PYOTB: A PYTHONIC EXTENSION OF OTB

Nicolas Narçon (INRAE)

OTB Users Days - 29 Nov. 2021

WHY ?

WHEN FIRST USING OTB, ONE MAY BE REPELLED BY THE VERBOSITY NEEDED FOR RUNNING AN APP

Example : undersampling a raster with OTB

```
import otbApplication

resampled = otbApplication.Registry.CreateApplication('RigidTransformResample')
resampled.SetParameterString('in', 'my_image.tif')
resampled.SetParameterString('interpolator', 'nn')
resampled.SetParameterString('transform.type', 'id')
resampled.SetParameterFloat('transform.type.id.scalex', 0.5)
resampled.SetParameterFloat('transform.type.id.scaley', 0.5)
resampled.SetParameterString('out', 'output.tif')
resampled.ExecuteAndWriteOutput()
```

8 lines of code, mainly just to set the parameters of the algorithm

WHY ?

OTHER PYTHON LIBRARIES NEED LESS "VERBOSE" CODE

Example : undersampling a raster with OpenCV

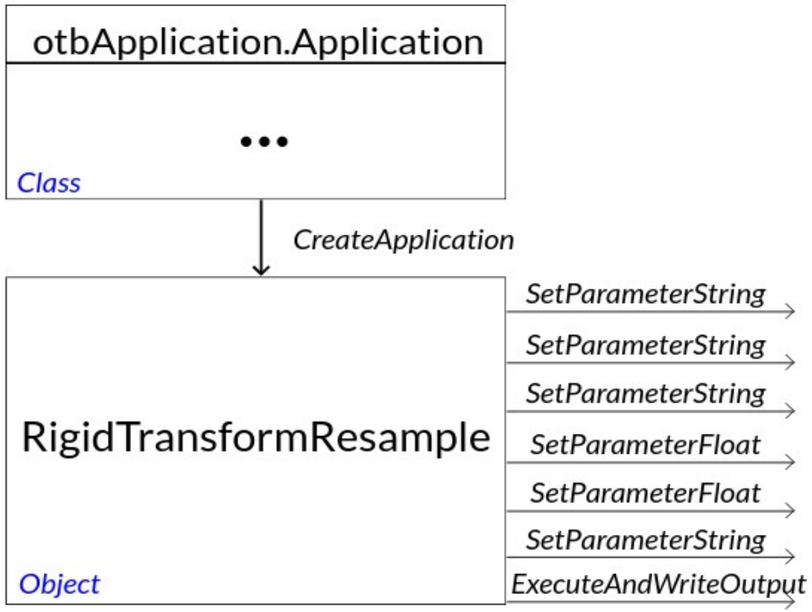
```
import cv2

resampled = cv2.resize(src, fx=0.5, fy=0.5, interpolation=cv2.INTER_NEAREST)
```

A "one-liner" code, something that we love in Python

PYOTB = OTB WITH LESS VERBOSE CODE

OTB Python API



However, for the user, what an Application is doing looks like a 'function'

EXAMPLE: RUNNING AN APP 'AS A FUNCTION'

Using OTB Python API:

```
import otbApplication

resampled = otbApplication.Registry.CreateApplication('RigidTransformResample')
resampled.SetParameterString('in', 'my_image.tif')
resampled.SetParameterString('interpolator', 'nn')
resampled.SetParameterString('transform.type', 'id')
resampled.SetParameterFloat('transform.type.id.scalex', 0.5)
resampled.SetParameterFloat('transform.type.id.scaley', 0.5)
resampled.SetParameterString('out', 'output.tif')
resampled.ExecuteAndWriteOutput()
```

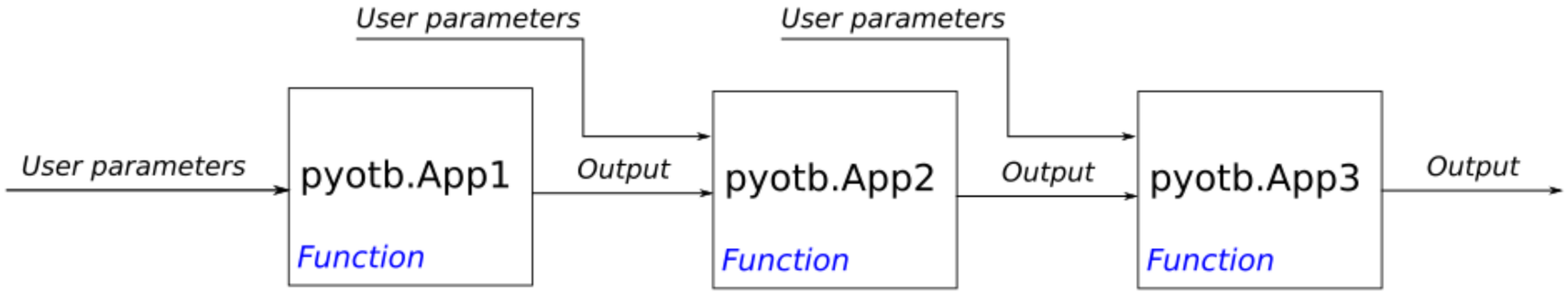
Using pyotb:

```
import pyotb

pyotb.RigidTransformResample({'in': 'my_image.tif', 'transform.type': 'id', 'transform.type.id.scaley': 0.5,
                              'transform.type.id.scalex': 0.5, 'interpolator': 'nn', 'out': 'output.tif'})
```

CONNECTION BETWEEN APPS

This 'functional' notation of pyotb is very convenient when using in-memory connections



CONNECTION BETWEEN APPS

This is what it looks like with OTB Python API

```
import otbApplication

resampled = otbApplication.Registry.CreateApplication('RigidTransformResample')
resampled.SetParameterString('in', 'my_image.tif')
resampled.SetParameterString('interpolator', 'linear')
resampled.SetParameterFloat('transform.type.id.scalex', 0.5)
resampled.SetParameterFloat('transform.type.id.scaley', 0.5)
resampled.Execute()

calibrated = otbApplication.Registry.CreateApplication('OpticalCalibration')
calibrated.ConnectImage('in', resampled, 'out')
calibrated.SetParameterString('level', 'toa')
calibrated.Execute()

dilated = otbApplication.Registry.CreateApplication('BinaryMorphologicalOperation')
dilated.ConnectImage('in', calibrated, 'out')
dilated.SetParameterString("filter", 'dilatation')
dilated.SetParameterString("structype", 'ball')
dilated.SetParameterInt("xradius", 3)
dilated.SetParameterInt("yradius", 3)
dilated.SetParameterString('out', 'output.tif')
dilated.ExecuteAndWriteOutput()
```

This is what it looks like with pyotb

```
import pyotb

resampled = pyotb.RigidTransformResample({'in': 'my_image.tif', 'interpolator': 'linear', 'transform.type.id.scaley': 0.5,
                                          'transform.type.id.scalex': 0.5})

calibrated = pyotb.OpticalCalibration(resampled, level='toa')

dilated = pyotb.BinaryMorphologicalOperation(calibrated, filter='dilatation', structype='ball', xradius=3, yradius=3)

dilated.write('output.tif', pixel_type='uint16')
```

COOL THINGS WE CAN DO WITH PYOTB

Prerequisite: having pyotb objects

```
import pyotb

# transforming filepaths to pyotb objects
input1, input2, input3 = pyotb.Input('image1.tif'), pyotb.Input('image2.tif'), pyotb.Input('image3.tif')
```

COOL THINGS WE CAN DO WITH PYOTB

Prerequisite: having pyotb objects

```
import pyotb

# transforming filepaths to pyotb objects
input1, input2, input3 = pyotb.Input('image1.tif'), pyotb.Input('image2.tif'), pyotb.Input('image3.tif')
```

Shape attribute

```
print(input1.shape)  # (width, height, channels)
```

COOL THINGS WE CAN DO WITH PYOTB

Prerequisite: having pyotb objects

```
import pyotb

# transforming filepaths to pyotb objects
input1, input2, input3 = pyotb.Input('image1.tif'), pyotb.Input('image2.tif'), pyotb.Input('image3.tif')
```

Shape attribute

```
print(input1.shape)  # (width, height, channels)
```

Slicing

```
input1[:, :, :3]  # selecting first 3 bands
input1[:, :, [0, 1, 4]]  # selecting bands 1, 2 and 5
input1[:1000, :1000]  # selecting 1000x1000 subset
```

Arithmetic operations

```
res = input1 * input2 - 2 * input2
res.write('output.tif', pixel_type='uint8')
```

Arithmetic operations

```
res = input1 * input2 - 2 * input2
res.write('output.tif', pixel_type='uint8')
```

Handling apps with several outputs

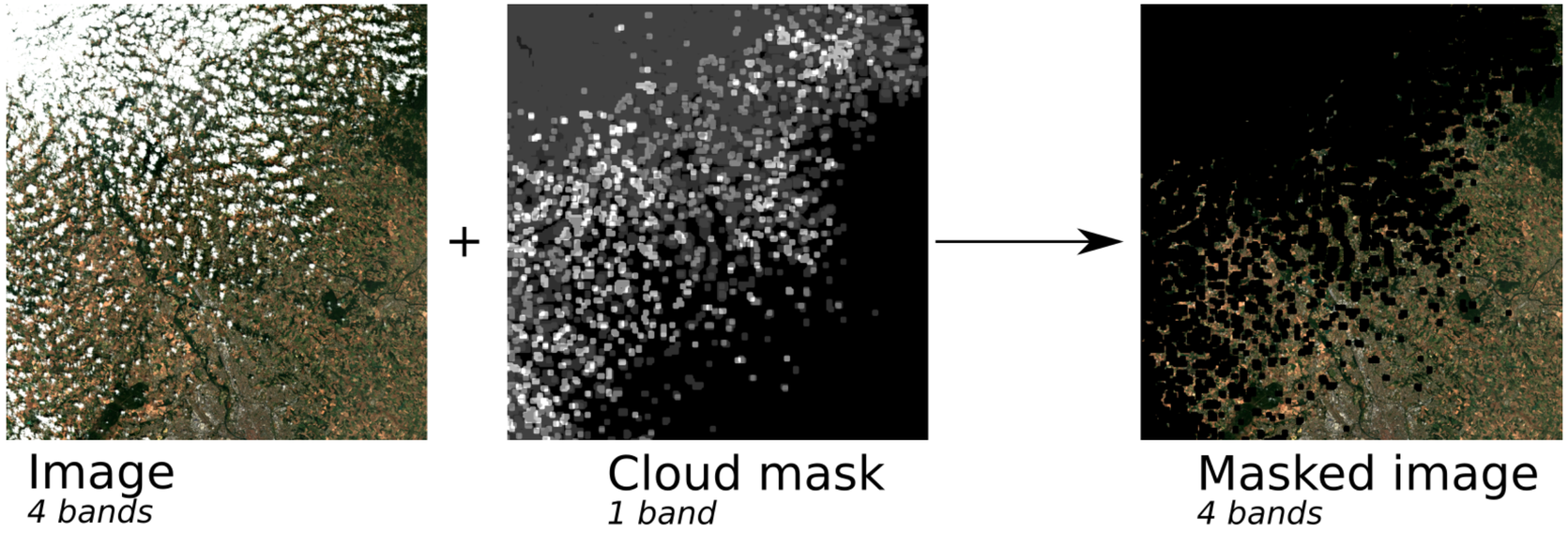
```
import pyotb

mean_smoothed = pyotb.MeanShiftSmoothing("my_image.tif", spatialr=16, ranger=16, thres=0.1, maxiter=100)

# we can access outputs of the app like this:
mean_smoothed.fout
mean_smoothed.foutpos
```

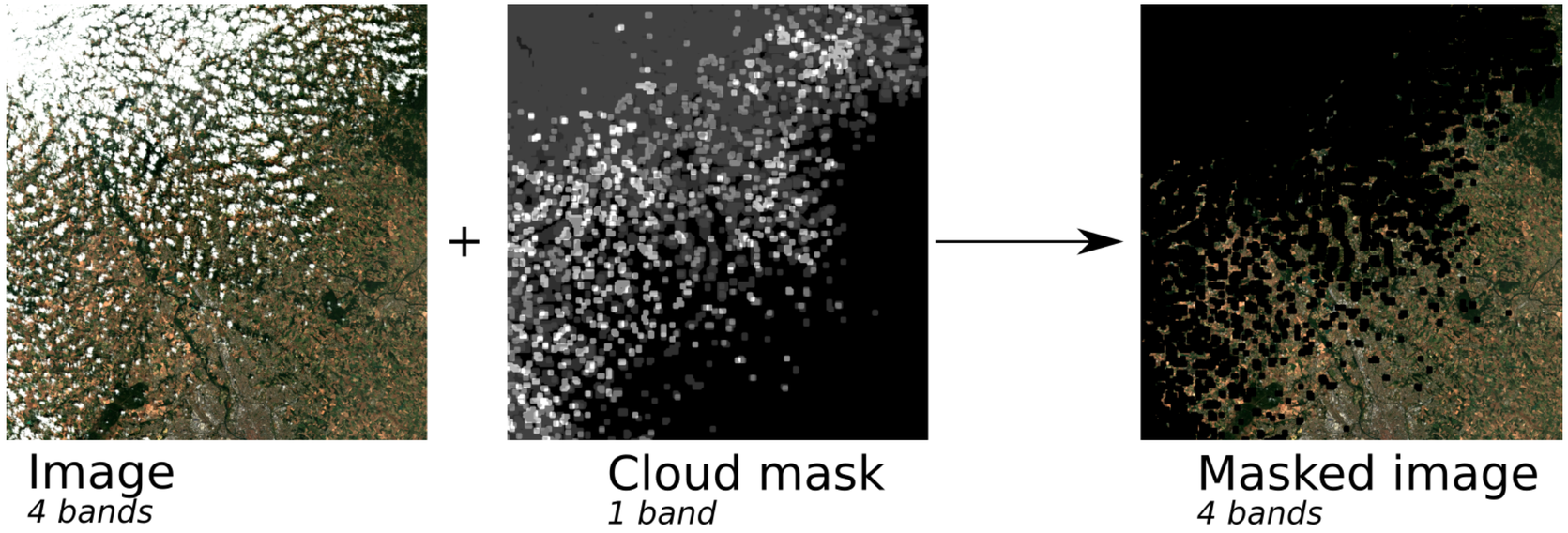
SIMPLE PROBLEMS REQUIRE SIMPLE CODE

Example: masking clouds on a satellite image



SIMPLE PROBLEMS REQUIRE SIMPLE CODE

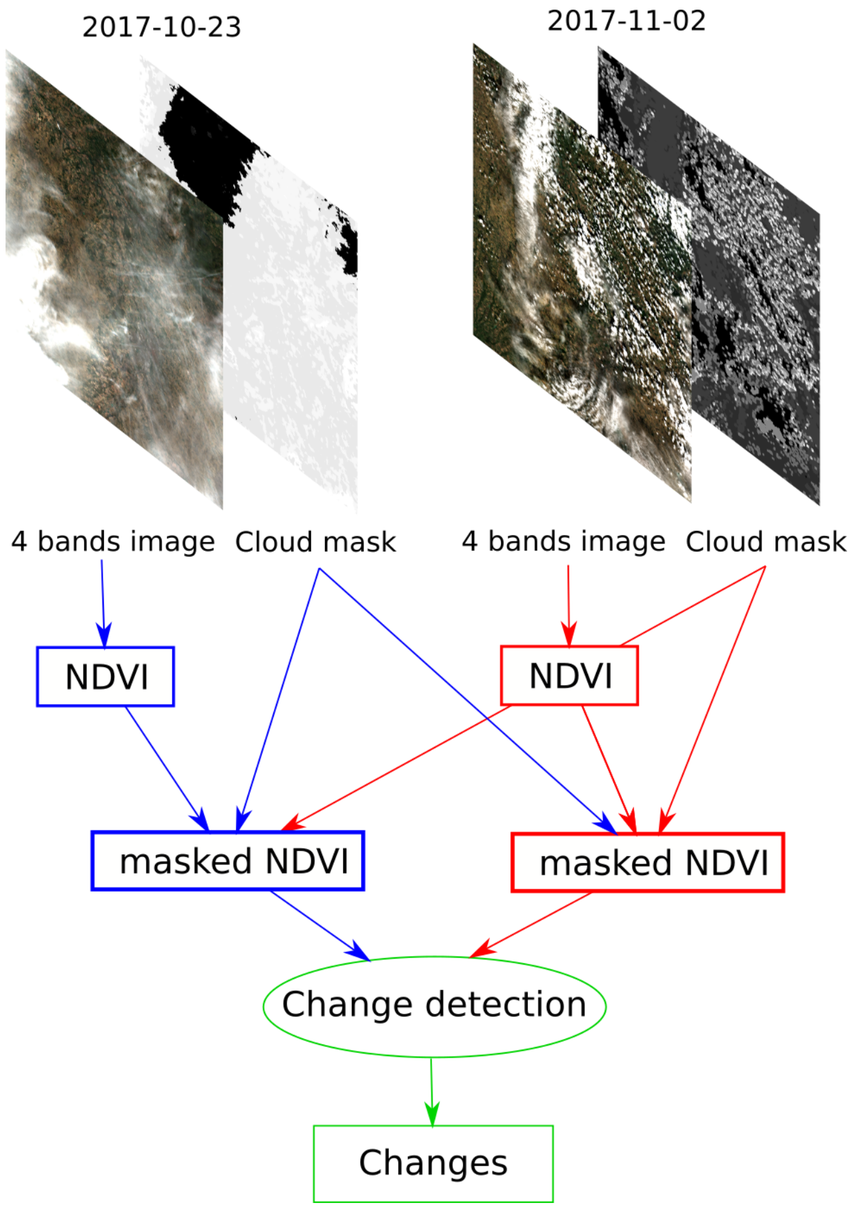
Example: masking clouds on a satellite image




```
import pyotb

masked_image = pyotb.where(pyotb.Input('mask.tif') > 0, 0, 'image.tif')
masked_image.write('masked_image.tif', pixel_type='int16')
```

LIVE CODING DEMO : CHANGE DETECTION ON SENTINEL-2 IMAGES



LIVE CODING DEMO : CHANGE DETECTION ON SENTINEL-2 IMAGES

Code:

```
import pyotb

image1 = pyotb.Input('SENTINEL2A_20171023-105154-753_L2A_T31TCJ_C_V2-2_FRE_10m.tif')
mask1 = pyotb.Input('SENTINEL2A_20171023-105154-753_L2A_T31TCJ_C_V2-2_CLM_R1.tif')
image2 = pyotb.Input('SENTINEL2A_20171102-105210-461_L2A_T31TCJ_D_V1-4_FRE_10m.tif')
mask2 = pyotb.Input('SENTINEL2A_20171102-105210-461_L2A_T31TCJ_D_V1-4_CLM_R1.tif')

ndvi1 = (image1[:, :, -1] - image1[:, :, 2]) / (image1[:, :, -1] + image1[:, :, 2])
ndvi2 = (image2[:, :, -1] - image2[:, :, 2]) / (image2[:, :, -1] + image2[:, :, 2])

merged_mask = (mask1 > 0) | (mask2 > 0)

masked_ndvi1 = pyotb.where(merged_mask == 1, -999, ndvi1)
masked_ndvi2 = pyotb.where(merged_mask == 1, -999, ndvi2)

pyotb.MultivariateAlterationDetector(in1=masked_ndvi1, in2=masked_ndvi2, out='changes.tif')
```

INTERACTION WITH NUMPY

```
import pyotb
import numpy as np

input = pyotb.Input('image.tif')  # this is a pyotb object

# Creating a numpy array of noise
white_noise = np.random.normal(0, 50, size=input.shape)  # this is a numpy object

# Adding the noise to the image
noisy_image = np.add(input, white_noise)  # magic: this is a pyotb object that has the same georeference as input
noisy_image.write('noisy_image.tif')
```

INTERACTION WITH TENSORFLOW

Let's consider a Python function using TensorFlow

```
import tensorflow as tf
def scalar_product(x1, x2):
    return tf.reduce_sum(tf.multiply(x1, x2), axis=-1)
```

INTERACTION WITH TENSORFLOW

Let's consider a Python function using TensorFlow

```
import tensorflow as tf
def scalar_product(x1, x2):
    return tf.reduce_sum(tf.multiply(x1, x2), axis=-1)
```

How to easily use it on images thanks to OTBTF ?

INTERACTION WITH TENSORFLOW

Let's consider a Python function using TensorFlow

```
import tensorflow as tf
def scalar_product(x1, x2):
    return tf.reduce_sum(tf.multiply(x1, x2), axis=-1)
```

How to easily use it on images thanks to OTBTF ?

```
import pyotb

def scalar_product(x1, x2):
    import tensorflow as tf
    return tf.reduce_sum(tf.multiply(x1, x2), axis=-1)

# Compute the scalar product
res = pyotb.run_tf_function(scalar_product)('image1.tif', 'image2.tif')  # magic: this is a pyotb object
res.write('scalar_product.tif')
```

CONCLUSION

https://gitlab.orfeo-toolbox.org/nicolasnn/pyotb/