Guide for OTB training courses (Solutions)

OTB Team

2018

Contents

1 Foreword

Solutions in the guide mostly use the command line interface of OTB applications. However, it is possible to apply the same commands by using the parameter values in the GUI (Mapla or Monteverdi).

2 Basics

2.1 Use Monteverdi and QGIS

Live demo done by the instructor to show image manipulations and functions.

2.2 The Orfeo ToolBox Application mechanism

2.2.1 Message 1

To reveal the first message, one can observe that a Pléiades image is encoded using 12 bits, and that there should not be any pixel value higher than $2^{12} - 1 = 4095$. We will therefore use the **BandMath** application to threshold pixels higher than this value:

```
$ otbcli_BandMath -il image1.tif           \
                  -out decoded1.tif uint8  \
                  -exp "im1b1>4095?255:0"
```

The encoded text will appear in white on a black background.

2.2.2 Message 2

To decode the second message we will compute the image gradient using the **EdgeExtraction** application:

```
$ otbcli_EdgeExtraction -in image2.tif    \
                        -filter gradient    \
                        -out decoded2.tif
```

2.2.3 Message 3

To decode the third message, one can compute a vegetation index such as the NDVI using the **RadiometricIndices** application:

```
$ otbcli_RadiometricIndices -in image3.tif      \
                            -channels.red 1       \
                            -channels.nir 4       \
                            -list Vegetation:NDVI \
                            -out decoded3.tif
```

One can also compute the NDVI using the **BandMath** application:

```
$ otbcli_BandMath -il image3.tif                    \
                  -out decoded3.tif                 \
                  -exp "(im1b4-im1b1)/(im1b4+im1b1)"
```

2.2.4 Message 4

To reveal the 4th message, we are going to isolate the 2 least significant bits using the **BandMath** application:

```
$ otbcli_BandMath -il image4.tif               \
                  -out decoded4.tif            \
                  -exp "im1b1-4*rint(im1b1/4)"
```

The $4 * rint(im1b1/4)$ expression does not contain the 2 least significant bits, and the difference with original image thus reveals the message.

2.2.5 Message 5

To reveal this 5th message, we are going to do a principal component analysis using the **DimensionalityReduction** application, and extract the last band, where the image noise is condensed, using the **ExtractROI** application.

```
$ otbcli_DimensionalityReduction -in image5.tif  \
                                 -out pca6.tif   \
                                 -method pca
$ otbcli_ExtractROI -in pca6.tif                 \
                    -out decoded6.tif            \
                    -cl Channel4
```

2.2.6 Message 6

To reveal the 6th message, we are going to use the idempotent property. If the message has been encoded using an idempotent transform, then $f(message) = message$, and therefore $f(message) - message = 0$, while outside of the image we will see $f(image)$.

```
$ otbcli_GrayScaleMorphologicalOperation -in image6.tif             \
                                         -out ouverture6.tif        \
                                         -structype.ball.xradius 1  \
                                         -structype.ball.yradius 1  \
                                         -filter opening
$ otbcli_BandMath -il image6.tif ouverture6.tif                     \
                  -out decoded6.tif                                 \
                  -exp "(im2b1-im1b1)"
```

2.3 Orfeo ToolBox internals

2.3.1 Encoding images

gdalinfo with *image1.tif* yields:

```
$ $ gdalinfo image1.tif
Driver: GTiff/GeoTIFF
Files: image1.tif
Size is 2000, 2000
Coordinate System is `'
Origin = (5400.000000000000000,4300.000000000000000)
Pixel Size = (1.000000000000000,1.000000000000000)
Image Structure Metadata:
INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (    5400.000,    4300.000)
Lower Left  (    5400.000,    6300.000)
Upper Right (    7400.000,    4300.000)
Lower Right (    7400.000,    6300.000)
Center      (    6400.000,    5300.000)
Band 1 Block=2000x1 Type=Float32, ColorInterp=Gray
Band 2 Block=2000x1 Type=Float32, ColorInterp=Undefined
Band 3 Block=2000x1 Type=Float32, ColorInterp=Undefined
Band 4 Block=2000x1 Type=Float32, ColorInterp=Undefined
```

Pixels are therefore encoded with 32 bits floating point numbers. By looking at the pixel values in **monteverdi**, you can see that the values are integers and between 100 and 1600 approximately. The 32 bit encoding is therefore needlessly expensive.

The **Convert** application allows to convert the encoded pixel type:

```
$ otbcli_Convert -in image1.tif -out image1_uint16.tif uint16
```

We can now compare image sizes, and see that the generated image only uses half the space of the original image:

```
$ du -h image1.tif
62M image1.tif

$ du -h image1_uint16.tif
31M image1_uint16.tif
```

Using the **CompareImages** application also show that the image content is identical.

```
$ otbcli_CompareImages -ref.in image1.tif -meas.in image1_uint16.tif
2016 Mar 08 13:59:24  :  Application.logger  (INFO) Using whole reference image
                         since the ROI contains no pixels or is not specified
2016 Mar 08 13:59:24  :  Application.logger  (DEBUG) Region of interest used
                         for comparison : ImageRegion (0x7ffcb6a6d930)
Dimension: 2
Index: [0, 0]
Size: [2000, 2000]

2016 Mar 08 13:59:24  :  Application.logger  (INFO) reference image channel 1
                         is compared with measured image channel 1
2016 Mar 08 13:59:24  :  Application.logger  (INFO) MSE: 0
2016 Mar 08 13:59:24  :  Application.logger  (INFO) MAE: 0
2016 Mar 08 13:59:24  :  Application.logger  (INFO) PSNR: 0
Output parameters value:
mse: 0
mae: 0
psnr: 0
```

To compute the NDVI, we use the following commands:

```
$ otbcli_RadiometricIndices -in image1.tif
                            -out image1_ndvi.tif uint16
                            -channels.red 1
                            -channels.green 2
                            -channels.blue 3 -channels.nir 4
                            -list Vegetation:NDVI
```

Looking at the generated image in **monteverdi**, we see that the image is 0 everywhere. The output encoding is therefore not appropriate. A floating point type should be used (the default for example).

2.3.2 .geom files

The `geom` file contains information necessary for geometric and radiometric image corrections.

2.3.3 Extended filenames

Read option The *skipgeom* read parameter allows to ignore the associated `geom` file. The ground pixel size is false and the acquisition date and sensor information has disappeared.

The `geom` extended filename parameter allows to attach a `geom` file to an existing image. Among other things, it is useful for radiometric and geometric processing. By default, Orfeo ToolBox (via OSSIM) looks for a `geom` file with the same name as the image.

Write options

```
$ otbcli_Convert -in image1.tif
  -out "image1_comp.tif?&gdal:co:NBITS=12&gdal:co:COMPRESS=LZW"  uint16
```

The size of the image is:

```
$ du -h image1_comp.tif
23M image1_comp.tif
```

8 MB are saved compared to the unsigned 16 bits image. The **CompareImages** application shows that the content stays the same.

The box parameter is used as such:

```
$ otbcli_Convert -in image1.tif
                 -out "image1_comp.tif?&box=1000:1000:100:100"  uint16
```

After this command, the output image is an extract of the total output, beginning at the $(1000, 1000)$ index and of size 100x100. This option can be used to visualize a result before processing the full image.

2.3.4 Streaming management

The **LocalStatiticsExtraction** application is used as follows:

```
$ otbcli_LocalStatisticExtraction -in image1.tif -out image1_ls.tif
                                  -radius 9
```

We notice that the computation is done in several steps, interleaved with disk I/O. By default, Orfeo ToolBox specifies the optimal tiling.

To completely disable streaming, use the following extended filename options:

```
$ otbcli_LocalStatisticExtraction -in image1.tif *
        -out "image1_ls.tif?&streaming:type=none" -radius 9
```

When processing is done in a single pass, only one disk write follows.

```
$ otbcli_LocalStatisticExtraction -in image1.tif
  -out "image1_ls.tif?&streaming:type=stripped \
  &streaming:sizemode=nbsplits&streaming:sizevalue=1000"
  -radius 9
```

This time, we see multiple computation and disk writing steps. Computation time can be almost twice as long, because over tiling is suboptimal.

2.3.5 Multithreading

Here's how to set the number of threads to 1:

```
$ export ITK_GLOBAL_DEFAULT_NUMBER_OF_THREADS=1
$ otbcli_LocalStatisticExtraction -in image1.tif
                                  -out "image1_ls.tif?&streaming:type=none"
                                  -radius 9
```

In that case, computation time is much more significant. It can also be seen that increasing the number too much does not improve performance above a certain point (typically the number of CPU cores).

3 Optical VHR image, from pre-processing to GIS

3.1 Pre-processing of VHR satellite images

3.1.1 Atmospheric corrections

Compute TOA reflectance:

```
$ otbcli_OpticalCalibration \
-in phr_xs_osr_mipy.tif \
-out phr_xs_osr_mipy_toa.tif uint16 \
-level toa \
-milli 1
```

```
$ otbcli_OpticalCalibration \
-in phr_pan_osr_mipy.tif \
-out phr_pan_osr_mipy_toa.tif uint16 \
-level toa \
-milli 1
```

Compute TOC reflectance:

```
$ otbcli_OpticalCalibration
-in phr_xs_osr_mipy.tif \
-out phr_xs_osr_mipy_toc.tif uint16 \
-level toc \
-milli 1
```

```
$ otbcli_OpticalCalibration
-in phr_pan_osr_mipy.tif \
-out phr_pan_osr_mipy_toc.tif uint16 \
-level toc \
-milli 1
```

We can use the **BandMathX** application to compute image differences:

```
$ otbcli_BandMathX
-il phr_xs_osr_mipy_toa.tif phr_xs_osr_mipy_toc.tif \
-out diff_xs_toa_toc.tif int16 \
-exp "im1-im2"
```

Then, for the panchromatic image:

```
$ otbcli_BandMath
-il phr_pan_osr_mipy_toa.tif phr_pan_osr_mipy_toc.tif \
-out diff_pan_toa_toc.tif int16 \
-exp "im1b1-im2b1"
```

The blue band is more impacted by atmospheric effects. Indeed molecular diffusion on the signal is high in this spectral range (λ^{-4} decay).

3.1.2 P+XS Fusion

```
$ otbcli_BundleToPerfectSensor \
-inp phr_pan_osr_mipy_toa.tif \
-inxs phr_xs_osr_mipy_toa.tif \
-mode phr \
-out phr_pxs_osr_mipy.tif uint16
```

3.1.3 Ortho-rectification

1. Orthorectification without DEM:

```
$ otbcli_OrthoRectification \
-io.in phr_pxs_osr_mipy.tif \
-io.out phr_orthopxs_osr_mipy.tif uint16
```

2. Orthorectification with DEM and geoid:

```
$ otbcli_OrthoRectification \
-io.in phr_pxs_osr_mipy.tif \
-io.out phr_orthopxs_osr_mipy.tif uint16 \
-elev.dem SRTM/ \
-elev.geoid Geoid/egm96.grd
```

3. Default projection is UTM. In our case the UTM zone is 32 North.

4. Orthorectification in WGS84 and in Lambert 93:

```
$ otbcli_OrthoRectification \
-io.in phr_pxs_osr_mipy.tif \
-io.out phr_orthopxs_osr_mipy.tif uint16 \
-elev.dem SRTM/ \
-elev.geoid Geoid/egm96.grd \
-map epsg -map.epsg.code 4326
```

```
$ otbcli_OrthoRectification \
-io.in phr_pxs_osr_mipy.tif \
-io.out phr_orthopxs_osr_mipy.tif uint16 \
-elev.dem SRTM/ \
-elev.geoid Geoid/egm96.grd \
-map lambert93
```

3.2 Image segmentation and export to GIS

3.2.1 Image smoothing using the MeanShift algorithm

Smoothing step:

```
$ otbcli_MeanShiftSmoothing -in phr_orthopxs_osr_mipy_xt.tif
  -fout meanshift.tif
  -foutpos meanshift_pos.tif
  -ranger 25
  -spatialr 3
  -maxiter 10 -modesearch 0
```

spatialr corresponds to the spatial radius. A higher value will lead to a stronger smoothing and will also take more time.

ranger is the spectral radius which means how pixels in the spatial radius and with close radiometry values will be averaged. A higher *ranger* will increase the smoothing effect.

foutpos can not be interpreted. It will be used in the next step of the exercise.

3.2.2 Segmentation

The segmentation step can be performed with the following command:

```
$ otbcli_LSMSSegmentation -in meanshift.tif
                          -inpos meanshift_pos.tif
                          -out init_seg.tif uint32
                          -ranger 10
                          -spatialr 2
```

It is pretty hard to interpret directly the output image. It can be colorized using the following command:

```
$ otbcli_ColorMapping -in init_seg.tif
                      -method optimal
                      -out init_seg_cm.tif uint8
```

This application performs color mapping and the algorithm analyzes nearest segments to maximize the contrast between polygons.

The segmented image can then be analyzed. We can see a lot of small regions which do not correspond to real objects. There are 2 ways to filter those regions by using the *minsize* parameter of the **LSMSSegmentation** application or by fusing them (next step).

3.2.3 Dealing with small regions

Small regions can be merged with the command:

```
$ otbcli_LSMSSmallRegionsMerging -in meanshift.tif
                                 -inseg init_seg.tif
                                 -out final_seg.tif uint32
                                 -minsize 100
```

Then, we use again the **ColorMapping** application to colorize the output.

```
$ otbcli_ColorMapping -in final_seg.tif
                      -method optimal
                      -out final_seg_cm.tif uint8
```

By comparing both colorized segmentations we can see that regions smaller than *minsize* where fused with adjacent regions close in terms of radiometry.

3.2.4 Vectorization

Start by computing the NDVI index from the initial image:

```
$ otbcli_RadiometricIndices -in phr_orthopxs_osr_mipy_xt.tif
                            -out phr_ndvi.tif
                            -list Vegetation:NDVI
                            -channels.blue 3
                            -channels.red 1
                            -channels.green 2
                            -channels.nir 4
```

Then, concatenate input image and NDVI index

```
$ otbcli_ConcatenateImages -il phr_orthopxs_osr_mipy_xt.tif phr_ndvi.tif
                           -out phr_radio_ndvi.tif
```

Finally, we can perform the vectorization:

```
$ otbcli_LSMSVectorization -in phr_radio_ndvi.tif
                           -inseg final_seg.tif -out segmentation.shp
```

In QGIS, we can see that for each polygon we've access to the mean and variance in each spectral band (including NDVI).

3.2.5 Polygons filtering in QGIS

To select all segments not belonging to shadow areas we are using the expression features as follows:

```
meanB0 > 140 or meanB1 > 140 or meanB2 > 140 or meanB3 > 140
```

Then, we use the field calculator to create a new virtual field (real type) called *compac* with the following formula:

```
sqrt(area($geometry)/perimeter($geometry)
```

Finally, to select small and compact objects with high NDVI we can use the following expression:

```
compac > 0.1 and nbPixels < 500 and meanB4 > 0.2
```

4 Supervised classification of a satellite image time series

In the following solution, the `$DATA` environment variable correspond to the folder containing the workshop data.

4.1 Single date training

The single date training can be achieved with the following command line:

```
$ otbcli_TrainImagesClassifier -io.il $DATA/images/20160607_T31TCJ_ROI_20m.tif \
                               -io.vd $DATA/references/training/training.shp   \
                               -io.out model.rf                                \
                               -sample.vfn CODE -classifier rf                 \
                               -classifier.rf.nbtrees 50 -classifier.rf.max 20 \
                               -classifier.rf.cat 13
```

This first run outputs the following results:

```
Confusion matrix (rows = reference labels, columns = produced labels):
       [10]  [31]  [32]  [34]  [36]  [41]  [42 ] [43]  [44]  [51] [211] [221] [222]
[ 10]   374     3     0    26     1     6    19    11     2     0    13    10    13
[ 31]     0   436     5     7    14     0     0     0     0     0     7     8     1
[ 32]     3    16   420     6    15     0     0     0     0     0    12     3     3
[ 34]    30    16    21   268    41     1    13     2     1     0    27    18    40
[ 36]    10     6    13    31   336     0     7     0     0     0    42    13    20
[ 41]     5     0     0     0     0   388    49    21    13     0     0     1     1
[ 42]    31     1     3    10     3    44   288    36    22     0     0     5    35
[ 43]    18     0     2    11     1    37    59   227   114     1     0     7     1
[ 44]     7     0     3     3     2     5    10    71   371     0     0     5     1
[ 51]     0     0     6     0     0     0     0     0     1   470     0     1     0
[211]    19     7    13    41    64     0     3     0     0     0   266    14    51
[221]    18    18     8    13    23     1    11     4     3     0    38   332     9
[222]    22     0     1    30    12     0    14     0     0     0    16     4   379

[...]

Global performance, Kappa index: 0.710774
```

The displayed performance is rather optimistic, since the samples used to estimate it comes from the same polygons as for the training. To obtain a more realistic evaluation of performances, it is better to select a dedicated polygon set for validation.

```
$ otbcli_TrainImagesClassifier -io.il $DATA/images/20160607_T31TCJ_ROI_20m.tif \
                               -io.vd $DATA/references/training/training.shp   \
                               -io.valid $DATA/references/testing/testing.shp  \
                               -io.out model.rf                                \
                               -sample.vfn CODE -classifier rf                 \
                               -classifier.rf.nbtrees 50 -classifier.rf.max 20 \
                               -classifier.rf.cat 13
```

Which outputs the following results:

```
Confusion matrix (rows = reference labels, columns = produced labels):
       [10]  [31]  [32]  [34]  [36]  [41]  [42]  [43]  [44]  [51] [211] [221] [222]
[ 10]   795     6     6    47     9     4    13    18     0     0    22    10    23
[ 31]     1   777    38    14    42     0     0     0     0     1    59    21     0
[ 32]     1    34   865     3    12     2    14     1     1     0     2    15     3
[ 34]    50   273   120    72    51     0     8     0     0    49   105   157    68
[ 36]    23    27    45   186   336     0     1     1     0     0   215    79    40
[ 41]     4     0     1     1     0   665   176    53    49     0     1     1     2
[ 42]    20     1     3    11     2    98   652    75    43     0     5     8    35
[ 43]    21     0     1    19     5    44   207   464   146     1     8    17    20
[ 44]     7     0     1     3     0    13    23   240   662     0     0     4     0
[ 51]     0     0     1     0     0     0     0     3     1   945     0     3     0
[211]    81    21    17    81   112     1    16     4     0     0   507    40    73
[221]    46    51    22    43    42     0    18    10     2     2   107   541    69
[222]    70     0     0    68    23     0    71     1     0     0    45    11   664

[...]

Global performance, Kappa index: 0.611403
```

If the `cleanup` option is deactivated by adding `-cleanup 0` parameter, the application does not erase temporary outputs.

The following XML files contain the statistics of available samples for each class, for training and validation set:

- `model.rf_statsTrain_1.xml`
- `model.rf_statsValid_1.xml`

The following shapefile files contain the samples used for training and for validation:

- `model.rf_samplesTrain_1.shp`
- `model.rf_samplesValid_1.shp`

Those files contain points corresponding to selected samples within the training and validation polygons. Each point has a set of fields corresponding to the radiometric measurement at the point location in the image. Those two files can be displayed in a GIS (in QGIS for instance).

4.2 Spot the date with the best performance

The following command line allows to do the training for each date:


```
$ for f in $DATA/images/*.tif; do echo $f;            \
      otbcli_TrainImagesClassifier -io.il $f          \
      -io.vd $DATA/references/training/training.shp   \
      -io.valid $DATA/references/testing/testing.shp  \
      -sample.vfn CODE -classifier rf                 \
      -classifier.rf.nbtrees 50 -classifier.rf.max 20 \
      -classifier.rf.cat 13 -io.out model.rf   | grep Kappa; done
```

Kappa coefficients for each date can be retrieved from the output:

Date	Kappa
2016-06-07	0.609741
2016-07-07	0.615163
2016-08-06	0.593739
2016-09-05	0.614463
2016-10-05	0.622246

We can see that the coefficient does not vary much, but that the `2016-10-05` date has slightly better performances.

4.3 Classifying and producing a colored classification map

To perform the classification, we use the `model.rf` file learnt on date `2016-10-05`, and use the following command line:

```
$ otbcli_ImageClassifier -in $DATA/images/20161005_T31TCJ_ROI_20m.tif \
                         -out classif_20161005.tif uint8              \
                         -model model.rf
```

The `classif_20161005.tif` contains for each pixel the label of the class which has been assigned to it. In ordered to ease the visualisation of the classification result, we can transform this image by setting a different color for each class, using the **ColorMapping** application:

```
$ otbcli_ColorMapping -in classif_20161005.tif            \
                      -out classif_20161005_rgb.tif uint8 \
                      -method custom -method.custom.lut   \
                      $DATA/support/color_map.txt
```

Another way of displaying the `classif_20161005.tif` results is to open it in QGIS and use the style file provided in `support/classif.qml`.

4.4 Evaluate global performance

To evaluate global performance over the whole validation set, one can use the **ComputeConfusionMatrix** application. This application allows to evaluate any classification map (for instance one that have been post-processed). Beware not to use as input the colored map created during previous step, which is only to be used for visualization purposes.

```
$ otbcli_ComputeConfusionMatrix -in classif_20161005.tif -ref vector  \
                  -ref.vector.in $DATA/references/testing/testing.shp \
                                 -out confusion_20161005.csv          \
                                 -ref.vector.field CODE
```

The performance measured using the whole validation set is as follows:

```
Confusion matrix (rows = reference labels, columns = produced labels):
       [  10] [  31] [  32] [  34] [  36] [  41] [  42] [  43] [  44] [  51] [ 211] [ 221] [ 222]
[  10] 113219    219   2240  10349   4090   2654   2469   1029    233    202   7387   2571   3453
[  31]      8  12282    265     66    346      0      5      1      0      0    174    192     27
[  32]      1     21   1143      5     27      0      2      0      0      0      3      7     10
[  34]    158     47     73   1469    146      9     68     25     12      0    187     11     70
[  36]     11      8      2      8    889      0      4      0      0      0     25      1     11
[  41]     45      0      4     34      7   4637    674    287    135      1      9      9     66
[  42]    800     14    107    675    355   5084  33947   3642   2684     13    468    490   3358
[  43]    816      4     97    417    130   2222   5399  18726   9404     66    137    171    857
[  44]     12      0      7      7      5     54    105    561   2807      6      0     13     27
[  51]    187     26     92     10     73      1     18    249    257  24330      4    300      4
[ 211]    367     73     55    882   1143      9     58      1      0      0   6755    126    301
[ 221]    244    337    372     79    338      2    197     49     16     32    174  10400    398
[ 222]     72      2     66     40    115      9    195      1      0      0     98     52   3474

Precision of class [10] vs all: 0.976531
Recall of class [10] vs all: 0.754215
F-score of class [10] vs all: 0.851095

[...]

Kappa index: 0.659139
```

One can note two things:

- First global performance is slightly better than the performance assessed during training. This comes from the fact that during the training step, we use the same number of samples for all classes, whereas when using the **ComputeConfusionMatrix**, all available samples are used. Some classes both have more available samples and are more easy to classify (such as class 51: water), and are therefore improving performances.

- Second, the annual crops class exhibit a lot of confusion with all other classes. It has a recall of 0.75 and a precision of 0.97. This means that if 97% of pixels identified as annul crops really belong to this class, 25% of pixel that belong to this class according to the reference validation set have been misclassified. We will see in last section that adding multi-temporal information in the classification allows to increase this performance.

4.5 Classification map regularization

To regularize the classification map using a majority voting algorithm, one can use the following command line:

```
$ otbcli_ClassificationMapRegularization -ip.radius 1 -ip.suvbool 0  \
                                         -io.in classif_20161005.tif \
                                         -io.out classif_20161005_reg.tif uint8
```

After regularization, we can evaluate the performances of the new classification map:

```
$ otbcli_ComputeConfusionMatrix -in classif_20161005_reg.tif -ref vector  \
                    -ref.vector.in $DATA/references/testing/testing.shp    \
                                   -out confusion_20161005_reg.csv         \
                                   -ref.vector.field CODE

Kappa index: 0.709103
```

Regularization improves significantly the classification map performance. This is due to the fact that reference data are rather regular, and applying this processing makes the classification map look more like the reference data.

4.6 Classification with multiple dates

Lets replay the different step with all the dates altogether:

First, training:

```
$ otbcli_TrainImagesClassifier -io.il $DATA/images/all.vrt                      \
                               -io.vd $DATA/references/training/training.shp    \
                               -io.valid $DATA/references/testing/testing.shp   \
                               -io.out model_all.rf                             \
                               -sample.vfn CODE -classifier rf                  \
                               -classifier.rf.nbtrees 50 -classifier.rf.max 20 \
                               -classifier.rf.cat 13
```

Then, classification:

```
$ otbcli_ImageClassifier -in $DATA/images/all.vrt    \
                         -out classif_all.tif uint8  \
                         -model model_all.rf
```

Next, regularisation:

```
$ otbcli_ClassificationMapRegularization -ip.radius 1 -ip.suvbool 0  \
                                         -io.in classif_all.tif \
                                         -io.out classif_all_reg.tif uint8
```

Finally, evaluation of performances:

```
$ otbcli_ComputeConfusionMatrix -in classif_all_reg.tif -ref vector       \
                    -ref.vector.in $DATA/references/testing/testing.shp   \
                                   -out confusion_all_reg.csv             \
                                   -ref.vector.field CODE
```

```
Confusion matrix (rows = reference labels, columns = produced labels):
       [  10] [  31] [  32] [  34] [  36] [  41] [  42] [  43] [  44] [  51] [ 211] [ 221] [ 222]
[  10] 140681     13     68   1893    790    280    494    545     27    108   3569    383   1264
[  31]     19  12732     87     77    200      3      8      0      0      0    104    131      5
[  32]      1      6   1211      0      1      0      0      0      0      0      0      0      0
[  34]      0     32     23   2131     34      0     12     13      0      0     19      5      6
[  36]      0      0      0      4    937      0      4      0      0      0      8      0      6
[  41]      2      0      1      0      0   5369    330    111     87      1      0      7      0
[  42]    148     10     75    465     84   3166  41818   2943   1850      2    316    362    398
[  43]    143      6     67    528     57   1545   5556  21781   8265     35    166    195    102
[  44]     11      0     14     13      0     34     60    354   3108      0      2      8      0
[  51]      7     18     53      0     12      0      3     37     24  25319      2     76      0
[ 211]    213     41     17    444    491      8     45      1      0      0   8326    104     80
[ 221]    187     87     66    123    159      0    109     83      3     14    208  11374    225
[ 222]     29      0      2     15     41      4     42      0      0      0     43     11   3937

Precision of class [10] vs all: 0.994627
Recall of class [10] vs all: 0.937155
F-score of class [10] vs all: 0.965036

[...]

Kappa index: 0.828411
```

Adding multi-temporal information in classification result in a significant performance improvement. One can note that recall of the annual crops class has raised to 93%, which means that now 93% of this class in the ground truth are correctly classified. This improvement was expected since crop classes usually have a strong temporal signature with respect to other classes.

Finally, we can generate the color classification map:

```
$ otbcli_ColorMapping -in classif_all_reg.tif            \
                      -out classif_all_reg_rgb.tif uint8 \
                      -method custom -method.custom.lut  \
                      $DATA/support/color_map.txt
```

5 SAR processing on Sentinel-1 images

5.1 Introduction to SAR image processing

Introduction to SAR image

1. The 2 extracts correspond to polarimetric combinations HH (for horizontal transmission and horizontal reception) and HV (for horizontal transmission and vertical reception).
2. The 2 bands correspond to the real and the imaginary parts of the complex signal.
3. We can use the **BandMath** application to compute the image intensity:

   For HH:

   ```
    $ otbcli_BandMath \
   -il s1_hh.tif \
   -out intensity_hh.tif int32 \
   -exp "im1b1*im1b1+im1b2*im1b2"
   ```

   For HV:

   ```
   $ otbcli_BandMath \
   -il s1_hv.tif \
   -out intensity_hv.tif int32 \
   -exp "im1b1*im1b1+im1b2*im1b2"
   ```

Radiometric calibration

1. **SARCalibration**
2. In the case of Sentinel-1, calibration coefficients are directly read in the product metadata

   ```
   $ otbcli_SARCalibration \
   -in s1_hh.tif \
   -out s1_hh_gamma0.tif \
   -lut gamma
   ```

   For HV:

   ```
   $ otbcli_SARCalibration \
   -in s1_hv.tif \
   -out s1_hv_gamma0.tif \
   -lut gamma
   ```

3. Warning: pixel <= 0 in the log expression!

   ```
   $ otbcli_BandMath \
   -in s1_hh_gamma0.tif \
   -out s1_hh_gamma0_db.tif \
   -exp "im1b1>0?10*log10(im1b1):0"
   ```

   For HV:

   ```
   $ otbcli_BandMath \
   -in s1_hv_gamma0.tif \
   -out s1_hv_gamma0_db.tif \
   -exp "im1b1>0?10*log10(im1b1):0"
   ```

Geometric corrections

1. Orthorectification without DEM:

```
$ otbcli_OrthoRectification \
-io.in s1_hh_gamma0.tif \
-io.out s1_hh_gamma0_ortho.tif uint16
```

2. With a DEM and a geoid:

```
$ otbcli_OrthoRectification \
-io.in s1_hh_gamma0.tif \
-io.out s1_hh_gamma0_ortho.tif uint16 \
-elev.dem SRTM/ \
-elev.geoid Geoid/egm96.grd
```

3. Default projection is UTM. 32 North.

Speckle filtering

1. Available methods are: Lee, Frost, Kuan and Gamma MAP. Speckle filtering allows to increase image quality and facilitate image analysis and object identification.

2. Using the **Despeckle** application and the *Frost* filter:

```
$ otbcli_Despeckle \
-in intensity_hh.tif \
-out intensity_hh_speckle.tif \
-filter frost \
-filter.frost.rad 3
```

   The effect of increasing the radius is to further smooth the image. It improves image quality in rather smooth areas but degrades details in more contrasted areas and on small structures.

3. The histogram of the filtered image tends to become *Gaussian* and differs from the Gamma distribution of the original image (right hand tail).

4. Increasing the *deramp* parameter will lead to take more into account pixels farther from the center and therefore increase the smoothing effects.

Polarimetry

1. HH-HV:

```
$ otbcli_BandMath \
-il intensity_hh_speckle.tif intensity_hv_speckle.tif \
-out hh-hv_speckle.tif \
-exp "im1b1-2*im2b1"
```

2. Then, image concatenation:

```
$ otbcli_ConcatenateImages \
-il intensity_hh_speckle.tif \
intensity_hv_speckle.tif hh-hv_speckle.tif \
-out intensity_compo.tif
```


1. Then convert in decibels:

```
$ otbcli_BandMath \
-in intensity_compo.tif \
-out intensity_compo_db.tif \
-exp "im1b1>0?10*log10(im1b1):0"
```

2. Comments:
   - layover is a geometric effect which makes the signal similar between HH and HV
   - vegetation area (forest)
   - HV is less sensible to roughness
   - water areas: low backscatter
3. Analysis of color composition:
   - Power lines around index (230,3700)
   - Reflector near index (3620,2925)
   - Anchor mast for boats

6 Develop with OTB

6.1 OTB Applications on Python API

6.1.1 Introduction : Water monitoring in the Laguna de la Nina(Peru) event

1. The color composition of each image of level 2 allows to see the regions as seen by the naked eye and seen as we did not have any atmosphere. The images show how this region evolves over three phases:
   - empty lagoon on December 2016
   - max extension of the flooded lagoon in April 2017
   - flooded lagoon decreasing in December 2017.

6.1.2 Sentinel 2 - Level 2A Format

1. FRE images have been corrected for the effect of slopes (which affects illumination) , and hence, the physical properties of the observed surfaces are better described.
2. B3 and B11 have different resolutions. Higher resolution (smaller pixel size) for the same area involves a higher number of pixels to be described in the file, and hence, a bigger file size.
3. Each band is independent from the others.
4. An zone covered with clouds does not present any information about the ground. It is better to ignore the clouded areas by tagging them as NODATA regions, in order to avoid false detections of water.
5. On water regions, B4(RED) has higher reflectance values than B8A(NIR). This relation between bands is used on the NDVI calculations to detect water pixels.

6.1.3 Simple OTB application in Python : exercise1.py

1. The line is `print(str(otbApplication.Registry.GetAvailableApplications()))` , that uses a method defined in the `Registry` module of the OTB Python library.
2. —
3. `Superimpose = otbApplication.Registry.CreateApplication("Superimpose")`
4. The new generated Near Infrared image has a finer resolution (pixel size = 10m) than the original image (pixel size = 20m). The new finer pixel values have been interpolated. You can use Monteverdi to zoom closer and compare the generated `B8A_10m.tif` and the original B8A image. You shall appreciate the smoothed interpolated values in the new image.

6.1.4 Chain OTB applications : exercise2.py

1. —
2. Gaps to fill:
   - Gap 2 :

     ```
     application1.SetParameterString("inr",str( d["input_path"]  \
                                     + d["B4_image"]))
     application1.SetParameterString("inm",str( d["input_path"]  \
                                     + d["B8A_image"]))
     application1.SetParameterString("out", "B8A_10.tif")
     ```

   - Gap 3 :

     ```
     application2.SetParameterStringList("il",["B8A_10.tif", \
                               str(d["input_path"] + d["B4_image"])])
     application2.SetParameterString("out", "ndvi.tif")
     application2.SetParameterString("exp", "(im1b1-im2b1)/(im1b1+im2b1)")
     ```

   - Gap 4 :

     ```
     application3.SetParameterStringList("il",["ndvi.tif"])
     application3.SetParameterString("out", "water_mask.tif")
     application3.SetParameterString("exp", "im1b1<0?1:0")
     ```

3. **ndvi.tif**: intermediary Geotif file that contains the Normalized Vegetation Index of the scene.

   water-mask.tif: binary mask (Geotif file) that contains the value 1 for those pixels with a value of NDVI under a given threshold, that is considered as Water Pixels. The rest of the pixels are considered as Land (pixel value = 0). Note: In Monteverdi, check the High/low values of the Dynamic Range to be 0 and 1. Also, deactivate the display of NODATA values.

6.1.5 Chain OTB applications in-memory: exercise3.py

1. —
2. The gap is filled as follows:

   ```
   application2.AddImageToParameterInputImageList("il", \
               application1.GetParameterOutputImage("out"))
   ```

3. The gap is filled as follows:

   ```
   application2.SetParameterString("exp", "(im1b1-im2b1)/(im1b1+im2b1)<0?1:0")
   ```

4. The lines with `ApplicationX.Execute()` will not launch immediately the ApplicationX. This line just describes that the ApplicationX will be launched in a pipeline sequence. When another `ApplicationY.ExecuteAndWriteOutput()` is further applied in the same pipeline, where the inputs of ApplicationY are dependent of the outputs of ApplicationX, then the ApplicationY will trigger the Execution of ApplicationX.

5. When `ApplicationX.Execute()` is used in the code, the results of ApplicationX are only used in RAM memory to resolve the dependence between the inputs/ouputs of different applications of the pipeline. Thus, the output file (`B8A_10m.tif`) is never written as file but it is used in the Application2 as input.

6. In order to reduce the size of the output image without degrading the image, we may use a simpler type of variable to store the values. Instead of using a Float type (which needs 2 or 4 bytes) we can use an Integer type(uint8= unsigned integer with 8 bits) which needs 1 byte per pixel. "uint8" type allows to use 256 different values, which is alright for the 2 values needed (0=land,1=water).

   We may go further using the Extended filenames mechanism on the output file definition:

   ```
   appX.SetParameterOutputImagePixelType("out", \
           "water_mask.tif?&gdal:co:NBITS=1")
   ```

   This solution will use just 1 bit per pixel. For a better understanding, see module "Internals".

6.1.6 Water detection chain with NoData management: exercise4.py

1. —

2. The gap is filled as follows:

   ```
   app3.SetParameterString("exp", "im2b1!=0?255:im1b1")
   ```

3. —

4. The straight lines are roads, which have similar spectral responses compared to water in the RED and NIR bands.

6.1.7 Comparison with a reference: exercise5.py

1. In the GSW product, the value = 90 means that water has been found in the 90% of the observations during the 32 years period. It corresponds to permanent waters.

2. —

3. The ComputeConfusionMatrix application can be used to compare raster images as follows:

   ```
   $ otbcli_ComputeConfusionMatrix -in water_mask_SENTINEL2A_20170407-154054-255_L2A
     -nodatalabel 255 \
     -ref raster \
     -ref.raster.in GSW_10.tif \
     -out conf_20170407_10.csv
   ```

   The best kappa results for each date are:

   20161218 » GSW_{75} (kappa = 0.8175)

   20170407 » GSW_{10} (kappa = 0.7867)

   20171203 » GSW_{20} (kappa = 0.5677)

4. The water surface contained in the 20161218 image (the driest one) is seen at least 75% of the times (during the 32 years of Landsat observations). The second image have a water extent that has been seen only 10% of the times and the third one only the 20% of the times. We may conclude that the second and third water extents are rare events given the low occurrence rate.