# CAL/VAL Prototype

## README File

**Change log**

January 07, 2024, V01.01.01

* Improved support for FLEX images.
* Installation & codes instructions improved.

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) L1C and L2A images and FLEX images.

First, this prototype processes all FLEX images and calculates the average and the standard deviation of the value “Sif Emission Spectrum“ inside the ROI. Besides, it will extract the date and the time of the FLEX images whose vegetation pixels inside the ROI are greater than a threshold.

After the process of FLEX images, this prototype will search for S2 images whose dates and time are the same to that of FLEX images; if there are not the same ones, it will find the S2 images with the nearest dates and time. Then, it will check 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.

The final output files are unanimously in .csv format. For FLEX images, this prototype will generate a .csv file for each FLEX image containing the average and the standard deviation of the value “Sif Emission Spectrum“ inside the ROI. After processing all FLEX images, it will create a single .csv file to record the dates and the time of the FLEX images that have sufficient vegetation pixels inside the ROI.

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 |
| VegetationPixel.nc | Vegetation Pixel mask image file | netcdf | Input Images | -Mandatory |
| Image.nc | Raw FLEX images | netcdf | Input FLEX 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 |
| \_vegetationPixel | The vegetation pixel threshold to determine whether a FLEX image is usable | Float | Optional Input.ini | -Optional; must be between 0 and 1; default 0.5 |

**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: V01.01.01
* Required Inputs:
  + Site.csv.
  + All S2 images inside “Input S2 Images” folder should be organized as follows: “Input S2 Images\$SiteName\$DateTTime\L1C\” and “Input S2 Images\$SiteName\$DateTTime\L2A\”. Change the part $SiteName to the actual names of sites, and $DateTTIme to the actual dates and times of your S2 images such as “20230821T100601”. The L1C and L2A images must cover the exact same region and have the same dates.
  + All FLEX images inside “Input FLEX Images” folder, organized as follows”Input FLEX Images\Site Name\”.
* Optional Inputs:
  + Optional Input.ini. Open it in a text editor.
* Expected Outputs:
  + SiteName\FlexImageName – sif.csv
  + Output\_S2.csv
  + Usable FLEX Images.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
3. Alternatively, if you are using Anaconda, you can directly create a virtual environment using “environment.yml”.

**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 the step 4. Otherwise, open “Sites.csv” and add their names, latitudes and longitudes. The “Site Name” must be unique.
4. For input FLEX images, firstly, you must create subfolders using all available site names inside “Input FLEX Images”, and then you can put your FLEX images in corresponding subfolders. Make sure the folder structure is identical to “Input Flex Images\SanRossore\PRS\_TD\_20230616\_101431.nc”.
5. To process S2 images, you also need to create subfolders using all available site names inside “Input S2 Images”.

The folder structure is as follows: “Input S2 Images\$SiteName\$DateTTime\L1C\” and “Input S2 Images\$SiteName\$DateTTime\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.

1. Then put your S2 images into the corresponding subfolders.
2. If you don’t want to modify those optional variables, skip this step and jump to the step 8. If you want, open “Optional Input.ini” and write your own values inside it, after carefully reading the comments inside that .ini file.
3. Open the “Main.py” in an IDE (such as Spyder or Visual Studio) inside the virtual python environment you have just created, like Spyder or VS Code, and run it.
4. Upon successful execution, the output results can be found inside the folder “Output”.

For FLEX images, expected output of a input image can be:

|  |  |  |
| --- | --- | --- |
| SIF | Average | STD |
| Sif Emission Spectrum\_sif\_wavelength\_grid=670 | 0.275985 | 0.043109 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=672 | 0.326899 | 0.051894 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=674 | 0.384171 | 0.061756 |
| Sif … | … | … |

Usable FLEX Images.csv

|  |  |  |  |
| --- | --- | --- | --- |
| Site | FLEX Filename | FLEX Date | FLEX Time |
| SanRossore | **PRS\_TD\_20230616\_101431.nc** | **20230616** | **101431** |

Output\_S2.csv

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Site | FLEX Filename | Valid Pixels L1C | Valid Pixels L2A | Valid Pixels Percentage L1C | Valid Pixels Percentage L2A | CV | Flag |
| SanRossore | PRS\_TD\_20230616\_101431.nc | 8100 | 8100 | 1 | 1 | 0.251332962 | 0 |

## Source Code

**Main.py**

# ---------------------------------------------------------------------------- #

#                            Import Python Packages                            #

# ---------------------------------------------------------------------------- #

import os

import time

from datetime import datetime

import shutil

import configparser

import numpy as np

import pandas as pd

import rasterio as rio

# Don't remove rasterio.mask this package otherwise it could cause errors

import rasterio.mask

import xarray as xr

# ---------------------------------------------------------------------------- #

#                                 Import Class                                 #

# ---------------------------------------------------------------------------- #

from Class import S2

from Class import FLEX

# ---------------------------------------------------------------------------- #

#                                   Main Code                                  #

# ---------------------------------------------------------------------------- #

start\_Time = time.time()

config = configparser.ConfigParser()

# Initiate classes

flex = FLEX()

s2 = S2()

print("Code starts")

# Read optional input

config.read(s2.path\_Optional)

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

    flex.\_vegetationPixel = config["Optional Input"]["threshold\_Vegetation"]

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

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

    s2.\_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"]:

    s2.\_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"]:

    s2.\_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":

    s2.bool\_DeleteCache = True

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

# ---------------------------------------------------------------------------- #

#                                For FLEX Images                               #

# ---------------------------------------------------------------------------- #

if len(os.listdir(flex.path\_Input)) !=0 :

    list\_Name\_Site = []

    list\_FLEX\_Date = []

    list\_FLEX\_Time = []

    list\_FLEX\_Filename = []

    # Read .csv files to retrieve info of each site

    df\_Site = flex.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]

        temp\_SitePath = flex.path\_Input + "\\" + temp\_SiteName

        if (not os.path.exists(temp\_SitePath)) or (len(os.listdir(temp\_SitePath)) == 0):

            print(f"WARNING: The site {temp\_SiteName} doesn't have input FLEX images. This site has been skipped! ")

        else:

            temp\_Path\_Output = flex.path\_Output + "\\" + temp\_SiteName

            os.makedirs(temp\_Path\_Output, exist\_ok = True)

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

            temp\_FileName = os.listdir(temp\_SitePath)

            temp\_NumFLEXImages = len(temp\_FileName)

            print(f"There are {temp\_NumFLEXImages} FLEX images available for the site {temp\_SiteName}!")

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

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

            for k in range(temp\_NumFLEXImages):

                print(f"Now starting with No.{k + 1} FLEX image '{temp\_FileName[k]}' of the site {temp\_SiteName}")

                temp\_ds = rio.open(f'netcdf:{flex.path\_Input + "\\" + temp\_SiteName + "\\" + temp\_FileName[k]}:Leaf Area Index')

                print(f"{temp\_FileName[k]} opened succesfully!")

                temp\_index\_x, temp\_index\_y = temp\_ds.index(temp\_SiteLon,temp\_SiteLat)

                # Veg pixel filter

                list\_Name\_Site.append(temp\_SiteName)

                list\_FLEX\_Filename.append(temp\_FileName[k])

                list\_FLEX\_Date.append(temp\_FileName[k].split('.')[0].split('\_')[-2])

                list\_FLEX\_Time.append(temp\_FileName[k].split('.')[0].split('\_')[-1])

                # Calculate SIF

                temp\_xr\_ds = xr.open\_dataset(flex.path\_Input + "\\" + temp\_SiteName + "\\" + temp\_FileName[k])

                temp\_Name\_Var = list(temp\_xr\_ds.data\_vars)

                temp\_list\_SIF\_Name = []

                temp\_list\_SIF\_AVG = []

                temp\_list\_SIF\_STD = []

                for var\_name in temp\_Name\_Var:

                    if "Sif Emission Spectrum\_sif\_wavelength\_grid" in var\_name:

                        temp\_Array = temp\_xr\_ds[var\_name][(temp\_index\_x-1):(temp\_index\_x+2),(temp\_index\_y-1):(temp\_index\_y+2)].values

                        # print(temp\_Array)

                        temp\_list\_SIF\_Name.append(var\_name)

                        temp\_AVG = np.average(temp\_Array).item()

                        temp\_list\_SIF\_AVG.append(temp\_AVG)

                        temp\_STD = np.std(temp\_Array).item()

                        temp\_list\_SIF\_STD.append(temp\_STD)

                temp\_df\_SIF = pd.DataFrame({

                    "SIF": temp\_list\_SIF\_Name,

                    "Average": temp\_list\_SIF\_AVG,

                    "STD": temp\_list\_SIF\_STD

                })

                temp\_df\_SIF.to\_csv(temp\_Path\_Output + "\\" + temp\_FileName[k] + " - Sif.csv", index = False)

    temp\_df\_Sites = pd.DataFrame({

        "Site": list\_Name\_Site,

        "FLEX Filename": list\_FLEX\_Filename,

        "FLEX Date": list\_FLEX\_Date,

        "FLEX Time": list\_FLEX\_Time

    })

    temp\_df\_Sites.to\_csv(flex.path\_Output + "\\Usable FLEX Images.csv", index = False)

else:

    print("WARNING: There are no FLEX images found inside the input folder. ")

# ---------------------------------------------------------------------------- #

#                             For Sentinel-2 Images                            #

# ---------------------------------------------------------------------------- #

# Read Site.csv files to retrieve info of each site

df\_Site = s2.get\_SiteInfo()

# Read usable FLEX images .csv file

df\_FLEX = pd.read\_csv(s2.path\_Output + "\\Usable FLEX Images.csv")

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

list\_FLEXName = []

list\_ValidPixelsL1C = []

list\_ValidPercentageL1C = []

list\_ValidPixelsL2A = []

list\_ValidPercentageL2A = []

list\_CV = []

list\_Flag = []

if len(os.listdir(s2.path\_Image)) !=0 :

    # Start the main part, a loop that will iterate each row in our output "Usable FLEX Images.csv" file

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

        temp\_StartTime = time.time()

        # Get FLEX image date and time

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

        temp\_FLEX\_Filename = df\_FLEX["FLEX Filename"][i]

        temp\_FLEX\_Date = df\_FLEX["FLEX Date"][i]

        temp\_FLEX\_Time = df\_FLEX["FLEX Time"][i]

        temp\_FLEX\_DateTime = datetime.strptime(str(temp\_FLEX\_Date)+str(temp\_FLEX\_Time), '%Y%m%d%H%M%S')

        # Get site name

        temp\_Site\_Index = df\_Site.index[df\_Site['Site'] == temp\_SiteName].tolist()[0]

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

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

        #

        temp\_Path\_Images\_Site = s2.path\_Image + "\\" + temp\_SiteName

        # Check usable FLEX images

        temp\_S2\_Image\_Final = os.listdir(temp\_Path\_Images\_Site)[0]

        for j in range(len(os.listdir(temp\_Path\_Images\_Site))):

            temp\_S2\_Image = os.listdir(temp\_Path\_Images\_Site)[j]

            temp\_S2\_Image\_DateTime = temp\_S2\_Image[0:8] + temp\_S2\_Image[-6:]

            temp\_S2\_Image\_DateTime = datetime.strptime(temp\_S2\_Image\_DateTime, '%Y%m%d%H%M%S')

            if j == 0:

                temp\_TimeDiff\_Final = temp\_S2\_Image\_DateTime - temp\_FLEX\_DateTime

            else:

                temp\_TimeDiff = temp\_S2\_Image\_DateTime - temp\_FLEX\_DateTime

                if abs(temp\_TimeDiff) < abs(temp\_TimeDiff\_Final):

                    temp\_TimeDiff\_Final = temp\_TimeDiff

                    temp\_S2\_Image\_Final = temp\_S2\_Image

        print(f"S2 image {temp\_S2\_Image\_Final} has the nearest date and time to the FLEX image {temp\_FLEX\_Filename}")

        list\_FLEXName.append(temp\_FLEX\_Filename)

        print(f"The calculation and validation of the S2 image {temp\_S2\_Image\_Final} of the 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 = s2.get\_PathImages(temp\_SiteName + "\\" + temp\_S2\_Image\_Final)

        # 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).astype(np.int32)

        values\_L2A\_B04 = image\_L2A\_B04.read(1).astype(np.int32)

        values\_L2A\_B08 = image\_L2A\_B08.read(1).astype(np.int32)

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

        gdf\_ROI = s2.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 = s2.cal\_ValidPixels(temp\_SiteName, image\_L1C\_B08, mask\_L1CCombined, gdf\_ROI, note = "L1C")

        temp\_PassL2A, temp\_ValidPixelsL2A, temp\_ValidPixelsPercentageL2A = s2.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 = s2.cal\_L2ANDVI(path\_L2A\_xml\_DS, values\_L2A\_B04, values\_L2A\_B08)

            temp\_Rad = s2.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(s2.path\_Cache + "\\" + temp\_SiteName + "\\NIRv.tif", 'w', \*\*out\_meta) as dest:

                dest.write(temp\_NIRv, 1)

            # Clip the NIRv.tif

            image\_NIRv = rio.open(s2.path\_Cache + "\\" + temp\_SiteName + "\\NIRv.tif")

            s2.clip\_RasterbySHP(temp\_SiteName, image\_NIRv, gdf\_ROI, suffix = "NIRv ROI")

            # Read the clipped ROI NIRv.tif

            image\_NIRv\_ROI = rio.open(s2.path\_Cache + "\\" + temp\_SiteName + "\\NIRv ROI.tif")

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

            temp\_CV = s2.cal\_CV(values\_NIRv\_ROI)

            temp\_Flag = s2.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\_FLEX["Site"]),

        "FLEX Filename": list\_FLEXName,

        "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: {s2.path\_Output}")

    df\_Output.to\_csv(s2.path\_Output + "\\Output\_S2.csv", index = False)

    # Delete cache folder?

    if s2.bool\_DeleteCache:

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

        shutil.rmtree(s2.path\_Cache)

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

else:

    print("WARNING: There are no Sentinel-2 images found inside the input folder. ")

# Elapsed Time

end\_Time = time.time()

elapsed\_Time = end\_Time - start\_Time

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

**Class.py**

import os

from bs4 import BeautifulSoup

import math

import numpy as np

import shapely as shp

import pandas as pd

import geopandas as gpd

import rasterio as rio

import rasterio.mask

class S2:

    # 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 + "\\Sites.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 FLEX:

    # Initialization

    def \_\_init\_\_(self):

        # Vegetation pixel

        self.\_vegetationPixel = 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 fake FLEX images

        self.path\_Input = self.path\_Main + "\\Input FLEX Images\\"

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

        self.path\_SiteCSV = self.path\_Main + "\\Sites.csv"

    @property

    def VegetationPixel(self):

        return self.\_vegetationPixel

        # 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

## 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.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\:**

…

**Input FLEX Images\**

**…**

**Optional Input.ini**

[OptionalInput]

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

; set the threshold of vegetation pixel for FLEX images. If the vegetation pixels inside the ROI of an image is lower than this threshold, the code will not record the date and the time of this FLEX image.

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

; Default 0.5

threshold\_Vegetation =

; 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:**

For FLEX images, expected output of a input image can be:

|  |  |  |
| --- | --- | --- |
| SIF | Average | STD |
| Sif Emission Spectrum\_sif\_wavelength\_grid=670 | 0.275985 | 0.043109 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=672 | 0.326899 | 0.051894 |
| Sif Emission Spectrum\_sif\_wavelength\_grid=674 | 0.384171 | 0.061756 |
| Sif … | … | … |

Usable FLEX Images.csv

|  |  |  |  |
| --- | --- | --- | --- |
| Site | FLEX Filename | FLEX Date | FLEX Time |
| SanRossore | **PRS\_TD\_20230616\_101431.nc** | **20230616** | **101431** |

Output\_S2.csv

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Site | FLEX Filename | Valid Pixels L1C | Valid Pixels L2A | Valid Pixels Percentage L1C | Valid Pixels Percentage L2A | CV | Flag |
| SanRossore | PRS\_TD\_20230616\_101431.nc | 8100 | 8100 | 1 | 1 | 0.251332962 | 0 |