A Google Earth Engine Processing Flow Package for Omanos Analytics: GEEOAPP

Oliver Atkinson

E-mail: o.atkinson.1@research.gla.ac.uk

Omanos Analytics
December 12, 2022

Abstract: In order to leverage the vast computational power and rich datasets available through Google Earth Engine, a Python package has been developed. This manual sets out the required background information on satellite Earth observations, before detailing how to install, use and expand upon this package.

Contents

1 Introduction

Since the advent of satellite technology with the famed Sputnik probe in 1957 the rate at which humans have launched objects into space has increased astronomically. From the far flung Voyager missions to the GPS system that powers SatNavs the world over, the applications of satellites have broadly fallen into two categories; to observe space and to observe our home, the pale blue dot so evocatively captured by Voyager 1.

Whilst the former category can often be more headline grabbing, thanks in large part to the cosmic imaging power of the Hubble and now James Webb space telescopes, the latter is crucial to understanding our fragile planet. Earth observations not only power weather forecasts, they allow us, among other things, to predict forest fires and map the changes wrought on our environment by nature and indeed by humanity itself.

In this document, the fundamental principles of Earth observation (EO) are set out, beginning with the orbital dynamics of the satellites that gather the data in section 2.1, before setting out some of the common terms in section 2.2, the various methods of imaging in section 2.3, with a focus on the key method of multi-spectral imaging in section 2.3.1. A brief introduction to Google Earth Engine (GEE) is given in section 2.4. A handful of the most notable missions and the nature of their datasets are collected in section 3. Beyond this, the Python package, going by the name `GEEOAPP` (Google Earth Engine Omanos Analytics Processing Package), is addressed, beginning with instructions on how to setup to use it in section 4, information on structure of the package in section 5, how to navigate the various input options available in section 6. Further instructions on how to expand `GEEOAPP` are given for concrete examples in section 7 and in more general development terms in section 8. Finally, a collection of information on the indices implemented in `GEEOAPP` is given in section 9, with useful links for further reading collected in section 10.

2 Background Information

2.1 Fundamentals of Orbital Dynamics

Here the physics involved in getting a lump of metal to continually fall toward the planet without ever hitting it is briefly recapped, with a focus on satellite orbits. For the purposes of this section, only circular orbits will be considered.

Beginning with Kelper's Laws, we can somewhat bypass the first law, that all orbits are elliptical, as satellite orbits have zero eccentricity and are therefore circular. The second law, that orbits sweep out equal areas in equal times, then becomes self-evident for a constant speed circular orbit. The third law, which links the period T and orbital radius r also becomes a little simpler, and reduces to (assuming the satellite is massless compared to planetary heft):

$$T^2 = \frac{4\pi^2 r^3}{G m_{\mathrm{E}}}, \tag{2.1}$$

where $G = 6.67 \times 10^{-11}$ $\mathrm{Nm^2kg^{-2}}$ is the gravitational constant and $m_\mathrm{E} = 5.97 \times 10^{24}$ kg is the mass of the Earth. It is important to note that r is not the altitude above the Earth, h, but the orbital radius, i.e. $r = h + r_\mathrm{E}$ for the radius of the Earth $r_\mathrm{E} = 6.37 \times 10^6$ m.

As the satellite orbits are circular, a number of equations and parameters simplify dramatically or are rendered irrelevant, though it is useful to note that the orbital velocity is given by

$$v^2 = \frac{Gm_\mathrm{E}}{r}. \tag{2.2}$$

2.2 Basics of Earth Observations

The majority of satellites that are involved in EO sit in low Earth orbits, with altitudes in the region of 800 km. Further up, in the 2000–36000 km range, the satellites are typically involved in navigation or have specific specialities. Most weather satellites sit beyond this range, with a geostationary orbits (having periods of exactly one day they stay above the same point on the Earth's surface) at 36000 km.

Orbital inclination is the angle between the Equator and the orbital path, defined such that 90° is a North to South pole path while 180° is an equatorial orbit in the opposite direction to the Earth's rotation.

Many EO satellites have inclinations close to 90° and periods around 90 minutes. Such orbits are sun-synchronous, meaning that they stay in the same time zone as the Earth rotates; they always cross the Equator at the same time for instance. This ensures that the sunlight incident on the surface below the satellite is consistent each time the satellite passes overhead, so there are minimal changes to the lighting between images (up to seasonal variations), thereby making comparisons over time more straightforward.

Such orbits are only viable in narrow paths, so regular adjustments must be made to correct for atmospheric drag, which is non-zero at 800 km, and the gravitational influence of the Sun and Moon.

For a near polar sun-synchronous orbit, the ascending pass, defined as the northward half of the orbit, typically passes over the side of the Earth in shadow, with the converse true for the descending path. Satellites that rely on reflections on the Sun's light by the planet therefore only image on the descending pass. This is part of the reason that some missions, such as Sentinel 1 and 2, are comprised of two satellites which are 180° out of phase with one another.

The surface imaged by the satellite on each pass is the swath, measured as width, meaning the satellite images a strip over time. Swathes range from a few kilometres to a few thousand kilometres. The rotation of the Earth under the satellites means that longitude of the satellite changes and thus the imaged strip moves. The time taken to completely cover the surface of the Earth is termed a cycle.

A cycle is distinct from the revisit time, the duration between consecutive images of the same point, as satellites are capable of imaging regions that do not sit directly underneath them.

2.3 Types of Satellite Imaging

Satellite imaging comes in two main kinds, active and passive. Passive imaging simply captures the emissions from the surface, whether local sources or reflected sunlight, leaving this technique dependant on the time of day, atmospheric conditions, clouds and any unwanted sources of light. Active imaging on the other hand, beams a signal at the Earth and captures the reflected signal and thus does not suffer from these issues. Despite this, passive imaging is more common amongst the major EO missions, and so this is addressed first here.

The wavelengths of light that satellites can capture, known as bands, are restricted by the atmosphere being largely opaque to many wavelengths. Visible light (λ = 400–700 nm) is of course transmitted by the atmosphere, and the Infra-red (IR, 700–1400 nm) is also generally accessible, with some peaks and troughs in the spectrum. Radio waves (10 cm – 10 m) are freely observable.

Panchromatic imaging captures a large range of wavelengths in a single channel, giving very high resolution images, with resolutions up to (or down to depending on semantics) 30 cm. This is often done across the visible or IR.

Multi-spectral imaging (MSI) is of great interest here as it is the primary mode of operation for many satellites and as such is discussed in more detail in the next section. As the name suggests, data is captured in a number of distinct bands at once. Whilst the spatial resolution is lower than that of panchromatic imaging, typically of the order of tens of metres, this approach allows important features, such as vegetation or soil type, to be detected.

Images can be pan-sharpened by combining data from the two above methods, giving high resolution full colour images. Hyper-spectral imaging is an extension of MSI to tens or hundreds of bands, capturing a nearly continuous spectrum for each pixel simultaneously. Thus technique is not yet widely available.

Microwave radimetry is useful for detecting the atmospheric water content, giving the integrated atmospheric water vapour column and the cloud liquid water content, which are important correction terms in radar altimetry. This data can be used to find the surface emissivity and soil moisture.

Synthetic aperture radar (SAR) is the most common of the active imaging techniques and is used by Sentinel 1. Radio waves are pulsed at the Earth, with the return time and intensity used to form an image of the surface. This technique is not sensitive to the day–night cycle, nor most weather conditions. The resolution can be enhanced by use of interferometry, which can give resolutions up to the centimetre level.

Lidar operates in much the same way, but using IR, visible or UV wavelengths to create vertical profiles of aerosols and thin cirrus cloud.

Radar altimetry uses radar techniques to map the topography of the Earth's surface.

2.3.1 Multi-spectral Imaging

The bands most commonly imaged in MSI are given below, along with features that they are useful for imaging - note that the wavelength ranges are not rigid:

- Blue: 450–515 nm, atmosphere and deep water
- Green: 515–600 nm, vegetation
- Red: 600–690 nm, man-made objects, soil and vegetation
- Near IR (NIR): 780–1400 nm, vegetation
- Mid-IR (MIR): 1550–1750 nm, vegetation , soil moisture and fires
- Far-IR: 2080–2350 nm, soil moisture, geological features, silicates, clays and fires.

Often the mid and far IR are grouped under the classification of short wave IR (SWIR).

The different bands captured in MSI can be combined in various ways. True colour images use the red, green and blue band data as the RGB channels in the image and thus represent the visual experience of a person. Often, false colour images offer more insight, wtih data from the non-visible part of the spectrum combined in the RGB channels in various configurations. These include

- Green, Red, IR, highlights vegetation in blue
- Blue, NIR, MIR, keeps vegetation green and shows water depth, soil moisture and fires
- SWIR, NIR, Green for floods and newly burned land
- Blue and two SWIR bands, for snow, ice and cloud differentiation

Another facet of MSI that makes it a particularly potent EO technique is that the bands can be combined to calculate indices that allow for classification. The most notable of these is the normalised differential vegetation index (NDVI), which makes use of the fact that healthy vegetation absorbs most visible and IR (VIR) light and reflects most NIR light. Thus

$$\mathrm{NDVI} = \frac{\mathrm{NIR} - \mathrm{VIR}}{\mathrm{NIR} + \mathrm{VIR}}. \tag{2.3}$$

Above some threshold value, pixels with high NDVI are classified as vegetation. Senescing, or old and deteriorating, vegetation is more reflective in Green than healthy vegetation which means the ratio of NIR to Green can be used to discriminate between vegetation types, with high values indicating senescing.

In a similar vein, the normalised difference snow index (NDSI) can be calculated, as snow reflects in the VIR and absorbs the SWIR, while cloud is generally reflective in both, so

$$\mathrm{NDSI} = \frac{\mathrm{VIR} - \mathrm{SWIR}}{\mathrm{VIR} + \mathrm{SWIR}}. \tag{2.4}$$

Strong negative (less than about -0.2) values indicate no cloud, values above around 0.2 are classed as cloudy, with intermediate values indicating a possibility of cloud.

Red reflectance also gives a cloud probability, with a low value indicating a pixel is cloud free. NIR can be used for snow detection, with low reflectance a sign of no snow; similarly for Blue. The ratio of Blue to SWIR allows for soil and water discrimination, with low values indicating soil and higher values water. Rocks and sands in deserts have a low NIR/SWIR ratio. Cirrus cloud can be detected in the SWIR at 1375 nm, with strong water vapour absorption at 1380 nm. Further examples are given in brief in section 9.

2.4 An Introduction to Google Earth Engine

GEE provides free access to a huge, petabyte level collection of datasets, drawn from a wide range of EO satellites. This vast amount of data can be put to work through the GEE infrastructure, which offers a web-based Javascript code editor as well as Python and Javascript APIs. This enables calculations to be performed on the image data to match the required real world applications.

One of the advantages of the GEE apparatus is that it allows for cloud computing, with much of the computational heavy lifting taking place on the server side, including data querying and many of the calculations.

In terms of use cases that are most relevant here, the Python API is the most important. The goal is to write local Python code that can interface with the ocean of GEE datasets, perform a suite of calculations (e.g. NDVI) and return images in whichever bands are appropriate for the task at hand, whilst also leveraging the server side computing power to the maximal possible extent.

3 A Brief EO Satellite Taxonomy

It's important to know what satellites are out there, and what operations they perform, the data they gather and so forth. This is a brief summary of the most important satellite missions, in no particular order.

3.1 Landsat

- Continuous EO since 1972 with 9 successive satellites and a 10^{th} planned, making for an incredibly rich dataset
- MSI, with bands ranging from Green, Red and two IR with resolutions of 60 m from Landsat 1, to 11 bands in Landsats 8 and 9, with 4 visible, 1 NIR and 3 SWIR with resolutions of 30 m, a panchromatic band at 15m and two thermal bands at 100 m
- Revisit time of around 16 days for each of Landsat 8 and 9, with an 8 day offset between the two
- The Tier 1 dataset meets the geometric and radiometric quality requirements

3.2 MODIS

- The Moderate Resolution Imaging Spectroradiometer is an instrument aboard the Terra and Aqua satellites which date to 2000
- The wide swath of 2330 km enables a revisit time of just 1–2 days
- Imaging across 36 bands with a wide range of wavelengths and purposes
- The downside is the relatively low spatial resolution of 500 m

3.3 Sentinel 1

- Active imagine (SAR) satellite operating in radio, dating back to 2014
- Bands differ in terms of polarisation and classified according to whether the emitted and received signal was horizontally (H) or vertically (V) polarised
- VV typically used, except in polar regions where HH is used
- 10 m resolution in each of these bands with a swath of 250 km in the interferometric wide mode, which is the primary channel
- Revisit time of the order of a few days, depending on location

3.4 Sentinel 2

- MSI with a pair of satellites with diametrically opposed orbits and wide swath and high resolution
- 13 bands; 4 visible and NIR (VNIR) at 10 m resolutions, 4 Red Edge (700–780 nm) and 2 SWIR at 20 m and 3 at 60 m, including bands to image aerosols, water vapour and cirrus clouds
- Revisit time of 5 days, operational since 2015 in the case of 2A and 2017 for 2B
- Commonly used and high quality dataset

3.5 Planet Labs SkySat

- Dataset from a private firm that has been made publicly available
- High resolution of 2 m in the RGB and NIR bands, and 0.8 m in the panchromatic
- Limited temporal availability

There are a large number of additional datasets available on the GEE database, many of which are designed with one specific case in mind, such as US lithography. The Landsat, MODIS and Sentinel datasets are the leading datasets that are publicly available, with further datasets derived from these.

4 Installation and Setup

The source code for `GEEOAPP` can be downloaded from the GitLab page here[1], which can be accessed once appropriate permissions have been given to your GitLab account. Expand the package by unzipping or untarring the compressed folder as appropriate, which should result in a single folder called `earth-engine-internship-main`. In this folder the source code of the package is stored, and it is also where the package is run from. Feel free to move and rename the entire folder as you wish.

In order to use Google Earth Engine and to make full use of this package, the user should have a Google account with an associated Google Drive. It is recommended that this Google Drive is accessible on the users local system and automatic syncing is turned on. This will allow any images generated to be automatically downloaded to the users machine.

Before using `GEEOAPP` itself, there are a few key dependencies that must be installed, namely an up to date Python build, and the Google Earth Engine Python API.

It is recommended that the open source Python platform Anaconda is installed, which can be found here[2]. This allows the user to set up a virtual environment which can be used specifically for running `GEEOAPP`, without interfering with any other Python builds present on the system. Once Anaconda has been installed, a virtual environment with all the required Python packages can be created. To do this, navigate to the location of the package in terminal (if you are unfamiliar with using command line tools, see the quick tutorial for navigating files here[3]), then run

```
$ conda create --name NAME --file GEEOAPP_CondaEnv.txt
```

from the command line, replacing `NAME` with your desired name for the virtual environment. The virtual environment can then be activated by running

```
$ conda activate NAME
```

Otherwise, instructions for installing the Google Earth Engine Python API can be found here[4], and the list of required Python packages found in `GEEOAPP_CondaEnv.txt`.

In order to use Google Earth Engine functionality, you must first be authenticated; to do this run

```
$ python GEE_Authenticator.py
```

which will prompt you to login and allow access by Google Earth Engine before redirecting to a page showing your account has been authenticated. As a test, this should then print a message to say that this has been successful.

Finally, the whole package can be tested by running

```
$ python GEEOAPP.py Examples/Example1.yml
```

This runs `GEEOAPP` for a simple example case, which collects Sentinel 2 imagery of Glasgow for June to September of 2018 through to 2022, filtering for images with less than 20%

[1] https://gitlab.com/omanosanalytics/earth-engine-internship
[2] https://www.anaconda.com/products/distribution
[3] https://ubuntu.com/tutorials/command-line-for-beginners#3-opening-a-terminal
[4] https://developers.google.com/earth-engine/guides/Python_install

cloud coverage, applying a cloud mask, then creating a median image from each period, and calculating the NDVI. The result should be series of progress statements to the terminal, ending in the message `All computations completed!`. The 10 images (5 showing the NDVI, 5 the more familiar true colour RGB image), will be saved to the users Google Drive in a folder named `Example1`, with filenames that offer some description of the image, such as `GlasgowExample_RGB_S2_2018-06-01_2018-10-01.tif`. For the purposes of comparison, the expected versions of these images have been complied into a pdf that can be found at `RunCards/Examples/Example1Outputs.pdf`.

A number of example run cards can be found in the `RunCards/Examples` directory, in order to demonstrate the possible options and for further clarification.

5 Structure

A summary of the functionality present in the modules actually involved in running `GEEOAPP` is given below. Run cards should be stored in the `RunCards` folder, and can be collected into folders for individual projects, though note that the filepath for such folders will need to be included at the command line when running `GEEOAPP`. Already present in `RunCards` folder is an `Examples` directory, which contains a handful of different ways to use `GEEOAPP`, in order to give a new user an introduction to some of the available options and demonstrate their usage.

`IndexFunctions.py` - A library of index calculating functions. Indices with common structures, e.g. normalised difference indices (ND(Vegetation)I, ND(Water)I, ND(Snow)I) are collected together into single functions for ease. A library of RGB composites can also be called. Each of the functions checks that the required bands are present for the satellite the image is from, does the required calculations and returns the original image, with a band added for each index calculated. In the case of RGB composites, three bands are added, with the naming convention `CompositeName_R`, `CompositeName_G`, `CompositeName_B`.

`VisFunctions.py` - Collects a number of functions for creating and processing visualisation parameters for image saving, including loading standard visualisations and capacity for user generation of new visualisations in a few different ways.

`CollectionLoading.py` - Holds the functions for requesting image collections from the Google Earth Engine database. The main loading function takes a satellite name, an area of interest and dates. Cloud masks can optionally applied, and collections filtered to only contain images with less than a given percentage of pixels cloud classified as cloud. If the requested satellite has no suitable images, other satellites of the same type that were operational at the same time will be checked and the user will be given the option to choose from alternatives (if any are available). In here are functions to generate image collections over a time series (e.g. every whole year, or every January of every year between the start and end dates).

`QuantFunctions.py` - Stores the functionality for the quantitative analyses (mean, standard deviation, etc.) that can be performed on a given area or point, as well as the functions to plot time series graphs of the results of these analyses.

`GEEOAPP.py` - The central file, that imports all the functionality and information from all other files and is used to actually run the project. After imports, numerous error checks are performed on the user input from the run card.

This code itself additionally contains key elements of the project, such as, all image masking capability (generating, saving and loading), and crucially, a function that brings all of the functionality together and serves to run the whole package.

`GEE_Authenticator.py` - A short piece of code that serves to check that the user is authenticated to use Google Earth Engine.

`GUI.py` - Controls the graphical user interface, one of the possible input modes for the user; this code is only called if the user chooses not to supply a run card at run time.

`Satellites.yml` - Contains the library of currently supported satellites, storing the name of the Google Earth Engine database reference, the type of instrumentation (MSI, SAR, etc.), a dictionary of the bands in each image and what they physically correspond to (e.g for Sentinel 2, the band named "B3" corresponds to green), the resolution of the highest resolution band, the dates over which the satellite is/was operational, the cloud coverage property name, the cloud masking function name, and the scaling information.

`Palettes.yml` - Holds a large number of colour palettes, each of which is in the form of a list of hex strings, which can be used to create visualisations.

`Visualisations.yml` - Holds the standard visualisations for each index, with the form, storing the minimum and maximum values, and the name of the colour palette as it appears in `Palettes.yml`.

In terms of outputs, all images and masks will be saved to Google Drive directories, while plots will be saved to a local directory. The names of these directories are set in the run card. Note that, unfortunately, subfolders cannot be generated in Google Drive.

6 Usage

`GEEOAPP` is operated using a run card formalism, whereby all the user input is collected into a single data file, which is then read in and checked when `GEEOAPP` is run. The package must be pointed to a valid run card stored in the `RunCards` directory at run time:

```
$ python GEEOAPP.py YourRunCardName.yml
```

This avoids the need for the user to edit the `GEEOAPP` code itself, as well as making it possible to store numerous different configurations for easy access or re-running at later times. A blank template run card is included in the package for ease of editing. The rest of this section outlines the nature of the run card inputs and how use each of the options. All flags follow the convention of being switched on with a value of 1, and off with a value of 0. If no value is passed, the flag is assumed to be off.

`GEEOAPP` can also be run with a user interface for input entry. This is initialised if no run card argument is passed to the command line call; i.e.

```
$ python GEEOAPP.py
```

Once the user inputs are checked for validity, this will generate a run card so that the user inputs can be accessed at later times. The user inputs to the interface should follow the

same conventions as outlined below for the run card; e.g. dates should still be formatted as 'YYYY-MM-dd' (including quotation marks).

6.1 Run card inputs

The `Names` block tells `GEEOAPP` where to save the outputs, with the `ImSaveDir` and `MaskSaveDir` options pointing to Google Drive folders where the images and masks will be saved respectively. The `PlotDir` variable is a local folder, relative to the location of the package. All these folders will be created if they do not already exist. The `Name` variable itself acts as a prefix for the filenames for all the outputs. It is recommended that this relates to the area of interest, the project at hand or something similar. All these names should be strings and will be checked for any illegal characters. Input is required for all variables here, even if no masking or plotting will be done.

The area of interest is defined in the next block, with the boundaries set by specifying the minimum and maximum values, in degrees, for the longitude and latitude. These will form the four corners of the generated imagery and are subject to validity checks, including if the resulting area is overly large or small compared to the resolution of the requested satellite. Inputs are always required here.

Choosing a satellite mission is one of the more important decisions to be made. Valid satellites are given in the `Satellites.yml` file and should be specified by their full name as it appears there. The satellite name will also be incorporated into any filenames of images as an abbreviation of the first character of the name and mission number; e.g. Sentinel 2 becomes S2.

Image processing input is also important. In particular, the `reducer` variable tells `GEEOAPP` what operation to perform to collapse image collections down to a single image. For instance, if the user requests data for each of the last 5 years, `GEEOAPP` will find 5 collections, each of which will consist of a number of images. The `reducer` operation then acts on each of these collections, turning them into 5 single images, creating one from each collection. Currently implemented options here are mosaic and median; one of these must always be chosen. The `cloud_cover` option sets the maximum allowed percentage of pixels registered as cloud. Only images with below this percentage will be included in the image collections. If no input is given, this will default to a value of 100. The `cloud_mask` is a flag for applying a cloud mask, which kills all pixels registered as cloud in the dataset.

6.2 Dates

The `Dates` block is slightly more involved. First, it is important to note that all dates should follow the format 'YYYY-MM-dd'; that is year, month in year and day in month (DD corresponds to day in year), and that any dates must be explicitly formatted as strings (with the quotation marks around them) in order to be read in correctly. The `start_date` and `end_date` options specify the overall period of interest. These are the only two required inputs here. The remaining options detail the sampling within the overall period. The `period` should be one of day, week, month, year, and images will be generated for every `delta_period period`, e.g. each image will cover a 2 month period, `delta_period` being a numerical value. These default to a 1 year sample if no input is given. Additionally, it

is possible to take samples for a sub period within the overall period, e.g. images for a single two month period every year. This sampling method is activated if input is given to `delta_sub`, which again should be numerical, while `subperiod` should be one of day, week, month, year. If the start of the sub period is different to the overall start date this can be set using the `sub_start` variable, which defaults to the `start_date` if not given. If the sub period sampling is longer than the period sampling, the resultant images form a moving average.

6.3 Masking

Masking is controlled in the next section of inputs. There are two ways to apply a mask; either by generating one at run time, or by loading a mask from a shapefile. Mask generation is defined in the first 5 parameters here. Note that a mask will be generated on an image by image basis according to the criteria given, i.e. each image will likely have a slightly different set of pixels masked. The `mask_bandName` is the name of the band used to calculate the mask (the mask is applied to all bands in the image). This will typically be some index such as NDWI. The `upper_bound` and `lower_bound` options are the numerical values of the mask; only pixels with values greater than the upper bound will be masked, as will those with values below the lower bound. When generating a mask, both these options must be included, even if only an upper or lower bound is desired. In this case, use a dummy value for the other bound that will not mask any pixels, e.g. an upper bound of 100000. The `save_raster` and `save_vector` flags allow for the mask to be saved if desired, as a `.tif` and a `.shp` respectively.

The last three options in this section concern loading a mask from a shapefile. The shapefile must first be uploaded to Google Earth Engine through the online code editor, instructions for which can be found here[5]. Once the file has been uploaded, the `load_name` tells `GEEOAPP` where to find it, specifying the full filepath here (something like `Users/username/file`). The loaded mask is applied to all images, so the same pixels in every image are masked. Unfortunately the loading process is not perfect. In some cases, the mask may be inverted; to correct this set the `load_invert` flag on. Small and intricate regions may also not be properly loaded in, which can be corrected to a good extent with the `load_overlap` option, which takes a value of 1 or 2. Trial and error is the best way to find out which values these last two variables to take. It is possible to both load and generate masks at the same time, allowing for masks from multiple bands to be applied.

6.4 Analytic information

The analytic information is contained in the next block, which defines what images will be generated, which indices will be calculated and if any time series analyses will be performed. The `indices` option should be a list of indices to be calculated, as well as any RGB composites. The available list of indices and composites is stored in the `GEEOAPP.py` file.

[5] https://developers.google.com/earth-engine/guides/asset_manager

The remaining options concern the quantitative analyses. The `areas` variable is should be a list of points and/or geometries to perform the analyses on. For example, `[[xMin1, yMin1, xMax1, yMax1], [xMin2, yMin2, xMax2, yMax2], [x1, y1]]` will perform the requested analyses for two rectangular areas and one point. The `bandName` is the band on which the analyses will be performed, and will typically be an index name. The names of the actual analyses are given in `quant_analyses`, which should be a list of names. Currently available analysis names are AreaMean, AreaStdDev, AreaThreshold and PointValue. The standard deviation will not be plotted on it's own, but will show as an uncertainty range when the mean of an area is plotted. The AreaThreshold analysis gives the percentage of a region with pixel values within the range specified by the `upper_bound` and `lower_bound` options, which follow the same conventions as the masking bounds, in that a dummy value of ± 100000 should be used if only an upper or lower bound is required. The final option in this block is a flag that if on will save a true colour RGB image with the analysed areas highlighted.

6.5 Visualisations

This input block is entirely optional. If no input is given, default visualisations, as defined in the `Visualisations.yml` file, will be used. Otherwise, this block allows the user to display images using customised visualisations, which can be generated to meet user needs in a number of ways. Visualisations consist of three parameters; `vis_min`, `vis_max` and `palette`. The first two options are numerical, specifying the minimum and maximum pixel values to colour, while the `palette` is a list of colours, where each colour can either be a list of RGB values or a hex string. Pixels with values below `vis_min` will be displayed as the first colour in the `palette`, while pixels with values above `vis_max` will be displayed as the last colour in the `palette`. If the `palette` is an empty list, greyscale will be used.

The user can specify a custom visualisation directly by providing `vis_min`, `vis_max` and `palette` to the `Visualisations` block. Additionally, the `palette` option can be the name of a pre-defined colour list stored in the `Palettes.yml` file. The `make` flag should be turned off, and no additional input is then required.

By turning the `make` flag on, the user can use `GEEOAPP` to generate a new visualisation. In this case, the `vis_min`, `vis_max` and `palette` should not be included as they will be overwritten by the generated visualisation. There are two further flags; `ramp_flag` and `steps_flag`. These refer to the two methods of generating a visualisation. If `ramp_flag` is switched on, the `vals` option should be a list that stores only the minimum and maximum values for the visualisation. The `floor` and `ceiling` options define the colours used for pixels with values below the minimum and above the maximum, defaulting to black and white respectively if no input is given. The `colours` variable should be a list of colours. With the `ramp_flag` switched on, a smooth colour ramp between each colour in `colours` is generated and used for the palette. The number of intermediate steps in this ramp can be set with the `n_steps` variable, which defaults to 10.

The `steps_flag` allows for a different approach. In this case, the `vals` list should be a series of steps, which can be uneven intervals. The `colours` should now be one colour for each interval in the `vals` list (i.e. one less than the length of `vals`). The `floor` and

`ceiling` options operate as before. The two options can be combined by turning both flags on, giving a smooth series of colours over uneven intervals, allowing for a fine grained palette. Turning on the `show` flag saves a plot of the custom visualisation.

A custom visualisation can be specified for each index in the image, so the structure of the `Visualisations` block should be as follows (for a simple example)

```
Visualisations:
    NDVI:
        vis_min: -0.2
        vis_max: 1
        palette: ['#fff', '#00cce5', '#007fe5', '#0000b2']
        make: 0
        show: 1
```

Examples of some custom visualisations are given in `VisualisationExamples.pdf` for further clarity.

7 Expansions

`GEEOAPP` is designed to be accessible and expandable for future usage, and how to do this is the focus of this section.

7.1 Adding an index

There are three steps to adding an index. In the `IndexFunctions.py` file, a new function should be created that take an image and a band dictionary as arguments. Following the pattern of the other functions in this file, the bands should be checked with the `BandChecker` function to ensure that the required bands are in the image. After doing the required calculations to find the index, the index band should be renamed with the name of the index. The function should return the image with this band added. In the `GEEOAPP.py` file, the `Analysis_lib` should be expanded with the name of the index. In the same file, in the `GEEOAPP` function, a line should be added, following the pattern of the lines already present there, e.g.

```
if 'NAME' in analyses: im_plus_ind = NAME_calc(im_plus_ind, band_dict)
```

In order to generate images, a visualisation will need to be defined in the run card, or alternatively included in the `Visualisations.yml` (see below).

7.2 Adding a visualisation

Visualisations are added in the `Visualisations.yml`, and should specify the `vis_min`, `vis_max` as well as the name of the palette as it appears in `Palettes.yml`. When adding a new index, the default can be set by defining a new visualisation with the name `IndexName_default`. A custom palette can be included in `Palettes.yml`, which is again set as the default by defining it as `IndexName_default`.

7.3 Adding a quantitative analysis

Adding a new quantitative analysis is comparatively straightforward, as there is a lot of unexplored room here with only a handful of analyses implemented so far. In the `QuantFunctions.py` file, define a new function, that takes the image, the sub region of interest, the image resolution and the name of the band to perform the calculation on. The operations available as Earth Engine Reducer objects are straightforward to implement; the function should return the value

```
im.reduceRegion(ee.Reducer.Name(), area, res).get(bandName).getInfo()
```

for some operation `Name`. The function name should start with either Area or Point, depending on the nature of the geometry for the analysis to be performed on. Then, in the `GEEOAPP.py` file, the `Quant_lib` should be expanded to include the name of the function.

7.4 Adding a satellite

A vast number of datasets are available through Google Earth Engine, only a handful of which have been implemented in `GEEOAPP` to date. The full selection of datasets can be found here[6]. To include a new satellite dataset, edit the `Satellites.yml` file to include a new dictionary named after the satellite. The `Dataset` variable is the name of the dataset as it appears in Google Earth Engine. Much of the remaining required information is accessible on the Earth Engine catalog page for the dataset. Particular attention should be paid to the `Bands` dictionary, which translates between the physical nature of the band (e.g. Green) to the name of that band in the images. This allows the index calculations to remain as general as possible and thus work for multiple satellite datasets, without needing further adjustments. The `mask` variable should be the name of a function that masks cloud in the dataset, which should defined within the scope of the `CollectionLoading.py` file.

8 Developer Notes

There are a few possible ways to further develop `GEEOAPP`, beyond the addition of further indices, composites or satellites. Some ideas for these developments are set out here for future consideration.

One of the more evident missing components of the package is automated change detection. Currently, only images from single times are generated, giving a snapshot in time. Often, real world applications focus on showing changes in time. `GEEOAPP` does generate a list of individual images, so it should be possible to take this list and analyse it as a whole, for example subtracting successive images from each other and thereby showing changes in time.

A further step in automation would be to make the satellite input optional. In this case, `GEEOAPP` could be altered to choose a best case satellite dataset. One question is how to define this best case scenario; whether to select for the most recently launched, highest resolution satellite or the dataset with the most images, or possibly some combination of these factors.

[6] https://developers.google.com/earth-engine/datasets/catalog

It might also be handy to create some standard analysis libraries, that run all required analyses for, e.g. vegetation health or fire detection. Similarly, some default masking libraries could be implemented that will always mask some ground feature such as snow. These could be stored in a series of run card templates, only needing date and area of interest input. This could make it straightforward to perform common analyses over different areas and times as required.

Further annotations on the images themselves would also be useful, including elements such as scale bars, the satellite name and stamping the acquisition dates onto the images. This seems to be involved, but perhaps possible.

9 Index of Indices

This section gives some brief details on the numerous indices implemented in the package, giving the relevant equations and an outline of their purpose.

9.1 Normalised Difference Index

This is the most common type of index in EO, and with the normalised difference vegetation index, one of the first formulations. The calculation takes the form

$$\mathrm{NDI} = \frac{\mathrm{Band\ 1} - \mathrm{Band\ 2}}{\mathrm{Band\ 1} + \mathrm{Band\ 2}}. \tag{9.1}$$

This allows features with high reflectance in Band 1 and low reflectance in Band 2 to be highlighted, with high NDI values. The naming convention is normalised difference [feature] index (ND[F]I) – in the following only the feature is stated in the section headings.

9.1.1 Vegetation

- Band 1: NIR, Band 2: R
- The most commonly used of all indices, this highlights healthy vegetation

9.1.2 Water

- Band 1: G, Band 2: NIR
- Best for detecting surface water, but can also pick out moisture from vegetation
- Values greater than 0.5 usually correspond to water bodies. Vegetation usually corresponds to much smaller values and built-up areas to values between zero and 0.2

9.1.3 Moisture

- Band 1: NIR, Band 2: SWIR
- Detects the moisture level in vegetation which allows checks for water stress and healthy vegetation
- Will also pick out surface water

9.1.4 Snow

- Band 1: G, Band 2: SWIR
- Differentiates between cloud and snow cover
- Values above 0.42 are usually snow

9.1.5 Red Edge

- Band 1: NIR, Band 2: RE
- Also highlights healthy vegetation

9.1.6 Normalised Burn Ratio

- Band 1: R, Band 2: SWIR
- Highlights burnt areas after a fire
- Can detect active wildfires and analyse the severity of the burn
- Typically used to calculate the difference pre- and post-fire

9.1.7 Green Normalised Difference Vegetation Index

- Band 1: NIR, Band 2: G
- Detects chlorophyll better than NDVI, used to show wilted or aging crops and monitor vegetation with dense canopies

9.2 Ratio Minus One

A ratio of two bands, normalised by subtracting one:

$$\mathrm{RM1} = \frac{\text{Band 1}}{\text{Band 2}} - 1. \tag{9.2}$$

9.2.1 Red Edge Chlorophyll (RECI)

- Band 1: NIR, Band 2: R
- Detects areas with yellow or shed foliage. most useful during active vegetation development

9.2.2 Green Chlorophyll (GCI)

- Band 1: NIR, Band 2: G
- Estimates leaf chlorophyll content, which is lower in stressed vegetation

9.3 Soil Adjusted Vegetation Indices

A few different methods of adjusting for the effects of soil on vegetation exist, given here are three of the most pertinent.

9.3.1 Soil Adjusted Vegetation (SAVI)

$$\text{SAVI} = (1 + L)\frac{\text{NIR} - \text{R}}{\text{NIR} + \text{R} + L}, \tag{9.3}$$

with $L \in [-1, 1]$, but typically $L = 0.5$. This mitigates the effects of soil on the NDVI, particularly useful for young and/or low coverage vegetation.

9.3.2 Optimised SAVI (OSAVI)

$$\text{OSAVI} = \frac{\text{NIR} - \text{R}}{\text{NIR} + \text{R} + 0.16}. \tag{9.4}$$

- Takes into account the standard value of the canopy background adjustment factor
- Greater soil variation than SAVI, and better sensitivity when the canpoy cover exceeds 50%
- Useful for low density vegetation with bare soil under the canopy

9.3.3 Modified SAVI (MSAVI)

$$\text{MSAVI} = 0.5(2\ \text{NIR} + 1 - \sqrt{(2\ \text{NIR} + 1)^2 - 8(\text{NIR} - \text{R})}) \tag{9.5}$$

Useful when vegetation is scarce, such as the beginning of the crop season.

9.4 Collected other indices

Several indices are uniquely defined; these are listed here.

9.4.1 Enhanced Vegetation Index (EVI)

$$\text{EVI} = 2.5\frac{\text{NIR} - \text{R}}{\text{NIR} + C_1\text{R} - C_2\text{B} + L}, \tag{9.6}$$

where typically $C_1 = 6, C_2 = 7.5, L = 1$. Adjusts the NDVI for atmospheric and soil noise, which is especially useful for dense vegetation. This also avoids saturated values, making it useful when the vegetation is dense and healthy. The range of values for EVI is -1 to 1, with healthy vegetation generally around 0.20 to 0.80.

9.4.2 Structure Intensive Pigment Vegetation Index (SIPI)

$$\text{SIPI} = \frac{\text{NIR} - \text{B}}{\text{NIR} - \text{R}} \tag{9.7}$$

High SIPI indicates chlorophyll loss, associated with diseased crops. SIPI values range from 0 to 2, where healthy green vegetation ranges from 0.8 to 1.8

9.4.3 Atmospherically Resistant Vegetation Index (ARVI)

$$\text{ARVI} = \frac{\text{NIR} - 2\ \text{R} + \text{B}}{\text{NIR} + 2\ \text{R} + \text{B}} \tag{9.8}$$

Insensitive to atmospheric and relief effects, useful for tropical mountains and regions where slash and burn agriculture is prevalent. Green vegetation has values $[0.20, 0.80]$.

9.4.4 Visible Atmospherically Resistant Index (VARI)

$$\text{VARI} = \frac{\text{G} - \text{R}}{\text{G} + \text{R} - \text{B}} \tag{9.9}$$

Enhances vegetation under strong atmospheric impacts and smooths variations in illumination, useful when the atmosphere must be accounted for.

10 Useful Resources

A collection of useful links, in no particular order:

1. Database of many useful analyses
2. Landsat 8 band information, includes useful examples and is relevant for Landsat 9
3. MODIS technical specifications
4. User guide for Sentinel 1
5. Sentinel 1 acquisition modes
6. User guide for Sentinel 2
7. Data processing flow for Sentinel 2, includes definitions of NDVI and other classification techniques
8. Landsat 9 dataset in GEE
9. Catalogue of MODIS data in GEE
10. Sentinel 1 dataset in GEE
11. Sentinel 2 dataset in GEE
12. GEE documentation