies or by communities to mitigate this problem.

2.3. What is an EO data cube?

Despite the term *data cube* being widely used by the EO community, it is not simple to provide a all-inclusive definition. An influential proposal has been made by Appel and Pebesma [3]. They state that a data cube is *"a four-dimensional array with dimensions x (longitude or easting), y (latitude or northing), time, and bands"*. They also propose that *"spatial dimensions refer to a single spatial reference system (SRS), while cells of a data cube have a constant spatial size with regard to the cube's SRS and have constant temporal duration"*. This definition is useful, but needs to be extended for EO data cubes that cover large areas. In these cases, it is natural to use the tile-based organisation of collections such as Landsat and Sentinel-2, so that the resulting data cubes will use multiple SRSs.

A second point concerns Appel and Pebesma's interpretation of *dimension*, which refers to spatial, temporal, and attribute components of data cubes. This choice allows space, time, and attribute-based operations to be generalised. Consider operations that combine one of the data dimensions to produce a result. To average each pixel in a image time series, one uses an API call such as `reduce dimension[time, mean]`. Similar `reduce dimension` calls can be applied to space and attribute operations. The resulting API is more concise than alternatives that require explicit mention of temporal, spatial, and attribute components.

In contrast, there is a line of thought in GIScience that considers space and time to be quite different, since time is directional while space is not [4, 5]. In this view, temporal interval operators such as `before` and `during` [6] are semantically different from spatial operators like `touches` and `crosses` [7]. Since we share this perspective, the authors opted for keeping space, time, and attribute operations separate in `sits`.

Both views on what is an EO data cube and what would be a matching API are valid and justified. At heart, there is a compromise between flexibility and strictness. This fact shows how hard it is to reach a universally-valid EO data cube definition.

2.4. Client-based or a server-based architecture?

A critical design decision involves how to distribute software between client and server machines. Client-based software uses an API on a single machine. This approach, adopted by `sits` and by Open Data Cube [8], builds an end-to-end environment wherein a single script can execute complex tasks. The downside is that users are constrained by the capacity of their computing environment. Optimal big data performance requires a large virtual machine, which may incur in significant processing costs.

An alternative is a server-based API, as in Google Earth Engine (GEE) [9]. GEE provides a collection of proprietary functions, optimised for pixel-based parallel processing. Tasks that align well with this API are executed efficiently. The drawback lies in its limited extensibility. New developments are hindered by the internal GEE pixel-based architecture. In GEE, it is hard to run deep learning algorithms that use image time series. The GEE case shows that it is not easy to build a server-based API that works equally well for different kinds of EO data analytics.

A third approach involves specifying an open-source API that can be implemented by different services, as in openEO [10]. The openEO initiative aims to allow users to choose between cloud services that implement its specification. This an ambitious solution for reproducibility across different EO services. To achieve its goals, it needs to solve issues such as how to deal with training samples

and deep learning algorithms in a language-independent way. While it is clear that openEO will have a positive impact on the area, its merits will be fully assessed when reference implementations of a complete API for big EO data are available.

These alternatives are not mutually exclusive. For example, as part of the EU-funded Open Earth Monitor project (https://earthmonitor.org/), the authors are implementing a back-end to the openEO specification using `sits`. This is an example of how open source initiatives can cooperate.

2.5. Implementing data cubes: open-ended scripting or structured APIs?

Given an abstract description of an EO data cube, developers need to consider how to implement it. The solution chosen by the Open Data Cube [8] and other efforts such as Pangeo is to use the Python data structure `xarray` to represent data cubes. `Xarrays` contain multidimensional numeric array data and metadata. When data fits in main memory, `xarrays` work well; scripts can use metadata to perform calculations, produce results, and view maps in a coordinate-aware fashion. Given its qualities and the popularity of Python, there has been significant progress in `xarray`-based scripts for big EO data analysis [8].

Despite their advantages, using `xarrays` as a foundation for big EO analytics is not without challenges. One significant issue arises when working with big areas —such as the Brazilian Amazon— that exceed the main memory capacity. A second problem relates is combining `xarrays` with `Dask` for parallel processing, which requires advanced software skills. Arguably, most users find it easier when parallel processing of large data runs under the hood, as in the case of GEE, openEO, and `sits`.

Further questions arise in terms of reuse. Since `xarray` is a data structure, related scripts tend to perform low-level actions that could have been encapsulated in a well-designed API. As a result, there are multiple implementations of the same base functions. Remote sensing experts without strong programming skills may find it difficult to understand such scripts. As a result, they might not be able to reuse and reproduce them easily. What works well for good programmers may be a limiting factor for capacity development.

When designing `sits`, we aimed at an audience of remote sensing experts without strong programming skills. For this reason, we opted for a high-level API that hides data structures and algorithm details. Each task of a typical land use classification workflow is mapped to an API function. The aim is to allow simple scripts to achieve complex tasks. The drawback of this approach is that is difficult for the community to extend the package. In practice, the tight API integration in `sits` compels all new developments to be done by the core programming team.

Comparing the two approaches, there is no unique best solution. Both open-ended scripting as is done by Open Data Cube and high-level APIs such as the ones used in GEE, openEO and `sits` are possible ways to process EO data. Each approach has drawbacks and advantages that need to be considered by software designers.

2.6. How to describe and share machine learning algorithms?

One key area of innovation in EO analytics is the development of machine learning (ML) algorithms. This is an active field with many new developments. However, there is limited availability of new AI methods in the current generation of EO analytic services and tools. Recent surveys of AI methods published in conferences show that only 6% of the papers include code [11]. The first author surveyed 1,300 papers presented in IGARSS 2019, and found out that only 35 (2.5%) included code. Furthermore, many papers are published without the full details that would allow reproducibility [12]. Therefore, despite alleged advances in EO deep learning methods, most claims made in the literature are unverifiable [13].

Consider also the split between EO researchers and EO platform developers. Even when shared, most new algorithms are developed in toy environments with limited data sets. Current platforms are not interoperable; algorithms developed using the `TensorFlow` and `Torch` libraries are not compatible. In `sits`, we spent considerable effort adapting published algorithms to a big EO data environment. In practice, the so-called last mile from research to operations is a long one.

A further problem concerns integration of deep learning algorithms in an existing API. This is an open problem with many possible solutions. The `sits`API has a single function for all machine learning models. This function takes two mandatory parameters: the training

doi:10.2760/46796

data and the ML algorithm. The result is a trained model that contains all information required for prediction. Training and classification are independent. This implementation is a compromise between user-friendly interfaces and internal software complexity. While users get the benefit of having simple API for model training, programmers need good skills to include new algorithms. In this respect, the open-ended approach used by `xarray`-based environment is more accessible to community contributions than the `sits` design. This is another example of the tradeoff between usability and extensibility that is a feature of big EO data analytics.

3. CONCLUSION: HOW FAR TO INTEROPERABILITY AND REPRODUCIBILITY?

The previous sections presented challenges that developers of big EO analytics software face, and decisions taken in `sits` to answer them. Designs that work well for 2D images are in general not compatible with the needs of time series analysis. Meeting the full abstract requirements for space-time data analytics while building operational software is hard. Thus, all current solutions have advantages and drawbacks.

Achieving broad interoperability in EO analytics would require an all-embracing definition of EO data cubes. It would also need an abstract API that handles training data and machine learning methods in a language independent way. Conflation between specification and implementation should be avoided. Cloud providers would implement access protocols in consistent ways. Users could run the same scripts in both client-based and server-based architectures. As discussed in the paper, none of these conditions holds currently.

There are on-going efforts by OGC to promote interoperability in EO applications. Differing responses to OGC testbeds show that it is hard to achieve consensus on an generic model that can be implemented is a consistent way. There is presently no agreement on an abstract API for data cubes, since current API are bottom-up and language-dependent. Thus, we are far from achieving interoperability in big EO data analytics.

Given the challenges to interoperability, arguably the community should focus on reuse and collaboration. Once the EO community acknowledges that unified standards are not achievable, it can focus on working together. If that happens, collective solutions can provide support to a broad range of applications that use big EO data.

Acknowledgments

The authors would like to thank Edzer Pebesma for many positive discussions on the issues covered in the paper. They also acknowledge the support of Microsoft Planetary Computer, the Amazon Fund (contract 17.2.0536.1) and EU-funded Open-Earth-Monitor Cyberinfrastructure project (grant No. 101059548).

Bibliography

[1] Vitor C. F. Gomes et al. “An Overview of Platforms for Big Earth Observation Data Management and Analysis”. In: *Remote Sensing* 12.8 (2020), p. 1253.

[2] Rolf Simoes et al. “Satellite Image Time Series Analysis for Big Earth Observation Data”. In: *Remote Sensing* 13.13 (2021), p. 2428.

[3] Marius Appel and Edzer Pebesma. “On-Demand Processing of Data Cubes from Satellite Image Collections with the Gdalcubes Library”. In: *Data* 4.3 (2019).

[4] Antony Galton. “Time Flies but Space Does Not: Limits to the Spatialisation of Time”. In: *Journal of Pragmatics* 43.3 (2011), pp. 695–703.

[5] Karine Ferreira et al. “An Algebra for Spatiotemporal Data: From Observations to Events”. In: *Transactions in GIS* 18.2 (2014), pp. 253–269.

[6] James F. Allen. “Maintaining Knowledge about Temporal Intervals”. In: *Communications of the ACM* 26.11 (1983), pp. 832–843. ISSN: 0001-0782, 1557-7317.

[7] Max Egenhofer and Robert Franzosa. “Point-Set Topological Spatial Relations”. In: *Int Journal of Geographical Information Systems* 5.2 (1991), pp. 161–174.

[8] Adam Lewis et al. “The Australian Geoscience Data Cube — Foundations and Lessons Learned”. In: *Remote Sensing of Environment* 202 (2017), pp. 276–292.

[9] Noel Gorelick et al. “Google Earth Engine: Planetary-scale Geospatial Analysis for Everyone”. In: *Remote Sensing of Environment* 202 (2017), pp. 18–27.

[10] Matthias Schramm et al. “The openEO API: Harmonising the Use of Earth Observation Cloud Services Using Virtual Data Cube Functionalities”. In: *Remote Sensing* 13.6 (2021), p. 1125.

[11] Matthew Hutson. “Artificial Intelligence Faces Reproducibility Crisis”. In: *Science* 359.6377 (2018).

[12] Joelle Pineau et al. “Improving Reproducibility in Machine Learning Research”. In: *Journal of Machine Learning Research* 22 (2021), p. 20.

[13] Daniel Nüst and Edzer Pebesma. “Practical Reproducibility in Geography and Geosciences”. In: *Annals of the AAG* 111.5 (2021), pp. 1300–1310.