# CAL/VAL Prototype

## README File

**Change log**

November 29, 2024, V01.01.00

* Add support for FLEX images.

September 2, 2024, V01.00.01

* Add more detailed instructions on how to prepare the files for input
* Add an optional standalone python file (.py) to facilitate the subfolder creation.

August 28, 2024, V01.00.00

* First version of the CAL/VAL Prototype Guideline

**Objective and Scope**

The objective of this CAL/VAL prototype is to calculate coefficients of variation (CV) of regions of interest (ROI) centred at field sites, using Sentinel-2 (S2) images and FLEX images.

First, this prototype processes S2 images. It checks the cloud coverage inside ROIs, and if it is below a specific threshold, it performs calculation of CV, and flags whether the CV is not greater than a given threshold, 1 if true and 0 if false. Otherwise, the S2 image will be skipped.

Then, this prototype checks FLEX images. It checks the vegetation pixels inside ROIs, and if the number of vegetation pixels is greater than a specific threshold, it performs calculation of average and standard deviation of all bands. Otherwise, the FLEX image will be skipped.

The final output is in .csv format. For the output of S2 images, it contains names of site, numbers of valid pixels, percentages of valid pixels, values of CV and the corresponding flags. For the output of FLEX images, if a site has sufficient vegetation pixels in its ROI, then this prototype will generate two .csv files, one containing vegetation pixels percentage and another containing the average and standard deviation values of all bands; otherwise, this prototype will generate only one .csv file, containing vegetation pixels percentage.

**Input variables:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable name** | **Descriptive Name** | **Type** | **Source** | **Remarks** |
| df\_Site | .csv file of sites | Pandas. Dataframe | Site.csv | -Mandatory |
| image\_L1C\_B08  image\_L2A\_B04  image\_L2A\_B08 | Raw S2 images | Rasterio.io.DatasetReader | Input S2 Images | -Mandatory |
| \_threshold\_CV | The threshold used to flag a CV value | Float | Optional Input.ini | -Optional; can’t be negative; default 0.2 |
| \_area | The side-length (meters) of squared ROI | Int | Optional Input.ini | -Optional; must be a multiple of 10; default 900 |
| \_cloud | The cloud coverage threshold to determine whether a S2 image is usable | Float | Optional Input.ini | -Optional; must be between 0 and 1; default 0.5 |
| bool\_DeleteCache | A Boolean value which determines whether the cache files will be deleted upon completion of the code | Boolean | Optional Input.ini | -Optional; default False |
| Image.nc | Raw FLEX images | Xarray | Input FLEX Images | -Mandatory |

**Output variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable name** | **Descriptive Name** | **Type** | **Remarks** |
| df\_Output | Dataframe of output | Pandas,Dataframe | - |

**Applicable Version of ATBD**

**Applicable Version of Test Data Set**

* Version: V01.01.00
* Required Inputs:
  + Site.csv.
  + All S2 images inside “Input S2 Images” folder should be organized as follows: “Input S2 Images\Site Name\L1C\” and “Input S2 Images\Site Name\L2A\”. The L1C and L2A images must cover the exact same region and have the same dates.
  + All FLEX images inside “Input FLEX Images” folder.
* Optional Inputs:
  + Optional Input.ini. Open it in a text editor.
* Expected Outputs:
  + .csv

**Installation Instructions**

Programming language: Python3

Libraries: (Different versions of packages might work as well)

* Numpy >= 1.26.4
* pandas >= 2.2.2
* shapely >= 2.0.5
* geopandas >= 1.0.1
* matplotlib >= 3.9.1
* lxml >= 5.2.2
* beautifulsoup4 >= 4.12.3
* GDAL == 3.9.1
* rasterio == 1.3.10
* configparser >= 7.1.0
* xarray >= 2024.10.0
* scipy >= 1.14.1

Steps:

1. Make sure python3 is installed
2. Install the necessary python libraries using pip
   1. pip install -r /path/to/requirements.txt

**Execution Instructions:**

1. Create a virtual python environment using Anaconda or any other similar managers.
2. Ensure you have Python, and the necessary libraries installed in the virtual env. (see Installation Instructions)
3. If you don’t want to add more sites, skip this step and jump to step 6. Otherwise, open “Sites\_S2.csv” or “Sites\_FLEX.csv” and add their names, latitudes and longitudes. Otherwise, skip this step.

If there are multiple sets of images for the same site, you should write the same lines but with distinct names of the site inside the .csv file to match the number of the sets of images. For example, if there are 3 sets of S2 images for San Rossore 2 on different dates, you can write down “SR2-1”, “SR2-2” and “SR2-3” in the column of “Site”.

1. To process S2 images, you also need to create subfolders inside “Input S2 Images”. This step is not needed for FLEX images, and you can jump to step 6.

The folder structure is as follows: “Input S2 Images\Site Name\L1C\” and “Input S2 Images\Site Name\L2A\”. The L1C and L2A images must cover the exact same region and have the same dates. Then put the S2 images into the corresponding subfolders.

You can open the python file “Batch create subfolders.py” in and IDE inside the virtual python environment and run it, so that it will automatically help you create all subfolders following correct folder structure

1. Then put your S2 images into the corresponding subfolders.
2. If you don’t want to modify those optional variables, skip this step. If you want to do so, open “Optional Input.ini” and write your own values inside it, after reading the comments inside that .ini file.
3. Open the “Main.py” in an IDE inside the virtual python environment you have just created, like Spyder or VS Code, and run it.
4. Upon successful execution, the output results will be written in the file

For S2 images, expected output is:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Site | Valid Pixels L1C | Valid Pixels L2A | Valid Pixels Percentage L1C | Valid Pixels Percentage L2a | CV | Flag |
| FR-FBn | 8100 | 8100 | 1 | 1 | 0.16535 | 1 |
| GF-GUY Clouds | 2550 | 2550 | 0.307934 | 0.307934 |  |  |
| IT-BFt | 8100 | 8100 | 1 | 1 | 0.199723 | 1 |
| IT-SR2 | 8100 | 8100 | 1 | 1 | 0.251333 | 0 |
| IT-SR2 Clouds | 0 | 0 | 0 | 0 |  |  |

For FLEX images, expected output of a site can be:

|  |  |  |
| --- | --- | --- |
|  | Average | STD |
| Sif 1 | 0.151997 | 0.099979 |
| Sif 2 | 0.178971 | 0.118222 |
| Sif 3 | 0.209371 | 0.138718 |
| Sif … | … | … |

|  |  |
| --- | --- |
|  | Value |
| Vegetation Pixels | 761 |
| Vegetation Percentage | 0.791883 |

## Source Code

**Main.py**

import os

import time

import shutil

import configparser

import numpy as np

import pandas as pd

import rasterio as rio

import rasterio.mask

import xarray as xr

### Class

from Class import CALVALPrototype

from Class import FakeFLEX

### Main Code

start\_Time = time.time()

# Initiate the class

calval\_Prototype = CALVALPrototype()

# Read optional input

config = configparser.ConfigParser()

config.read(calval\_Prototype.path\_Optional)

if config["OptionalInput"]["threshold\_CV"]:

    calval\_Prototype.\_threshold\_CV = config["Optional Input"]["threshold\_CV"]

    print(f"The threshold of CV has been modified to {config["Optional Input"]["threshold\_CV"]}!")

if config["OptionalInput"]["threshold\_cloud"]:

    calval\_Prototype.\_cloud = config["Optional Input"]["threshold\_cloud"]

    print(f"The threshold of cloud coverage has been modified to {config["Optional Input"]["threshold\_cloud"]}!")

if config["OptionalInput"]["area\_ROI"]:

    calval\_Prototype.\_area = config["Optional Input"]["area\_ROI"]

    print(f"The ROI has been modified to {config["Optional Input"]["area\_ROI"]} m by {config["Optional Input"]["area\_ROI"]} m!")

if config["OptionalInput"]["bool\_DeleteCache"] == "True":

    calval\_Prototype.bool\_DeleteCache = True

    print("The cache foldeer will be deleted upon the completion of the code! ")

# Read .csv files to retrieve info of each site

df\_Site = calval\_Prototype.get\_SiteInfo()

# Initiate some empty lists to save the output values of each loop

list\_ValidPixelsL1C = []

list\_ValidPercentageL1C = []

list\_ValidPixelsL2A = []

list\_ValidPercentageL2A = []

list\_CV = []

list\_Flag = []

# For Sentinel-2 Images

if len(os.listdir(calval\_Prototype.path\_Image)) !=0 :

    # Start the main part, a loop that will iterate each site in our input .csv file

    for i in range(df\_Site.shape[0]):

        temp\_StartTime = time.time()

        # Get site name

        temp\_SiteName = df\_Site["Site"][i]

        temp\_SiteLat = df\_Site["Latitude"][i]

        temp\_SiteLon = df\_Site["Longitude"][i]

        print(f"The calculation and validation of site {temp\_SiteName} has started! ")

        # Get paths to B8 of L1C and B4, B8 of L2A

        path\_L1C\_B08\_raw, path\_L2A\_B04\_raw, path\_L2A\_B08\_raw, path\_L1C\_Mask, path\_L2A\_Mask, path\_L1C\_xml\_DS, path\_L1C\_xml\_TL, path\_L2A\_xml\_DS = calval\_Prototype.get\_PathImages(temp\_SiteName)

        # Read images

        image\_L1C\_B08 = rio.open(path\_L1C\_B08\_raw)

        image\_L2A\_B04 = rio.open(path\_L2A\_B04\_raw)

        image\_L2A\_B08 = rio.open(path\_L2A\_B08\_raw)

        # Get the values of the images

        values\_L1C\_B08 = image\_L1C\_B08.read(1)

        values\_L2A\_B04 = image\_L2A\_B04.read(1)

        values\_L2A\_B08 = image\_L2A\_B08.read(1)

        # Create a shapefile of our ROI and another one of the mask

        gdf\_ROI = calval\_Prototype.create\_Shapefile(image\_L1C\_B08, image\_L2A\_B04, temp\_SiteName, temp\_SiteLat, temp\_SiteLon)

        # Read masks of opaque clouds, cirrus clouds and snow ice areas

        mask\_L1C = rio.open(path\_L1C\_Mask)

        mask\_L2A = rio.open(path\_L2A\_Mask)

        mask\_L1C\_OpaqueClouds = mask\_L1C.read(1)

        mask\_L1C\_CirrusClouds = mask\_L1C.read(2)

        mask\_L1C\_SnowIceAreas = mask\_L1C.read(3)

        mask\_L2A\_OpaqueClouds = mask\_L2A.read(1)

        mask\_L2A\_CirrusClouds = mask\_L2A.read(2)

        mask\_L2A\_SnowIceAreas = mask\_L2A.read(3)

        # Check if all three masks are empty. If not empty, we should check if the masked

        mask\_L2ACombined = mask\_L2A\_OpaqueClouds + mask\_L2A\_CirrusClouds + mask\_L2A\_SnowIceAreas

        mask\_L1CCombined = mask\_L1C\_OpaqueClouds + mask\_L1C\_CirrusClouds + mask\_L1C\_SnowIceAreas

        temp\_PassL1C, temp\_ValidPixelsL1C, temp\_ValidPixelsPercentageL1C = calval\_Prototype.cal\_ValidPixels(temp\_SiteName, image\_L1C\_B08, mask\_L1CCombined, gdf\_ROI, note = "L1C")

        temp\_PassL2A, temp\_ValidPixelsL2A, temp\_ValidPixelsPercentageL2A = calval\_Prototype.cal\_ValidPixels(temp\_SiteName, image\_L2A\_B04, mask\_L2ACombined, gdf\_ROI, note = "L2A")

        list\_ValidPixelsL1C.append(temp\_ValidPixelsL1C)

        list\_ValidPixelsL2A.append(temp\_ValidPixelsL2A)

        list\_ValidPercentageL1C.append(temp\_ValidPixelsPercentageL1C)

        list\_ValidPercentageL2A.append(temp\_ValidPixelsPercentageL2A)

        if temp\_PassL1C and temp\_PassL2A:

            # NDVI, Rad, NIRv

            temp\_NDVI = calval\_Prototype.cal\_L2ANDVI(path\_L2A\_xml\_DS, values\_L2A\_B04, values\_L2A\_B08)

            temp\_Rad = calval\_Prototype.cal\_L1CRad(path\_L1C\_xml\_DS, path\_L1C\_xml\_TL, values\_L1C\_B08)

            temp\_NIRv = temp\_NDVI \* temp\_Rad

            print(f"The NIRv of {temp\_SiteName} has been calculated successfully!")

            # Save NIRv.tif to cache folder

            src = image\_L1C\_B08

            out\_meta = src.meta

            out\_meta.update({

                "driver": "GTiff",

                "dtype": 'float64'

            })

            with rio.open(calval\_Prototype.path\_Cache + "\\" + temp\_SiteName + "\\NIRv.tif", 'w', \*\*out\_meta) as dest:

                dest.write(temp\_NIRv, 1)

            # Clip the NIRv.tif

            image\_NIRv = rio.open(calval\_Prototype.path\_Cache + "\\" + temp\_SiteName + "\\NIRv.tif")

            calval\_Prototype.clip\_RasterbySHP(temp\_SiteName, image\_NIRv, gdf\_ROI, suffix = "NIRv ROI")

            # Read the clipped ROI NIRv.tif

            image\_NIRv\_ROI = rio.open(calval\_Prototype.path\_Cache + "\\" + temp\_SiteName + "\\NIRv ROI.tif")

            values\_NIRv\_ROI = image\_NIRv\_ROI.read(1)

            temp\_CV = calval\_Prototype.cal\_CV(values\_NIRv\_ROI)

            temp\_Flag = calval\_Prototype.cal\_Flag(temp\_CV)

            list\_CV.append(temp\_CV)

            list\_Flag.append(temp\_Flag)

            temp\_EndTime = time.time()

            temp\_ElapsedTime = temp\_EndTime - temp\_StartTime

            print(f"The calculation and validation of site {temp\_SiteName} has been finished successfully, which took {temp\_ElapsedTime:.2f} seconds! ")

        else:

            list\_CV.append(None)

            list\_Flag.append(None)

            continue

    # Loop finished, now we save the output to a new .csv file

    df\_Output = pd.DataFrame({

        "Site": list(df\_Site["Site"]),

        "Valid Pixels L1C": list\_ValidPixelsL1C,

        "Valid Pixels L2A": list\_ValidPixelsL2A,

        "Valid Pixels Percentage L1C": list\_ValidPercentageL1C,

        "Valid Pixels Percentage L2A": list\_ValidPercentageL2A,

        "CV": list\_CV,

        "Flag": list\_Flag

    })

    print(f"Please find the final output.csv in the following folder: {calval\_Prototype.path\_Output}")

    df\_Output.to\_csv(calval\_Prototype.path\_Output + "\\Output.csv", index = False)

    # Elapsed Time

    end\_Time = time.time()

    elapsed\_Time = end\_Time - start\_Time

    num\_Sites = df\_Site.shape[0]

    average\_Time = elapsed\_Time / num\_Sites

    # Delete cache folder?

    if calval\_Prototype.bool\_DeleteCache:

        del image\_L1C\_B08, image\_L2A\_B04, image\_L2A\_B08, image\_NIRv, image\_NIRv\_ROI

        shutil.rmtree(calval\_Prototype.path\_Cache)

        print("The cache folder and all its contents has been deleted permanently! ")

    print(f"This python code has finished its work, and in totale it has taken {elapsed\_Time:.2f}!")

    print(f"All {num\_Sites} sites have been validated, and the average process time for each site is {average\_Time:.2f} seconds! ")

# For Fake FLEX Images

fakeFLEX = FakeFLEX()

if len(os.listdir(fakeFLEX.path\_Original)) !=0 :

    list\_VegCount = []

    list\_Sif = []

    # Read .csv files to retrieve info of each site

    df\_Site = fakeFLEX.get\_SiteInfo()

    # Start the main part, a loop that will iterate each site in our input .csv file

    for i in range(df\_Site.shape[0]):

        temp\_StartTime = time.time()

        # Get site name

        temp\_SiteName = df\_Site["Site"][i]

        print(f"{temp\_SiteName} starts!")

        temp\_SiteLat = df\_Site["Latitude"][i]

        temp\_SiteLon = df\_Site["Longitude"][i]

        gdf\_Site = fakeFLEX.create\_SiteShp(temp\_SiteLat,temp\_SiteLon)

        ds = xr.open\_dataset(fakeFLEX.path\_Original + "\\" + os.listdir(fakeFLEX.path\_Original)[i])

        print(f"{os.listdir(fakeFLEX.path\_Original)[i]} opened succesfully!")

        lat = ds['Lat'].values

        lon = ds['Lon'].values

        veg = ds['Veg'].values

        gdf\_NC = fakeFLEX.create\_NCShp(veg, lat, lon)

        index\_Original\_1, index\_Original\_2, index\_Resampled\_1, index\_Resampled\_2 = fakeFLEX.locateSite(fakeFLEX.findNearestPoint(gdf\_Site,gdf\_NC), lat, lon)

        veg\_Count = fakeFLEX.cal\_Veg(veg, index\_Original\_1, index\_Original\_2)

        if veg\_Count / 900 >= 0.5:

            print(f"There are {veg\_Count} valid vegetation pixels, and the valid percentage is {veg\_Count / 900:.2%}, greater than 50%, so we keep this image!")

            ds\_Resampled = xr.open\_dataset(fakeFLEX.path\_Resampled + "\\" + os.listdir(fakeFLEX.path\_Resampled)[i])

            sif = ds\_Resampled['Sif'].values

            for num\_Sif in range(sif.shape[0]):

                temp\_Sif = sif[num\_Sif]

                sif\_ROI = temp\_Sif[(index\_Resampled\_1 - 1):(index\_Resampled\_1 + 2), (index\_Resampled\_2 - 1):(index\_Resampled\_2 + 2)]

                sif\_ROI\_Avg = np.average(sif\_ROI).item()

                sif\_ROI\_STD = np.std(sif\_ROI).item()

                list\_Sif.append([sif\_ROI\_Avg,sif\_ROI\_STD])

            index\_labels = [f'Sif {i+1}' for i in range(len(list\_Sif))]

            df\_Sif = pd.DataFrame(list\_Sif, index = index\_labels, columns = ['Average','STD'])

            df\_Veg = pd.DataFrame(

                [veg\_Count, veg\_Count / 961],

                index = ["Vegetation Pixels", "Vegetation Percentage"],

                columns = ['Value']

            )

            df\_Sif.to\_csv(fakeFLEX.path\_Output + "\\" + temp\_SiteName + " - Sif.csv", index = True)

            df\_Veg.to\_csv(fakeFLEX.path\_Output + "\\" + temp\_SiteName + " - Vegetation Pixel.csv", index = True)

        else:

            print(f"There are {veg\_Count} valid vegetation pixels, and the valid percentage is {veg\_Count / 961:.2%}, not greater than 50%, so we abandone this image!")

            df\_Veg = pd.DataFrame(

                [veg\_Count, veg\_Count / 961],

                index = ["Vegetation Pixels", "Vegetation Percentage"],

                columns = ['Value']

            )

            df\_Veg.to\_csv(fakeFLEX.path\_Output + "\\" + temp\_SiteName + " - Vegetation Pixel.csv", index = True)

**Class.py**

import os

from bs4 import BeautifulSoup

import math

import numpy as np

import shapely as shp

from shapely.geometry import Point

import pandas as pd

import geopandas as gpd

import rasterio as rio

import rasterio.mask

from scipy.spatial import cKDTree

class CALVALPrototype:

    # Initialization

    def \_\_init\_\_(self):

        # Set a threshold of CV. You can change this value according to your own need, but note that there is no sense if the threshold is not greater than 0. The threshold of CV is set to 0.2 by default.

        self.\_threshold\_CV = 0.2

        # Set the side-length of the ROI. The ROI must be a squared area. This value must be a multiple of 10. The side-length of the ROI is 900 meters by default.

        self.\_area = 900

        # Cloud coverage

        self.\_cloud = 0.5

        # The path to some essential folders and files. Don't modify these unless you know what you are going to do.

        # The absolute path of the current working directory

        self.path\_Main = os.path.realpath(os.path.dirname(\_\_file\_\_))

        # The absolute path of the output

        self.path\_Output = self.path\_Main + "\\Output"

        # The absolute path of the S2 images

        self.path\_Image = self.path\_Main + "\\Input S2 Images"

        # The absolute path to the input .csv file, where the info of all sites are saved.

        self.path\_SiteCSV = self.path\_Main + "\\Site\_S2.csv"

        # The absolute path to the folder, where interim files are saved. If you want to modify this path, be cautious.

        self.path\_Cache = self.path\_Main + "\\Cache"

        # The absolute path to the folder of "Main - Optional Input.ini"

        self.path\_Optional = self.path\_Main + "\\Optional Input.ini"

        # A boolean variable which determines whether the cache files will be deleted unpon the completion of the code. False by default.

        self.bool\_DeleteCache = False

        if not os.path.exists(self.path\_Cache):

            print("Creating cache folder")

            os.makedirs(self.path\_Cache)

        else:

            if not self.bool\_DeleteCache:

                print("The cache files will be saved in the following folder: " + self.path\_Cache)

        if not os.path.exists(self.path\_Output):

            print("Creating cache folder")

            os.makedirs(self.path\_Output)

        def cache\_Folder():

            temp\_df = pd.read\_csv(self.path\_SiteCSV)

            for i in range(temp\_df.shape[0]):

                temp\_Name = temp\_df["Site"][i]

                if not os.path.exists(self.path\_Cache + "\\" + temp\_Name):

                    os.makedirs(self.path\_Cache + "\\" + temp\_Name)

            if not self.bool\_DeleteCache:

                print("Subfolders created successfully inside cache folder!")

        cache\_Folder()

    @property

    def threshold\_CV(self):

        return self.\_threshold\_CV

    @property

    def area(self):

        return self.\_area

    @property

    def cloud(self):

        return self.\_cloud

    @threshold\_CV.setter

    def threshold\_CV(self, value):

        if value <=0:

            raise ValueError("The threshold of CV should be greater than 0!!!")

        self.\_threshold\_CV = value

    @area.setter

    def area(self, value):

        if value % 10 != 0:

            raise ValueError("The side-length of ROI must be a multiple of 10 meters!!!")

        self.\_area = value

    @cloud.setter

    def cloud(self, value):

        if value < 0 or value > 1:

            raise ValueError("The cloud coverage must be between 0 and 1!!!")

        self.\_cloud = value

    # Get names, lat, lon of all sites from .csv file, returning a pandas dataframe

    def get\_SiteInfo(self):

        temp\_CSV = pd.read\_csv(self.path\_SiteCSV)

        temp\_SiteName = list(temp\_CSV["Site"])

        temp\_SiteLat = list(temp\_CSV["Latitude"])

        temp\_SiteLon = list(temp\_CSV["Longitude"])

        if len(temp\_SiteName) == len(temp\_SiteLat) == len(temp\_SiteLon):

            return temp\_CSV

        else:

            print("User Error: Please make sure there is no missing data in the .csv file!")

            return

    # Create subfolders inside Input S2 Images

    def create\_InputSubfolders(self, dataFrame):

        for i in range(dataFrame.shape[0]):

            if not os.path.exists(self.path\_Image + "\\" + dataFrame.loc[i,"Site"]):

                os.makedirs(self.path\_Image + "\\" + dataFrame.loc[i,"Site"])

                os.makedirs(self.path\_Image + "\\" + dataFrame.loc[i,"Site"] + "\\L2A")

                os.makedirs(self.path\_Image + "\\" + dataFrame.loc[i,"Site"] + "\\L1C")

                print(f"Subfolders of {dataFrame.loc[i,"Site"]} have been created succesfully inside input S2 image folder!")

    # Get paths to S2 images and mask images. In this case we need B8 image of L1C and B4, B8 images of L2A

    def get\_PathImages(self, site\_Name):

        # L1C

        temp\_PathL1C = self.path\_Image + "\\" + site\_Name + "\\L1C"

        for path, subdirs ,files in os.walk(temp\_PathL1C):

            for name in files:

                temp = os.path.join(path, name)

                if "IMG\_DATA" in temp and temp[-3:] == 'jp2' and "B08" in temp:

                    path\_L1C\_B08\_raw = temp

                # Path to the mask file

                if "MSK\_CLASSI\_B00" in temp and temp[-3:] == 'jp2':

                    path\_L1C\_Mask = temp

                # Get the path to the XML file "MTD\_DS" of L1C raster, where there are values of "U", "Solar Irradiance", "Quantification Value" and "Radiometric Offset"

                if "MTD\_DS.xml" in temp:

                    path\_L1C\_xml\_DS = temp

                # Get the path to the XML file "MTD\_TL" of L1C raster, where there are values (matrices) of "Solar Zenith Angle".

                if "MTD\_TL.xml" in temp:

                    path\_L1C\_xml\_TL = temp

        # L2A

        temp\_PathL2A = self.path\_Image + "\\" + site\_Name + "\\L2A"

        for path, subdirs, files in os.walk(temp\_PathL2A):

            for name in files:

                temp = os.path.join(path, name)

                if temp[-3:] == 'jp2'in temp and "10m" in temp and "B04" in temp :

                    path\_L2A\_B04\_raw = temp

                if temp[-3:] == 'jp2'in temp and "10m" in temp and "B08" in temp :

                    path\_L2A\_B08\_raw = temp

                # Path to the mask file

                if "MSK\_CLASSI\_B00" in temp and temp[-3:] == 'jp2':

                    path\_L2A\_Mask = temp

                # Get the path to the XML file "MTD\_DS" of L1C raster, where there are values of "Quantification Value" and "Radiometric Offset"

                if "MTD\_DS.xml" in temp:

                    path\_L2A\_xml\_DS = temp

        if not os.path.exists(temp\_PathL1C) or not os.path.exists(temp\_PathL2A):

            print("User Error: Please organise the input S2 images in correct folder hierarchy. Input S2 Images\\SiteName\\L1C\\ and Input S2 Images\\SiteName\\L2A\\")

            return

        else:

            return path\_L1C\_B08\_raw, path\_L2A\_B04\_raw, path\_L2A\_B08\_raw, path\_L1C\_Mask, path\_L2A\_Mask, path\_L1C\_xml\_DS, path\_L1C\_xml\_TL, path\_L2A\_xml\_DS

    # Create a ROI shapefile, centered at the site. You can modify the side-length of the shapefile by changing the value of "area" of this class.

    def create\_Shapefile(self, img\_L1C, img\_L2A, site\_Name, site\_Lat, site\_Lon, bool\_toSave = False):

        # Get the crs of input L1C image and L2A image

        crs\_L1C = img\_L1C.crs.data["init"].split(":")[1]

        crs\_L2A = img\_L2A.crs.data["init"].split(":")[1]

        # In the case that L1C and L2A have different crs, give an error. But this won't happen if the input images are correct.

        if crs\_L2A != crs\_L1C:

            raise SystemExit("Stop right there!")

            return

        crs\_Final = 'EPSG:' + crs\_L1C

        # Create a point shapefile based on the site, using Lon-Lat

        df\_4326 = pd.DataFrame({

            "Site": [site\_Name],

            "Latitude": [site\_Lat],

            "Longitude": [site\_Lon]

        })

        gdf\_4326 = gpd.GeoDataFrame(

            df\_4326,

            geometry = gpd.points\_from\_xy(df\_4326['Longitude'], df\_4326['Latitude']),

            crs = "EPSG:4326"

        )

        gdf\_New = gdf\_4326.copy()

        gdf\_New = gdf\_New.to\_crs(crs\_Final)

        # First we retrieve the x, y coordinate of our site

        site\_x = gdf\_New.geometry.x.values[0]

        site\_y = gdf\_New.geometry.y.values[0]

        site\_row, site\_col = img\_L2A.index(site\_x, site\_y)

        site\_pixel\_x, site\_pixel\_y = img\_L2A.xy(site\_row, site\_col)

        # Calculate the "cardinal" distance

        side\_length\_half = self.\_area / 2

        if side\_length\_half % 2 == 0:

            # If the half of the side length is even, we need to add another 5 meters to make sure the pixels on the borders will not be omitted when we clip the raster images.

            length\_Cardinal = side\_length\_half + 5

        else:

            length\_Cardinal = side\_length\_half

        site\_x\_left\_New = site\_pixel\_x - length\_Cardinal

        site\_x\_right\_New = site\_pixel\_x + length\_Cardinal

        site\_y\_top\_New = site\_pixel\_y + length\_Cardinal

        site\_y\_bottom\_New = site\_pixel\_y - length\_Cardinal

        # Create a bounding box

        shp\_New = shp.box(site\_x\_left\_New, site\_y\_bottom\_New, site\_x\_right\_New, site\_y\_top\_New)

        # Create shapefile!

        gdf\_New = gpd.GeoDataFrame(

            pd.DataFrame({"0": ["0"]}),

            geometry=[shp\_New],

            crs = crs\_Final

        )

        return gdf\_New

    def cal\_L2ANDVI(self, path\_L2A\_xml\_DS, values\_L2A\_B04, values\_L2A\_B08):

        # Read the DS xml file of L2A

        with open(path\_L2A\_xml\_DS, 'r') as f:

            data = f.read()

        BS\_L2A\_dS = BeautifulSoup(data, "xml")

        # Get the quantification value!

        quantification\_L2A = int(BS\_L2A\_dS.find("BOA\_QUANTIFICATION\_VALUE").text)

        # Get the radiometric offset!

        offset\_L2A\_B04 = int(BS\_L2A\_dS.find("BOA\_ADD\_OFFSET", {"band\_id": "3"}).text)

        offset\_L2A\_B08 = int(BS\_L2A\_dS.find("BOA\_ADD\_OFFSET", {"band\_id": "7"}).text)

        # Calculate NDVI of L2A!

        temp\_NDVI = ((values\_L2A\_B08 + offset\_L2A\_B08).astype(float) / quantification\_L2A - (values\_L2A\_B04 + offset\_L2A\_B04).astype(float) / quantification\_L2A) / ((values\_L2A\_B08 + offset\_L2A\_B08).astype(float) / quantification\_L2A + (values\_L2A\_B04 + offset\_L2A\_B04).astype(float) / quantification\_L2A )

        return temp\_NDVI

    def cal\_L1CRad(self, path\_L1C\_xml\_DS, path\_L1C\_xml\_TL, values\_L1C\_B08):

        with open(path\_L1C\_xml\_DS, 'r') as f:

            data = f.read()

        BS\_L1C\_dS = BeautifulSoup(data, "xml")

        # Get the quantification value!

        quantification\_L1C = int(BS\_L1C\_dS.find("QUANTIFICATION\_VALUE").text)

        # Get the radiometric offset!

        offset\_L1C = int(BS\_L1C\_dS.find("RADIO\_ADD\_OFFSET", {"band\_id": "7"}).text)

        # Get the U

        U\_L1C = float(BS\_L1C\_dS.find("U").text)

        # Get the solar irradiance

        SolarIrr = float(BS\_L1C\_dS.find("SOLAR\_IRRADIANCE", {"bandId": "7"}).text)

        # Read the TL xml file of L1C

        with open(path\_L1C\_xml\_TL, 'r') as f:

            data = f.read()

        BS\_L1C\_dS = BeautifulSoup(data, "xml")

        # Get the sun zenith angle! There should be a 23 x 23 arrays in the xml. Now we save each row as an array and keep all these arrays into a list

        list\_SunZenith = []

        for row in BS\_L1C\_dS.find("Sun\_Angles\_Grid").find("Zenith").find\_all("VALUES"):

            temp\_List = row.text.split(" ")

            temp\_Arr = np.array(temp\_List)

            temp\_Arr = temp\_Arr.astype(float)

            list\_SunZenith.append(temp\_Arr)

        # Now we stack these nested-in-list arrays into a 2d array

        index = 0

        for arr in list\_SunZenith:

            if index == 0:

                arr\_SunZenith = arr

            else:

                arr\_SunZenith = np.vstack((arr\_SunZenith, arr))

            index = index + 1

        # Get the shape of L1C image, which should be (10980, 10980)

        shape\_L1C = values\_L1C\_B08.shape

        # Repeat each element of sun zenith angle array, in both axies. The final array should have a shape of (11500, 11500)

        arr\_SunZenith\_Repeat = np.repeat(arr\_SunZenith, 500, axis = 1)

        arr\_SunZenith\_Repeat = np.repeat(arr\_SunZenith\_Repeat, 500, axis = 0)

        # Index only the first 10980 of each dimension

        arr\_SunZenith\_Assigned = arr\_SunZenith\_Repeat[0:shape\_L1C[0], 0:shape\_L1C[1]]

        # radiance = reflectance \* cos(radians(SunZenithAngle)) \* solarIrradiance \* U / pi

        temp\_Radiance = (values\_L1C\_B08 + offset\_L1C).astype(float)  \* np.cos(np.radians(arr\_SunZenith\_Assigned)) \* SolarIrr / quantification\_L1C / (math.pi \* (1 / U\_L1C))

        return temp\_Radiance

    def clip\_RasterbySHP(self, site\_Name, raster, shp, suffix = None):

        out\_image, out\_transform = rio.mask.mask(raster, shp.geometry, crop=True)

        out\_meta = raster.meta

        out\_meta.update({"driver": "GTiff",

                        "height": out\_image.shape[1],

                        "width": out\_image.shape[2],

                        "transform": out\_transform})

        with rio.open(self.path\_Cache + "\\" + site\_Name + "\\" + suffix + ".tif", "w", \*\*out\_meta) as dest:

            dest.write(out\_image)

        return

    def cal\_ValidPixels(self, site\_Name, raster, mask\_Combined, shp, note = None):

        if np.max(mask\_Combined) >= 1:

            # Upscale 60mx60m mask to 10mx10m without modifying any pixel values

            mask\_Combined\_Upscale = np.repeat(mask\_Combined, 6, axis = 0)

            mask\_Combined\_Upscale = np.repeat(mask\_Combined\_Upscale, 6, axis = 1)

            # Save this mask

            mask\_meta = raster.meta

            with rio.open(self.path\_Cache + "\\" + site\_Name + "\\Mask.tif", "w", \*\*mask\_meta) as dest:

                dest.write(mask\_Combined\_Upscale, indexes = 1)

            # clip and save

            temp\_Mask = rio.open(self.path\_Cache + "\\" + site\_Name + "\\Mask.tif")

            self.clip\_RasterbySHP(site\_Name, temp\_Mask, shp, suffix = "Mask ROI")

            temp\_MaskClipped = rio.open(self.path\_Cache + "\\" + site\_Name + "\\Mask ROI.tif")

            temp\_MaskCombined = temp\_MaskClipped.read(1)

            if np.max(temp\_MaskCombined) >= 1:

                temp\_ValidPixels = np.count\_nonzero(temp\_MaskCombined == 0)

                temp\_InvalidPixels = np.count\_nonzero(temp\_MaskCombined != 0)

                temp\_TotalPixels = temp\_ValidPixels + temp\_InvalidPixels

                temp\_ValidPixelsRatio = temp\_ValidPixels / temp\_TotalPixels

                print(f"There are {temp\_ValidPixels} valid pixels in the S2 {note} image of {site\_Name}!")

                if temp\_ValidPixelsRatio >= self.\_cloud:

                    print(f"But the ratio of valid pixels is {temp\_ValidPixelsRatio:.2%}, equal to or greater than {self.\_cloud:.2%}, so we can use these S2 images. ")

                    temp\_Pass = True

                    return temp\_Pass, temp\_ValidPixels, temp\_ValidPixelsRatio

                else:

                    print(f"And the ratio of valid pixels is {temp\_ValidPixelsRatio:.2%}, lower than {self.\_cloud:.2%}, so we can't use these S2 images and hence we can't proceed. ")

                    temp\_Pass = False

                    return temp\_Pass, temp\_ValidPixels, temp\_ValidPixelsRatio

            else:

                print(f"All pixels in the S2 {note} image of {site\_Name} are valid! ")

                temp\_Pass = True

                return temp\_Pass, (self.\_area / 10) \*\* 2, 1

        else:

            print(f"All pixels in the S2 {note} image of {site\_Name} are valid! ")

            temp\_Pass = True

            return temp\_Pass, (self.\_area / 10) \*\* 2, 1

    def cal\_CV(self, value):

        return np.std(value) / np.mean(value)

    def cal\_Flag(self, value):

        if value <= self.\_threshold\_CV:

            return 1

        else:

            return 0

class FakeFLEX:

    # Initialization

    def \_\_init\_\_(self):

        # The path to some essential folders and files. Don't modify these unless you know what you are going to do.

        # The absolute path of the current working directory

        self.path\_Main = os.path.realpath(os.path.dirname(\_\_file\_\_))

        # The absolute path of the output

        self.path\_Output = self.path\_Main + "\\Output"

        # The absolute path of the fake FLEX images

        self.path\_Original = self.path\_Main + "\\Input Fake FLEX Images\\Original"

        self.path\_Resampled = self.path\_Main + "\\Input Fake FLEX Images\\Resampled"

        # The absolute path to the input .csv file, where the info of all sites are saved.

        self.path\_SiteCSV = self.path\_Main + "\\Site\_FLEX.csv"

    @property

    def threshold\_CV(self):

        return self.\_threshold\_CV

    @property

    def area(self):

        return self.\_area

    @property

    def cloud(self):

        return self.\_cloud

        # Get names, lat, lon of all sites from .csv file, returning a pandas dataframe

    def get\_SiteInfo(self):

        temp\_CSV = pd.read\_csv(self.path\_SiteCSV)

        temp\_SiteName = list(temp\_CSV["Site"])

        temp\_SiteLat = list(temp\_CSV["Latitude"])

        temp\_SiteLon = list(temp\_CSV["Longitude"])

        if len(temp\_SiteName) == len(temp\_SiteLat) == len(temp\_SiteLon):

            return temp\_CSV

        else:

            print("User Error: Please make sure there is no missing data in the .csv file!")

            return

    def create\_SiteShp(self, lat,lon):

        df = pd.DataFrame({

            "Latitude": [lat],

            "Longitude": [lon]

        })

        gdf = gpd.GeoDataFrame(

            data = [[0]],

            crs = "EPSG:4326",

            geometry=gpd.points\_from\_xy(df['Longitude'],df['Latitude'])

        )

        return gdf

    def create\_NCShp(self, veg, np\_lat, np\_lon):

        df = pd.DataFrame({

            "Veg": veg.reshape(-1),

            "Latitude": np\_lat.reshape(-1),

            "Longitude": np\_lon.reshape(-1)

        })

        gdf = gpd.GeoDataFrame(

            data = df,

            crs = "EPSG:4326",

            geometry=gpd.points\_from\_xy(df['Longitude'],df['Latitude'])

        )

        return gdf

    def findNearestPoint(self, gdfA, gdfB):

        nA = np.array(list(gdfA.geometry.apply(lambda x: (x.x, x.y))))

        nB = np.array(list(gdfB.geometry.apply(lambda x: (x.x, x.y))))

        btree = cKDTree(nB)

        dist, idx = btree.query(nA, k=1)

        gdfB\_nearest = gdfB.iloc[idx].drop(columns="geometry").reset\_index(drop=True)

        gdf = pd.concat(

            [

                gdfA.reset\_index(drop=True),

                gdfB\_nearest,

                pd.Series(dist, name='dist')

            ],

            axis=1)

        return gdf

    # Find the pixel where the site is located

    def locateSite(self, gdf,lat,lon):

        lat\_Site = gdf['Latitude'].values

        lon\_Site = gdf['Longitude'].values

        for m in range(1000):

            for n in range(1000):

                if (lat[m][n] == lat\_Site) and (lon[m][n] == lon\_Site):

                    index\_1 = m

                    index\_2 = n

                    break

        index\_1\_Resampled = index\_1 // 10

        index\_2\_Resampled = index\_2 // 10

        print(f"The site falls on {index\_1} row and {index\_2} column!")

        return index\_1, index\_2, index\_1\_Resampled, index\_2\_Resampled

    def cal\_Veg(self, veg, index\_1, index\_2):

        veg\_ROI = veg[(index\_1 - 15):(index\_1 + 16), (index\_2 - 15):(index\_2 + 16)]

        # veg = 1 is vegetation pixel

        veg\_Count = sum(veg\_ROI.reshape(-1))

        return veg\_Count

**Batch create subfolders.py**

from Class import CALVALPrototype

calval\_Prototype = CALVALPrototype()

df\_Site = calval\_Prototype.get\_SiteInfo()

calval\_Prototype.create\_InputSubfolders(df\_Site)

## TDS and Auxiliar Files

**Example of installation file:**

**requirements.txt:**

Numpy >= 1.26.4

pandas >= 2.2.2

shapely >= 2.0.5

geopandas >= 1.0.1

matplotlib >= 3.9.1

lxml >= 5.2.2

beautifulsoup4 >= 4.12.3

GDAL == 3.9.1

rasterio == 1.3.10

configparser >= 7.1.0

xarray >= 2024.10.0

scipy >= 1.14.1

**Input files:**

**Site\_S2.csv:**

|  |  |  |
| --- | --- | --- |
| Site | Latitude | Longitude |
| FR-FBn | 43.24079 | 5.67865 |
| GF-GUY Clouds | 5.2787 | -52.9248 |
| IT-BFt | 45.19775 | 10.74197 |
| IT-SR2 | 43.732 | 10.291 |
| IT-SR2 Clouds | 43.732 | 10.291 |

**Site\_FLEX.csv**

|  |  |  |
| --- | --- | --- |
| Site | Latitude | Longitude |
| castelPorziano | 41.7043 | 12.3573 |
| JolandaDiSavoia | 44.87431 | 11.9792 |
| Nebraska | 41.1797 | -96.4404 |
| SanRossore | 43.732 | 10.291 |

**Input S2 Images\:**

FR-FBn\L1C\

FR-FBn\L2A\

GF-GUY Clouds\L1C\

GF-GUY Clouds\L1C\

…\…\

**Input FLEX Images\**

**…**

**Optional Input.ini**

[OptionalInput]

; if left blank, these variables will use their default values.

; set the threshold of coefficient of variation (CV). If the CV is lower than this threshold, the code flags it as 1. Otherwise it will be flagged as 0.

; It must be a positive value.

; Default 0.2

threshold\_CV =

; set the threshold of cloud coverage. If the cloud coverage of an image is lower than this threshold, the code will pass the calculation of CV on that image.

; It must be a value between 0 and 1 (can be equal to either 0 or 1).

; Default 0.5

threshold\_cloud =

; set the side-length (meters) of the squared ROI.

; It must be a multiple of 10m since we are using Sentinel-2 images whose spatial resolution is 10m.

; Default 900 (m)

area\_ROI =

; set this to True so that the "Cache" folder will be deleted when the code completes its work.

; It must be either True or False.

; Default False

bool\_DeleteCache = True

**Output file:**

**Output.csv**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Site | Valid Pixels L1C | Valid Pixels L2A | Valid Pixels Percentage L1C | Valid Pixels Percentage L2a | CV | Flag |
| FR-FBn | 8100 | 8100 | 1 | 1 | 0.16535 | 1 |
| GF-GUY Clouds | 2550 | 2550 | 0.307934 | 0.307934 |  |  |
| IT-BFt | 8100 | 8100 | 1 | 1 | 0.199723 | 1 |
| IT-SR2 | 8100 | 8100 | 1 | 1 | 0.251333 | 0 |
| IT-SR2 Clouds | 0 | 0 | 0 | 0 |  |  |