SENTINEL-3 SLSTR PRODUCT ANALYSIS AND VALIDATION

ANNA ANZALONE AND ALESSANDRO BRUNO

INAF IASF PALERMO

PYTHON SLSTR VALIDATION TOOL

- SLSTR Validation Tool consists of 5 python files to be applied for validation of SLSTR cloud mask product using CLloudFCN Deep Learning model;
- Here is the list of the package files:
  - Gui.py
  - Rad_to_reflectance.py
  - Image_loading.py
  - Patch_extraction.py
  - Unpack_and_show.py

SLSTR DATA VALIDATION (USE CASE DESCRIPTION)

- In the use case, user first goes down to download all SLSTR products of interest in a unique directory called SLTSTR_root
- Then, the user runs gui.py as many times as many folders of SLSTR products of interest to apply over each SLSTR directory the conversion from radiance to reflectance;
- At that point data can be «prepared» into data_cubes using the numpy package, this is accomplised by running image_loading.py (Users can browse all at once all the SLSTR folders of interest and then run the data preparation; which save all bands and cloud flags in two data_cubes as two npy files)
- After the above steps patch_extraction.py can be run to save all bands and cloud flags to numpy data cubes which fit the input size spatial resolution of the first Neural Network layer adopted to validate SLSTR products (in our use case it is 398x398);
- As miscellaneous function, unpack_and_show.py is useful whether the user want to display the cloud mask of each SLSTR product.

SLSTR DATA VALIDATION (USE CASE)

Conversion of Bands

GUI.py

RAD_TO_REFLECTANCE.py

The above step is needed to have all radiance bands converted to reflectance (for each SLSTR product folder)

Data Preparation

Image_loading.py

Patch_extraction.py

SLSTR product go through some transformation to be saved as images into a numpy array (datacube). Then patch_extraction.py allows for discriminating cloudy patches from the clear ones.

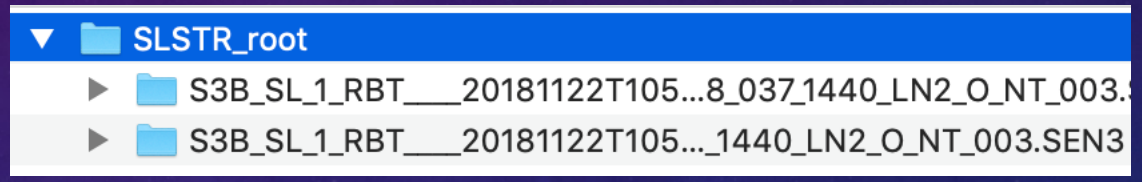
HANDLING WITH SLSTR PRODUCT DATA

Premise:

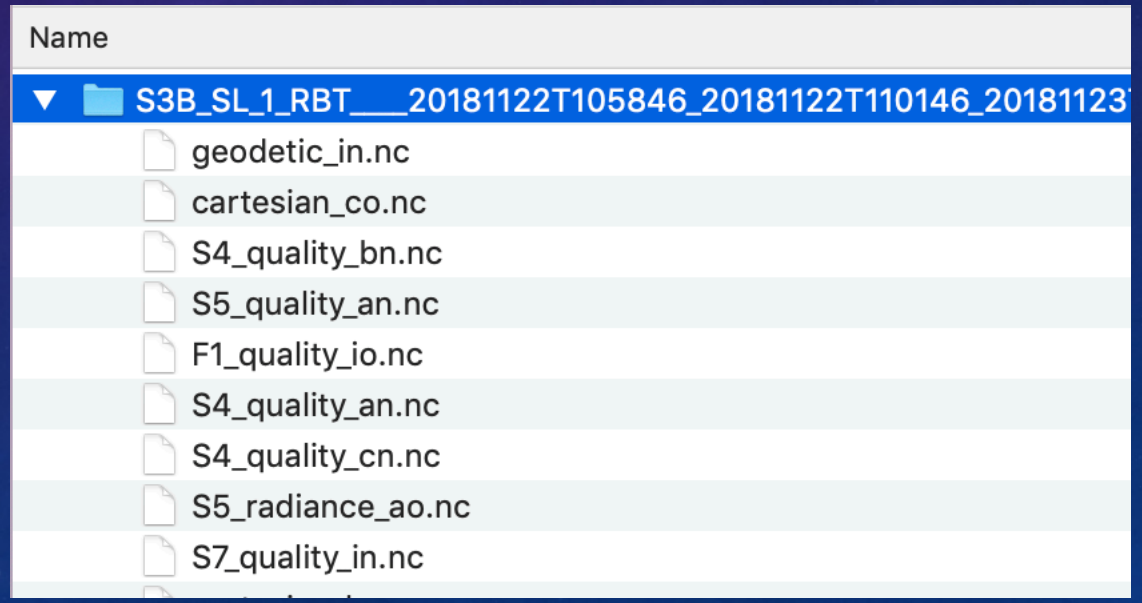
SLSTR products as downloaded from https://colhub.copernicus.eu/dhus/#/home include many ancillary data and TOA radiance bands;

We want radiance channels to be converted to reflectance as we want to work with absolute measurements; which can be compared (reflectance and brightness)

SLSTR_root folder contains all SLSTR products of our interest



Inside the SLSTR_root folder the user browse the product to be analysed



SLSTR S1-S9 BANDS

- S1-S6 Bands provide TOA **radiances** (page 32 from SLSTR handbook), which might not be useful when the user need to compare or merge data coming out of different or multiple sensors;
- A more useful unit to compare sensors is normalised **reflectance**, which can be generated from the radiance using the following formula:

reflectance = PI * (ToA radiance / solar irradiance / COS(solar zenith angle))

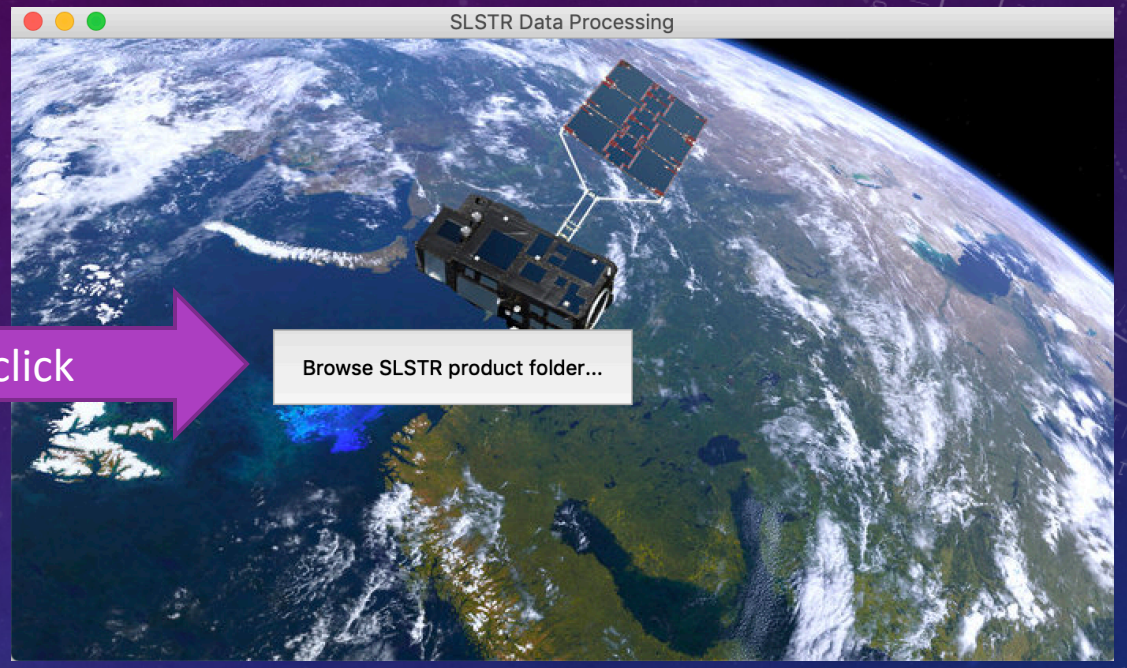
- S7-S9 bands provide brightness temperature values which needed to be normalised

SLSTR DATA CONVERSION

Order of steps:

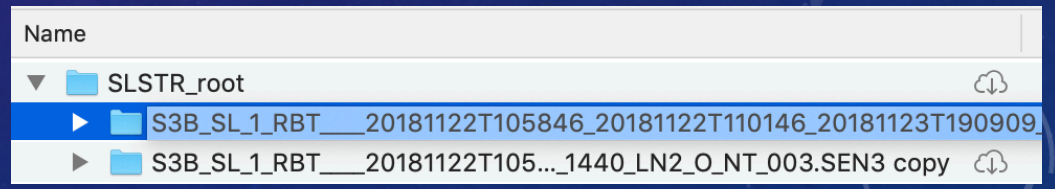
1) **GUI.py** is launched to convert Radiance to Reflectance

click



2) Browse the more nested SLSTR folder containing the product of interest

click



3) Once user browses the SLSTR product folder, **rad_to_reflectance.py** is run

Rad_to_reflectance.py

Ancillary Data

SLSTR BAND CONVERSION (RADIANCE TO REFLECTANCE)

- Before doing the conversion we need some ancillary data. S1 related data are reported as example (extracted from rad_to_reflectance.py):

s1_an = open_dataset(product + "/S1_radiance_an.nc")["S1_radiance_an"].values[:]

solar_irradiance_s1 = open_dataset(product + "/S1_quality_an.nc")["S1_solar_irradiance_an"].values[:]

detectors = open_dataset(product + "/indices_an.nc")["detector_an"].values[:]

sza = open_dataset(product + "/geometry_tn.nc")["solar_zenith_tn"].values[:]

s1_reflectance = max(0, (1., math.pi * (s1_an[y][x] / solar_irradiance_s1[int(detectors[y][x])]) / math.cos(math.radians(sza_corrected))))

SLSTR DATA PREPARATION

- SLSTR PRODUCT Preparation consists of extracting data of both bands and cloud flags; which are saved into two different data cubes (this job is done by **loading_image.py**);
- Once cloud and band data cubes are saved, patch_extraction.py allows for extracting cloudy and clear patches from the data cubes;
- After the step above, Deep Learning models can be applied to validate SLSTR cloud flags concerning all the levels of cloud masks provided;
- As a miscellanous function, unpack_and_show.py displays the cloud_flags among the following:

[0]visible
[1]1.37_threshold
[2]1.6_small_histogram
[3]1.6_large_histogram
[4]2.25_small_histogram
[5]2.25_large_histogram
[6]11_spatial_coherence
[7]gross_cloud
[8]thin_cirrus medium_high
[9]fog_low_stratus
[10]11_12_view_difference
[11]3.7_11_view_difference
[12]thermal_histogram
[13]spare
[14]spare

PYTHON DEPENDENCIES AND PACKAGES

- import os
- import numpy as np
- import glob
- import cv2
- import sys
- from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QLabel, QToolTip, QMessageBox
- from PyQt5.QtGui import *
- from PyQt5.QtCore import pyqtSlot, QDir, QCoreApplication
- from PyQt5.QtWidgets import QFileDialog, QDialog, QListView, QAbstractItemView, QTreeView
- from PIL import ImageTk, Image
- from netCDF4 import Dataset
- import glob
- import cv2
- from skimage import color

- Experimental Setup
- gui.py, image_load.py, patch_extraction.py have been tested on a computer with the following configuration:
- MacBook pro 2019, 8 Gb Ram, Mac OS Mojave 10.14.6 version, 2,3 GHz Intel Core i5
- Python 3 libraries:
  - Tkinter, PIL, Image, os, numpy, netCDF4, glob, cv2, skimage, **PyQt5**,
- Remarks
  - Some python libraries are not Mac Os compliant (Tkinter) or maybe it is only a problem of python version (tkinter was released with ptyhon 2);
  - If user is using a windows operating system computer it won't be a problem to run codes with tkinter graphic library, otherwise it will be necessary to work out with PyQt5;
  - Some fixes need to be done to make all tool functions Mac Os compliant…