Broceliande Manual

User Guide for broceliande

triskele (at) irisa.fr

May 26, 2020

Revision index	Type of modification	Author	Date
0	creation	François Merciol	8th Aug 2017
1	illustrations	Jérôme Moré	9th Sept 2019
2	feature profile, multi-classes, pantex	François Merciol	23th March 2020

Equipe OBELIX – UMR 6074 IRISA / Université Bretagne Sud & CNRS

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

Contents

Introduction **5**

1 First examples of use (LINUX only) **5**
1.1 Preliminary conditions to run the following examples . . . 5
1.2 Examples using only the "Triskele" options (i.e. for tree representation) . . . 5
1.3 Examples using only the "Broceliande" options (i.e. for classification) . . . 6

2 Hierarchical representation of images **7**
2.1 Tree representation . . . 7
2.2 Efficient implementation . . . 7
2.2.1 Tree and attributes implementation . . . 7
2.3 Attribute profiles . . . 8
2.4 Feature profiles . . . 9

3 Software overview **9**
3.1 Functional architecture . . . 9
3.2 Optimization . . . 9
3.3 Computational steps . . . 9
3.3.1 Reading the configuration file . . . 9
3.3.2 Parsing options and reading metadata . . . 12
3.3.3 Cropping the image . . . 12
3.3.4 Reading the image . . . 12
3.3.5 Finding the inside and the outside of the image . . . 12
3.3.6 Computing NDVI . . . 12
3.3.7 Computing Sobel band . . . 12
3.3.8 Selecting the bands . . . 12
3.3.9 (Pre)computing texture profiles . . . 12
3.3.10 (Pre)computing Attribute Profiles (AP) . . . 13
3.4 Configuration files (a.k.a. "channel files") . . . 13

4 Installation **13**
4.1 Operating system . . . 13
4.2 Import Broceliande source code . . . 13
4.3 Installation procedure . . . 13
4.4 Visualization of output images . . . 14

5 Formatting rules for options **14**
5.1 Short and long version of options . . . 14
5.2 Options using filenames . . . 14
5.3 Options using strings . . . 14
5.4 Options using integers . . . 14
5.5 Options for both software, or for broceliande only . . . 14

6 Input and output images (either as parameters or as options) **15**
6.1 -i fileName, --input fileName . . . 15
6.2 -o fileName, --output fileName (broceliande only) . . . 15

7 General options (information display) **15**
7.1 --help . . . 15
7.2 --version (broceliande only) . . . 15
7.3 --debug . . . 15
7.4 --timeFlag (broceliande only) . . . 15
7.5 --countFlag (broceliande only) . . . 16

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

7.6 --countingSortCeil arg . . . 16
7.7 --showConfig arg (broceliande only) . . . 16
7.8 --includeNoDataFlag (broceliande only) . . . 16
7.9 --noDataValue arg (broceliande only) . . . 16
7.10 --coreCount arg (broceliande only) . . . 16
7.11 --threadPerImage arg (broceliande only) . . . 16
7.12 --tilePerThread arg (broceliande only) . . . 16
7.13 --seqTileFlag (broceliande only) . . . 16
7.14 -- autoThreadFlag (broceliande only) . . . 16
7.15 -- memImpactFlag (broceliande only) . . . 16

8 Options for spatial selection and connectivity (broceliande only) 17
8.1 -x dimImg, --left dimImg . . . 17
8.2 -y dimImg, --top dimImg . . . 17
8.3 -w dimImg, --width dimImg . . . 17
8.4 -h dimImg, --height dimImg . . . 17
8.5 --connectivity arg (broceliande only) . . . 17

9 Option for bands selection (spectral selection) 18
9.1 -b selection, --withBand selection . . . 18
9.2 -d arg, --spectralDepth arg . . . 19
9.3 --mixedBandFlag . . . 19
9.4 --noMixedBand . . . 19

10 Options for bands production 19
10.1 -c arg, --copyBands arg . . . 19
10.2 -n arg, --ndviBands arg . . . 19
10.2.1 Computation steps . . . 20
10.2.2 Precision . . . 20
10.2.3 Complexity . . . 20
10.3 -p arg, --PantexBands arg . . . 20
10.3.1 Computation steps . . . 21
10.3.2 Precision . . . 22
10.3.3 Complexity . . . 22
10.4 -s arg, --sobelBands arg . . . 22
10.4.1 Computation steps . . . 23
10.4.2 Precision . . . 23
10.4.3 Complexity . . . 23
10.5 --noCopy . . . 24
10.6 --noNdvi . . . 24
10.7 --noPantex . . . 24
10.8 --noSobel . . . 24

11 Options for Attribute Profiles (AP) and Features Profiles (FP) 24
11.1 -t arg, --treeType arg . . . 24
11.2 -a arg, --attributeType arg . . . 24
11.3 --thresholds arg . . . 24
11.4 -f arg, --featureType arg . . . 25

12 Options for Differential Attribute Profiles (DAP) 29
12.1 --dapPos arg . . . 30
12.2 --dapWeight . . . 30

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

13 Options for textures **32**
13.1 Note about Integral Images . . . 32
13.1.1 Definition . . . 32
13.1.2 Creation . . . 33
13.1.3 Usage . . . 33
13.2 -H arg, --haarSizes arg . . . 34
13.2.1 Computation . . . 34
13.2.2 Optimization (vertical and horizontal features) . . . 34
13.2.3 Optimization (diagonal features) . . . 35
13.2.4 Complexity . . . 35
13.3 -T arg, --statSizes arg . . . 36
13.3.1 Computation . . . 36
13.3.2 Complexity . . . 36
13.4 --noHaar . . . 37
13.5 --noStat . . . 37

14 Options for configuration files ("channel" files) **37**
14.1 --loadChannel arg . . . 37
14.2 --saveChannel arg . . . 37
14.3 --cfgName arg . . . 37
14.4 --showChannel . . . 37

15 Options for ground truth (broceliande only) **37**
15.1 -g arg, --groundTruth arg . . . 37
15.2 --showGroundTruth arg . . . 37
15.3 --tagValues arg (=1) . . . 38
15.4 --bgValue arg (=nan) . . . 38
15.5 --tagsRate arg (=100%) . . . 38
15.6 --bgTagRate arg (=80%) . . . 38
15.7 --bgRate arg (=125%) . . . 38

16 Options for training (broceliande only) **38**
16.1 --cleanTagsFlag . . . 38
16.2 --cleanSamplesFlag . . . 38
16.3 --nTrees arg (=64) . . . 38
16.4 --maxPerRFBatch arg (=65536) . . . 38
16.5 --loadTrain arg . . . 38
16.6 --saveTrain arg . . . 38

17 Options for batch processing (broceliande only) **38**
17.1 --batchDir arg . . . 39
17.2 --batchPrefixResult arg . . . 39
17.3 --batchResultDir arg . . . 39
17.4 --batchOverwriteFlag . . . 39

Introduction

This work is the result of academic researches[4, 3, 2, 1, 5].

The *broceliande* software (as well as *broceliandeConfig* software) is an executable file that allows to run the C++ broceliande library to process an image. Broceliande is a project from Obelix team in IRISA aimed to perform some classification for remote sensing images. It is mainly based on 2 other pieces of software, *Triskele* from Obelix team used to produce hierarchical representations of images, and *Shark* librairy used for Machine Learning.

The *broceliande* software must be called from command line, using at least input and output image files, as well as some options. Besides, those options can be stored to and loaded from a configuration file. In addition we can also provide a ground truth image as a input for classification.

The *broceliandeConfig* software is a simplified version of broceliande that does not produce any output image, but still can produce a configuration file to check the set of options.

1 First examples of use (LINUX only)

Below are some examples of calls of broceliandeConfig or broceliande, using command lines under Linux.

1.1 Preliminary conditions to run the following examples

In the following examples, we assume that you have fully installed and build the Broceliande software, and that you have the following folder structure:

`~/git/broceliande/`	folder for source code of Broceliande
`~/git/build/`	folder for executable files
`~/data/`	folder for input images (e.g.: arles.tif)

Also we assume that a image called “arles.tif” already exists in the folder for input images.

1.2 Examples using only the “Triskele” options (i.e. for tree representation)

```
# add executable folder to linux PATH :
export PATH = $PATH :~/ git / build /
# go to the folder containing images (‘‘arles.tif’’, ...):
cd ~/data/

# read input image, keeps only band 0, save the output image and the
    ↪ config file
broceliande arles.tif o.tif --saveChannel config --copyBands 0

# read input image, create a new NDVI band from band 0 and 1, then save
    ↪ the output image and the config file
# option copyBand keeps the newly created band (=band 3) in config. By
    ↪ default it is not in the config ! If this option is omitted, it
    ↪ will produce this message: ‘‘error: No output band selected’’
broceliande arles.tif o.tif --saveChannel config --ndviBands 0,1 --
    ↪ copyBand 3

# read input image, from band 0 create a AP image using Min-Tree and
    ↪ Area attribute with thresholds 1000 and 2500, then save the output
    ↪  image and the config file
```


Université Bretagne Sud - IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex - +33/0 2 97 01 72 35
www.univ-ubs.fr

```
# (the 2 following lines are equivalent, with short then long parameters
   ↪ )
broceliande arles.tif o.tif --saveChannel config -b 0 -t Min -a Area --
   ↪ thresholds 1000,2500
broceliande arles.tif o.tif --saveChannel config --withBands 0 --
   ↪ treeType Min --attributeType Area --thresholds 1000,2500

# read input image, perform the same AP as previously, add a DAP band,
   ↪ then save the output image and the config file
broceliande arles.tif o.tif --saveChannel config -b 0 -t Min -a Area --
   ↪ thresholds 1000,2500 --dapPos apOrigPosBegin

# read input image, produce Sobel band from band 1, then save the output
   ↪  image and the config file
# option copyBand keeps the newly created band (=band 3) in config. By
   ↪ default it is not in the config ! If this option is omitted, it
   ↪ will produce this message: ‘‘error: No output band selected’’
broceliande arles.tif o.tif --saveChannel config --sobel 1 --copyBand 3

# read input image, produce Haar band from band 1, then save the output
   ↪ image and the config file
# By default the band is in the config
broceliande arles.tif o.tif --saveChannel config -b 0 --haarSizes 5x3

# loads the content of previously created config file (config.ch) then
   ↪ show its content
# Warning: as there is no output image, we must use broceliandeConfig,
   ↪ not broceliande
broceliandeConfig --loadChannel config --showChannel
```

1.3 Examples using only the “Broceliande” options (i.e. for classification)

```
# Classification using a min-tree and a ground truth image (‘‘
   ↪ arlesGT.tif’’).
broceliande arles.tif o.tif --saveChannel config -b 0 -t Min -a Area --
   ↪ thresholds 1000,2500 -g arlesGT.tif --fgValue 3

# Classification using a max-tree and a ground truth image.
broceliande arles.tif o.tif --saveChannel config -b 0 -t Max -a Area --
   ↪ thresholds 1000,2500 -g arlesGT.tif --fgValue 3

# Classification using a median-tree and a ground truth image.
broceliande arles.tif o.tif --saveChannel config -b 0 -t Med -a Area --
   ↪ thresholds 1000,2500 -g arlesGT.tif --fgValue 3

# Classification using a max-tree, a NDVI band and a ground truth image.
broceliande arles.tif o.tif --saveChannel config -n 0,1 -c 3 -b 3, -t
   ↪ Max -a Area --thresholds 1000,2500,5000,7500 -g arlesGT.tif, --
   ↪ fgValue 3
```

2 Hierarchical representation of images

2.1 Tree representation

A hierarchical representation of the content of the images can be made by using morphological trees, here most often inclusion trees.

Those trees are constructed in such a way that the leaves are small regions of pixels with constant value, and the nodes are connected regions created by adding surrounding pixels with very close values. In that way all possible pixel values are represented up to the value of the root node, which can be either the minimum, the maximum or the median value, according to the type of tree chosen.

Let's take an example using pixel values between 0 (black) and 255 (white). We will build the tree node by node, from the leaves to the root by browsing all this interval of pixels values. And the order that we choose to browse the interval will define the type of tree we build:

- If we order the pixel values from 0 (dark) to 255 (white), then the leaves will be local minima, and the tree will be called a "**min-tree**".
- If we order the pixel values from 255 (white) down to 0 (dark), then the leaves will be local maxima, and the tree will be called a "**max-tree**".
- If we order the pixel values using the distance to the median value, beginning from the farthest values to the median (either greater or lower) then getting closer and closer to the median, then the leaves will be local extrema (minima and/or maxima), and the tree will be called a "**tree of shape**". We can mention that the name "tree of shape" comes from the fact that it creates the different nodes by mixing in parallel "bright" and "dark" nodes from the median node, therefore focuses more of the shape of the nodes than on their gray-scale.
- Actually the tree described before is not the official tree-of-shape, but rather an optimized version which is called "**median-tree**", using the median value to compute it faster.

Also, note that we can also visualize this tree representation with a "lake model". First think of the initial image with a 3D representation, where the intensity would be replaced by a third dimension (altitude), so that the image is like a landscape made with mountains (bright pixels) and valleys (dark pixels). Then each node would be a "lake" partially filling the valleys and partially reduced by the mountains, and having a constant altitude (i.e. on the 2D image having a constant intensity). Then the min-tree is a set of all possible lakes from the lowest case to the highest. The max-tree describes the dual situation, that is a set of all possible emerged surface regions from the highest case to the lowest. Finally the tree of shape describes all the possible lakes and/or emerged surface regions from the most extreme (lowest and/or highest) ones to the average ones (see figure 3, for now look only at column "Orig. images").

2.2 Efficient implementation

To maintain an efficient implementation, all algorithms integrated into the software have linear or quasi-linear time complexity [4]. This is a brief presentation of the hidden data structure with compact and linear information that help us to design attribute algorithms.

2.2.1 Tree and attributes implementation

First of all, we use the fundamental properties of hierarchical data representation. Consider a gray scale image (left side of Figure 1). This image is a matrix of pixels. We can build different types of trees (min-tree, max-tree, ...) to manage the image. We choose to build a max-tree (right side of figure 1).

This tree is a hierarchical representation. All pixels have parent. These parents are called nodes. All nodes (except the root) also have a parent node. This defines an inclusion tree. Because we choose a max-tree for this example, the nodes are organized from light gray (low level of the tree) to dark gray (high level). All nodes can have properties. We represent them with colored bullets in Figure 1.

The tree is defined only with an array of parents (see Figure 2). All pixels and nodes are associated with a unique index. The indexes under "pixel cardinality" refer to the pixels from the first row to the last row of the image. Indexes with higher "cardinality of pixels", refer to tree nodes. Thanks to this

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

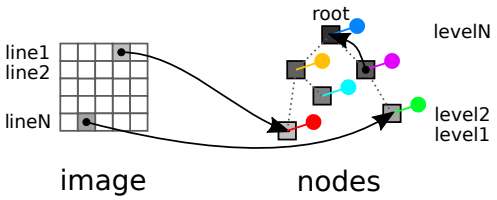


Figure 1: Hierarchical representation of image using a max-tree

building process, all nodes are sorted according to their level. So the light-gray nodes are on the right and the darker-gray ones are on the left.

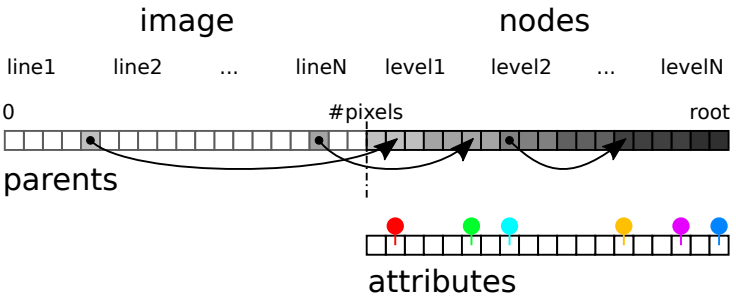


Figure 2: Tree implemented with array

Attributes are only defined for nodes. The attributes can be simple scalar (surface, perimeter, gray value) or multivalued (centroid, bounding box), and they are also stored in another array. The position of an attribute in an array gives the position of the associated node. Here are our efficient calculations of new attributes.

2.3 Attribute profiles

The main goal of the tree previously build is to create what we call the Attribute Profiles (AP) of the image. The attribute profiles are obtained by pruning the tree (i.e. excluding some leaves). More precisely the different steps are:

- create a tree from the image, where each node of the tree contains a gray scale value
- for each node, compute also some extra attributes, such as area or standard deviation
- prune the tree, according to either gray scale levels or any type of attributes, and do this several times using a series of thresholds [1]. We can notice that the tree pruning is a projection
- for each pruned tree, rebuild the image using only the remaining nodes (filtered image)
- put all the filtered images together in a stack, which will be called the "attribute profiles"

According to the type of tree we consider, here is the result of this process on the tree and on the image (see figure 3):

- in the case of min-tree, the tree pruning removes the nodes from its darkest to its brightest, so that the image filtering replaces the darkest areas by merging them into brighter and brighter surrounding zones (corresponding to their parent nodes).
- in the case of max-tree, the tree pruning removes the nodes from its brightest to its darkest, so that the image filtering replaces the brightest areas by merging them into darker and darker surrounding zones.
- in the case of tree of shape, the tree pruning removes the nodes from its most extreme (brightest and darkest) to its most average (with median value), so that the image filtering replaces the most

[1] For optimization reason, the algorithm does not actually prune the tree, but instead walks through the entire tree from the leaves to the root until it reaches a node above a given threshold, which will then be considered as a new "leaf" of the pruned tree

extreme (brightest and/or darkest) areas by merging them into less and less extreme surrounding zones.

2.4 Feature profiles

The Feature Profiles (FP) is a concept very similar to the Attribute Profiles, except that instead of producing a stack of images containing gray-values (just as the original image), we will produce a similar stack of images but containing a given feature, such as area, moment of inertia, or standard deviation.

Note that they are two independent characteristics used in this process (which could either be the same or be different):

1. First the characteristics called "attribute" used to prune the tree, according to a series of thresholds. For example we can compute for each node the area of the node, then prune the tree according to different thresholds of the area. For a series of N thresholds, this will produce N pruned trees.
2. Then the characteristics called "feature" used to reconstruct the corresponding stack of images (also called "filtered images").
   - if we choose the gray scale-level, the stack will be called "Attribute Profile"
   - if we choose any other types (area, standard deviation, ...), the stack will be called "Feature Profile"

Also note in the future sections, we will also talk about "Haar-like features" and "statistical features". Those features are completely different characteristics, absolutely independent of the attributes or features we are discussing now.

3 Software overview

3.1 Functional architecture

Figure 4 show a global view of the process chain of the software. The next figure 5 show a more detailed view of the processes involved.

3.2 Optimization

The Obelix team has developed optimized algorithms which are fast and not much memory consuming. Every tree is build with using linear complexity (with the number of pixels), regardless of the type of tree or other parameters. Also the building process can be parallelized, so that the overall computational time will be divided by the number of available processors.

3.3 Computational steps

Below we present a global view of the different computational steps that this program will run into, in order to have a general idea of the different processes applied. Almost all steps are optional, except reading the metadata and the content of the input image.

3.3.1 Reading the configuration file

The very first step is to read the configuration file (if required). The configuration file can be seen as a set of default options, that might be completed and/or overridden by all the following manual options.

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

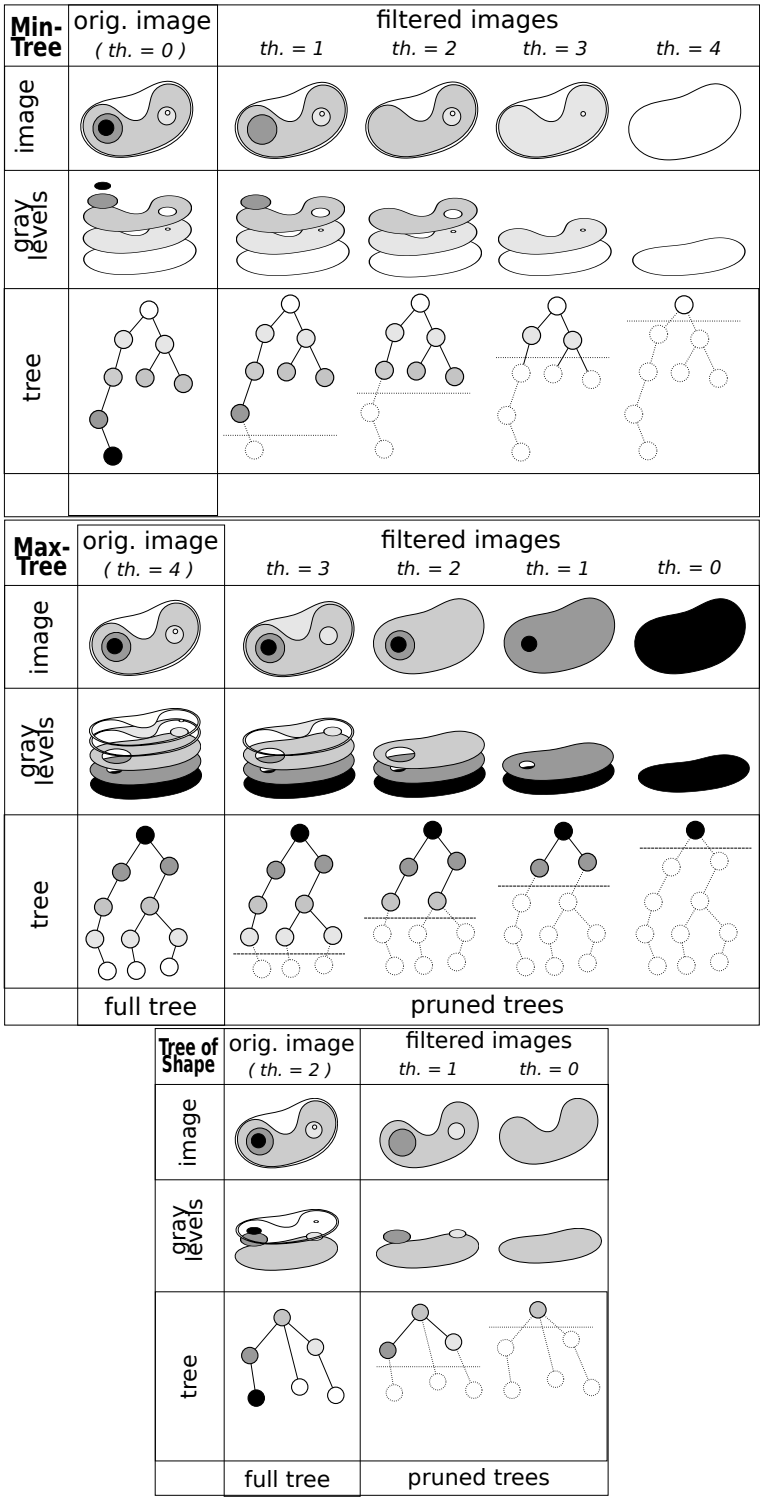


Figure 3: Principle of a Min-Tree, a Max-Tree and a Tree-of-Shape

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

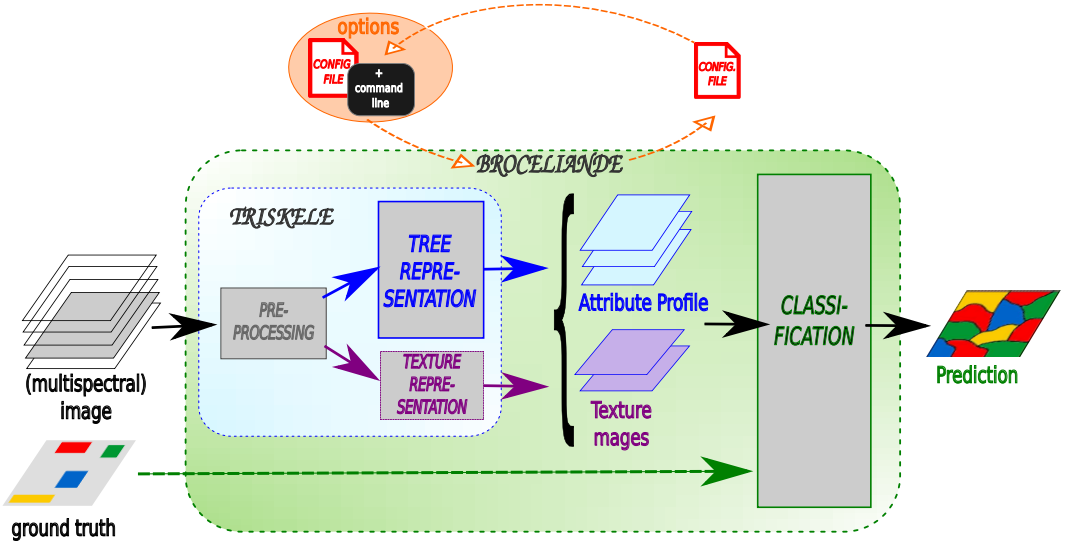


Figure 4: Functional architecture (global view)

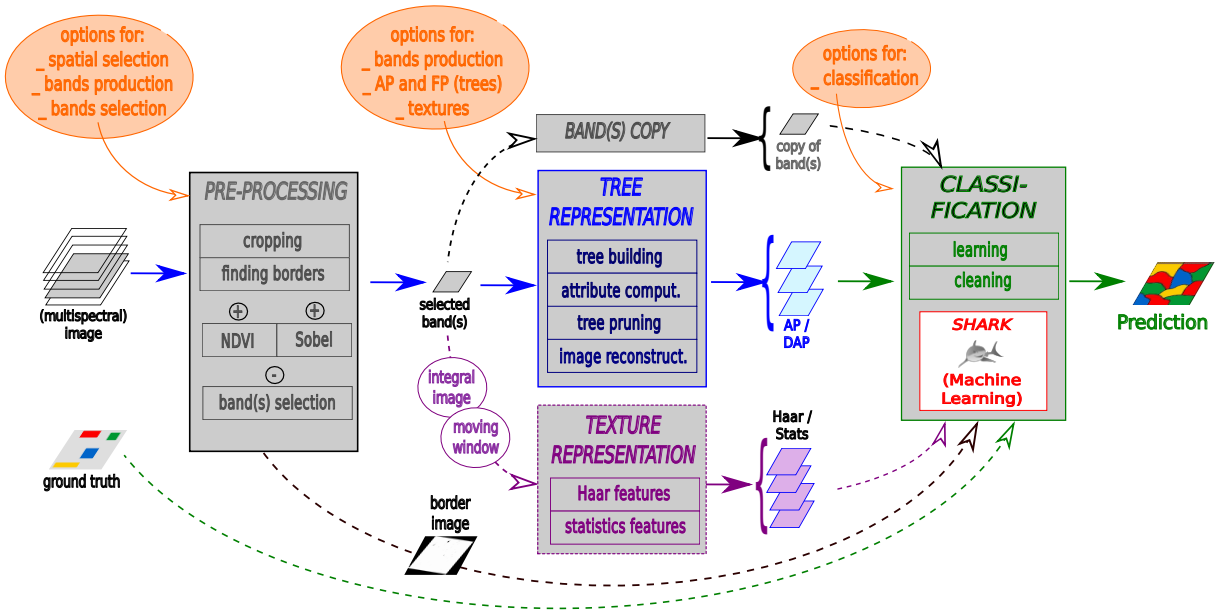


Figure 5: Functional architecture (detailed view)

Equipe OBELIX – UMR 6074 IRISA / Université Bretagne Sud & CNRS

3.3.2 Parsing options and reading metadata

So the very next step is to analyze all the different options given by the users, including input and/or output image filenames if provided. Please refer to section 5 and following for more details about those options. A this stage, the program also reads the metadata in the input image, allowing the program to set up for the proper pixel type (integer of floating point, so far complex type like in SAR images is not allowed).

3.3.3 Cropping the image

The input image is cropped according to a rectangular selection zone. By default this rectangular zone is the whole image. A partial selection can be extremely useful when the input image is to large to be analyzed in a single go.

3.3.4 Reading the image

All the bands of the image are read (which is useful at least to identify the outside of the image, where the pixels have a "no-data" value). We use the GDAL library for this stage, which means that our software can read any image format supported by this library.

3.3.5 Finding the inside and the outside of the image

Before reading the image, by default all the pixels are considered as being outside the image, i.e. having a "no-data" values. Then when reading the image, all the pixels that has a non-zero value in any of the bands will be promoted as being inside the image to be analyzed. Otherwise they will still be considered as being outside the image.

3.3.6 Computing NDVI

Whenever reading a image band, we check whether it is part of the NDVI computation. In that case, the program could keep in memory either the content of this band, either the result of NDVI. Note that so far we do not consider computing more than one NDVI band. Please refer to the section 10 for more details.

3.3.7 Computing Sobel band

Whenever reading a image band, we check whether it is part of the Sobel band computation. In that case the program can run the computation (linear time complexity) on this band. Please refer to section 10 for more details.

3.3.8 Selecting the bands

For each band selected by the user (and/or the configuration file) the computation for attribute profiles and texture profiles will be done. For texture profiles, only "integral images" are produced and kept (see section 13.1). If required, Haar features or statistics features will be produced on the fly when necessary. For attribute profiles, they are created and kept. If required, the DAP (Difference of Attribute Profile) will be produced on the fly when necessary.

3.3.9 (Pre)computing texture profiles

All texture profiles are produced using linear time complexity algorithms, as their computation is based on "integral images". Actually, an integral image is defined as being the sum of all the pixel values in the region above and to the left of the current pixel. The computation of the integral image requires only a linear time complexity. Then once computed, it can provide very quickly the sum of pixel values on any rectangular zone of the image in an extremely short and constant time, using only differences in the integral image. For more information on integral images, please refer to section 13.1.

3.3.10 (Pre)computing Attribute Profiles (AP)

The computation of Attribute profiles (AP) is based on the computation of morphological trees: min-tree, max-tree or median-trees. So for each band, for example a min-tree is created then the attribute profiles are computed. For each AP, trees are pruned according to the list of thresholds given by the user (and/or the configuration file). Here there is an implementation issue for this software about choosing the best possible memory usage when computing trees and attribute profiles. In other words we want to determine what is best choice between keeping in memory the trees or keeping in memory the attribute profiles. This choice relies on the number and the type of pixels, but also on the number of user thresholds. To give a typical example, we can consider that below 13 thresholds it is better for memory usage to compute the AP then delete the tree structure, whereas above 13 thresholds it is better to keep only the tree structure. This example is based on an image of 20 000 x 90 000 pixels using 16-bit depth. However, for any different situation, or for example if we consider only a sub-part of the image, the previous criterion should be reconsidered.

3.4 Configuration files (a.k.a. "channel files")

Configuration files is a useful way to store and retrieve a set of options used to compute the output images. The configuration files are stored in special files called "channel files" (extension *.ch). For readability they are written in XML format. Here is an example of such a channel file:

```
<?xml version="1.0" ?>
<Triskele type="UInt16" bandCount="3" created="2019-03-07 17:33:08"
    modified="2019-03-07 17:33:08">
    <AP id="3" from="0" treeType="Min" featureType="AP" attributeType="Area
    ">
        <Cut threshold="1000" />
        <Cut threshold="2500" />
    </AP>
    <DAP id="4" doWrite="true" from="3" apOrigPos="apOrigPosBegin" />
</Triskele>
```

4 Installation

The installation of broceliandeConfig and/or broceliande needs the following steps

4.1 Operating system

We suggest to use the latest stable version of Linux Debian (e.g. version 10.0 ("Buster")). The installation can be made from the official Debian site or any mirror, choosing if possible a mirror not too far from your home place. For example go to:
`https://www.debian.org/CD/http-ftp/#stable`

4.2 Import Broceliande source code

You can import the source code from the official repository using Git:
`https://gitlab.inria.fr/obelix/broceliande`

4.3 Installation procedure

Please follow the complete installation/compilation procedure described in the repository page given above. We strongly advice to include all the suggested packages, and it is actually better to install them before any other command.

4.4 Visualization of output images

Though not mandatory, we also recommend to use software QGIS to visualize the different images (input and/or output). To install it please follow the instructions from the original QGIS site:
`https://qgis.org/en/site/forusers/alldownloads.html#debian-ubuntu`

5 Formatting rules for options

5.1 Short and long version of options

Many of the options used for this software follow the Linux option standard, i.e. can exist in 2 versions:
- a short version option, using the character "-" and a letter (e.g. "-h")
- a long version option, using the 2 characters "--" and a word (e.g. "--help")

Note that some options only exist in long version (usually when a single letter would be ambiguous with another option starting with the same letter).

5.2 Options using filenames

An example of such options is the input or the output image filename (see section 6).

Here a filename means either a simple name (e.g. "image.tif") or a path using folders and file name (e.g. "~/Images/image.tif"). Folders can be absolute or relative (using ".", ".." under Linux) and must use the proper separator character, which is operating-system dependent (e.g. "\" for Windows, "/" for Linux).

5.3 Options using strings

Strings must follow the rules used by the operating system (such as the separator character). It is highly recommended to avoid special characters such as punctuation marks or accents.

5.4 Options using integers

An example of such options is the input band parameter (see section 9). Integers should be unsigned and contain no other characters than digits (such as a point or comma sign). Note that image sizes are just integers limited to a certain range of values. For options using sets of integers, several notations are allowed, using the following characters: "," (separator), "-" (interval), "*" (min or max of allowed values). For example the set 2,3,4,5 can be written as:
- 2,3,4,5
- 2-3,4-5
- 2-5
- 2-* (assuming the maximum possible value is 5)

5.5 Options for both software, or for broceliande only

Unless otherwize stated, the options shown are available both for broceliandeConfig and for broceliande. Whenever dealing with options for broceliande only, a special mention will be added. The following table gives a global idea of which software has which options:

Option group	broceliandeConfig	broceliande
Main options	(some of them)	YES
Channel config options	YES	YES
Crop options	NO	YES
Groundtruth options	NO	YES
Training options	NO	YES
Batch options	NO	YES

6 Input and output images (either as parameters or as options)

The command lines used to call broceliandeConfig or broceliande uses:

- several options using their proper options names
- 2 parameters which are input image and output image (for broceliande) or 1 parameter which is input image (for broceliandeConfig).

Unlike the options that can be placed anywhere, the parameters have to be placed in the right order (input image then output image). However a way to get around this rule is to declare input and output images as options, using the following specific option keywords:

6.1 -i fileName, --input fileName

where fileName is the path for the input image

6.2 -o fileName, --output fileName (broceliande only)

where fileName is the path for the output image

7 General options (information display)

This section explains the general options used only to display some information.

7.1 --help

Display a quick help. Usually shows the main usage and a list of options.

7.2 --version (broceliande only)

Display the software version. This option can be useful for checking if your current version supports an option or a functionality that you would like to use. The version used for this User Guide is:

```
broceliande version 2.4 2020-03-23 multiclasses (Linux)
```

7.3 --debug

Display a detailed information for debugging purpose. Basically every start and end of every function is displayed, as well as some extra information, in the following format:

```
[MM/YY] HH:mm  name> expr
```

where MM/YY is the date, HH:mm is the time, name is the name of a class or a function, then a symbol ('>' for start, '|' for middle, '<' for end), and possibly some extra information (content of some variables).

This option is available only if software is compiled with c++ log option "$-DENABLE\_LOG$". That easily done with the cmake command:

```
cmake -DCMAKE_BUILD_TYPE=Debug ../broceliande
```

7.4 --timeFlag (broceliande only)

Display information about execution time of the main steps of the software. This option is also intended for debugging purpose.

7.5 --countFlag (broceliande only)

Display information about the number of “leaves” (pixels in the image) as welle as the number of “components” (nodes) used to build the tree. This option is also intended for debugging purpose.

7.6 --countingSortCeil arg

(=2)
Set the ceiling value (i.e. maximum value) for the counting sort algorithm.

7.7 --showConfig arg (broceliande only)

Write the configuration information into a file.

7.8 --includeNoDataFlag (broceliande only)

Build a tree using all the available pixels in the image (i.e. including the "no-data" flagged pixels). In that case the metadata in image header file are ignored.

7.9 --noDataValue arg (broceliande only)

(=nan)
Sets the special value that will be considered as no-data in the image. The default value is “nan”. The “nan” option use the metadata value indicated in image header file (but with the includeNoDataFlag).

7.10 --coreCount arg (broceliande only)

General information used when the software can split computation activity. That can save computation time but also increase the memory impact. The default value is set with the number of cores.

7.11 --threadPerImage arg (broceliande only)

Number of cores used in parallel to build a tree for an image. This number must me less than the host capability. The default value is the quarter of number of cores (at least one).

7.12 --tilePerThread arg (broceliande only)

if threadPerImage or tilePerThread is greater than one, the image is divided by tiles. The number of tiles is the product of the two terms threadPerImage tilePerThread.

7.13 --seqTileFlag (broceliande only)

Force sequential tile computation. The image is split into several tiles that are computed one after the other, in order to save memory. The number of tiles depends on threadPerImage and tilePerThread (or autoThreadFlag).

7.14 -- autoThreadFlag (broceliande only)

Set the threadPerImage and tilePerThread parameters to maximized the memory used and reduce the comptation time.

7.15 -- memImpactFlag (broceliande only)

Give estimation of memory impact according sequential or parallel computation.

8 Options for spatial selection and connectivity (broceliande only)

This section deals with spatial selection (i.e. cropping image) and pixel connectivity.

Working on a cropped image can be useful to speed up a given process by testing it on a smaller image. Cropping an image means reducing it to a smaller rectangle. Now in order to define the rectangle, we usually provide 4 parameters (left border, top border, width, height). Here it is possible to provide only some of the 4 previous parameters, and the missing parameters will be deduced (see details below). The approach is that a missing position indicates centered cropping. And that a missing size indicates the maximum occupancy. Note that if none of the cropping parameters are provided, then the entire image will be taken into account.

As parameters, one can provide a number of pixels or a ration of the maximum size.

8.1 -x dimImg, --left dimImg

(=-1)

Defines the left border of the cropped image, in pixels. This values must be in the interval $[0, imageWidth-1]$ or $[0\%, 100\%]$. If not provided, the value will be chosen so that the cropped image will be horizontally centered.

8.2 -y dimImg, --top dimImg

(=-1)

Defines the top border of the cropped image, in pixels. This values must be in the interval $[0, imageHeight-1]$ or $[0\%, 100\%]$. If not provided, the value will be chosen so that the cropped image will be vertically centered.

8.3 -w dimImg, --width dimImg

(=-1)

Defines the width of the cropped image, in pixels. This values must be in the interval $[0; imageWidth-1]$ or $[0\%, 100\%]$. If not provided, the value will be chosen so that the cropped image fill the right side of the image (all the remaining width after the new left border is used).

8.4 -h dimImg, --height dimImg

(=-1)

Defines the height of the cropped image, in pixels. This values must be in the interval $[0; imageHeight-1]$ or $[0\%, 100\%]$. If not provided, the value will be chosen so that the cropped image fill the bottom side of the image (all the remaining height below the new top border is used).

Examples

parameter	-1	0	100	10%	100%
-x	centered	left border	100 pixels from left	1/10 from left	nonsense
-y	centered	top border	100 pixels from top	1/10 from top	nonsense
-w	like 100%	nonsens	100 pixels	1/10 of width	full width but left border
-h	like 100%	nonsens	100 pixels	1/10 of height	full width but top border

```
broceliande i.tif crop.tif -c 0-* -x -1 -y 1000 -w "50%"
```

8.5 --connectivity arg (broceliande only)

Force to use a given pixel connectivity definition. Pixel connectivity defines which surrounding pixels are considered as neighbors. Here are the different possible definition for a pixel neighborhood (see figure 6):

- C4: 4 surrounding pixels : one above, one below, one at the left and one at the right.
- C6P: 6 surrounding pixels: the 4 C4 pixels plus one at the top-left corner and one the bottom-right corner.
- C6N: 6 surrounding pixels: the 4 C4 pixels plus one at the top-right corner and one the bottom-left corner.
- C8: 8 surrounding pixels.
- CT (for time series):
  - no neighbors in the current image (time t)
  - in the previous image (time t-1) and the the next image (time t+1) : use same pixel
- CTP (for time series):
  - no neighbors in the current image (time t)
  - in the previous image (time t-1) and the the next image (time t+1) : use C4 neighbors
- CTN (for time series):
  - no neighbors in the current image (time t)
  - in the previous image (time t-1) and the the next image (time t+1) : use "C4X" neighbors, that is consider the 4 possible corners (top-left, top-right, bottom-left and bottom-right corners)

The connectivity can be combine with the operator '|'. Note this special character can be interpreted by the shell.

```
broceliande i.tif o.tif -c 0 --showConfig 'tty' --connectivity "C4|CT"

broceliande i.tif o.tif -c 0 --showConfig 'tty' --connectivity C4\|CT
```

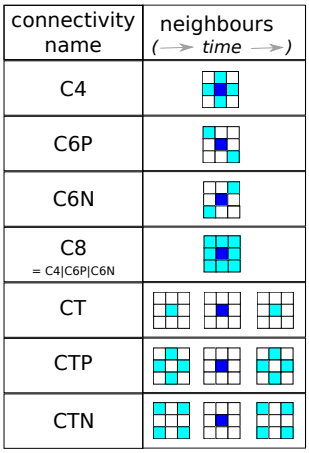


Figure 6: Different pixel connectivities

9 Option for bands selection (spectral selection)

This section deals with selecting the different bands in the input image, and how to consider the role of the different bands. Note about band numbering: in this software, the bands are numbered starting from 0 (which means that the first band is band 0). However in QGIS software for example, the bands are numbered starting from 1.

9.1 -b selection, --withBand selection

When dealing with multi-band images (multispectral or hyperspectral images), it is possible to select just some of the available bands. For example is the input image is a RGB image, selecting bands "0,2"

means working with only band “Red” and “Blue”.

9.2 -d arg, --spectralDepth arg

(=0)
The option is only used to build tree from cubic images (3D or volume). It define the number of spectral band in an image. To explain the option we need to precise terms.

- layers: all different frames in an image. A gray-scale image has 1 layer. A color image has 3 layers.
- spectral depth: number of spectral band in an image. A gray-scale image has 1 spectra. A color image has 3 spectra.
- layers per spectral band: define the layers used for 3D approach = $layers/spectralDepth$. For 2D images, the depth is 1.

By default the value is 0. It replaces the maximum value (number of bands of the image).
Consider a 6 bands images (a,b,c,d,e,f) not mixed.

spectralDepth	number of sub-images	depth of sub-images	bands of the first sub-image
1	1	6	a,b,c,d,e,f
2	2	3	a,b,c
3	3	2	a,b
6 (or 0)	6	1	a

9.3 --mixedBandFlag

Flag indicating that the bands are mixed in frames (time series).
The same a 6 bands images (a,b,c,d,e,f) with mixed option.

spectralDepth	number of sub-images	depth of sub-images	bands of the first sub-image
1	1	6	a,b,c,d,e,f
2	2	3	a,c,e
3	3	2	a,d
6 (or 0)	6	1	a

9.4 --noMixedBand

Flag indicating that the bands are not mixed in frame.

10 Options for bands production

This section deals with choosing the number and nature of the bands to produce in the output image.

10.1 -c arg, --copyBands arg

Copy the selected bands. Please note that those bands can be either:

- bands from input image (numbers starting from 0)
- NDVI or Sobel bands just produced. In that case you have to known the corresponding number.

Please see some examples in section 1.

10.2 -n arg, --ndviBands arg

Produces NDVI bands from 2 input bands. We remind that the bands are numbered starting from 0. We remind that this option needs to be used with option --copyBands in order to keep this band.

```
channelGenerator arles.tif ndvi.tif -n 0,1 -c 3
```


Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

source

NDVI (0,1)

0.014 s

10.2.1 Computation steps

The computation is made in 2 stages:

- first band is read and stored
- second band is read and the computation is such as:

$NDVI = [(A - B)/(A + B) + 1] * MaxVal/2$ where $A + B \neq 0$

$NDVI = 0$ where $A + B = 0$ (i.e. $A = 0$ and $B = 0$ simultaneously)

Note that as the ratio $(A - B)/(A + B)$ would be in interval $[-1; 1]$, we added an extra normalization to make it fit in in the interval $[0; MaxVal]$ where $MaxVal$ is the maximum possible value of the initial images. Also note that if the 2 input bands are inverted, then the resulting band will have opposite values.

10.2.2 Precision

The resulting band is set at the same precision than the input bands (ex. 8 bits or 16 bits).

10.2.3 Complexity

The complexity is linear with the number of pixels (each pixel from the input bands is read once).

10.3 -p arg, --PantexBands arg

Produces Pantex bands of the input bands. TThis command produces Pantex band of the input image according to the method introduced in [6]. This feature relies on gray-level co-occurrence matrix (GLCM).

First, all the pixels of the image are sorted. 15% of the pixels with lowest values are removed and substituted with the smallest remaining pixel value. Also 15% of the pixels with highest values are removed and substituted with the largest remaining pixel value. Then, dynamic range of the pixels is scaled to the interval of $[0, 127]$.

A $w * w$ sliding window moves on the image. Inside each sliding window we consider 10 different offsets corresponding to 10 different combinations of distances and angles (see 7). A co-occurrence matrix is computed separately for each offset.

Co-occurrence of an image is a matrix that has a row and a column for each gray level of the image. (See 8). The position (i,j) in co-occurrence matrix shows the number of times that gray level j occurs at a specific offset of gray level i in original image.

For Pantex 10 co-occurrence matrices are calculated based on 10 offsets for each sliding window. One of these offsets is shown in 9 corresponding to red square in 7. Each co-occurrence matrix is normalized by dividing by the sum of its elements. So the sum of all elements in final co-occurrence matrix is equal to 1.

Contrast of each gray-level co-occurrence matrix (GLCM) is defined as

$$contrast = \sum_{i=0}^{127} \sum_{j=0}^{127} (i - j)^2 . GLCM_{i,j}.$$

Pantex is the minimum of 10 contrasts that are calculated for 10 co-occurrence matrices.

Université Bretagne Sud - IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex - +33/0 2 97 01 72 35
www.univ-ubs.fr

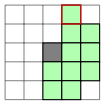


Figure 7: 10 vectors

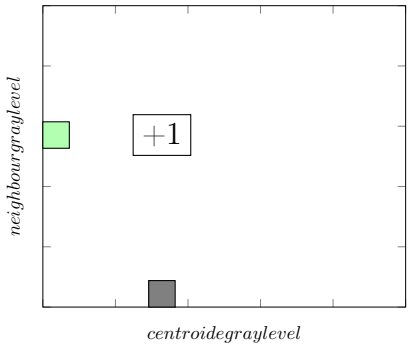


Figure 8: GLCM

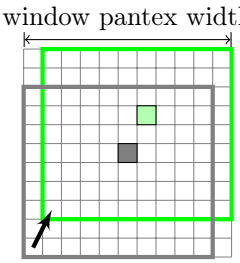


Figure 9: Sliding window

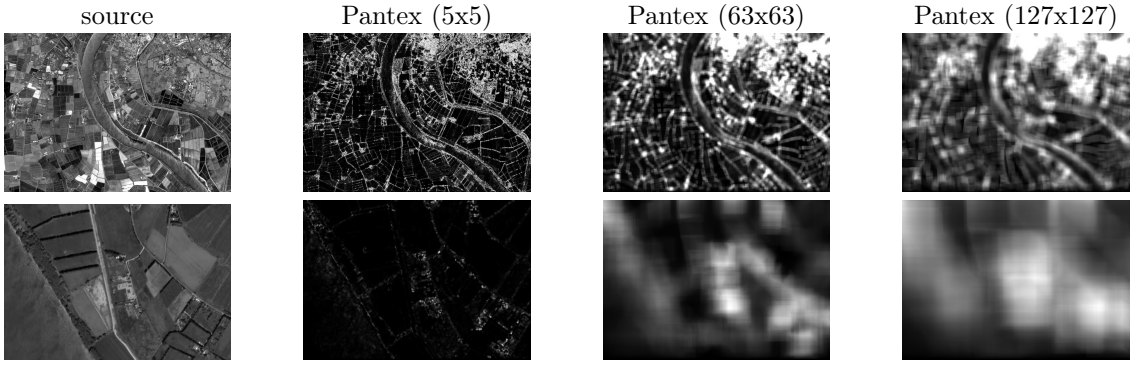
```
gdal_calc.py -R arles.tif --R_band=1 -G arles.tif --G_band=2 -B
    ↪ arles.tif --B_band=3 --outfile=arlesGS.tif '--calc=R*0.2989+G*0
    ↪ .5870+B*0.1140'

broceliande arlesGS.tif pantex5.tif -p 0 -c 1 --pantexWindowSide 5

broceliande arlesGS.tif pantex63.tif -p 0 -c 1 --pantexWindowSide 63

broceliande arlesGS.tif pantex127.tif -p 0 -c 1 --pantexWindowSide 127

broceliande arlesGS.tif pantex35.tif -p 0 -c 1
```



10.3.1 Computation steps

The complexity of the original algorithm depend of g number of neighbours, w the side of the sliding window, n the number of pixels.

$O(g.w^2.n)$

Because the Pantex index could be customized by the size of the sliding windows our approach is to reduce the impact of w.

We first of all define a pre-computed matrix S for square delta gray scale values.

$S_{i,j} = (i-j)^2$

We delay the normalization of GCLM. We call M the true occurrence matrix.

In that case, we rewrite the definition as

$$pantex = \min_{sw} \sum_{i=0}^{127} \sum_{j=0}^{127} (i-j)^2 . GLCM_{i,j} = \min_{sw} \frac{\sum_{i=0}^{127} \sum_{j=0}^{127} S_{i,j} . M_{i,j}}{card(slidingWindow)} = \min_{sw} \frac{C}{card(slidingWindow)}$$

Let call C the sum of co-occurrence by the square of delta gray level according a class neighbour. The difference between C of to consecutive pixel depend on forgotten pairs of neighbour in a side and new pairs of neighbours in the opposite side.

As shown in figure 10 to find the Pantex index of the next pixel, we remove impact of the old side (orange one) and add impact of the new side (blue one).

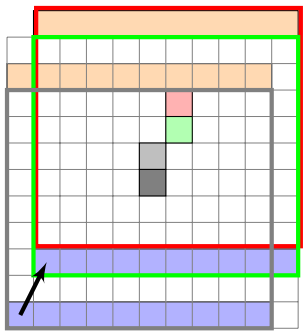


Figure 10: Pantex linear

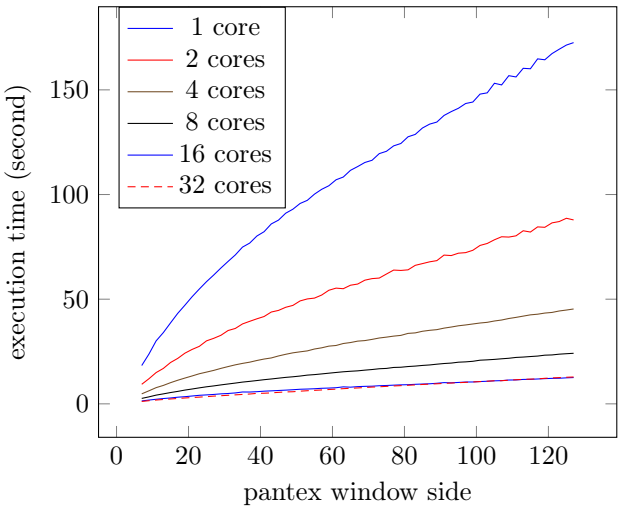


Figure 11: Pantex performance

The algorithm become (where i and j come from the position of pixels *old* and *new*) :
$C_i = C_{i-1} - \sum_{old} S_{i,j} + \sum_{new} S_{i,j}$
The complexity become linear $O(g.w.n)$

10.3.2 Precision

The Pantex index depend on the square of gray scale value ([0..127] in our case). So it is in [0..127^2]

If the input Pixel is a byte, we scale the Pantex by taking the square root. Other else the value is copy as the same type as the input pixels.

10.3.3 Complexity

The complexity is linear with the number of pixels and the side of the Pantex window.

We chose to move the sliding window according the y axis to reduce the memory default pages.

10.4 -s arg, --sobelBands arg

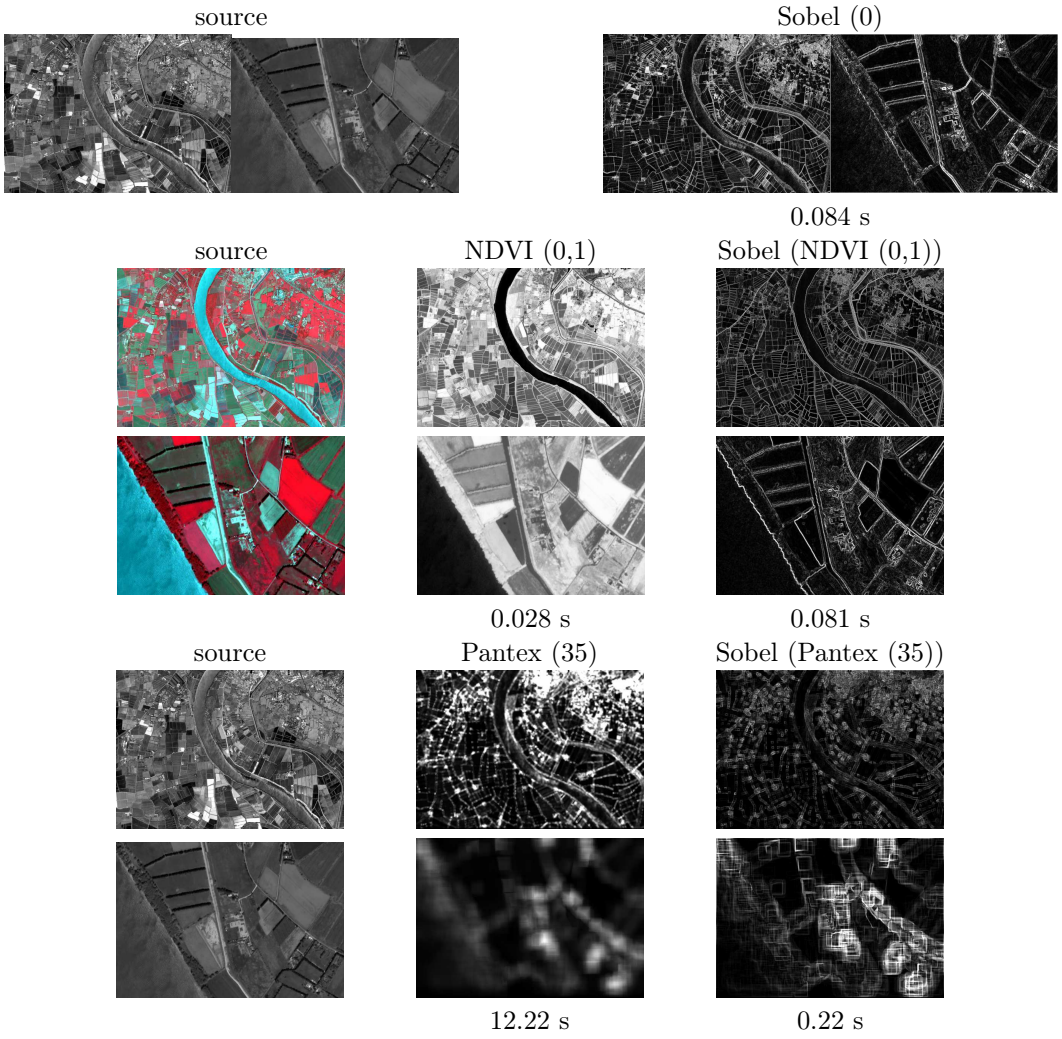
Produces Sobel bands, i.e. bands containing the Sobel gradient of the input band. We remind that this option needs to be used with option --copyBands in order to keep this band.

```
broceliande arlesGS.tif sobel.tif -s 0 -c 1

broceliande arles.tif sobel.tif -n 0,1 -s 3 -c 4

broceliande arlesGS.tif sobel.tif -p 0 -s 1 -c 2
```


Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr



10.4.1 Computation steps

The Sobel band computation consists of the 2 following 3 steps:

$$G_x = I * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = I * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

$$G_y = I * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = I * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

$$G = \sqrt{{G_x}^2 + {G_y}^2} \simeq |G_x| + |G_y|$$

10.4.2 Precision

The resulting band is set at the same precision than the input bands (ex. 8 bits or 16 bits).

10.4.3 Complexity

The complexity is linear with the number of pixels (each pixel from the input bands is read 4 times).

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

10.5 --noCopy

Do not keep the selected bands (either input image, or NDVI, Sobel or Pantex bands) in the list of bands to produce

10.6 --noNdvi

Do not keep the NDVI band in the list of bands to produce

10.7 --noPantex

Do not keep the Pantex bands in the list of bands to produce

10.8 --noSobel

Do not keep the Sobel bands in the list of bands to produce

11 Options for Attribute Profiles (AP) and Features Profiles (FP)

To fully understand the meaning of those options, please refer to section 2.

11.1 -t arg, --treeType arg

Select the type of tree of create.
The possible values are:

- **Min** (Min-Tree)
- **Max** (Max-Tree)
- **Med** (Median-Tree)
- **Alpha** (Alpha-Tree)

By default a type of tree is **Min**.

11.2 -a arg, --attributeType arg

Select the type of attributes used to prune the tree, according to a given list of thresholds.
The possible values are:

- **Area**
- **Perimeter**
- **ZLength** ($\Delta(BoundingBox.z)$)
- **Compactness** ($= area/perimeter^2$)
- **Complexity** ($= perimeter/area$)
- **Simplicity** ($= area/perimeter$)
- **Rectangularity** ($= area/BoundingBoxArea$)
- **Weight** (i.e. gray-scale level)
- **SD** ($StandardDeviation^2$ base on mean gray value of nodes)
- **SDW** ($StandardDeviation^2$ base on weight of nodes)
- **MoI** (Moment of Inertia)

By default the attributes for pruning trees is the **Area**.

11.3 --thresholds arg

List of thresholds used to prune the tree, according to the attribute previously chosen.

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

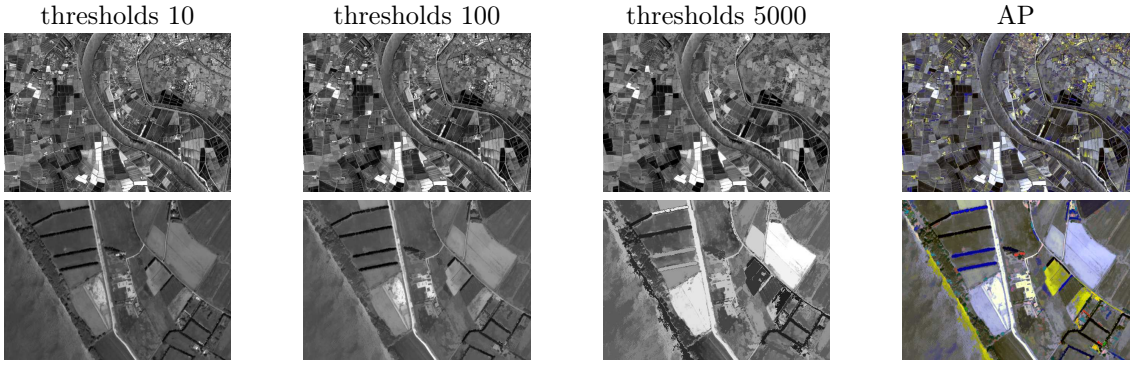
11.4 -f arg, --featureType arg

Select the type of features used to reconstruct the images, from the previously pruned trees, and gather them into a stack of filtered images. We remind that if this feature is the gray-scale level, the stack of images will be called Attribute Profiles (AP), whereas if this feature is any other feature, the stack of images will be called Feature Profiles (FP)
The possible values are:

- **AP** (Attribute Profile, i.e. gray-scale level)
- **Area**
- **Perimeter**
- **Zlength**
- **Compactness**
- **Complexity**
- **Simplicity**
- **Rectangularity**
- **Min** (min pixels value of the node)
- **Max** (max pixels value of the node)
- **Mean** (mean pixels value of the node)
- **SD**
- **SDW**
- **MoI**

By default the feature for reconstructing the images is **AP** (Attribute Profiles), in other words the gray-scale level.

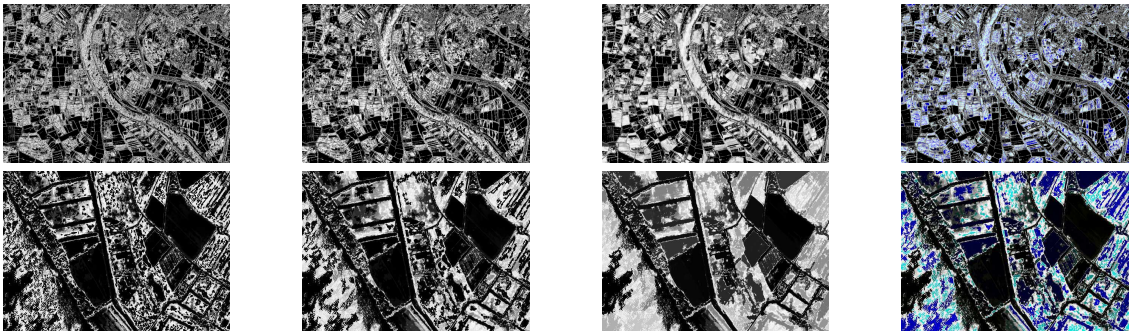
```
broceliande arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds
    ↪ 10,100,5000 --time
```




```
broceliande arlesGS.tif o.tif -b 0 -f Area -t Med -a Area --thresholds
    ↪  10,100,5000 --time
```

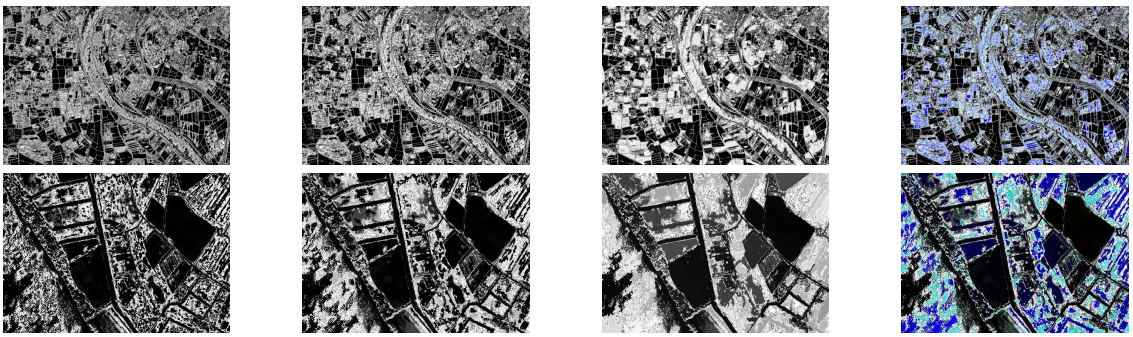

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

thresholds 10 | thresholds 100 | thresholds 5000 | Area



```
broceliande arlesGS.tif o.tif -b 0 -f Perimeter -t Med -a Area --
    ↪ thresholds 10,100,5000 --time
```

thresholds 10 | thresholds 100 | thresholds 5000 | Perimeter

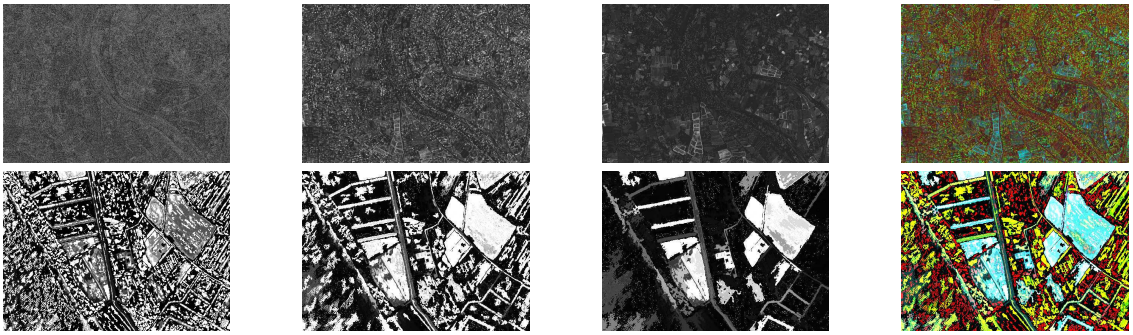


```
broceliande arlesGS.tif o.tif -b 0 -f Compactness -t Med -a Area --
    ↪ thresholds 10,100,5000 --time
```

If you use QGIS, you can highlight the picture by tune some parameters :

- Red band range [0, .3]
- Green band range [0, .1]
- Blue band range [0, .1]
- Brightness : 80
- Saturation : 100
- Contrast : 15

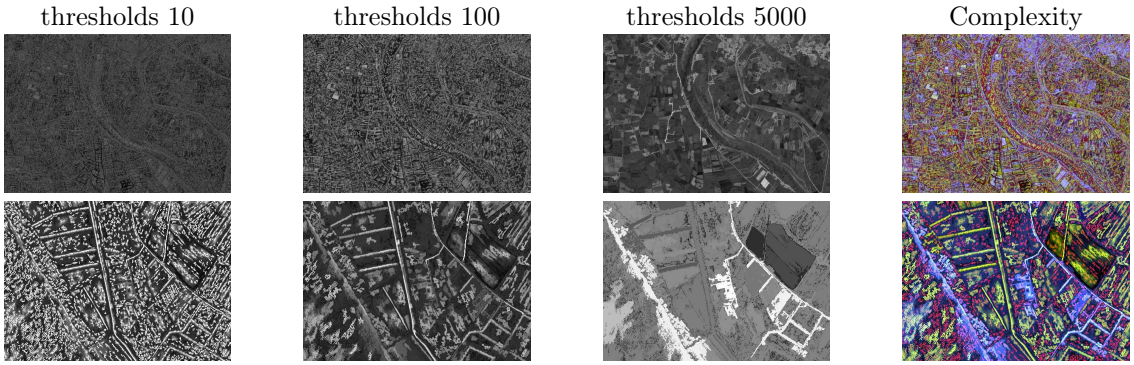
thresholds 10 | thresholds 100 | thresholds 5000 | Compactness



```
broceliande arlesGS.tif o.tif -b 0 -f Complexity -t Med -a Area --
    ↪ thresholds 10,100,5000 --time
```

Equipe OBELIX – UMR 6074 IRISA / Université Bretagne Sud & CNRS

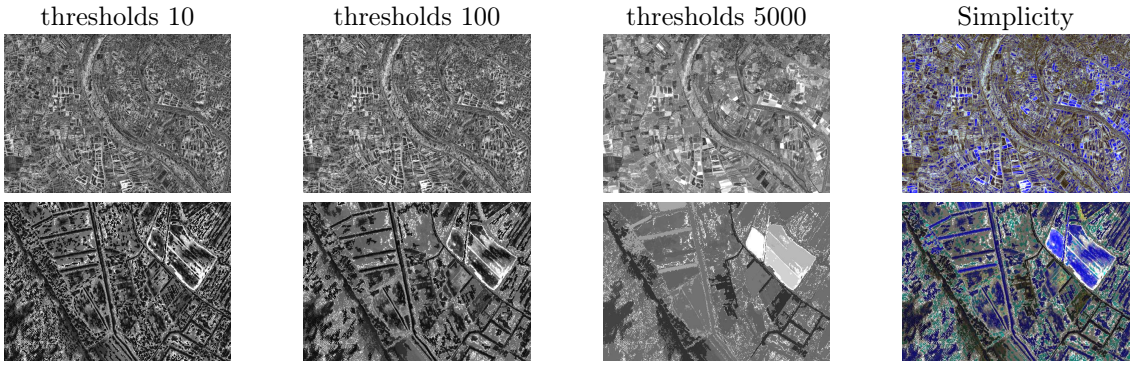
Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr



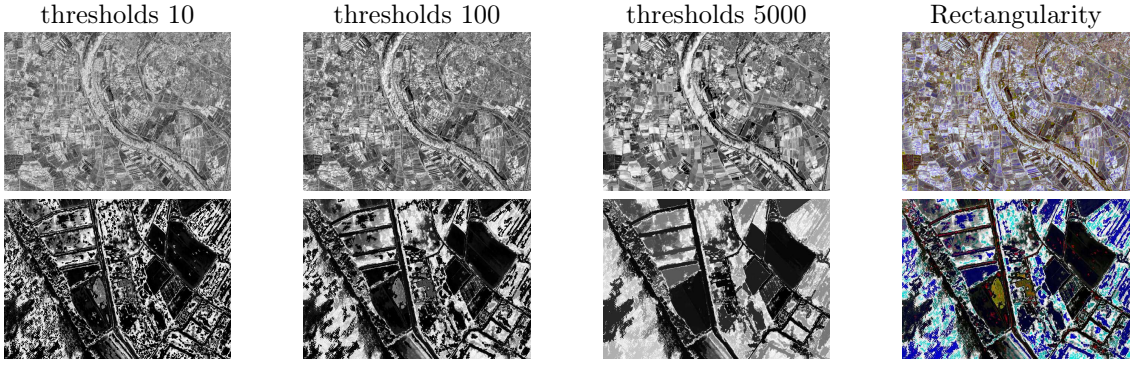

```
broceliande arlesGS.tif o.tif -b 0 -f Simplicity -t Med -a Area --
    ↪ thresholds 10,100,5000 --time
```

If you use QGIS, you can highlight the picture by tune some parameters :

- Red band range [.5, 11.1]
- Green band range [.8, 11.1]
- Blue band range [1.3, 11.1]

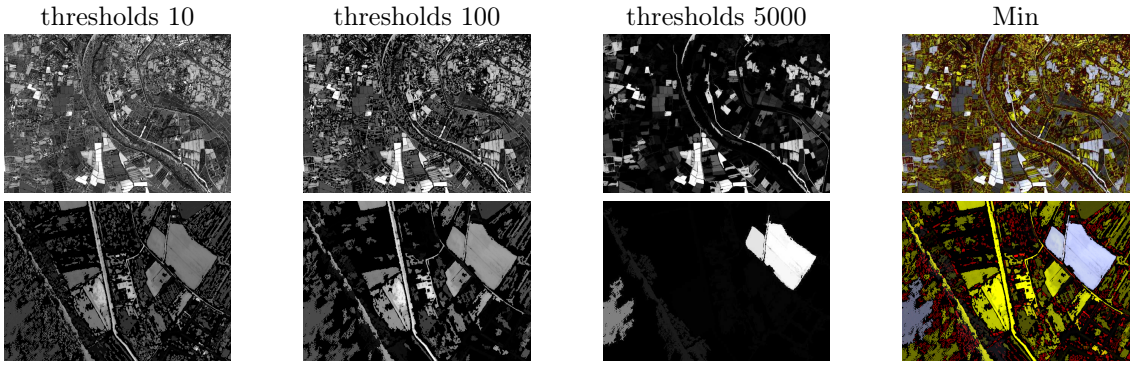



```
broceliande arlesGS.tif o.tif -b 0 -f Rectangularity -t Med -a Area --
    ↪ thresholds 10,100,5000 --time
```

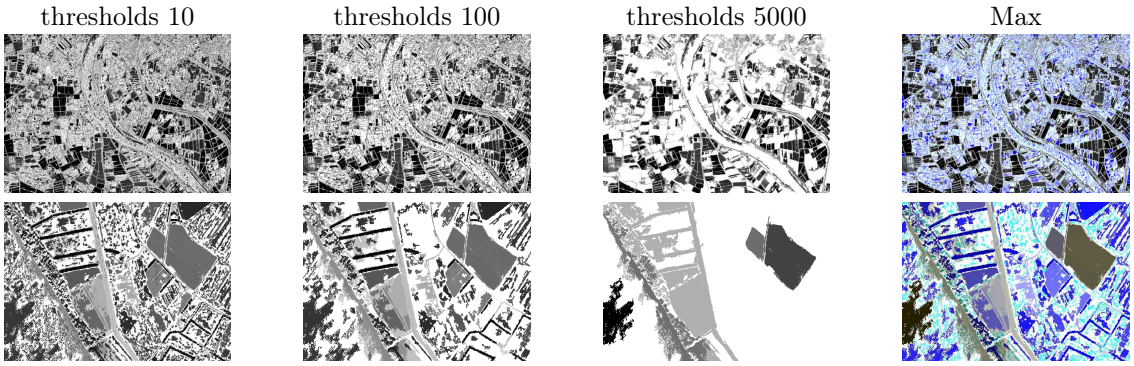



```
broceliande arlesGS.tif o.tif -b 0 -f Min -t Med -a Area --thresholds
    ↪ 10,100,5000 --time
```

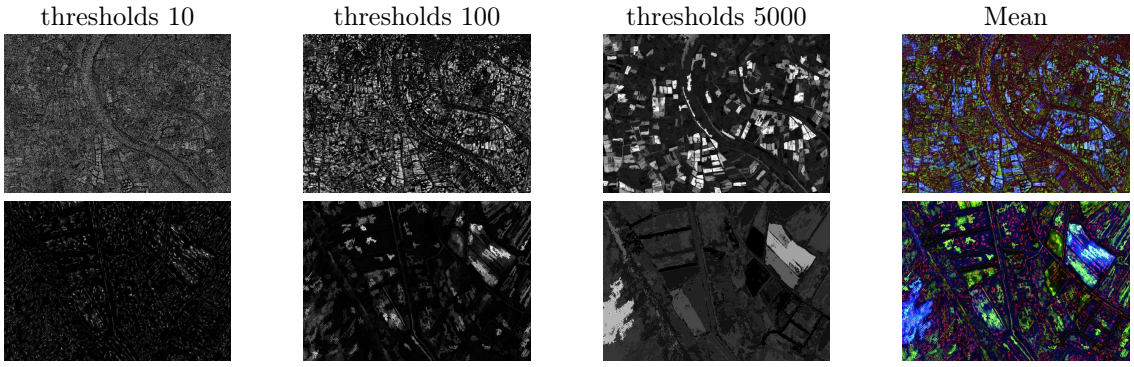

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr




```
broceliande arlesGS.tif o.tif -b 0 -f Max -t Med -a Area --thresholds
    ↪ 10,100,5000 --time
```

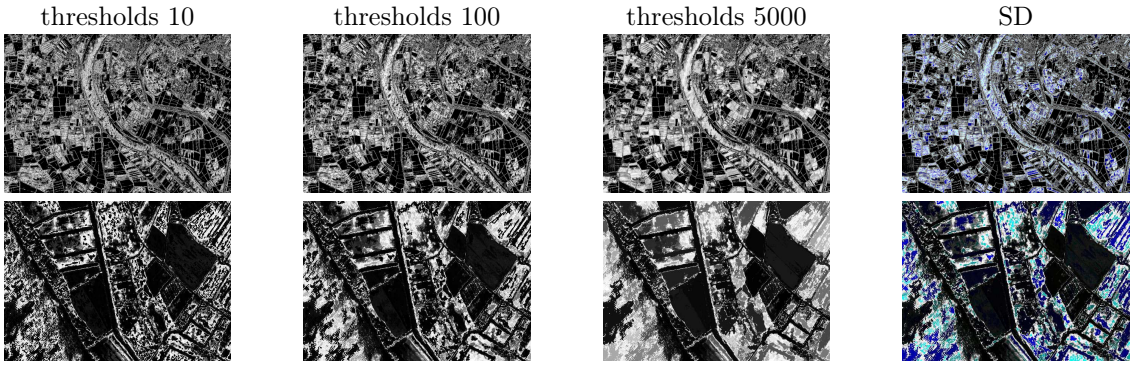



```
broceliande arlesGS.tif o.tif -b 0 -f Mean -t Med -a Area --thresholds
    ↪  10,100,5000 --time
```



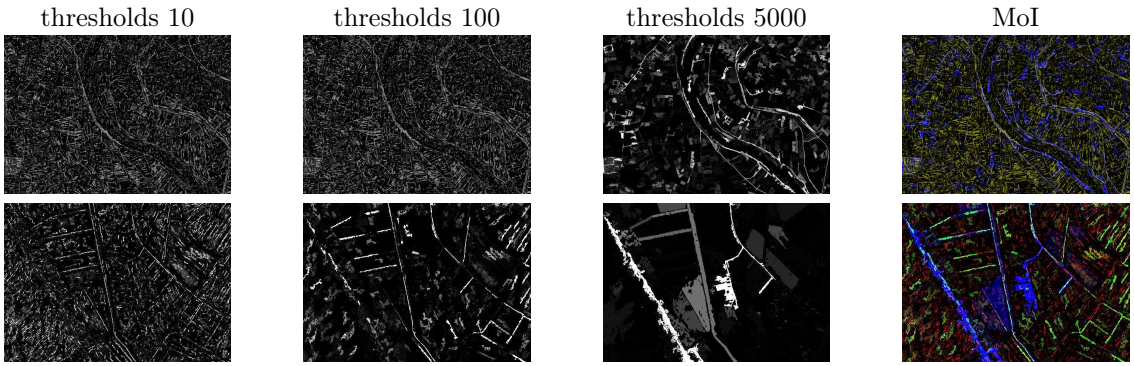

```
broceliande arlesGS.tif o.tif -b 0 -f SD -t Med -a Area --thresholds
    ↪ 10,100,5000 --time
```


Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr




```
broceliande arlesGS.tif o.tif -b 0 -f SDW -t Med -a Area --thresholds
    ↪ 10,100,5000 --time
```


```
broceliande arlesGS.tif o.tif -b 0 -f MoI -t Med -a Area --thresholds
    ↪ 10,100,5000 --time
```



The figure 12 indicates time of computation for attributes. It was launch on AMD Ryzen Threadripper 1950X (2*16 cores). That don't take in account build tree depending the type of it.

12 Options for Differential Attribute Profiles (DAP)

First note that the following option dapPos will automatically trigger the production of Differential Attribute Profiles (DAP) instead of Attribute Profiles (AP). The Differential Attribute Profiles are simply a series of images that are the difference between 2 different Attribute Profiles.

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

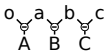
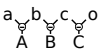
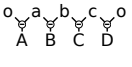
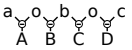
	Area	perimeter	BoundingBox	xyz mean	"the function"	filtering
Attribut Profile	0.04 s				n/a	0.18 s
Area	0.04 s				done	0.19 s
Perimeter	0.04 s	3.40 s			done	0.19 s
Compactness	0.04 s	3.40 s			0.003 s	0.14 s
complexity	0.04 s	3.40 s			0.003 s	0.19 s
Simplicity	0.04 s	3.40 s			0.003 s	0.19 s
Rectangularity	0.04 s		0.19 s		0.007 s	0.19 s
Min	0.04 s				0.05 s	0.16 s
Max	0.04 s				0.05 s	0.17 s
SD	0.04 s			0.08 s	0.09 s	0.19 s
SDW	0.04 s				0.06 s	0.19 s
MoI	0.04 s			0.15 s	0.16 s	0.19 s

Figure 12: computation time according Feature Profiles

12.1 --dapPos arg

The DAP images are usually computed between 2 filtered image. However in addition we can also compute the difference between a filtered image and the original image. In that case we have to specify between which filtered image and the original the subtraction will be made. Actually this situation is equivalent to inserting the original image in the list of filtered image, then make the difference image of all pairs of successive image of the list. So this option will specify in which position the original will be inserted inside the list of filtered images. Note that we might want to insert it more than once. The following table shows all the possible positions. In order to visualize the position of the original in the list of other images, we also add a diagram representation of the list of images using the following notations:

- o: original image
- a, b, c, ...: series of filtered images found in the attribute profiles, made with the different thresholds in this series, we assume that the thresholds are ordered from the lowest to the highest
- A, B, ...: difference between 2 successive images

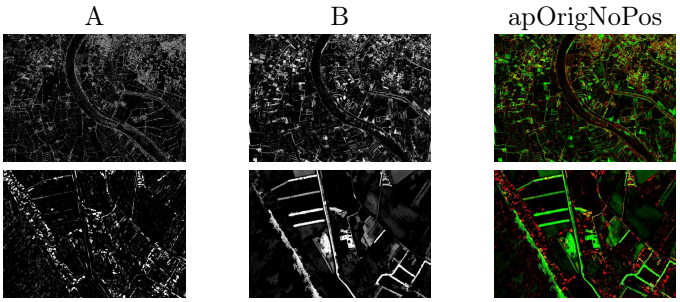
Name	Diagram representation	Compute the difference between:
apOrigNoPos	0 	- all successive filtered images
apOrigPosBegin	1 	- all successive filtered images - the original image and the first filtered image (lowest threshold)
apOrigPosEnd	2 	- all successive filtered images - the original image and the last filtered image (highest threshold)
apOrigPosBoth	3 	- all successive filtered images - the original image and the first filtered image (lowest threshold) - the original image and the last filtered image (highest threshold)
apOrigPosEverywhere	4 	- all the filtered image and the original image

12.2 --dapWeight

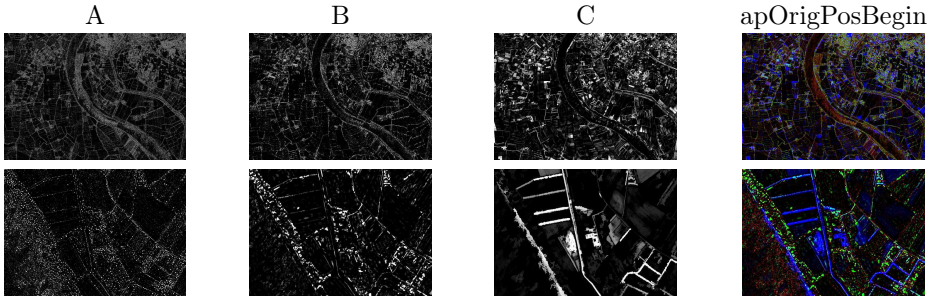
Multiply the DAP images by the original image. This option is useful to obtain DAP images with a better contrast.

Université Bretagne Sud - IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex - +33/0 2 97 01 72 35
www.univ-ubs.fr

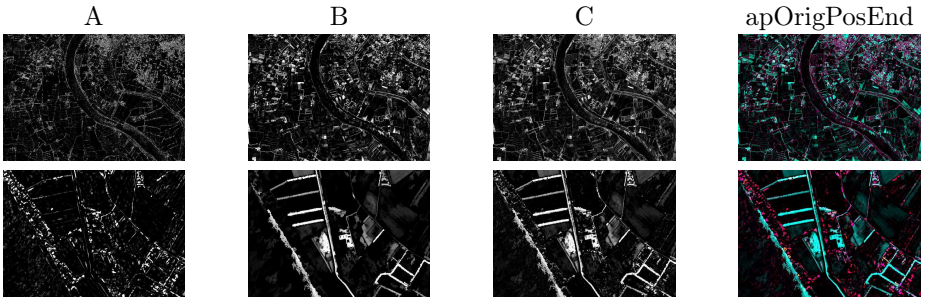
```
broceliande arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds
    ↪ 10,100,5000 --dapPos apOrigNoPos --auto --time
```




```
broceliande arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds
    ↪ 10,100,5000 --dapPos apOrigPosBegin --auto --time
```

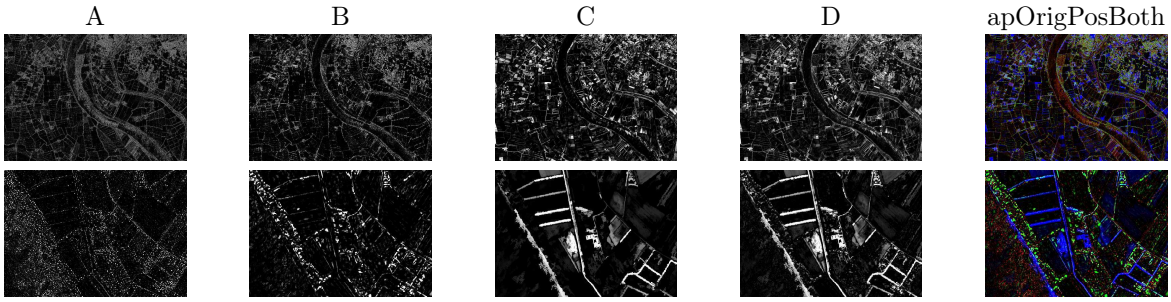



```
broceliande arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds
    ↪ 10,100,5000 --dapPos apOrigPosEnd --auto --time
```

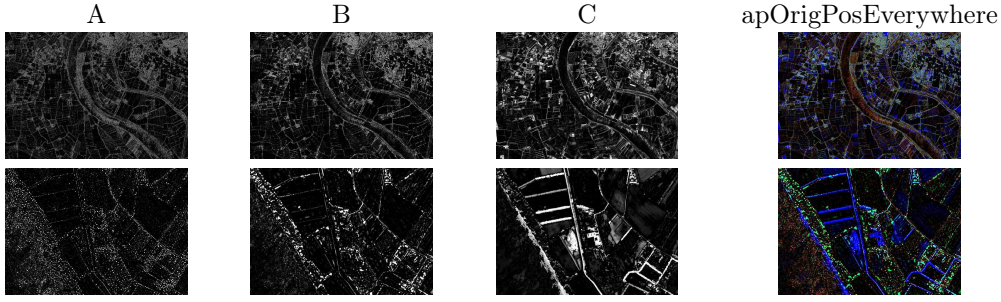



```
broceliande arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds
    ↪ 10,100,5000 --dapPos apOrigPosBoth --auto --time
```


Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr




```
broceliande arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds
    ↪ 10,100,5000 --dapPos apOrigPosEverywhere --auto --time
```



13 Options for textures

13.1 Note about Integral Images

For the following options for textures, we will first compute a special image called "integral image" which is a computational optimization allowing us to perform the computation with a linear complexity.

13.1.1 Definition

The integral image (a.k.a. "Summed Area Table") I of an original image i is defined such as a point (x_0, y_0) of the integral image is the sum of all pixel values in the original image above and to the left of this point (x_0, y_0), inclusive (see figure 13). In other words:

$$I(x_0, y_0) = \sum_{\substack{0 \leq x \leq x_0 \\ 0 \leq y \leq y_0}} i(x, y)$$

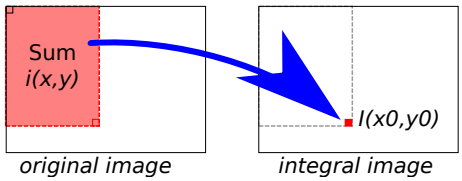


Figure 13: Definition of an Integral Image

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

13.1.2 Creation

Actually the fastest way to create the Integral Image is not by using its definition but instead by using the following property (where i is the original image and I the integral image currently being creating):

$$I(x_0, y_0) = i(x_0, y_0) + I(x_0, y_0 - 1) + I(x_0 - 1, y_0) - I(x_0 - 1, y_0 - 1)$$

This property is interesting as it allows to build the integral image simply and incrementally, in other word each point of the integral image is deduced from 3 other points already computed in the integral image and 1 point from the original image. So that actually we never have to perform a multiple sum.

It is also important to notice that for any term $I(..., ...)$ in the formula, whenever one of the coordinate (or both) is equal to 0, we should replace the corresponding this term by 0. This situation is equivalent to adding one extra columns on the left and one extra row at the top of the integral image, both filled with 0.

In figure 14 we can see a numerical example of such a creation process. The point with value 33 in the integral image corresponds to the summed-area of the red zone in original image and is computed as follow:

$$33 = 10 + 14 + 15 - 6$$

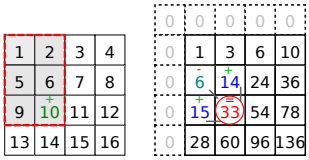


Figure 14: Creation of an Integral Image
(left:original image, right:integral image)

13.1.3 Usage

An interesting property is that the summed-are of any rectangular region $[x_1; x_2,] \times [y_1; y_2]$, in the original image can be computed using only 4 points of the integral image I:

$$\sum_{\substack{x_1 \leq x \leq x_2 \\ y_1 \leq y \leq y_2}} i(x, y) = I(x_2, y_2) + I(x_1 - 1, y_1 - 1) - I(x_2, y_1 - 1) - I(x_1 - 1, y_2)$$

Actually we can notice than the first property that was used for creation is just a special case of this second property (the rectangular region being only a mere point (x, y) so that $x_1 = x_2 = x_0$ and $y_1 = y_2 = y_0$).

In figure 15 we can see an numerical example of usage of the integral matrix:

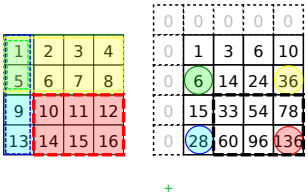


Figure 15: Usage of an Integral Image
(left:original image, right:integral image)

- on the original image, in order to compute the summed-area on the red rectangle, we can compute the sum of the whole image, then subtract the sum on the blue rectangle, subtract the sum on the yellow rectangle, then add the sum on the green rectangle (because it was subtracted twice).

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

- on the integral image, we can obtain the same result with a much more simple operation using only 4 points (note that each colored point on the integral image corresponds to the bottom-right corner of the colored rectangle in the original image):

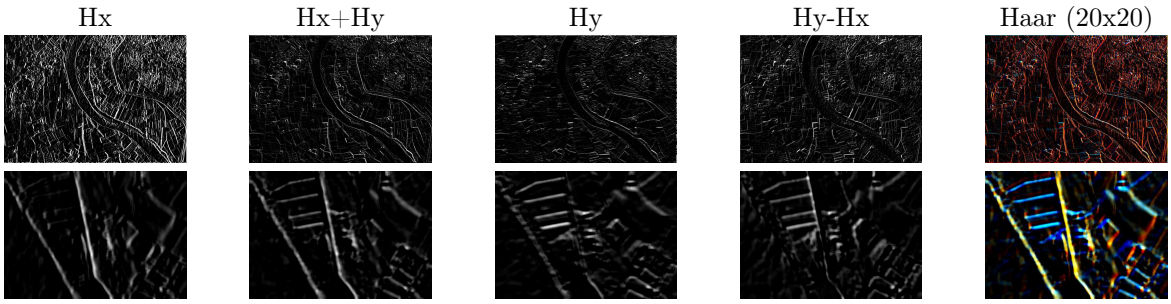
$$72 = 136 - 36 - 28 + 6$$

13.2 -H arg, --haarSizes arg

Compute Haar-like features for a given window size given in argument (in format $H \times W$). Here 4 different features are produced and put into 4 different bands:

- H_x
- $H_x + H_y$
- H_y
- $H_y + H_x$

```
broceliande arlesGS.tif o.tif -b 0 --haar 20x20 --time
```



13.2.1 Computation

The Haar-like features are named upon the Haar wavelets as they follow the same principle. Historically they were use for face recognition and more generally for object recognition. They are usually computed in a moving rectangular A windows, which is itself divided into several regions in order to detect a precise pattern. Here we will only use rectangular features which means each window is divided into 2 rectangular regions, in order to detect horizontal, vertical or diagonal patterns.

For example as we can see in figure 16 the H_x feature is meant to detect vertical patterns (edges of change in texture for example). It gives a non-zero value each time we have a difference between the 2 vertical sub-rectangles. In the same way the H_y feature is meant to detect horizontal patterns. The diagonal patterns are discussed below.

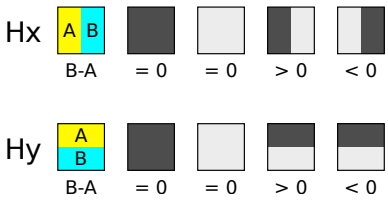


Figure 16: Vertical and horizontal Haar features

13.2.2 Optimization (vertical and horizontal features)

As we said previously, the Haar features are computed from integral images. The summed area is computed on each sub-rectangle of the moving window by using 4 points in the integral image for each

sub-rectangle. For example to compute H_x or H_y we will need to compute the difference of 2 summed-area on surfaces A and B, which will be done by using the values of points a, b, c, d, e, f in the integral images, as follow (see figure 16):

$$B - A = (f + b - e - c) - (e + a - b - d) = f + 2b + d - a - c - 2e$$

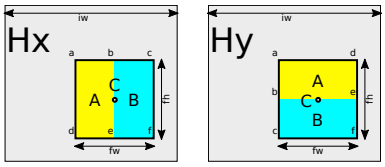


Figure 17: Vertical and horizontal Haar features: optimization using integral image

13.2.3 Optimization (diagonal features)

Theoretically, the diagonal features H_a and H_b with directions $\pi/4$ and $3\pi/4$ computed from should be H_x and H_y as:

$$H_a = \begin{pmatrix} \cos(\pi/4).H_x \\ \sin(\pi/4).H_y \end{pmatrix}$$

$$H_b = \begin{pmatrix} -\sin(\pi/4).H_x \\ \cos(\pi/4).H_y \end{pmatrix}$$

Actually we can avoid the cost of computing the trigonometric functions, because usually we are only interested by the direction of those patterns. One of the reason is that we want to use those features for future classification and we will mainly focus on a "random forest" type of classifier, which is indifferent to sign and scale. So in that case, as we can see in figure 18, the following features will give the same results:

$$H_{x+y} = \begin{pmatrix} H_x \\ H_y \end{pmatrix}$$

$$H_{x-y} = \begin{pmatrix} -H_x \\ H_y \end{pmatrix}$$

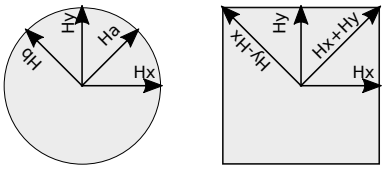


Figure 18: Diagonal Haar features: optimization using horizontal and vertical features

13.2.4 Complexity

The Haar-like features will be based on the computation of an integral image. In a similar way, the statistics features will be based on the computation of a square integral image. A square integral image is simply the integral image of the square values of the original image.

The complexity of creation for an integral image is linear (with the number of pixels), while the complexity of usage is constant. The same result apply for the square integral image that implies only a additional step with linear complexity (computing the square value of all pixels). So the overall complexity of the integral image (or square integral image) is always linear.

	integral image	square integral image	"the function"
Haar	0.0498 s		0.1126 s
Stat	0.0374 s	0.0475 s	0.0923 s

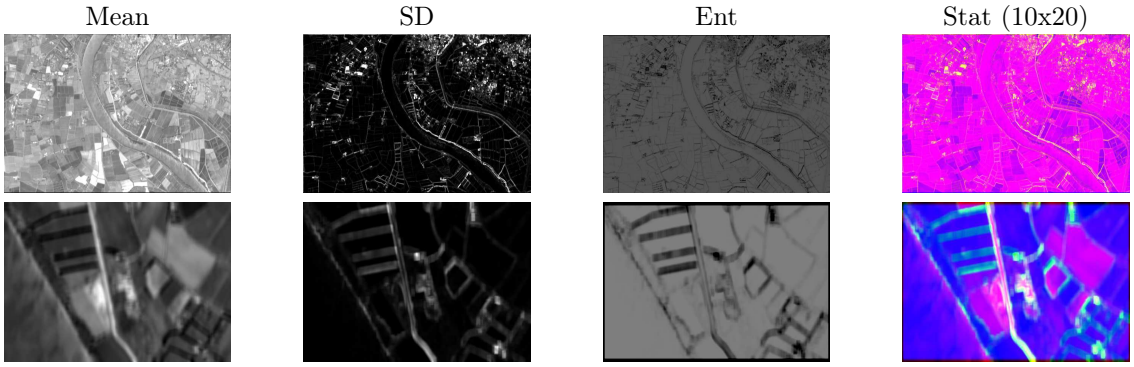
Figure 19: computation time according texture

13.3 -T arg, --statSizes arg

Compute statistics features for a given window size given in argument (in format $H \times W$). Currently the statistics produce the following bands:

- Mean
- Standard-deviation
- Entropy

```
broceliande arlesGS.tif o.tif -b 0 --stat 10x20 --time
```



13.3.1 Computation

We consider a window containing n pixels named p_i (with $0 \leq i \leq n-1$)
Then the mean value is computed as:

$$mean = \frac{\sum_{i=0}^{n-1} p_i}{n}$$

The variance and standard-deviation are computed as:

$$var = \frac{\sum_{i=0}^{n-1} p_i^2}{n} - mean^2$$

$$std = \sqrt{var}$$

13.3.2 Complexity

We remind the in all the previous formula, the sums are not performed directly, but instead taken either from the integral image (for $\sum p_i$) or from the square integral image (for $\sum p_i^2$).

For that reason, the complexity of all the statistics features remain linear (with the number of pixels).

The figure 19 indicates time of computation for texture. It was launch on AMD Ryzen Threadripper 1950X (2*16 cores). That don't take in account build tree depending the type of it.

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

13.4 --noHaar

Do not keep the Haar feature band in in list of bands to produce

13.5 --noStat

Do not keep the statistics feature band in the list of bands to produce

14 Options for configuration files ("channel" files)

For an example of real configuration file, please see section 3.4 The configuration files can be loaded, saved and/or displayed on screen

14.1 --loadChannel arg

Loads a configuration file ("channel" file), which uses XML format. The file extension can be omitted, in that case it uses the default extension *.ch Not that the configuration file is loaded before taking into account the possible manual options of the current command line, so that:

- if the options in the configuration files and the manual options are independent, they will just add up in the final configuration
- if the options in the configuration files and the manual options are contradictory, the manual options will override the options found in the configuration file and be in the final configuration

14.2 --saveChannel arg

Saves a configuration file ("channel" file), using XML format. The file extension can be omitted, in that case it uses the default extension *.ch

14.3 --cfgName arg

Give a name embedded in a configuration file. The name have no impact to the behavior of the software. It is just a comment.

14.4 --showChannel

Displays the current configuration on the screen. Note that the presentation of the information displayed on screen differs from the presentation used in configuration files. Actually this option showChannel is more useful to check afterwards what has really been done inside the output image. For example it shows every single band produced on a separate line (which is not always the case the configuration file).

15 Options for ground truth (broceliande only)

This section deals with using a ground truth image for the classification.

15.1 -g arg, --groundTruth arg

Ground truth file name image for classification.

15.2 --showGroundTruth arg

Show selected ground truth.

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

15.3 --tagValues arg (=1)

Values use as labels in ground truth. This option define list or range of integer.

15.4 --bgValue arg (=nan)

Value use as non foreground in ground truth (nan => no bg tag).

15.5 --tagsRate arg (=100%)

Number of labeled pixels to use per class: 200 means 200 pixels, 40% means 40% per class of ground truth.

15.6 --bgTagRate arg (=80%)

Number of tagged background of ground truth (the other are taken randomly).

15.7 --bgRate arg (=125%)

Number of background of ground truth rate of foreground.

16 Options for training (broceliande only)

This section deals with settings for the training of the classification.

16.1 --cleanTagsFlag

Clean all labelled pixels in ground truth according accuracy of 50% samples chosen randomly.

16.2 --cleanSamplesFlag

Clean selected foreground according accuracy

16.3 --nTrees arg (=64)

Number of decision trees for random forest algorithm.

16.4 --maxPerRFBatch arg (=65536)

Size of batch for random forest algorithm

16.5 --loadTrain arg

Load training

16.6 --saveTrain arg

Save training

17 Options for batch processing (broceliande only)

This section deals with batch processing, i.e. performing processing on multiple images using only one command .

Université Bretagne Sud – IRISA
Campus de Tohannic, BP 573, 56017 Vannes Cedex – +33/0 2 97 01 72 35
www.univ-ubs.fr

17.1 --batchDir arg

Use all image in a batch directory

17.2 --batchPrefixResult arg

Add prefix for batch result

17.3 --batchResultDir arg

Put batch result in a directory

17.4 --batchOverwriteFlag

Overwrite previous batch result

References

[1] Loïc Faucqueur, François Merciol, Bharath Bhushan Damodaran, Pierre-Yves Rémy, Baudoin Desclée, Fabrice Dazin, Sébastien Lefèvre, and Christophe Sannier. Scalable extraction of small woody features (swf) at the pan-european scale using open source solutions. In *ESA EO Open Science*, Frascati, Italy, 2017.

[2] François Merciol, Thibaud Balem, and Sébastien Lefèvre. Efficient and large-scale land cover classification using multiscale image analysis. In *Big Data from Space*, Toulouse, France, 2017.

[3] François Merciol, Loïc Faucqueur, Bharath Bhushan Damodaran, Pierre-Yves Rémy, Baudoin Desclée, Fabrice Dazin, Sébastien Lefèvre, and Christophe Sannier. GEOBIA at the Terapixel Scale: From VHR Satellite Images to Small Woody Features at the Pan-European Level. In *GEOBIA 2018 - From pixels to ecosystems and global sustainability*, Montpellier, France, June 2018. Centre d'Etudes Spatiales de la BIOsphere (CESBIO) and Office national d'études et de recherches aérospatiales (ONERA) and Espace pour le développement (ESPACE DEV) and Société T.E.T.I.S.

[4] François Merciol, Loïc Faucqueur, Bharath Bhushan Damodaran, Pierre-Yves Rémy, Baudouin Desclée, Fabrice Dazin, Sébastien Lefèvre, Antoine Masse, and Christophe Sannier. GEOBIA at the Terapixel Scale: Toward Efficient Mapping of Small Woody Features from Heterogeneous VHR Scenes. *ISPRS International Journal of Geo-Information*, 8(1):46, January 2019.

[5] François Merciol, Antoine Sauray, and Sébastien Lefèvre. Interoperability of multiscale visual representations for satellite image big data. In *Conference on Big Data from Space (BiDS)*, Santa Cruz de Tenerife, Spain, 2016.

[6] M. Pesaresi, A. Gerhardinger, and F. Kayitakire. A robust built-up area presence index by anisotropic rotation-invariant textural measure. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 1(3):180–192, 2008.