GEE SAR Fetcher

Release 0.3.8

Thomas Di Martino

Jun 20, 2022

TABLE OF CONTENTS

This documentation describes the different functionalities available in the **GEE SAR Fetcher** project. This tool is useful to download large areas over long period of time by successively tiling the input parameters along spatial and temporal axis.

An easy-to-use Python library to download SAR GRD imagery from Google Earth Engine.

CHAPTER

ONE

INTRODUCTION

Access Google's multi-petabytes of SAR Imagery data from your python code with *no dimension restraint*. Simply supply coordinates, a time interval and obtain a stack of Sentinel-1 preprocessed PolSAR images. This enables quick data analysis of GRD images to get better insights of the temporal dimension in SAR data without having to bother with essential but potentially time-consuming steps such as coregistration or calibration.

Compatible with python 3.

CHAPTER

TWO

ADDITION WITH VERSION 0.3.3

The development of version 0.3.3 added two new functionalities to the library:

- the ability to select the orbit number of the downloaded temporal stack. It can directly be supplied by the user, or the said user can supply a keyword “min” or “max” and the adequate orbit number will automatically be extracted, given the orbit type and coordinates.
- the ability to retrieve metadata from the downloaded stack, one per temporal image.

CHAPTER

THREE

USAGE

The library allows the user to retrieve GEE Sar data eiter over a rectangular area or over a single coordinates tuple.

3.1 Fetch data over an area

The function to call in order to retrieve data over an area is the `fetch` function:

```
from geesarfetcher import fetch
from datetime import datetime, timedelta

d = fetch(
   top_left=[-116.17556985040491, 60.527371254744246],
   bottom_right=[-116.1364310564596, 60.54425859382555],
   start_date=datetime(year=2021, month=5, day=20) - timedelta(days=365),
   end_date=datetime(year=2021, month=5, day=20),
   ascending=False,
   scale=10,
   orbit_number="max",
   verbose=2
) # returns a dictionnary with access to the data through the 'stack' keyword, to its␣
↪timestamps through the 'timestamps' keyword, to pixels' coordinates with 'coordinates'␣
↪key and to metadata with the 'metadata' key.
```

It returns a `dict` object with 4 keys:

`"stacks"`
: 4-D array containing db intensity measure (*numpy.ndarray*), `(height, width, pol_count, time_series_length)`

`"coordinates"`
: 3-D array containg coordinates where `[:,:,0]` provides access to latitude and `[:,:,1]` provides access to longitude, (*numpy.ndarray*), `(height, width, 2)`

`"timestamps"`
: list of acquisition timestamps of size (time_series_length,) (*list of str*)

`"metadata"`
: Dictionnary describing data for each axis of the stack and the coordinates

3.2 Fetch data over an area and save as a GeoTIFF

The function to call in order to retrieve & save data over an area is the `fetch_and_save` function:

```
from geesarfetcher import fetch_and_save
from datetime import datetime, timedelta

fetch_and_save(
   save_dir = ".",
   top_left = [-104.77431630331856, 41.729889598264826],
   bottom_right = [-104.65140675742012, 41.81515375846025],
   start_date = datetime(2019, 6, 1),
   end_date = datetime(2019, 6, 3),
   ascending = False,
   scale = 10,
   orbit_number="max",
   n_jobs = 8,
   verbose = 2
) # saves each timestep of the multitemporal SAR image in the directory specified by the␣
↪keyword 'save_dir'
```

It saves each timestep as a GeoTIFF file using the following naming pattern: 't_{date}_{subcoordinate_index}.tiff'. Metadata are saved as .json files following the same naming convention (i.e. 't_{date}_{subcoordinate_index}.json').

Subcoordinate indexes are generated when splitting the initial whole area into smaller areas. Each of the subregion is then saved as a separate GeoTIFF, for less memory consumption. Every GeoTIFF contains in its first band VV values and in its second band VH values.

3.3 Fetch data for a single point

To fetch over a single point, the process is similar to the difference that we use another function, called `fetch_point` and only provide a single coordinates tuple rather than either two or 5 tuples for the area query.

```
from geesarfetcher import fetch_point
from datetime import date, timedelta

d = fetch_point(
   coords = [-104.88572453696113, 41.884778748257574],
   start_date = date.today()-timedelta(days=15),
   end_date = date.today(),
   ascending = False,
   scale = 10,
   orbit_number="max",
   verbose = 2
)
```

For data consistency, the returned object is of the same nature as with the `fetch` method, i.e a `dict` with 4 keys:

`"stacks"`
: 4-D array containing db intensity measure (*numpy.ndarray*), `(1, 1, pol_count, time_series_length)`

`"coordinates"`
: 3-D array containg coordinates where `[:,:,0]` provides access to latitude and `[:,:,1]` provides access

to longitude, (*numpy.ndarray*), `(1, 1, 2)`

`"timestamps"`
list of acquisition timestamps of size (time_series_length,) (*list of str*)

`"metadata"`
Dictionnary describing data for each axis of the stack and the coordinates

CHAPTER

FOUR

INSTALLATION

Access to Google Earth Engine is conditioned by the obtention of a GEE account. Once created, you can install the **geesarfetcher** API and register an identifying token for your Python working environment using the following commands:

```
pip install geesarfetcher
earthengine authenticate
```

CHAPTER

FIVE

CONTRIBUTING

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change. Please make sure to update tests as appropriate.

CHAPTER

SIX

LICENSE

MIT

In this page, you will be able to consult the code documentation of the main modules of the library

CHAPTER

SEVEN

GEESARFETCHER

The main module of interest, where you will find the `fetch` functions, for different areas.

7.1 geesarfetcher

geesarfetcher

geesarfetcher.**fetch**(*top_left=None*, *bottom_right=None*, *coords=None*, *start_date: datetime = datetime.date(2021, 6, 20)*, *end_date: datetime = datetime.date(2022, 6, 20)*, *ascending: bool = True*, *orbit_number: Optional[object] = None*, *scale: int = 20*, *n_jobs: int = 8*, *verbose: int = 0*)

Fetches SAR data in the form of a dictionnary with image data as well as timestamps

Parameters

- **top_left** (*tuple of float, optional*) – Top left coordinates (lon, lat) of the Region
- **bottom_right** (*tuple of float, optional*) – Bottom right coordinates (lon, lat) of the Region
- **coords** (*tuple of tuple of float or list of list of float, optional*) – If *top_left* and *bottom_right* are not specified, we expect *coords* to be a list (resp. tuple) of the form `[top_left, bottom_right]` (resp. `(top_left, bottom_right)`)
- **start_date** (*datetime.datetime, optional*) – First date of the time interval
- **end_date** (*datetime.datetime, optional*) – Last date of the time interval
- **ascending** (*boolean, optional*) – The trajectory to use when selecting data
- **orbit_number** (*int or str, optional*) – The orbit number to restrict the download to. If provided with an integer, the S1 temporal stack is filtered using the provided orbit number. If provided with a string value, we expect one of these keywords:
  - "max" for the orbit number with the highest number of image in the stack
  - "min" for the orbit number with the smallest number of image in the stack

  If `None`, then no filter over the orbit number is applied.
- **scale** (*int, optional*) – Scale parameters of the getRegion() function. Defaulting at `20`, change it to change the scale of the final data points. The highest, the lower the spatial resolution. Should be at least `10`.
- **n_jobs** (*int, optional*) – Set the parallelisation factor (number of threads) for the GEE data access process. Set to 1 if no parallelisation required.

- **verbose** (*int, optional*) – Verbosity mode (0: No info, 1: Info, 2: Detailed info, with added timestamp)

Returns

Dictionnary with four keys:

`"stacks"`
4-D array containing db intensity measure (*numpy.ndarray*), `(height, width, pol_count, time_series_length)`

`"coordinates"`
3-D array containg coordinates where `[:,:,0]` provides access to latitude and `[:,:,1]` provides access to longitude, (*numpy.ndarray*), `(height, width, 2)`

`"timestamps"`
list of acquisition timestamps of size (time_series_length,) (*list of str*)

`"metadata"`
Dictionnary describing data for each axis of the stack and the coordinates as well as the properties (orbit number, slice, acquisition time…) of each image of the temporal stack

Return type
dict

geesarfetcher.**fetch_and_save**(*save_dir: Optional[str] = None*, *top_left=None*, *bottom_right=None*, *coords=None*, *start_date: datetime = datetime.date(2021, 6, 20)*, *end_date: datetime = datetime.date(2022, 6, 20)*, *ascending: bool = True*, *orbit_number: Optional[object] = None*, *scale: int = 20*, *n_jobs: int = 8*, *verbose: int = 0*)

Fetches SAR data by looping other each timestep and each generated subregion and saves extracted images as GeoTIFF in the supplied *save_dir* folder

Parameters

- **save_dir** (*str*) – Path toward an *existing* directory where to save the images. If non-existing, an Exception is raised.
- **top_left** (*tuple of float, optional*) – Top left coordinates (lon, lat) of the Region
- **bottom_right** (*tuple of float, optional*) – Bottom right coordinates (lon, lat) of the Region
- **coords** (*tuple of tuple of float or list of list of float, optional*) – If *top_left* and *bottom_right* are not specified, we expect *coords* to be a list (resp. tuple) of the form `[top_left, bottom_right]` (resp. `(top_left, bottom_right)`)
- **start_date** (*datetime.datetime, optional*) – First date of the time interval
- **end_date** (*datetime.datetime, optional*) – Last date of the time interval
- **ascending** (*boolean, optional*) – The trajectory to use when selecting data
- **orbit_number** (*int or str, optional*) – The orbit number to restrict the download to. If provided with an integer, the S1 temporal stack is filtered using the provided orbit number. If provided with a string value, we expect one of these keywords:
  - "max" for the orbit number with the highest number of image in the stack
  - "min" for the orbit number with the smallest number of image in the stack

  If `None`, then no filter over the orbit number is applied.

- **scale** (*int, optional*) – Scale parameters of the getRegion() function. Defaulting at 20, change it to change the scale of the final data points. The highest, the lower the spatial resolution. Should be at least 10.
- **n_jobs** (*int, optional*) – Set the parallelisation factor (number of threads) for the GEE data access process. Set to 1 if no parallelisation required.
- **verbose** (*int, optional*) – Verbosity mode (0: No info, 1: Info, 2: Detailed info, with added timestamp)

Returns

Dictionnary with four keys:

"stack"
4-D array containing db intensity measure (*numpy.ndarray*), (height, width, pol_count, time_series_length)

"coordinates"
3-D array containg coordinates where [:,:,0] provides access to latitude and [:,:,1] provides access to longitude, (*numpy.ndarray*), (height, width, 2)

"timestamps"
list of acquisition timestamps of size (time_series_length,) (*list of str*)

"metadata"
Dictionnary describing data for each axis of the stack, the coordinates as well as the properties (orbit number, slice, acquisition time....) of each image of the temporal stack

Return type
dict

geesarfetcher.**fetch_point**(*coords*, *start_date: datetime = datetime.date(2021, 6, 20)*, *end_date: datetime = datetime.date(2022, 6, 20)*, *ascending: bool = True*, *orbit_number: Optional[object] = None*, *scale: int = 20*, *n_jobs: int = 8*, *verbose: int = 0*)

Fetches SAR data from a single coordinate point in the form of a dictionnary with image data as well as timestamps

Parameters

- **coords** (*tuple of float*) – Coordinates (lon, lat) of the point of interest
- **start_date** (*datetime.datetime, optional*) – First date of the time interval
- **end_date** (*datetime.datetime, optional*) – Last date of the time interval
- **ascending** (*boolean, optional*) – The trajectory to use when selecting data
- **orbit_number** (*int or str, optional*) – The orbit number to restrict the download to. If provided with an integer, the S1 temporal stack is filtered using the provided orbit number. If provided with a string value, we expect one of these keywords:
  - "max" for the orbit number with the highest number of image in the stack
  - "min" for the orbit number with the smallest number of image in the stack

  If None, then no filter over the orbit number is applied.
- **scale** (*int, optional*) – Scale parameters of the getRegion() function. Defaulting at 20, change it to change the scale of the final data points. The highest, the lower the spatial resolution. Should be at least 10.

- **n_jobs** (*int, optional*) – Set the parallelisation factor (number of threads) for the GEE data access process. Set to 1 if no parallelisation required.
- **verbose** (*int, optional*) – Verbosity mode (0: No info, 1: Info, 2: Detailed info, with added timestamp)

Returns

Dictionnary with four keys:

`"stack"`

4-D array containing db intensity measure (*numpy.ndarray*),

`(1, 1, pol_count, time_series_length)`

`"coordinates"`
3-D array containg coordinates where `[:,:,0]` provides access to latitude and `[:,:,1]` provides access to longitude, (*numpy.ndarray*), `(1, 1, 2)`

`"timestamps"`
list of acquisition timestamps of size (time_series_length,) (*list of str*)

`"metadata"`
Dictionnary describing data for each axis of the stack and the coordinates as well as the properties (orbit number, slice, acquisition tim....) of each point of the temporal stack

Return type
dict

7.2 geesarfetcher.filter

geesarfetcher.filter.**filter_sentinel1_data**(*start_date*, *end_date*, *geometry*, *orbit='ASCENDING'*, *orbit_number=None*)

Filters Sentinel-1 products to get images collected in interferometric wide swath mode (IW) and on i) a date range, ii) a geometry and iii) ascending or descending orbit.

Parameters

- **start_date** (*str*) – str following the pattern `'yyyy-mm-dd'` describing the start date of the time interval
- **end_date** (*str*) – str following the pattern `'yyyy-mm-dd'` describing the end date of the time interval
- **geometry** (*ee.Geometry*) – Geometry object defining the area of process
- **orbit** (*str, optional*) – Defines the orbit to set for the data retrieval process
- **Returns** –
- **--------** –
- **ee.ImageCollection** – Filtered ImageCollection left to be queried

7.3 geesarfetcher.fetcher

geesarfetcher.fetcher.**fetch_sentinel1_data**(*start_date*, *end_date*, *geometry*, *scale*, *orbit='ASCENDING'*, *orbit_number=None*)

Retrieves and queries ImageCollection using input parameters and return data as a tuple of header and values.

Parameters

- **start_date** (*str*) – str following the pattern `'yyyy-mm-dd'` describing the start date of the time interval
- **end_date** (*str*) – str following the pattern `'yyyy-mm-dd'` describing the end date of the time interval
- **geometry** (*ee.Geometry*) – Geometry object defining the area of process
- **scale** (*int*) – Scale parameters of the getRegion() function. Defaulting at `20`, change it to change the scale of the final data points. The highest, the lower the spatial resolution. Should be at least `10`.
- **orbit** (*str, optional*) – Defines the orbit to set for the data retrieval process
- **orbit_number** (*int or str, optional*) – The orbit number to restrict the download to. If provided with an integer, the S1 temporal stack is filtered using the provided orbit number. If provided with a string value, we expect one of these keywords:
  - "max" for the orbit number with the highest number of image in the stack
  - "min" for the orbit number with the smallest number of image in the stack

  If `None`, then no filter over the orbit number is applied.

Returns
Returns a dictionnary of the properties of the Image retrieved from GEE.

Return type
dict

geesarfetcher.fetcher.**fetch_sentinel1_properties**(*start_date*, *end_date*, *geometry*, *orbit='ASCENDING'*, *orbit_number=None*)

Retrieves and queries ImageCollection using input parameters and return Image properties.

Parameters

- **start_date** (*str*) – str following the pattern `'yyyy-mm-dd'` describing the start date of the time interval
- **end_date** (*str*) – str following the pattern `'yyyy-mm-dd'` describing the end date of the time interval
- **geometry** (*ee.Geometry*) – Geometry object defining the area of process
- **orbit** (*str, optional*) – Defines the orbit to set for the data retrieval process
- **orbit_number** (*int or str, optional*) – The orbit number to restrict the download to. If provided with an integer, the S1 temporal stack is filtered using the provided orbit number. If provided with a string value, we expect one of these keywords:
  - "max" for the orbit number with the highest number of image in the stack
  - "min" for the orbit number with the smallest number of image in the stack

  If `None`, then no filter over the orbit number is applied.

Returns
val_header corresponds to the `list of str` describing the fields of the val array. The val array is a `list` of data records, each represented as a `list` of the same size as the val_header array.

Return type
tuple

7.4 geesarfetcher.utils

The utils module contains functions used to prepare data before fetching sar images.

`geesarfetcher.utils.`**cmp_coords**(*a*, *b*)

Given two coordinates dict a and b, compare which one is closer to the North-Eastern direction

Parameters

- **a** (*dict*) – dict with keys `"lon"` and `"lat"`
- **b** (*dict*) – dict with keys `"lon"` and `"lat"`

Returns
-1 if `a > b`, **1** if `a < b`, **0** if `a == b`

Return type
int

`geesarfetcher.utils.`**get_date_interval_array**(*start_date*, *end_date*, *day_timedelta=1*)

Initialize a list of days interval of size `day_timedelta` iteratively created between `start_date` and `end_date`.

Parameters

- **start_date** (*datetime.datetime*) – first date time of the array
- **end_date** (*datetime.datetime*) – last date of the array
- **day_timedelta** (*int*) – size, in days, of every interval

`geesarfetcher.utils.`**make_polygon**(*top_left*, *bottom_right*)

Given two (lon, lat) coordinates of both the top left and bottom right corner of a polygon, return the list of corner coordinates of this polygon

Parameters

- **top_left** (*list of int or tuple of int*) – Top Left coordinates of the polygon
- **bottom_right** (*list of int or tuple of int*) – Bottom right coordinates of the polygon

Returns
2-D list of the 5 coordinates need to create a Rectangular Polygon `[top_left, top_right, bottom_right, bottom_left, top_left]`.

Return type
list

`geesarfetcher.utils.`**retrieve_max_pixel_count_from_pattern**(*error_str*)

Given an input getRegion error from GEE, extract the provided points count.

Parameters
error_str (*str*) – the str text of the GEE error (e.g. the function caled on `"ImageCollection.getRegion: Too many values: x points ..."` will output x)

Returns
Returns the number of points specified in the input image

Return type
int

geesarfetcher.utils.**tile_coordinates**(*total_count_of_pixels*, *coordinates*, *max_gee=1048576*)

Given a coordinates array describing a Polygon, a count of pixes within that polygons, tiles this polygon into a grid a sub-Polygons where each sub-Polygon size matches the max_gee pixel count given as a parameter.

Parameters

- **total_count_of_pixels** (*int*) – Total number of pixels of the designated area
- **coordinates** (*array of array of floats*) – Can be a 5-sized list of every coordinates defining the polygon `[[long1, lat1],[long2, lat1]...,[long1, lat1]]` or a 2-sized list of coordinates defining the top left and bottom right corner of the Polygon `[[long1, lat1],[long2, lat2]]`
- **max_gee_threshold** (*int, optional*) – Total number of points allowed for one data query. Default: 1048576

Returns
3-dimensional list of coordinates with pixel count inferior or equal to the maximum GEE threshold (shape: `(number of images, number of coordinates per image, 2)`)

Return type
list

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX