Sen4CAP - Sentinels for Common Agricultural Policy

Design Justification File
ATBD for L3A biophysical indicators

Milestone	Milestone 1
Authors	Nicolas BELLEMANS, Sophie BONTEMPS, Pierre DEFOURNY, Cosmin UDROIU, Marie WEISS
Distribution	ESA - Benjamin KOETZ

This page is intentionally left blank

Table of contents

List of figures

List of tables

List of algorithms

Applicable documents

ID	Title	Code	Issue	Date
AD.1	Sen4CAP Technical Specification document	Sen4CAP_TS_1.0	1.0	29/03/2018
AD.2	Sen4CAP Design Justification File – Benchmarking for L3A biophysical indicators	04_Sen4CAP_DJF_v1.1_ BenchmarkingBiophysicalIndicators	1.1	29/07/2018

Reference documents

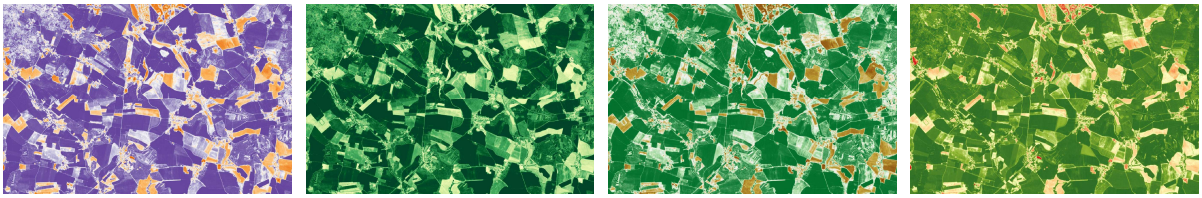
ID	Title
RD.1	Weiss M. and Baret F.. 2016. S2 Toolbox Level 2 products: LAI, FAPAR, FCOVER, V1.1.
RD.2	Weiss, M., Baret, F., Leroy, M., Hautecoeur, O., Bacour, C., Prévot, L., and Bruguier, N. (2002). Validation of neural net techniques to estimate canopy biophysical variables from remote sensing data. Agronomie, 22, 547-554
RD.3	Rouse, J.W., Haas, R.H., Schell, J.A. and Deering, D.W., 1973, Monitoring vegetation systems in the Great Plains with ERTS. In 3rd ERTS Symposium, NASA SP-351 I, pp. 309–317.
RD.4	Tucker, C. J., 1979: Red and photographic infrared linear combinations for monitoring vegetation. Remote Sens. Environ., 8, 127–150
RD.5	Garrigues, S., Allard, D., Baret, F., & Weiss, M. (2006). Influence landscape spatial heterogeneity on the non-linear estimation of leaf area index from moderate spatial resolution remote sensing data. Remote Sensing of Enviroment, 105, 286-298
RD.6	Weiss, M., Baret, F., Myneni, R., Pragnère, A., & Knyazikhin, Y. (2000). Investigation of a model inversion technique for the estimation of crop charcteristics from spectral and directional reflectance data. Agronomie, 20, 3-22
RD.7	Chen, J.M., Menges, C.H., & Leblanc, S.G. (2005). Global mapping of foliage clumping index using multi-angular satellite data. Remote Sensing of Environment, 97, 447-457
RD.8	Weiss M., Baret F., Lero M., Hautecoeura O., Bacourd C., Prévotb L. and Bruguierb N., 2002. Validation of neural net techniques to estimate canopy biophysical variables from remote sensing data. Agronomie,22, 547-554.
RD.9	Prince, S.D. (1991). A model of regional primary production for use with coarse resolution satellite data. International Journal of Remote Sensing
RD.10	Baret, F., Leroy, M., Roujean, J.L., Knorr, W., Lambin, E., & Linderman, M. (2003). CYCLOPES User Requirement Document. In. Avignon: INRA-CSE
RD.11	Inglada J., Bontemps S., Defourny P., Koetz B. Sen2-Agri Begetation Status Algorithm Theoritical Basis Document , 2015

List of acronyms

Acronym	Definition
ANN	Artificial Neural Network
ATBD	Algorithm Theoretical Basis Document
BI	Biophysical Indicators
BOA	Bottom-Of-Atmosphere
CAP	Common Agricultural Policy
ESA	European Space Agency
FAPAR	fraction of Absorbed Photosynthetically Active Radiation
FCOVER	fraction of Vegetation Cover
GAI	Green Area Index
L2A	Level 2A
L8	Landsat 8
LAI	Leaf Area Index
MACCS	Multisensor Atmospheric Correction and Cloud-Screening
NDVI	Normalized Difference Vegetation Index
NIR	Near InfraRed
S2	Sentinel-2
Sen4CAP	Sentinel For CAP

1. Introduction

Four different products are proposed as Biophysical Indicators (BIs): Normalized Difference Vegetation Index (NDVI), Leaf Area Index (LAI), fraction of Vegetation Cover (FCOVER) and fraction of Absorbed Photosynthetically Active Radiation (FAPAR) (Figure 1-1). Each BI retrieval will be performed for every 10-meter pixel covering the area of interest and for each acquisition date.



(a) (b) (c) (d)

Figure 1-1. Illustration of the four BIs targeted by the Sen4CAP project: (a) NDVI, (b) LAI, (c) FAPAR and (d) FCOVER

Following definitions are extracted from RD.1.

- **NDVI**

The well-known NDVI introduced in [RD.3] and [RD.4] is the most commonly used index to qualify the vegetated areas. It is a simple optical band ratio computed from the surface reflectance in the Near-InfraRed (NIR) and the red channels using the following formula:

$$\mathrm{NDVI} = \frac{\rho(NIR) - \rho(RED)}{\rho(NIR) + \rho(RED)}$$

NDVI values range from -1 to 1.

- **LAI**

LAI is defined as half the developed area of photosynthetically active elements of the vegetation per unit horizontal ground area. It determines the size of the interface for exchange of energy (including radiation) and mass between the canopy and the atmosphere. This is an intrinsic canopy primary variable that should not depend on observation conditions. LAI is strongly non-linearly related to reflectance. Therefore, its estimation from remote sensing observations will be strongly scale dependent [RD.5, RD.6]. Note that vegetation LAI as estimated from remote sensing will include all the green contributors, i.e. including understory when existing under forests canopies. However, except when using directional observations [RD.7], LAI is not directly accessible from remote sensing observations due to the possible heterogeneity in leaf distribution within the canopy volume. Therefore, remote sensing observations are rather sensitive to the "effective" LAI, i.e. the value that would produce the same remote sensing signal as that actually recorded one, while assuming a random distribution of leaves. The difference between the actual and effective LAIs may be quantified by the clumping index [RD.7] that roughly varies between 0.5 (very clumped canopies) and 1.0 (randomly distributed leaves) [RD.8]. The LAI measured from remote sensing is often refered to as the Green Area Index (GAI) since it includes all the green contributors of the canopy.

LAI values range from 0 to 8.

- **FAPAR**

FAPAR corresponds to the fraction of photosynthetically active radiation absorbed by the canopy. The FAPAR value results directly from the radiative transfer model in the canopy which is computed instantaneously. It depends on canopy structure, vegetation element optical properties and illumination conditions. FAPAR is very useful as input to several primary productivity models based on simple

efficiency considerations [RD.9]. Most of the primary productivity models using this efficiency concept are running at the daily time step. Consequently, the product definition should correspond to the daily integrated FAPAR value that can be approached by computation of the clear sky daily integrated FAPAR values as well as the FAPAR value computed for diffuse conditions. To improve the consistency with other FAPAR products that are sometimes considering the instantaneous FAPAR value at the time of the satellite overpass under clear sky conditions (e.g. MODIS), a study was proposed to investigate the differences between alternative FAPAR definitions [RD.10]. Results show that the instantaneous FAPAR value at 10:00 (or 14:00) solar time is very close to the daily integrated value under clear sky conditions. FAPAR is relatively linearly related to reflectance values and will be little sensitive to scaling issues. Note also that the FAPAR refers only to the green parts (leaf chlorophyll content higher that 15µg.cm-2) of the canopy.

fAPAR values range from 0 to 1.

- **FCOVER**

It corresponds to the gap fraction for nadir direction. FCOVER is used to separate vegetation and soil in energy balance processes, including temperature and evapotranspiration. It is computed from the leaf area index and other canopy structural variables and does not depend on variables such as the geometry of illumination as compared to FAPAR. For this reason, it is a very good candidate for the replacement of classical vegetation indices for the monitoring of green vegetation. Because of its quasi-linear relationship with reflectances, FCOVER will be only marginally scale dependent [RD.6]. Note that similarly to LAI and FAPAR, only the green elements (leaf chlorophyll content higher that 15µg.cmP -2 P) will be considered.

FCOVER values range from 0 to 1.

2. Logical model

This Algorithm Theoretical Basis Document (ATBD) describes the algorithm developed to produce the L3A - Biophysical Indicators product from Sentinel-2 (S2) and Landsat 8 (L8) acquisitions, following the specifications defined in [AD.1]. This algorithm will be implemented in the Sen4CAP system.

For the LAI, FCOVER and FAPAR, this ATBD is derived from one already proposed by Marie Weiss and Frédéric Baret in the frame of the ESA Sentinel-2 toolbox [RD.1].

The LAI, FCOVER and FAPAR BI retrieval is performed by applying a global Artificial Neural Network (ANN) on each pixel considering the reflectance values of all the available bands pre-processed at the Level 2A (L2A) and some geometric configuration as input.

The training of the ANN, that consists in generating the training database, defining the neural network architecture and calibrating the network, will not be performed within the Sen4CAP system. Instead, the Sen4CAP system will benefit from an already trained ANN, made openly and freely available by INRA. From the system implementation point of view, this trained ANN will be given as auxiliary data to the processor.

As for the BI, NDVI spectral index are also provided for each acquisition.

Two delivery modes are foreseen for the vegetation status indicator: single-date and multi-date delivery. The single-date L3A will consist in delivering the product for each acquisition while in the multi-date L3A, the single-date L3A is smoothed by doing a weighted average of a set of neighouring acquistions.

Figure 2-1 presents the general worklow of the single-date L3A BI product algorithm. In order to get the multi-date LAI, one post-processing option will be made available to smooth out the BI retrieval. This step is detailed in the section 5.

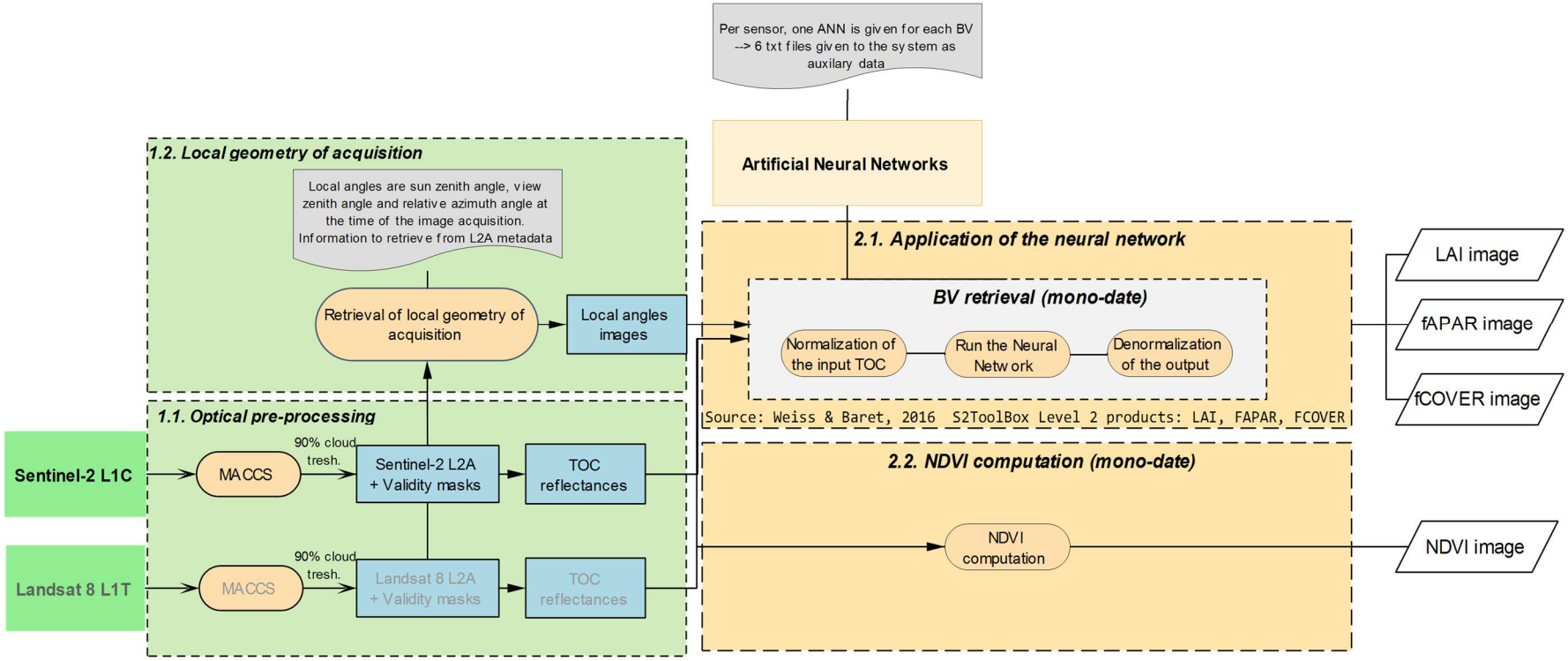


Figure 2-1. General workflow of the mono-date L3A Biophysical Indicators product algorithm

3. Algorithm description for the single-date BI

3.1 Inputs

The ANN will by applied on each valid L2A Bottom-Of-Atmosphere (BOA) reflectance data obtained using the Multisensor Atmospheric Correction and Cloud-Screening (MACCS) software. The following sub-sections describe the inputs required for each considered pixel.

3.1.1 BOA reflectance value

Individual daily observations are considered. All S2 bands are used except the blue one (B2): B3, B4, B5, B6, B7, B8a, B8, B11 and B12. All L8 bands are used except the blue one (B2): B3, B4, B5, B6, B8. This band selection is derived from a benchmarking exercise and proved to achieve the best results when compared to ground data and with the aim of using S2 and L8 in combination [AD.2].

Table 3-1 and Table 3-2 presents the spectral characteristics of the S2 and L8 bands used for the BI retrieval.

Table 3-1. S2 spectral characteristics

Acronym	Central wavelength (nm)	Width (nm)	Spatial resolution (m)	Band name
B3	560	35	10	Green
B4	665	30	10	Red
B5	705	15	20	Red edge 1
B6	740	15	20	Red edge 2
B7	783	20	20	Red edge 3
B8	842	115	10	NIR wide
B8a	865	20	20	NIR narrow
B11	1610	90	20	SWIR 1
B12	2190	180	20	SWIR 2

Table 3-2. L8 spectral characteristics

Acronym	Central wavelength (nm)	Width (nm)	Spatial resolution (m)	Band name
B3	560	30	30	Green
B4	650	20	30	Red
B5	865	15	30	NIR narrow
B6	1610	40	30	SWIR 1
B7	2200	90	30	SWIR 2

3.1.2 Geometry of acquisition

The choosen implementation of the BI retrieval makes use of a global ANN, meaning that the ANN takes as input the characteristics of the acquisition geometry. Consequently, the directional information must be considered when training the neural networks, similarly to the spectral information. The cosine of the sun zenith angle (θs), view zenith angle (θv) and relative azimuth angle (φ) at the time of the image acquisition are required. The step of extracting the observation angles of each pixel is described here below.

3.1.2.1 Extracting the local viewing angles of Sentinel-2 products

The local observation angles (i.e. for every pixel) of S2 acquisitions can be derived from the general L2A HDR file which is providing the observation angles for the 12 detectors of each band on a 5x5 km grid. This can be done using the pseudo-code provided in Algorithm 3-1. The output is a 3-band raster file respectively containing the information of the cosine of the sun zenith angle (θs) (band 1), view zenith angle (θv) (band 2) and relative azimuth angle (φ) (band 3) of each pixel.

Algorithm 3-1. Extraction of local observation angles in S2 L2A acquisition

```
import os
import argparse
from lxml import etree
import numpy as np
from osgeo import gdal, osr

parser = argparse.ArgumentParser(description='Take the 5x5 km angles and convert it to a user defined
resolution vrt with bilinear interpolation. The code produces three .tif (5kmx5km) and .vrt (10mx10m)
containing "cos(angle)" with angle being view_zenith, solar_zenith and solar_azimuth-view_azimuth. NB:
since we store integers, the output is actually 10000*cos(x).)
parser.add_argument('--inputAngleFile', help='Path to the input file containing the angles (xml/HDR
format).', required=True)
parser.add_argument('--inputRasterFile', help='Path to an input raster corresponding to the
inputAngleFile (used to extract origin and projection.', required=True)
parser.add_argument('--outputFile', help='Path and name of the output files. Do not include extension,
the script writes a .tif with 5x5km and a .vrt with the user defined resolution.', required=True)
parser.add_argument('--target_res', default=10, help='Target vrt resolution (do not support non-square
pixels). Default value is 10m.')

args = parser.parse_args()
nodata = 32767

# Dump array information into a raster
def array2raster(outRasterName, rasterOrigin, pixelWidth, pixelHeight, projection, array,
dataType=gdal.GDT_Int16):
    cols = array.shape[1]
    rows = array.shape[0]
    originX = rasterOrigin[0]
    originY = rasterOrigin[1]
    driver = gdal.GetDriverByName('GTiff')
    outRaster = driver.Create(outRasterName, cols, rows, 1, dataType, ["TILED=YES", "COMPRESS=LZW"])
    outRaster.SetGeoTransform((originX, pixelWidth, 0, originY, 0, pixelHeight))
    outband = outRaster.GetRasterBand(1)
    outband.SetNoDataValue(nodata)
    outband.WriteArray(array)
    outRasterSRS = osr.SpatialReference(projection)
    outRaster.SetProjection(outRasterSRS.ExportToWkt())
    outband.FlushCache()
    print("File %s written."%outRasterName)

# Convert the string matrices to numpy arrays and computes the 10000*cos(angle)
def string2array(string, computeCos=True):
  if computeCos :
    return np.array([[int(10000 * np.cos(np.radians(float(angle)))) for angle in line.split(" ")] for
line in string.splitlines()])
  else:
    return np.array([[float(angle) for angle in line.split(" ")] for line in string.splitlines()])

# Get input tif informations
input_raster = gdal.Open(args.inputRasterFile)
projection = input_raster.GetProjectionRef()
raster_x_size = input_raster.RasterXSize
raster_y_size = input_raster.RasterYSize
raster_origin = [input_raster.GetGeoTransform()[0], input_raster.GetGeoTransform()[3]]

# Parse the xml file
namespaces = {'ns' : 'http://eop-cfi.esa.int/CFI'}
tree = etree.parse(args.inputAngleFile)
```

```
# Get the angle matrix as string
def get_matrixAngle_string(xpath):
  out_string = ""
  for entry in tree.xpath(xpath, namespaces=namespaces)[0].getchildren():
    out_string += entry.text +"\n"
  return out_string

def write_tif_and_vrt(matrixAngle_array, stringToAppend):
  # Dump the array with 1000*cos(angle) in a tif
  angle_rawResolution_outName = args.outputFile + "_rawResolution_" + stringToAppend + ".tif"
  #print(angle_rawResolution_outName)
  #print(matrixAngle_array)
  array2raster(angle_rawResolution_outName,          raster_origin,          pixel_x_size_input_angles,
pixel_y_size_input_angles * -1, projection, matrixAngle_array)
  #Produce a vrt with the user define resolution
  print("Writing %s vrt:"%stringToAppend)
  os.system("gdalbuildvrt %s %s -tr %d %d -r bilinear -srcnodata %d -vrtnodata %d"%(args.outputFile +
"_userResolution_"   +   stringToAppend   +   ".vrt",   angle_rawResolution_outName,   args.target_res,
args.target_res, nodata, nodata))
  #os.system("gdal_translate %s %s"%(args.outputFile + "_userResolution_" + stringToAppend + ".vrt",
args.outputFile + "_userResolution_" + stringToAppend + ".tif"))

# Let's assume for now that the resolution are the same for all the angles (Solar, Observation, Zenith
and Azimuth
pixel_x_size_input_angles                                                                                =
float(tree.xpath("/ns:Earth_Explorer_Header/ns:Variable_Header/ns:Specific_Product_Header/ns:Product_In
formation/ns:Solar_Angles/ns:Zenith/ns:COL_STEP", namespaces=namespaces)[0].text)
pixel_y_size_input_angles                                                                                =
float(tree.xpath("/ns:Earth_Explorer_Header/ns:Variable_Header/ns:Specific_Product_Header/ns:Product_In
formation/ns:Solar_Angles/ns:Zenith/ns:ROW_STEP", namespaces=namespaces)[0].text)

# Solar Zenith Angle (never nodata...)
solarAngle_Zenith_string                                                                                 =
get_matrixAngle_string("/ns:Earth_Explorer_Header/ns:Variable_Header/ns:Specific_Product_Header/ns:Prod
uct_Information/ns:Solar_Angles/ns:Zenith/ns:Values_List")
write_tif_and_vrt(string2array(solarAngle_Zenith_string), "cosSolarAngleZenith")

# for the view angle, we average over the detectors and bands
def get_mean_over_matrices(xpath):
  view_angle_matrices = []
  for value_list in (tree.xpath(xpath, namespaces=namespaces)):
    temp_matrix_string = ""
    for line in value_list.getchildren():
      temp_matrix_string += line.text + "\n"
    view_angle_matrices.append(np.array([[float(angle)  for  angle  in  line.split("  ")]  for  line  in
temp_matrix_string.splitlines()]))
  #print(len(view_angle_matrices))
  #print(view_angle_matrices[0].shape)
  view_angle_mean = np.zeros(view_angle_matrices[0].shape)
  view_angle_countNonNan = np.zeros(view_angle_matrices[0].shape)
  for matrix in view_angle_matrices:
    for (i,j), value in np.ndenumerate(matrix):
      if not (np.isnan(value)):
          view_angle_mean[i,j] += value
          view_angle_countNonNan[i,j] += 1
          # to see variations across the bands and det id:
  #        if (i,j) == (1,1):
  #          print(value)
  #print(view_angle_mean)
  #print(view_angle_countNonNan)
  for (i,j), value in np.ndenumerate(view_angle_countNonNan):
    if value != 0:
      view_angle_mean[i, j] = view_angle_mean[i, j] / value
    else:
      view_angle_mean[i, j] = nodata
  #print(view_angle_mean)
  return view_angle_mean

#View Zenith Angle
```

```
view_angle_zenith_mean                                                                                =
get_mean_over_matrices("/ns:Earth_Explorer_Header/ns:Variable_Header/ns:Specific_Product_Header/ns:Prod
uct_Information/ns:List_of_Viewing_Angles/ns:Viewing_Incidence_Angles_Grids/ns:Zenith/ns:Values_List")
cos_view_angle_zenith_mean = nodata*np.ones(view_angle_zenith_mean.shape)
for (i,j), value in np.ndenumerate(view_angle_zenith_mean):
  if not value == nodata:
    cos_view_angle_zenith_mean[i,j] = int(10000 * np.cos(np.radians(value)))
write_tif_and_vrt(cos_view_angle_zenith_mean, "cosViewAngleZenith")

#View Azimuth Angle
view_angle_azimuth_mean                                                                               =
get_mean_over_matrices("/ns:Earth_Explorer_Header/ns:Variable_Header/ns:Specific_Product_Header/ns:Prod
uct_Information/ns:List_of_Viewing_Angles/ns:Viewing_Incidence_Angles_Grids/ns:Azimuth/ns:Values_List")

# Solar Azimuth Angle
solarAngle_Azimuth_string                                                                             =
get_matrixAngle_string("/ns:Earth_Explorer_Header/ns:Variable_Header/ns:Specific_Product_Header/ns:Prod
uct_Information/ns:Solar_Angles/ns:Azimuth/ns:Values_List")
solarAngle_Azimuth_array = string2array(solarAngle_Azimuth_string, computeCos=False)

relative_angle_azimuth = nodata*np.ones(view_angle_azimuth_mean.shape)
for (i,j), value in np.ndenumerate(view_angle_azimuth_mean):
  if value == nodata or solarAngle_Azimuth_array[i,j] == nodata:
    relative_angle_azimuth[i,j] = nodata
  else:
    relative_angle_azimuth[i,j] = solarAngle_Azimuth_array[i,j] - value
cos_relative_angle_azimuth = nodata*np.ones(relative_angle_azimuth.shape)
for (i,j), value in np.ndenumerate(relative_angle_azimuth):
  if not value == nodata:
    cos_relative_angle_azimuth[i,j] = int(10000 * np.cos(np.radians(value)))
write_tif_and_vrt(cos_relative_angle_azimuth, "cosRelativeAngleAzimuth")
```

3.1.2.2 Extracting the local viewing angles of Landsat-8 products

The local observation angles (i.e. for every pixel) of L8 acquisitions can be derived from the Angle Coefficient File contained in the L1C product ("ANG.txt"). To do so, an executable file is provided on the USGS web site (https://landsat.usgs.gov/solar-illumination-and-sensor-viewing-angle-coefficient-file).

3.2 Algorithm implementation

Considering that the ANN models are provided to the processor as auxiliary datasets, the algorithm implementation can be summarized in 3 steps:

1. normalizing the inputs;
2. running the ANN;
3. denormalizing the outputs.

From the system implementation point of view, a specific effort should be made to allow easy update of the L3A processor in the future. Therefore, all the coefficients useful for the normalization/denormalization, the neural network models and the domain definition should be stored in the form of parameter tables that can easily be changed whenever an update is available.

These parameters tables are provided by INRA in the form of Excel files: there is one file for each product and for each sensor. Six parameters tables are therefore needed to be able to generate the 3 BI (LAI, FCOVER and FAPAR) using both S2 and L8 acquisitions.

To be easily readable by the processor, these excel sheets are converted into text files using the Python script provided in Algorithm 3-2.

Algorithm 3-2. Conversion of excel files into text files, for the parameters' tables

```
import openpyxl, sys, pystache

wb = openpyxl.load_workbook(filename=sys.argv[1], read_only=True)

data = {
  'inputs': [], # {min_norm, max_norm(, label)}
  'layers': [], # {(inputs: [labels,..],) neurons: [{bias, weights}], function, number}
  'output': {}  # min_norm, max_norm, min, max, tolerance
}

ws = wb['Normalisation']
inputs=[]
for row in ws.iter_rows(row_offset=3):
  if row[0].value == None:
    if data['inputs']:
      break
    continue
  data['inputs'].append({'label':row[0].value, 'min_norm':row[1].value, 'max_norm':row[2].value})
  inputs.append(row[0].value)

row_output = tuple(ws.rows)[-1] # defined as the last data of this sheet
data['output']['name'] = row_output[0].value
data['output']['min_norm'] = row_output[1].value
data['output']['max_norm'] = row_output[2].value

ws = wb['Weights']

lay_indexes = []
index = 0
for row in ws.iter_rows():
  if row[0].value == 'neuron1':
    lay_indexes.append(index)
  index += 1

nbInputs = len(inputs)
for index in lay_indexes:
  newlayer = {'neurons':[]}
  newlayer['number'] = 0
  for row in ws.iter_rows(row_offset=index):
    if row[0].value == None:
      break
    elif row[0].value == 'bias':
      for i in range(newlayer['number']):
        newlayer['neurons'][i]['bias'] = row[i+1].value
    elif row[0].value == 'Transfert Fct':
      newlayer['function'] = row[1].value
    else: # neuron
      newlayer['neurons'].append({'weights': [row[i+1].value for i in range(nbInputs)]})
      newlayer['number'] += 1
  if not data['layers']: # first
    newlayer['inputs'] = inputs
  else:
    newlayer['inputs'] = ['neuron'+str(i+1) for i in range(nbInputs)]
  data['layers'].append(newlayer)
  nbInputs = newlayer['number']

ws = wb['Extreme Cases']

data['output']['tolerance'] = ws['B10'].value
data['output']['min'] = ws['C10'].value
data['output']['max'] = ws['D10'].value

with open(sys.argv[2], 'r') as tpl_f:
  tpl = tpl_f.read()
  rendered = pystache.render(tpl, data)
  with open(sys.argv[3], 'w') as out_f:
    out_f.write(rendered)
```

Figure 3-1 provides an example of the extracted text file for the LAI ANN using S2 images.

Figure 3-1. Example of parameters text file for LAI generation using S2 acquisitions

```
# number of layers, defined by their transfert function and size
tansig 5 purelin 1
# min/max for normalisation of inputs
0 0.24227150564744757 0 0.29180643729483435 0.023617798406067352 0.7940468337225911 0 0.3352184967986723
0.02202129476436714  0.6082236185467771  0.0282828940537622  0.7593032519236953  0.027033167693721202
0.7762830093591506  0.008452168326508781  0.4951157687164573  0  0.4939202941174727  0.9185974648964176
0.9999999998075627 0.3420445117819629 0.9359109848078707 -0.9999999998938771 0.9999999949266675
# definition of layers (first column is the bias)
# bias B3 B4 B8 B5 B6 B7 B8A B11 B12 cos(View_Zenith) cos(Sun_Zenith) cos(Rel_Azimuth)
-1.2942679301598696  -0.3742492703985799  -1.0879940321273938  0.9073468650336846  0.3615250088966526  -
3.118927273401686  2.77498648736835  -0.057300763562444526  -1.234084059620391  -0.8564958529719705  -
0.4841042554196577 0.1515712348243339 0.6152433844319842
0.8734788297065322  0.44320224366190497  0.8887592231132775  -0.5056153268593457  0.9098195307014335  -
0.21344245581533045  -2.4093186017926205  1.5692209777841764  1.19567905031086  0.08276325717685079  -
0.10053882578929706 -0.3750623159288322 -0.023841736554610356
1.6077395606205696  -0.04979059593238454  0.05545236241312816  -2.050827282340712  -0.038057479931813895
2.7314636384400606  0.0621089209933646  -2.520042453088982  1.6787370971513154  0.014589040863996896  -
0.026092775177453207 0.004630525433094241 0.04475965644686002
-0.7690197124128048  -0.2809034831800579  -0.6919131312848309  0.425638116628382  -0.9578575668912771
0.5258728878442835  1.3642064064465487  -0.5814229260282106  -0.7416025252569921  -0.21122777745932927
0.1374457596308841 0.3242486913111365 -0.04969549164971736
-4.3601288190008844  0.0326867623298655  -0.4572803297467024  -0.33274696775405693  -0.26145005507079555
1.6331052298364763  2.19881936121678  0.06499309497213597  0.2848516621138459  -0.6458758003998314
0.024346869568991975 -0.22837933063794427 -0.24834066469162608
# bias neuron1 neuron2 neuron3 neuron4 neuron5
1.468201298006802  0.06116052624682594  0.33858438067795643  -0.39458071876841083  0.46139774610556217
1.878696702247
# min/max form denormalisation of outputs
3.614536369589416e-05 14.238223860863263
# min, max and tolerance for output
0 8 0.2
```

The 3 steps of the BI algorithm impementation are detailed here below.

3.2.1 Normalization of the inputs

The input variables (9 S2 bands and 3 observation angles in the case of S2 and 5 bands and 3 observation angles in the case of L8) should be normalized before running the neural network, using the following equation:

$$X^* = 2 \times \left(\frac{X - Xmin}{Xmax - Xmin}\right) - 1$$

where X* is the nornalized input value, and X_{min} and X_{max} are the minimum and maximum values computed over the neural network training dataset.

3.2.2 Neural network

The neural network is described by its architecture, i.e. the number of hidden layers and the output layer.

Each layer is described by its number of neurons, associated weight and biases, and transfer function.

For the neurons of the hidden layers, the transfer function is a tangent sigmoid function given by:

$$Y = Tansig(x) = \left(\frac{2}{1 + exp(-2x)}\right) - 1$$

For the output layer, the transfer function is linear (y=x).

For each neural net (one per product and per sensor), the weight and biases for each neuron are provided in the form of tables.

3.2.3 Denormalization of the output

This last step simply consists in applying the inverse function than the one used for input normalization:

$$Y = 0.5 \times (Y^* + 1) * (Ymax - Ymin) + Ymin$$

3.2.4 Pseudo-code

The generic C++ code provided in Algorithm 3-3 can be used to run the 3 steps in a row, similarly for S2 and L8 acquisitions.

Algorithm 3-3. BI retrieval implementation, including inputs normalization, neural network run and outputs denormalization

```
#include "itkImageRegionIterator.h"
#include "belcamApplyTrainedNeuralNetworkFilter.h"


template <class TI, class TO>
void belcamApplyTrainedNeuralNetworkFilter<TI, TO>
::ThreadedGenerateData(const typename TO::RegionType& outputRegionForThread, itk::ThreadIdType threadId)
{
  unsigned int nbBands = this->GetInput(0)->GetNumberOfComponentsPerPixel();
  unsigned int nbLayers = m_net.layers.size();

  // initialize iterators
  typename TI::RegionType inputRegionForThread;
  this->CallCopyOutputRegionToInputRegion(inputRegionForThread, outputRegionForThread);
  itk::ImageRegionIterator     <TO> oIt(this->GetOutput(), outputRegionForThread);
  itk::ImageRegionConstIterator<TI> iIt(this->GetInput(), inputRegionForThread);

  typename TO::PixelType pixelO(1);
  typename TI::PixelType pixelI(nbBands);

  if(m_net.layers[0].weights[0].size() != nbBands) {
    printf("the number of bands and the number of inputs in txt file have to be identical !! (%d vs
%d)\n", nbBands, m_net.layers[0].weights[0].size());
    exit(1);
  }

  // init size of processing variables
  std::vector<std::vector<double> > inputs(nbLayers+1);
  for(int iN = 0; iN < nbLayers; ++iN)
    for(int i = 0; i < m_net.layers[iN].weights[0].size(); ++i)
      inputs[iN].push_back(0.);
  inputs[nbLayers].push_back(0.); // one output

  // loop over pixels
  for(oIt.GoToBegin(), iIt.GoToBegin(); ! oIt.IsAtEnd(); ++oIt, ++iIt)
  {
    // normalisation of inputs
    pixelI = iIt.Get();
    for(int iB = 0; iB < nbBands; ++iB)
      inputs[0][iB] = 2.0 * (pixelI[iB] / m_scale - m_net.min_norm_in[iB]) / (m_net.max_norm_in[iB] -
m_net.min_norm_in[iB]) - 1.0;

    // loop over layers
    for(int iL = 0; iL < nbLayers; ++iL) {
      trained_network_layer & lay = m_net.layers[iL];

      // loop over neurons (apply layer rules)
      for(int iN = 0; iN < lay.weights.size(); ++iN) {
        inputs[iL+1][iN] = lay.bias[iN];
```

```
        for(int iB = 0; iB < lay.weights[iN].size(); ++iB)
          inputs[iL+1][iN] += inputs[iL][iB] * lay.weights[iN][iB];
        inputs[iL+1][iN] = (*lay.func)(inputs[iL+1][iN]);
      }

    }

    // de-normalisation of output
    pixelO[0] = 0.5 * (inputs[nbLayers][0] + 1.0) * (m_net.max_norm_out - m_net.min_norm_out) +
m_net.min_norm_out;

    oIt.Set( pixelO );
  }

}

template <class TI, class TO>
void belcamApplyTrainedNeuralNetworkFilter<TI, TO>
::GenerateOutputInformation()
{
  Superclass::GenerateOutputInformation();
  this->GetOutput()->SetNumberOfComponentsPerPixel( 1 );
}
```

3.3 Outputs

The outputs are provided over each valid pixel of each S2 or L8 L2A images.

One image is produced for each BI: LAI, FAPAR and FCOVER.

The spatial resolution of the output is the finer spatial resolution available, i.e. 10 meters for S2 (even if 20 meters bands are used, they are re-scaled to 10 meters) and 30 meters for L8.

The non-valid pixels (snow/cloud/cloud shadow) should be masked and replaced with no data. Water mask is sometimes too approximative, pixels flagged with water should not be masked.

3.4 Quality flags

In addition to the output BI values, two quality flags are generated:

- the status of the input L2A pixels (no data / cloud / snow / water / land);
- the definition domain of the input spectral bands and of the output variables.

3.4.1 Status flag

This first flag gives the status of each pixel (nodata, cloud, snow, water, land). It is directly derived from the similar quality flag of the corresponding S2 or L8 L2A image.

3.4.2 Input/output values range

Several quality flags shall also be generated to inform about the validity of the input reflectance values and the output BI values. The bands values are put to 0 if the values are valid and to 1 if the values are invalid, i.e. out of range. The two following sections defined the validity range of the input and outputs.

3.4.2.1 Input definition domains

The definition domain of the input reflectance values for each spectral band are presented in Table 3-3 for S2 and in Table 3-4 for L8. These definition domains are also provided to the system in the Excel file along with the algorithm parameters.

Table 3-3. Definition domain of the S2 reflectance data

Acronym	Minimum value	Maximum value
B3	0	0.242
B4	0	0.292
B5	0	0.335
B6	0	0.608
B7	0	0.759
B8a	0	0.776
B8	0	0.790
B11	0	0.495
B12	0	0.494

Table 3-4. Definition domain of the L8 reflectance data

Acronym	Minimum value	Maximum value
B3	0	0.300
B4	0	0.266
B8a	0	0.788
B11	0	0.476
B12	0	0.504

3.4.2.2 Output definition domain and tolerance

In the situation where the retrieved BI values are outside of their definition domain (given in Table 3-5), the band 2 value of the quality flag is set to 1 (invalid value) and the product value is adjusted and set to the closest bound of the domain (i.e. either the minimum or the maximum bound). In the event that the output value is within a tolerance interval around the bound, the initial value is kept; yet, the "output out of range" flag is raised.

Table 3-5. Definition domain for the output product values

Product	Tolerance	Minimum value	Maximum value
LAI	0.2	0	8.00
FAPAR	0.1	0	0.94
FCOVR	0.1	0	1.00

4. Algorithm description for the single-date NDVI

4.1 Input

The inputs required for the NDVI computation correspond to the BOA reflectance values for each considered pixel.

Individual daily observations are used. For S2, bands 4 and 8 are used and for L8, bands 4 and 5 are used. Their spectral characteristics are described in Table 4-1 and Table 4-2.

Table 4-1. S2 spectral characteristics for NDVI computation

Acronym	Central wavelength (nm)	Width (nm)	Spatial resolution (m)	Band name
B4	665	30	10	Red
B8	842	115	10	NIR wide

Table 4-2. L8 spectral characteristics for NDVI computation

Acronym	Central wavelength (nm)	Width (nm)	Spatial resolution (m)	Band name
B4	650	20	30	Red
B5	865	15	30	NIR narrow

4.2 Algorithm implementation

Computing the NDVI over each acquisition is a simple band math operation:

$$\text{NDVI} = \frac{\rho(NIR) - \rho(RED)}{\rho(NIR) + \rho(RED)}$$

4.3 Output

The output NDVI S2 and L8 images are respectively at 10-m and 30-m spatial resolution.

The non-valid pixels (snow/cloud/cloud shadow) should be masked and replaced with no data. Water mask is sometimes too approximative, pixels flagged with water should not be masked.

4.4 Quality flag

One status flag is generated informing about the status of each pixel (no data / valid / cloud / snow / water). It is directly inferred from the similar quality flag of the corresponding S2 or L8 L2A image.

5. Algorithm description for the multi-date BI

5.1 Input

The inputs required for the reprocessed version of the BI are the mono-date BI time series generated from both S2 and L8 acquisitions.

An option should allow to use either the mono-date BI time series based S2 alone or the combination of the timer series based on S2 and L8.

5.2 Algorithm implementation

This algorithm will be implemented based on the `ProfileReprocessing` otb application available in the Sen2-Agri system and introduced in RD.11. It is summarized here after.

The reprocessing of the acquisition *n* will use all the acquisitions in a temporal window of *k* days to perform a linear combination weighted by the temporal distance between the date to reprocess and each available date. The processing will wait for the next observation:

$$LAI_{reprocessed}(n) = \sum_{tfirst}^{t_{n+1}} w_i LAI_{mono-dat}$$

The weight w_i given to each BI value used in the reprocessing can be computed as follows:

$$w_i = \alpha \frac{1}{1+|t_i - t_n|}$$

where α is the relative weight given to the temporal distance between the acquisition to reprocess and one acquisition used in the reprocessing, t_i is the date of the BI for which the weight is computed and t_n is the date of the BI to reprocess.

Compared to the initial implementation of the reprocessed BI product available in the Sen2-Agri system v1.7, several parameters were added, and existing parameters were adjusted:

- The relative weight α given to the acquisitions used in the reprocessing have been increased and set to 2;
- The number of acquisitions used in the reprocessing is no more limited to 3. Instead, all the acquisitions available in a temporal window should be used. This temporal window *k* is by-default set to 50 days;
- Along with past acquisitions, the reprocessing also uses the next acquisition value (n+1). Four situations might happen:
    - The next acquisition date is in a temporal window of 50 days and the next acquisition is valid (at pixel level) -> the next BI value is used in the reprocessing;
    - The next acquisition date is in a temporal window of 50 days but the next acquisition is not valid (eg. a cloudy pixel) -> the next BI value is not used in the reprocessing and only backward pixel are used;
    - The next acquisition date is not in a temporal window of 50 days > the reprocessing is not waiting longer than 50 days and will be launched and uses only backward acquisitions;
    - The season is finished before a next acquisition is available -> the reprocessing is launched and uses only backward acquisitions.
- To avoid too aggressive smoothing effects on the lowest BI values that might appear after an important drop of the BI – e.g. typical of a harvest event – a check of the residue of the reprocessing is done on the lowest values of the BI range. If the residue is higher than a

percentage of the original value, the reprocessing is limited to this percentage. For this, some default values are defined. Note that this default value could be adapted and should be set as parameters:

- maxResidue (Max percentage of change for small value) = 20%
- threshold4smallValueLAI = 2
- threshold4smallValueFAPAR = 0.2
- threshold4smallValueFCOVER = 0.2

For the latest case, a pseudo code is given below:

```
residue = monodate_BIvalue - reprocessed_BIvalue
residue_prop = (residue/monodate_BIvalue)
if residue_prop > maxResidue && monodate_BIvalue < threshold4smallValue ;
  then reprocessed_BIvalue = monodate_BIvalue + sign(residue) * maxChange * monodate_BIvalue
else
  reprocessed_BIvalue = reprocessed_BIvalue
```

5.3 Output

The outputs are provided over each valid pixel of the mono-date S2 or L8 L3B product. The product makes available the reprocessed version of each BI: LAI, FAPAR and FCOVER.

The non-valid pixels in the reprocessed mono-date image (snow/cloud/cloud shadow) should be masked and replaced with no data. Water mask is sometimes too approximative, pixels flagged with water should not be masked.

The spatial resolution of the output is 10 meters.

5.4 Quality flags

In addition to the output BI values, two quality flags are generated:

- the status of the input L2A pixels (no data / cloud / snow / water / land);
- the number of images used for the reprocessing.

5.4.1 Status flag

This first flag gives the status of each pixel (nodata, cloud, snow, water, land). It is directly derived from the similar quality flag of the corresponding S2 or L8 L3B image.

5.4.2 Number of images used for the reprocessing

The second status flag gives the number of images used for the reprocessing apart from the reprocessed image. Different cases must be considered:

- 0 = the mono-date image is non-valid;
- 1 = the mono-date image is valid, but no other images are available in the temporal window;
- 2 = only the date n+1 is available to reprocess the date n;
- 3 = 3 dates are available: the date n-1, the date n and the date n+1
- 4 = 4 dates are available: the date n-2, the date n-1, the date n and the date n+1
- …
- 10*X* = only backward dates are available (either because the next acquisition is outside the 50 days temporal window limit or because the next date is non-valid)
  - 102 = 2 dates are used: the date n-1 and the date n

- 103 = 3 dates are used: the date n-2, the date n-1 and the date n
- ….