Tutorial on Evaluation of Background Subtraction Algorithms

A practical introduction to the ChangeDetection.NET dataset, BGSLibrary, and C++ programming for evaluating background subtraction algorithms

Benjamin Laugraud

Montefiore Institute, University of Liège, Belgium

August 28th, 2018

VISMAC 2018

Vico Equense, Naples, Italy
Modern background subtraction (BGS) algorithms must be evaluated rigorously!

Such an evaluation should consider all the challenges associated with the field.

Important to convince reviewers and readers of the efficacy of a method.

How to perform a rigorous evaluation in practice?
## Important Topics to Study

### ChangeDetection.NET (CDnet) Dataset
- Content of the website.
- Structure and conventions of the dataset.
- Evaluation tools associated with the dataset.

### BGSLibrary
- Presentation.
- Content of the website.
- Structure and conventions of the library.

### C++ Programming
- How to use an algorithm from the BGSLibrary in your own C++ code?
- How to apply an algorithm from the BGSLibrary on CDnet?
- How to integrate your own algorithm in the BGSLibrary?
Prerequisites

- The **Ubuntu** (or derived) GNU/Linux distribution.\(^1\)
- The **OpenCV** library.\(^2\)
- The **CMake** compilation utility.\(^3\)
- The **Matlab** programming environment.\(^4\)

---

\(^1\)https://www.ubuntu.com  
\(^2\)https://opencv.org  
\(^3\)https://cmake.org  
\(^4\)https://www.mathworks.com/products/matlab.html
CDnet Dataset [2] [4]
The identification of changing or moving areas in the field of view of a camera is a fundamental pre-processing step in computer vision and video processing. Example applications include visual surveillance (e.g., people counting, action recognition, anomaly detection, post-event forensics), smart environments (e.g., room occupancy monitoring, fall detection, parking occupancy detection), and video retrieval (e.g., activity localization and tracking). Although subsequent processing may be different in each case, typically one has to start with the identification of regions of interest which, in the case of video, are either short-term changes, i.e., video dynamics (motion), or long-term changes, i.e., appearing/disappearing objects and structural changes. Clearly, motion and change detection are only pre-processing steps for subsequent tracking, classification, or estimation, albeit important ones.

To date, many motion and change detection algorithms have been developed that perform well in some types of videos, but most are sensitive to sudden illumination changes, environmental conditions (night, rain, snow, air turbulence), background/camera motion, shadows, and camouflage effects (photometric similarity of object and background). There is no single algorithm today that seems to be able to simultaneously address all the key challenges that accompany real-world (non-synthetic) videos. In fact, no single, realistic, large-scale dataset exists that covers a range of challenges present in the real world and includes accurate ground truths.

This website encapsulates a rigorous and comprehensive academic benchmarking effort for testing and ranking existing and new algorithms for change and motion detection. It will be revised/expanded from time to time based on received feedback, and will maintain a comprehensive ranking of submitted methods for years to come.
### Overall Results

Methods with the "(supervised method)" tag involve a supervised machine learning algorithm potentially trained on the groundtruth data used to produce the metrics reported in this page. Thus, these methods should not be compared directly with the other unsupervised methods without further investigation and careful analysis. Please refer to the original papers for more details.

#### Click on method name for more details.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average ranking across categories</th>
<th>Average ranking</th>
<th>Average Re</th>
<th>Average Sp</th>
<th>Average FPR</th>
<th>Average FNR</th>
<th>Average PWC</th>
<th>Average F-Measure</th>
<th>Average Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>FaSeNet V2 (Supervised Method) [45]</td>
<td>1.36</td>
<td>1.29</td>
<td>0.9891</td>
<td>0.9989</td>
<td>0.0002</td>
<td>0.0108</td>
<td>0.0402</td>
<td>0.9847</td>
<td>0.9823</td>
</tr>
<tr>
<td>FvSeNet 5 (FPMV; Supervised Method) [44]</td>
<td>1.91</td>
<td>2.14</td>
<td>0.9886</td>
<td>0.9997</td>
<td>0.0003</td>
<td>0.0104</td>
<td>0.0461</td>
<td>0.9804</td>
<td>0.9751</td>
</tr>
<tr>
<td>FvSeNet/Fourground Separability Network (Supervised Method) [39]</td>
<td>2.73</td>
<td>2.57</td>
<td>0.9036</td>
<td>0.9989</td>
<td>0.0002</td>
<td>0.0104</td>
<td>0.0559</td>
<td>0.9770</td>
<td>0.9756</td>
</tr>
<tr>
<td>BPSVGAN (supervised method) [41]</td>
<td>4.00</td>
<td>4.00</td>
<td>0.9544</td>
<td>0.9990</td>
<td>0.0010</td>
<td>0.0456</td>
<td>0.2272</td>
<td>0.9501</td>
<td>0.8472</td>
</tr>
<tr>
<td>BSGAN (supervised method) [43]</td>
<td>4.91</td>
<td>5.29</td>
<td>0.9476</td>
<td>0.9863</td>
<td>0.0017</td>
<td>0.0524</td>
<td>0.3281</td>
<td>0.9339</td>
<td>0.9232</td>
</tr>
<tr>
<td>Cascade CNN-supervised method) [29]</td>
<td>6.45</td>
<td>5.71</td>
<td>0.9506</td>
<td>0.9968</td>
<td>0.0032</td>
<td>0.0494</td>
<td>0.4052</td>
<td>0.9203</td>
<td>0.8967</td>
</tr>
<tr>
<td>LJTIS 5 [27]</td>
<td>9.45</td>
<td>10.71</td>
<td>0.7849</td>
<td>0.9948</td>
<td>0.0052</td>
<td>0.2151</td>
<td>1.1986</td>
<td>0.7717</td>
<td>0.8087</td>
</tr>
</tbody>
</table>
## Results for CD.net 2014

### Bad Weather Category

Click on method name for more details.

<table>
<thead>
<tr>
<th>Method</th>
<th>Average ranking</th>
<th>Average R</th>
<th>Average Sp</th>
<th>Average FPR</th>
<th>Average FNR</th>
<th>Average PWC</th>
<th>Average Measure</th>
<th>F- Measure</th>
<th>Average Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>FySNet_v2 (Supervised Method) [45]</td>
<td>1.71</td>
<td>0.9969</td>
<td>0.9999</td>
<td>0.0001</td>
<td>0.0131</td>
<td>0.0296</td>
<td>0.9904</td>
<td>0.9939</td>
<td></td>
</tr>
<tr>
<td>FySNet_v2_5 (CPMv2, Supervised Method) [44]</td>
<td>2.14</td>
<td>0.9886</td>
<td>0.9999</td>
<td>0.0001</td>
<td>0.0112</td>
<td>0.0221</td>
<td>0.9897</td>
<td>0.9907</td>
<td></td>
</tr>
<tr>
<td>FySNet (Foreground Spontaneous Network) (Supervised Method) [39]</td>
<td>3.43</td>
<td>0.9793</td>
<td>0.9998</td>
<td>0.0022</td>
<td>0.0207</td>
<td>0.0644</td>
<td>0.9845</td>
<td>0.9938</td>
<td></td>
</tr>
<tr>
<td>BSRVGAN (supervised method) [41]</td>
<td>4.43</td>
<td>0.9566</td>
<td>0.9966</td>
<td>0.0004</td>
<td>0.0434</td>
<td>0.1004</td>
<td>0.9644</td>
<td>0.9725</td>
<td></td>
</tr>
<tr>
<td>BSRGAN (supervised method) [40]</td>
<td>6.71</td>
<td>0.9335</td>
<td>0.9983</td>
<td>0.0007</td>
<td>0.0665</td>
<td>0.1272</td>
<td>0.9465</td>
<td>0.9559</td>
<td></td>
</tr>
<tr>
<td>Cascade CNN (Supervised method) [29]</td>
<td>8.29</td>
<td>0.9312</td>
<td>0.9983</td>
<td>0.0007</td>
<td>0.0888</td>
<td>0.1911</td>
<td>0.9431</td>
<td>0.9555</td>
<td></td>
</tr>
<tr>
<td>DeepBS (supervised method) [54]</td>
<td>10.71</td>
<td>0.7517</td>
<td>0.9986</td>
<td>0.0004</td>
<td>0.2483</td>
<td>0.3784</td>
<td>0.8301</td>
<td>0.9477</td>
<td></td>
</tr>
<tr>
<td>SemanticBGS [39]</td>
<td>13.86</td>
<td>0.7420</td>
<td>0.9994</td>
<td>0.0006</td>
<td>0.2580</td>
<td>0.5112</td>
<td>0.8200</td>
<td>0.9518</td>
<td></td>
</tr>
<tr>
<td>SubSENSE [13]</td>
<td>14.00</td>
<td>0.8213</td>
<td>0.9989</td>
<td>0.0011</td>
<td>0.1787</td>
<td>0.4527</td>
<td>0.6619</td>
<td>0.9051</td>
<td></td>
</tr>
<tr>
<td>WornetMD [42]</td>
<td>14.43</td>
<td>0.8213</td>
<td>0.9989</td>
<td>0.0011</td>
<td>0.1787</td>
<td>0.4524</td>
<td>0.6616</td>
<td>0.9084</td>
<td></td>
</tr>
<tr>
<td>HTTSA [17]</td>
<td>14.47</td>
<td>0.7848</td>
<td>0.9643</td>
<td>0.0017</td>
<td>0.2917</td>
<td>0.6269</td>
<td>0.8968</td>
<td>0.9411</td>
<td></td>
</tr>
</tbody>
</table>
Details of the dataset:

- This dataset contains 11 video categories with 4 to 6 videos sequences in each category.
- Each individual video file (.zip or .7z) can be downloaded separately. Alternatively, all videos files within one category can be downloaded as a single .zip or .7z file.
- Each video file when uncompressed becomes a directory which contains the following:
  1. a sub-directory named “input” containing a separate JPEG file for each frame of the input video.
  2. a sub-directory named “groundtruth” containing a separate BMP file for each frame of the groundtruth.
  3. an empty folder named “results” for binary results (1 binary image per frame per video you have processed).
  4. files named “ROI.bmp” and “ROI.jpg” showing the spatial region of interest.
  5. a file named “temporalROI.dat” containing two frame numbers. Only the frames in this range will be used to calculate your score.
- The groundtruth images contain 5 labels namely:
  - 0 : Static
  - 50 : Hard shadow
  - 85 : Outside region of interest
  - 170 : Unknown motion (usually around moving objects, due to semi-transparency and motion blur)
  - 255 : Motion

Click here to download the entire dataset: dataset2014.zip | 7z

Click on the tabs below to view sample frames and download individual videos and complete video categories.

If you use this facility in any publication, we request you to kindly acknowledge this website (www.changedetection.net) and cite the following overview paper:


Pedestrian detection dataset

As a subset of Changedetection2014 dataset, this dataset contains 10 videos which mostly contain pedestrians.
Once uncompressed, the dataset is in a dataset folder.

Inside, there is a folder per category gathering a folder per sequence.

For each sequence, there is a folder for the input and the groundtruth.

In temporalROI.txt there are 2 frame numbers defining the evaluation interval.
- The input folder of a given sequence contains one .jpg file per frame.
- The name of a file is in, the frame number encoded with 6 digits, and .jpg.
- The groundtruth folder contains one .png file per frame.
- The name of a file is gt, the frame number encoded with 6 digits, and .png.
Each pixel of a ground-truth (GT) .png file has a value among:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Background</td>
</tr>
<tr>
<td>50</td>
<td>Shadow</td>
</tr>
<tr>
<td>85</td>
<td>Outside the ROI (pixel ignored during the evaluation)</td>
</tr>
<tr>
<td>170</td>
<td>Impossible to determine (pixel ignored during the evaluation)</td>
</tr>
<tr>
<td>255</td>
<td>Foreground</td>
</tr>
</tbody>
</table>

Note: A frame outside the evaluation interval has a GT full of 85 values.
Some tools to compute metrics (e.g. F1) are given along with the dataset.

- There are Matlab and Python versions.
- In this tutorial, we will show how to use the Matlab version.
Small problem with GNU/Linux

- To work on GNU/Linux, the CDnet evaluation tool requires some modifications.
  - In `processVideoFolder.m` line 35:
    ```
fID = fopen([path, '\temporalROI.txt']); % Before
fID = fopen([path, '/temporalROI.txt']); % After
    ```
  - In `Stats.m` line 45:
    ```
% Before
f = fopen([this.path '\category \video \cm.txt'], 'wt');
% After
f = fopen([this.path '/category '/video '/cm.txt'], 'wt');
    ```
  - In `Stats.m` line 52:
    ```
f = fopen([this.path '\category \cm.txt'], 'wt'); % Before
f = fopen([this.path '/category '/cm.txt'], 'wt'); % After
    ```
  - In `Stats.m` line 76:
    ```
f = fopen([this.path '\cm.txt'], 'wt'); % Before
f = fopen([this.path '/cm.txt'], 'wt'); % After
    ```
BGSLibrary [3] [1]
- Open-source (GPL 3) C++ library full of BGS algorithms.
- Based upon OpenCV.
- Maintained by Andrews Sobral.
- Numerous algorithms have been implemented by the authors!
- Provides also: GUI; wrappers for Java, Python, and Matlab; Docker images; etc.
- Everyone is free to send its algorithm (as long as a reference is associated).
- For any support related to the BGSLibrary, please contact Andrews Sobral.
BGSLibrary

A Background Subtraction Library

Release v2.0.0  License GPL v3  Platform Windows, Linux, OS X  OpenCV 2.x, 3.x  Wrapper Java, Python, MATLAB  Algorithms 43

Page Update: 01/04/2017

Library Version: 2.0.0 (see Build Status and Release Notes for more info)

The BGSLibrary was developed by Andrews Sobral and provides an easy-to-use C++ framework based on OpenCV to perform foreground-background separation in videos. The bgslibrary is compatible with OpenCV 2.x and 3.x, and compiles under Windows, Linux, and Mac OS X. Currently the library contains 43 algorithms. The source code is available under GNU GPLv3 license, the library is free and open source for academic purposes.

- List of available algorithms
- Algorithms benchmark
**A background subtraction library**  
https://github.com/andrewssobral/bgslib

<table>
<thead>
<tr>
<th>Branch: master</th>
<th>New pull request</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>172</strong> commits</td>
<td><strong>1</strong> branch</td>
</tr>
<tr>
<td><strong>9</strong> contributors</td>
<td></td>
</tr>
</tbody>
</table>

- **172** commits
- **1** branch
- **12** releases
- **9** contributors

### Clone with HTTPS

Use Git or checkout with SVN using the web URL.

https://github.com/andrewssobral/bgslib

**Download ZIP**
Once uncompressed, you have all the files to compile the library and its tools.

- There are Java and QT GUIs, but...
- ...in this tutorial, we will focus on the inclusion of the library in your own program.
- This requires to copy in your project **package_analysis** and **package_bgs**.
- The folder **package_bgs** contains the implementations of the BGS algorithms.
The wiki on the website documents every aspect of the library.

You can find the list of available BGS algorithms.
C++ Programming
Step 1: C++ Project with the BGSLibrary
Create a folder for your project and put inside:

- An empty **build** folder (location of the compiled files).
- An empty **build/config** (location of the BGSLibrary config files).
- A copy of the **package_analysis** folder of the BGSLibrary.
- A copy of the **package_bgs** folder of the BGSLibrary.
- An empty **CMakeLists.txt** file.
cmake_minimum_required(VERSION 2.8)

# Project name
project(cdnet-bgs)

# Enable C++11 support of the compiler
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")

# Find OpenCV
find_package(OpenCV REQUIRED)

# Add include directories of OpenCV
include_directories(SYSTEM ${OpenCV_INCLUDE_DIRS})

# Find all C++ and C implementation files in package_bgs
file(GLOB_RECURSE bgslibrary_src
    package_bgs/*.cpp
    package_bgs/*.cc
    package_bgs/*.c)

# Declare the BGSLibrary with the files being found
add_library(bgslibrary STATIC ${bgslibrary_src})

# Link the BGSLibrary with OpenCV
target_link_libraries(bgslibrary ${OpenCV_LIBS})
The empty project can be compiled using the following commands:

- $ cd build
- $ cmake -DCMAKE_BUILD_TYPE=Release ..
- $ make
Once compiled, the BGSLibrary is a static library `libbgslibrary.a` in `build`.

Thus, we have a project enabling to create a program with the BGSLibrary.
Step 2: List of Sequences
Procedure

- For applying BGS on the CDnet dataset, we need a list of sequences.
- We can add to our project a `cdnet.txt` file containing such a list.
- Thus, we start our program with a function to read `cdnet.txt`.
- The code of the program will be put in a file called `bgs-subtractor.cpp`.
badWeather/blizzard
badWeather/skating
badWeather/snowFall
badWeather/wetSnow
baseline/highway
baseline/office
baseline/pedestrians
baseline/PETS2006
cameraJitter/badminton
cameraJitter/boulevard
cameraJitter/sidewalk
cameraJitter/traffic
dynamicBackground/boats
dynamicBackground/canoe
dynamicBackground/fall
dynamicBackground/fountain01
dynamicBackground/fountain02
dynamicBackground/overpass
intermittentObjectMotion/abandonedBox
intermittentObjectMotion/parking
intermittentObjectMotion/sofa
intermittentObjectMotion/streetLight
intermittentObjectMotion/tramstop
... 
turbulence/turbulence3
We need to add some lines at the end of *CMakeLists.txt* to compile our program!

```cmake
# Produce the executable bgs-subtractor from the C++ code
add_executable(bgs-subtractor bgs-subtractor.cpp)

# Link bgs-subtractor to the BGSLibrary and OpenCV
target_link_libraries(bgs-subtractor bgslibrary ${OpenCV_LIBS})
```
Function `list_seqs()` to read cdnet.txt in bgs-subtractor.cpp

```c++
#include <cstddef>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// Path to the cdnet.txt file
#define SEQS_PATH "~/home/user/Bureau/cdnet-bgs/cdnet.txt"

// Function to read the cdnet.txt file
vector<string> list_seqs() {
    vector<string> seqs; // Vector containing the seq. names
    ifstream ifs(SEQS_PATH); // Stream to read the cdnet.txt file
    string buffer; // String buffer

    // For each line in the cdnet.txt file
    while (getline(ifs, buffer))
        // Add the current sequence name in the vector
        seqs.push_back(buffer);

    // Return the vector of sequence names
    return seqs;
}
```
int main() {
    // Vector with the sequence names
    vector<string> seqs = list_seqs();

    // For each sequence
    for (size_t seq_idx = 0; seq_idx < seqs.size(); ++seq_idx) {
        // Print the current sequence name
        cout << seqs[seq_idx] << endl;
    }
}
You can compile the code as in the slide 20.

The program `bgs-subtractor` is in the build folder.

You can launch it with the command `./bgs-subtractor`.

If everything is correct, the list of CDnet sequences should be printed.
Step 3: Reading a Temporal ROI
In CDnet, each sequence is provided with a `temporalROI.txt` file.

In contains two integers separated by a space on a unique line.

The first is the frame number beginning the *evaluation period*.

The second is the frame number ending the *evaluation period*.

Our program will save the segmentation maps computed during this period.

Note that the period before the evaluation period is the *training period*. 
Add the path to the CDnet dataset.

```cpp
#define CDNET_PATH "/home/user/Bureau/dataset2014/dataset"
```

Add a function to read the temporal ROI given a sequence name `seq`.

```cpp
vector<int> read_temporal_roi(string seq) {
    /* The temporal ROI is a vector of two integers:
    *   - The first is the frame number beginning the evaluation
    *   - The second is the frame number ending the evaluation
    */
    vector<int> temporal_roi;

    // Stream to read temporalROI.txt
    ifstream ifs(string(CDNET_PATH) + "/" + seq + "/temporalROI.txt);
    int frame; // Integer buffer

    ifs >> frame; // Read the first integer
    temporal_roi.push_back(frame); // Add it into the vector
    ifs >> frame; // Read the second integer
    temporal_roi.push_back(frame); // Add it into the vector

    // Return the temporal ROI vector
    return temporal_roi;
}
```
For this purpose, we can add the following code in the loop iterating the sequences:

```cpp
... int main() {
  ...
  // For each sequence
  for (size_t seq_idx = 0; seq_idx < seqs.size(); ++seq_idx) {
    ...
    // Read the temporal ROI of the current sequence
    vector<int> temporal_roi = read_temporal_roi(seqs[seq_idx]);
    // Put the first frame number in a variable frame_begin
    int frame_begin = temporal_roi[0];
    // Put the second frame number in a variable frame_end
    int frame_end = temporal_roi[1];
    // Print frame_begin and frame_end
    cout << frame_begin << " - " << frame_end << endl;
  }
}
```
You can compile (resp. launch) the code as in the slide 20 (resp. 27).

If everything is correct, the temporal ROI of each sequence is printed.
Step 4: Reading the Frames of a Sequence
In the sequence folder, we need to read each image file.

To each image file corresponds a frame.

For each frame, the image file name has to be formatted correctly.

We can read the image file using OpenCV.
Modification of `bgs-subtractor.cpp`

- Add the following includes.

```cpp
... 
#include <iomanip>
#include <sstream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
```

- Use the OpenCV namespace.

```cpp
... 
using namespace cv;
```
Code to read and display the frames in `bgs-subtractor.cpp`

```cpp
... int main() {
... // For each sequence
    for (size_t seq_idx = 0; seq_idx < seqs.size(); ++seq_idx) {
... // For each frame
        for (int f_num = 1; f_num <= frame_end; ++f_num) {
            // Stream to format the image file name
            stringstream frame_path;
            // Path to the image file corresponding to the frame
            frame_path << CDNET_PATH << '/' << seqs[seq_idx] << '/input/in' << setw(6) << setfill('0') << f_num << '.jpg';

            // Ask OpenCV to read the image file and put it in a Mat
            Mat frame = imread(frame_path.str());
            // Put the input frame in a graphical window
            imshow("Input frame", frame);
            // Display the graphical window
            waitKey(1);
        }
    }
}
```
You can compile (resp. launch) the code as in the slide 20 (resp. 27).

If everything is correct, a window displaying the current sequence appears.
Step 5: Applying a BGS Algorithm
We want to instantiate a given BGS algorithm in the BGSLibrary.

Apply it on each frame of each CDnet sequence.

By default, the BGSLibrary displays the results in a graphical window.
Include the BGSLibrary.

```cpp
...  
#include "package_bgs/bgslibrary.h"
```
... int main() {
...
// For each sequence
for (size_t seq_idx = 0; seq_idx < seqs.size(); ++seq_idx) {
...
// Instantiate a BGS algorithm (the frame difference here)
IBGS* subtractor = new FrameDifference;

// For each frame
for (int f_num = 1; f_num <= frame_end; ++f_num) {
...
// Instantiate an empty segmentation map
Mat seg_map(frame.rows, frame.cols, CV_8UC3);
// Instantiate an empty background model
Mat bg_model(frame.rows, frame.cols, CV_8UC3);

// Apply BGS algorithm on the current frame
subtractor->process(frame, seg_map, bg_model);
}
}

// Delete the instantiated BGS algorithm
delete subtractor;
You can compile (resp. launch) the code as in the slide 20 (resp. 27).

If everything is correct, a window displaying the segmentation maps appears.
A New File Appeared!

- We used the frame difference algorithm in our code.

- A file `FrameDifference.xml` appeared in the `config` folder.
An XML file is automatically created the first time a BGS algorithm is used.

This file enables to tune the parameters of the frame difference.

This tuning must be done before launching our `bgs-subtractor`.

For instance, the threshold can be modified by tuning the value surrounded by the `<threshold>` tag (here, the value is 15).
Using Another BGS Algorithm is Easy!

- To use another BGS algorithm, we must change a unique line:

```c
IBGS* subtractor = new FrameDifference;
```

- For instance, to use $\Sigma - \Delta$, we can modify this line as follows:

```c
IBGS* subtractor = new SigmaDelta;
```

- You can find the available algorithms in the file `package_bgs/bgslibrary.h`. 
Step 6: Saving the Segmentation Maps
For assessing a BGS algorithm, we must save the results.

Specifically, we must save the maps produced during a temporal ROI.

They can be saved in the `results` folder of the CDnet dataset.

It contains empty category and sequence folders.

The name of a map is `bin`, the frame number encoded with 6 digits, and `.png`.
Add the path to the CDnet results.

```cpp
...  
#define RESULTS_PATH "/home/user/Bureau/dataset2014/results"
```
int main() {
...
// For each sequence
for (size_t seq_idx = 0; seq_idx < seqs.size(); ++seq_idx) {
...
// For each frame
for (int f_num = 1; f_num <= frame_end; ++f_num) {
...
// If we are in the evaluation period
if (f_num >= frame_begin) {
// Stream to format the output image file name
stringstream write_path;
/* Path to the output image file containing the
 * current segmentation map
 */
write_path << RESULTS_PATH << "/" << seqs[seq_idx] 
  << "/bin" << setw(6) << setfill(’0’) 
  << f_num << ".png";
// Ask OpenCV to write the segmentation map in the file
imwrite(write_path.str(), seg_map);
}
}
}
You can compile (resp. launch) the code as in the slide 20 (resp. 27).

If everything is correct, the maps should be saved in the results folder.
Performance Evaluation
What’s Next?

- We want to assess a given BGS algorithm on the CDnet dataset.

- We compare the resulting segmentation maps with the ground-truth.

- The result of such a comparison is expressed by metrics/scores.

- An evaluation tool computing those metrics is given with CDnet (see slide 11).

- We will see how to use the Matlab version of this tool.

- Feel free to use the Python version if it is more convenient to you!
Procedure

- In Matlab, use the function `processFolder()`.  
- The first parameter is the path to the CDnet dataset. 
- The second parameter is the path to the results to assess.
When `processFolder()` is over, a `cm.txt` file is generated.

- The file is located in the dataset folder.
- For each category, it contains the scores averaged over sequences.
- It contains also the scores averaged over all CDnet sequences.
- For instance, for the F. Diff., F1 is $\approx 0.22$ on PTZ, and $\approx 0.23$ on the dataset.
If your results are convincing, you can upload them on the CDnet website. This enables to discover your position in the ranking. Moreover, the website performs a deeper evaluation. More ground-truth is available internally on the server (for avoiding cheating). Your results are even more convincing? Let's publish your paper!
Integrate Your Own Algorithm
Until now, we saw how to apply a BGS algorithm from the BGSLibrary on CDnet.

Also, we saw how to assess quantitatively the results.

New question: Is it possible to do those operations with your own algorithm?

Answer: Yes!

The solution is to integrate your algorithm to the BGSLibrary.

Consists in creating a class (e.g. MyAlgo) inheriting from the class IBGS.

MyAlgo must be in the namespace bgslibrary::algorithms.

In MyAlgo, you must override the relevant methods of IBGS.

You can start this work by creating a header and implementation file (MyAlgo.h and MyAlgo.cpp) in package_bgs.
In your class inheriting from IBGS, you must override 3 methods:

```cpp
void process(const cv::Mat& img_input,
             cv::Mat& img_output,
             cv::Mat& img_bgmodel);

void saveConfig();
void loadConfig();
```

- **process()** applies the algorithm on the current frame. The parameters are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img_input</code></td>
<td>The current frame (input).</td>
</tr>
<tr>
<td><code>img_output</code></td>
<td>The resulting segmentation map (output).</td>
</tr>
<tr>
<td><code>img_bgmodel</code></td>
<td>An image representing the current background model (output).</td>
</tr>
</tbody>
</table>

- **saveConfig()** saves the parameter values in the algorithm XML config file.
- **loadConfig()** loads the parameter values in the algorithm XML config file.
- Let's analyze the code of the frame difference!
#pragma once
#include "IBGS.h"

namespace bgslibrary {
    namespace algorithms {
        // Inherits from IBGS
        class FrameDifference : public IBGS {
            private:
                bool enableThreshold; // Parameter 1  
                int threshold; // Parameter 2

            public:
                FrameDifference(); // Constructor
                ~FrameDifference(); // Destructor
                void process(const cv::Mat &img_input, // Method 1
                              cv::Mat &img_output,
                              cv::Mat &img_bgmodel);

            private:
                void saveConfig(); // Method 2
                void loadConfig(); // Method 3
        };
    }
}
#include "FrameDifference.h"

using namespace bgslibrary::algorithms;

// Constructor
FrameDifference::FrameDifference() :
// Parameters default values
enableThreshold(true), threshold(15) {
    // Display the name of the instantiated algorithm
    std::cout << "FrameDifference()" << std::endl;
    /* Call the setup function of the BGSLibrary to initialize
     * the algorithm XML config file
     */
    setup("./config/FrameDifference.xml");
}

// Destructor
FrameDifference::~FrameDifference() {
    // Display the name of the destroyed algorithm
    std::cout << "~FrameDifference()" << std::endl;
    // Nothing to destroy for this algorithm
}
void FrameDifference::process(const cv::Mat &img_input, 
    cv::Mat &img_output, 
    cv::Mat &img_bgmodel) {
    /* Call the initialization function of the BGSLibrary to 
     * allocate memory related to img_output and img_bgmodel 
     * whether they are empty. 
     */
    init(img_input, img_output, img_bgmodel);

    // If internal background model is empty (first frame)
    if (img_background.empty()) {
        // Copy the first frame as background model
        img_input.copyTo(img_background);
        // Stop (we cannot detect motion from a unique frame)
        return;
    }

    // Absolute difference between model and current frame
    cv::absdiff(img_background, img_input, img_foreground);

    // If input frame is RGB
    if (img_foreground.channels() == 3)
        // Convert it to grayscale
        cv::cvtColor(img_foreground, img_foreground, CV_BGR2GRAY);
    ...
}
... // If threshold operation is required (yes by default)
if (enableThreshold)
  /* Apply threshold on input frame and save it as the
     * internal segmentation map
     */
  cv::threshold(img_foreground, img_foreground, threshold, 255, cv::THRESH_BINARY);

// Code to show the segmentation map in a graphical window
#ifndef MEX_COMPILE_FLAG
  if (showOutput)
    // Give the name of your algorithm to the graphical window
    cv::imshow("Frame Difference", img_foreground);
#endif

// Copy the internal segmentation map to the output one
img_foreground.copyTo(img_output);
// Copy the input frame as the internal background model
img_input.copyTo(img_background);
// Copy the internal background model as the output one
img_background.copyTo(img_bgmodel);
// The first frame has been processed
firstTime = false;
}
```cpp
void FrameDifference::saveConfig() {
    // Ask OpenCV to open the XML file to write
    CvFileStorage* fs = cvOpenFileStorage(config_xml.c_str(), nullptr, CV_STORAGE_WRITE);

    // Write enableThreshold parameter value as an integer
    cvWriteInt(fs, "enableThreshold", enableThreshold);
    // Write threshold parameter value as an integer
    cvWriteInt(fs, "threshold", threshold);
    // Write showOutput parameter value as an integer
    cvWriteInt(fs, "showOutput", showOutput);

    // Writing is over (closing)
    cvReleaseFileStorage(&fs);
}
```

Note that OpenCV limits the parameter types that can be written. You can use:

- `cvWriteInt` to write an integer parameter.
- `cvWriteReal` to write a floating-point parameter.
- `cvWriteString` to write a string parameter.
```cpp
void FrameDifference::loadConfig() {
    // Ask OpenCV to open the XML file to read
    CvFileStorage* fs = cvOpenFileStorage(config_xml.c_str(), nullptr, CV_STORAGE_READ);

    // Read enableThreshold as an integer (true if not defined)
    enableThreshold = cvReadIntByName(fs, nullptr, "enableThreshold", true);

    // Read threshold as an integer (15 if not defined)
    threshold = cvReadIntByName(fs, nullptr, "threshold", 15);

    // Read showOutput as an integer (true if not defined)
    showOutput = cvReadIntByName(fs, nullptr, "showOutput", true);

    // Reading is over (closing)
    cvReleaseFileStorage(&fs);
}
```

Note that, once again OpenCV limits the parameter types that can be read to integers, floating-points, and strings.
Do not forget to add your algorithm into `bgslibrary.h`.

You can send it to Andrews Sobral to be integrated in the official BGS Library.

You are now able to:

- Integrate your own BGS algorithm in the BGS Library.
- Apply it on the CDnet dataset.
- Assess it with metrics/scores.
thank you
References
References


