Homework. 9: Bag of Visual Words Histograms

Ignacio Vizzo, E-Mail ivizzo@uni-bonn.de

Handout : 22.06.2020
Handin: 03.07.2020 at 23:59:59 (CET)

General rules

1. You need to provide the build system for this homework. This means, you need to provide as many **CMakeLists.txt** files as you think is needed.
2. Your code will be checked using `clang-format`. If your code in not properly formatted, then the tests will not run.
3. Your code will be static-analyzed by `clang-tidy`, if your code does not comply with the analysis, the tests will not run. If you are using VSCode with the reccomended setup, most of the errors will be visible while you program.
4. **ALL** tasks should be solved within the `homework_9` folder, no need to create `task_x` folders.
5. The design of how to solve this exercise is up to you. The **ONLY** requirement to pass the tests are:
   - You should provide one header file (put it where you preffer) and call it `homework_9.h`.
   - All the functions of this exercise must be accessible form this header file
   - You should guarantee this header file can be included thorugh your build system.
6. Do not add binary files or images to your repository. The only images you are allowed to submit is the ones that comes with this homework. If any **binary** file or image file(**.png**) is found on your submission, then your homework will be completely discarded.

A Bag of Visual Words Dictionary Refactoring (2 points)

Writing the **ipb::BowDictionary** class was a ton of fun, but if you look close, there is too much borderplate code in the implementation, and the "type" does not represent close enough the **concept** of Dictionary(for example, a dictionary **does not have** "a" maximun number of iterations, that's a property of the clustering algorithm).

In this task you will simplify a bit your Dictionary implementation, making it easier to read and mantain. If you wish to carry on with previous functionality, you could do it for your final project, but for this exercise, let's refactor the class:

- Remove **ALL** data members of the **ipb::BowDictionary** class except for the vocabulary.
- Remove **ALL** setters/getters from the class, except for the vocabulary ones.
- In case you didn't have, add one **set_vocabulary()** method. This will be handy in case you have a pre-computed dictionary and you want to instansiate it as an object of the class.
- Keep the singletton pattern untouched.
- Rename the **set_params()** function for **build()**. But don't change the implementation.
- Keep the **size()** and **empty()**.

A.1 Requirements to pass the tests

- Provide a library called "bow_dictionary" that implements the class.
- Your implementation **must** be refactored, this means, you should delete all the old stuff from homework_7.

A.2 *Tips*

1. You can use code from previous homeworks.
2. You can add additional data members to your class implementation if you consider it necessary.

B Bag of Visual Words Histograms (8 points)

One key component in a Bag of Visual Words pipelines are the histograms. As the C++ creator(Bjarne Stroustrup) says: "A program that provides types that closely match the concepts of the application tends to be easier to **understand**, easier to **reason** about, and easier to **modify** than a program that does not."

In this exercise you will be following this approach and implementing your own **ipb::Histogram** type. Keep in mind than a histogram might be easily represented as a **std::vector** and this might look like an overkill. But following C++ best practices we are trying to make our code easier to reason, to mantain, and to use.A draft of the type of idea that you will be representing with your histogram type is depicted in Figure 1:

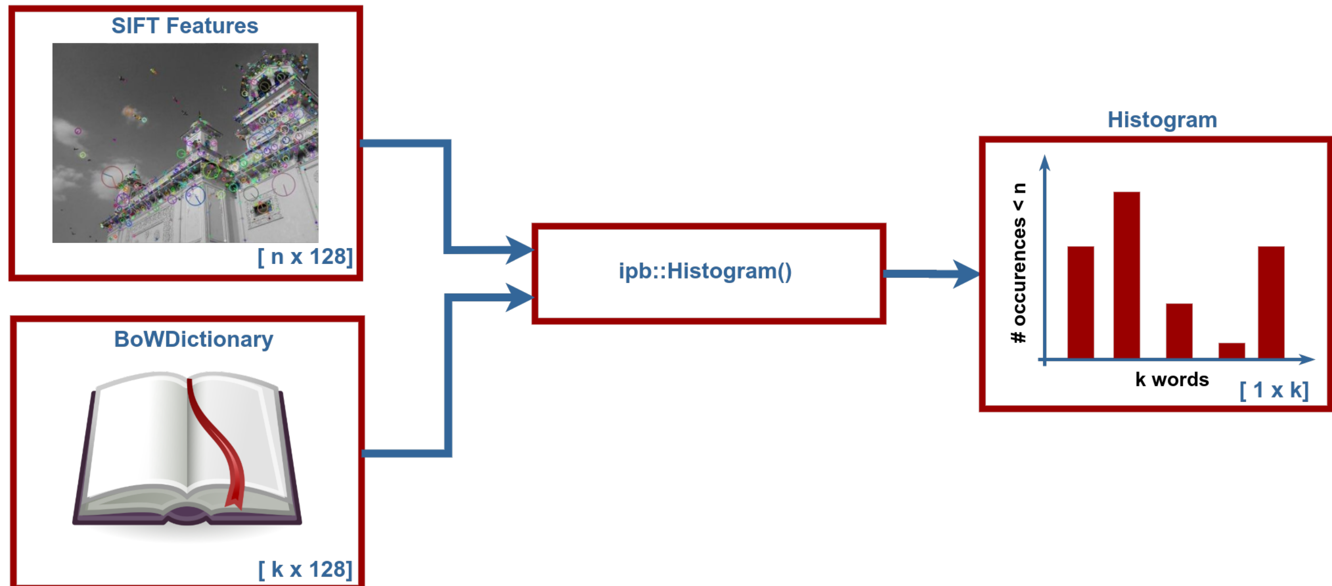

Figure 1: Histogram class representation. **NOTE:** Pay attention to the dimensions of the input data and the histogram properties.

A **pseudo-code** of the type interface is the following:(Note that the interface is stripped from any type, const qualifer, arguments, etc).

```
class Histogram {
  Histogram();
  Histogram(data):
  Histogram(descriptors, dictionary);

  operator<<();
  WriteToCSV();
  Histogram ReadFromCSV();

  // Imitate stl_vector functionality
  operator[]();
  data():
  size();
  empty();
  begin();
  cbegin();
  end();
  cend();

  // data members:
  std::vector<int> data_;
};
```

B.1 Requirements to pass the tests

- Create a new type called **Histogram** under the **ipb** namespace.
- Implement the 3 constructors depicted in the type interface above.
- Add "printable" functionality to your type, one should be able to print the content of the histogram to the standard output with no pain(overload the « operator).
- Add two handy methods to read and write histograms to csv files. Note that this could be really usefull for debugging your implementation, dumping the content of the histogram to the disk and inspecting the data with MATLAB/Python.
- Implement the access operator([]).
- Implement STL vector-like functionality, this will give you access for free to all the STL algorithm library.
- Provide a library called "histogram" that implements this class.

B.2 *Tips*

1. If your think is necessary you could use OpenCV::flann to speedup your histogram computations.
2. If you do so, keep in mind that flann build a **KDTree** of the input dataset, this is typically time costly. It's important to keep in mind that if you build the same tree for the same type of input data multiple times, this **will** hurt performance. On a 5-image dataset you won't probably notice any difference, but if you carry this to your final project it will impact performance.
3. You can add as much functionality as you consider necessary to this new type. At the end, is the type that you will most likely be using in your final project implementation.