Image Retrieval using Visual Dictionaries

Nikolaos Papathanasiou, Thomas Halvey
Einsteinweg 55, Leiden, Leiden University, NL
halvey@mirsociety.org

1 Abstract

In this report an implementation of a Content-based Image Retrieval System(CIRS) [1] [3]. in C++ is being presented that takes as a query part of an image from the original data set and produces a ranked list of the most similar photos. The dataset used for this implementation is MICROSOFT COCO train set 2014 consisting of 80.000 images and 80 categories. In order to implement this idea some tools where used from the OpenCV library. The feature detection and extraction is made with SURF detector/descriptor and K-means method is used for clustering these descriptors(visual words) into documents creating a visual dictionary [2].

2 Introduction

During the last years many large scale Content-based Image Retrieval Systems [2] rely on the Bag of (Visual) Words model [1] [6] [7] BoW, which allows to follow an approach similar to the well-know text retrieval technique by using salient descriptors (e.g. [4], [5]). This approach provides some interesting results for CIRSs especially when time is not an obstacle and there is enough RAM to find the optimum clustering between a large number of descriptors (several millions). OpenCV library provides computer vision algorithms to researchers, includes some useful automated techniques for this approach and therefore is used for the implementation of this sub-image search engine. In general the scope of this report is to provide the methodology of creating a CIR system that works and can be easily adjusted with the help of OpenCV library.

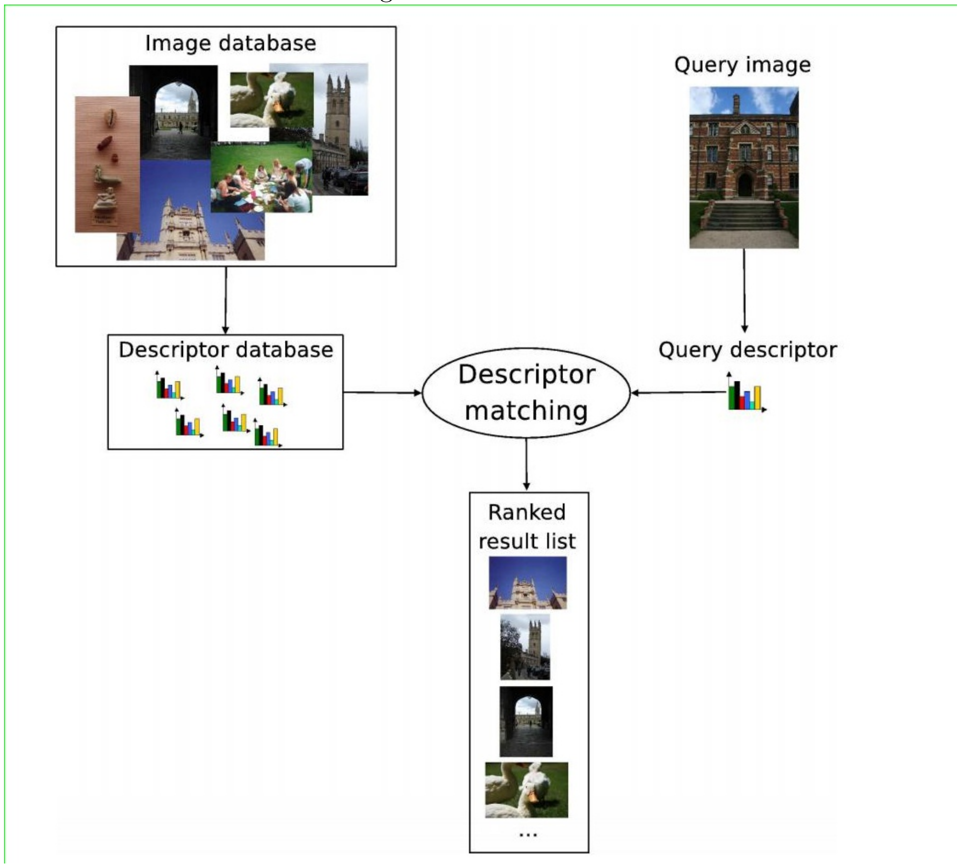
3 Dataset: Microsoft COCO

Microsoft COCO is an image recognition database with 300.000 photos in total, 80 object categories and provides also annotated segmentations from the images with 5 captions per image on average. It is used for image recognition and more specifically for recognition in context. The choice of this dataset was made based on two aspects, first it helps a lot to have nice and organized ground truth in order to be able to evaluate the performance of the feature classification (Bag of Words) and second because as the images contain multiple objects, thus there is a lot of noise in the background, and having many different categories helps to compare the results of this image retrieval system with real world problems. Finally it was observed during the first testing with the dataset there are some cases of segmentation examples that were mislabeled thus this can provide extra noise from the standard 10 categories that were chosen.

4 Implementation Details

The basic architecture in a CIRS as suggested in [1] is shown in the following figure:

Figure 1: architecture



This implementation is using several OpenCV structures and methods and the source code consists 4 mains parts:

1. Preprocessing/Sampling of the images (segmentations).
2. Creating the dictionary (BoW).
3. Indexing of dictionary Histograms.
4. Query an image and creation of the ranked list.

4.1 System Requirments

All experiments and runtimes were recorded a in desktop pc with Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz and 8 GB RAM. Only the creation of the dictionary run at titan liacs computer with 64GB RAM. Thus the system requirement for this implementation to work is pretty basic except of the BoW dictionary which can take up to 15-16 GB for 24 thousand images (10 million descriptors).

4.2 OpenCV Libraries/ C++11

In order to keep things relatively simple only OpenCV was used as an external library and some C++11 functions were used that provided some nice and fast solutions. The OpenCV library does not have to be installed in order for the program to compile and run as all necessary files can be found in the /home/nikolaos/mir2015 folder in titan liacs computer and the only thing to do is run make to produces the executables. The choice of OpenCV library was made because it provides very helpful and automated tools that can helped to work with this implementation fast and efficiently. The most important tools that were used are:

- matrix container cv::Mat is used in order to store images, histograms, dictionary etc.
- cv::Filestorage for storing the same things in yaml files.
- cv::SURF implementation of Surf detector/descriptor.
- cv::BOWKmeansTrainer for clustering (K-means) and cv::bowImgDescriptorExtractor that produces a normalized histogram of occurrences.
- cv::FlannBasedMatcher for the creation of the BoW Histogram (Fast Approximate Nearest Neighbors matching of SURF descriptors).
- image read/write with cv::imread and cv::imwrite respectively.
- cv::imshow for showing the image query and cvSetMouseCallback in order to crop the query image and feed it to the program.
- cv::compareHist() was used to compare BoW histograms.

4.3 Preprocessing

From the 80 categories 10 categories have been chosen: *TV, oven, surfboard, elephant STOP sign, car, train, dog, skateboard, pizza.* The choice of these categories was at random and this list was kept as there are some harder items included, like the skateboard/surfboard has usually noise from human legs thus similar segmentations and they can be confused easily, TV from this dataset can be any screen(laptop or tv) which usually is turned on containing some captured moment from the display. Also there are mislabeled examples of cat as a dog or there are objects of very small scale in the background of some images (for example cars in the background)

The COCO dataset provides a *.json* file containing the annotations of the images. A python script was written in order to retrieve the images that belonged to this dataset and take their segmentations to create a sample of the original dataset where in this sample each image is a segmentation from the original image. This was implemented with the help of *cv2.fillPoly* OpenCV function for python and is a little bit time consuming. These categories resulted to 24.872 unique photos from which 51.147 segmentation's photos where grabbed. The reason behind the segmentation grabbing of the sample is that this way each group of descriptors give a more clear view of what one object is instead of several inside a whole image. No other preprocessing took place in the final implementation.

4.4 Bag of Visual Words

The main idea is to first quantize image descriptors, that describe local areas around the keypoints that were detected in the image, into visual words. In order to do this, as describing keypoints can result to a lot of descriptors from each image, the use of BoW model is suggested by Josef Sivic and Andrew Zisserman [6]. SURF was chosen as the method of detecting/describing keypoints and image areas respectively, as it is faster than the well known SIFT and also stores vectors of length, $l = 64$, instead of 128 that SIFT produces [4]. For the vocabulary generation a basic K-means clustering is used that is implemented from OpenCV library and provides the functionality to add descriptors of type cv::Mat to a bowTrainer class and cluster automatically into the number that is given. The choice of the number of clusters is 10.000 and the runtime was 8 hours for both the collection of the descriptors and the clustering.

4.5 Indexing

For the indexing implementation the scheme of the architecture shown in Figure.1 was followed:
First the BoW histograms were extracted from each image(segmentation) from the 51.147 sample on the SURF keypoints that were detected, where the BoW histograms are represented in cv::Mat by a vector of length, $l = \#cluster$, containing the normalized occurrence value of each given image group of descriptors for all clusters (documents). Then the algorithm creates a folder named respectively for each of the 24.872 unique whole images, which contain the histogram (in a yaml file) that belongs to each segmented image that belongs to that unique whole image. Many schemes are suggesting considering tf_idf (term frequency.inverse document frequency) value but as Pierre Tirilly , Vincent Claveau , Patrick Gros suggested in [1] and as some naive experiments also has proven there is no significant change in the performance especially when using classical td_idf.

4.6 Query/Ranked List

Figure 2: sub image query functionality 1

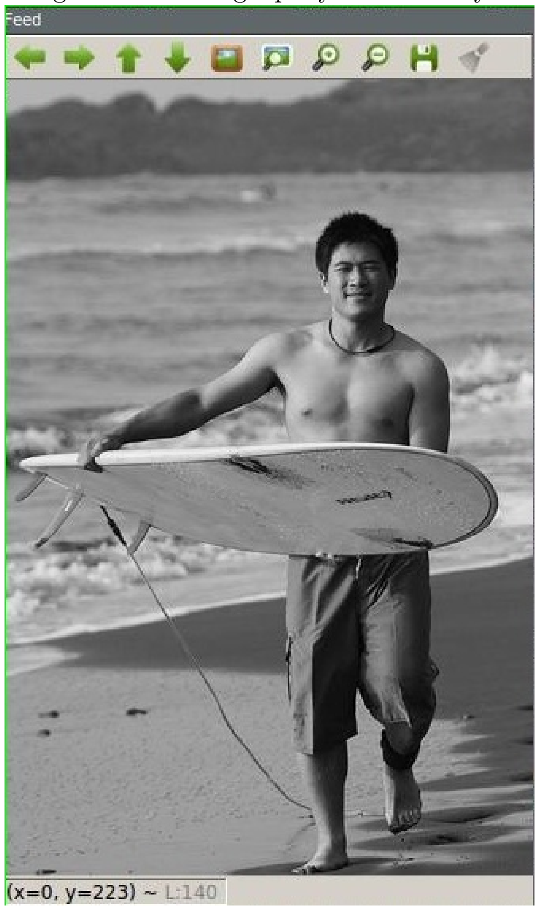
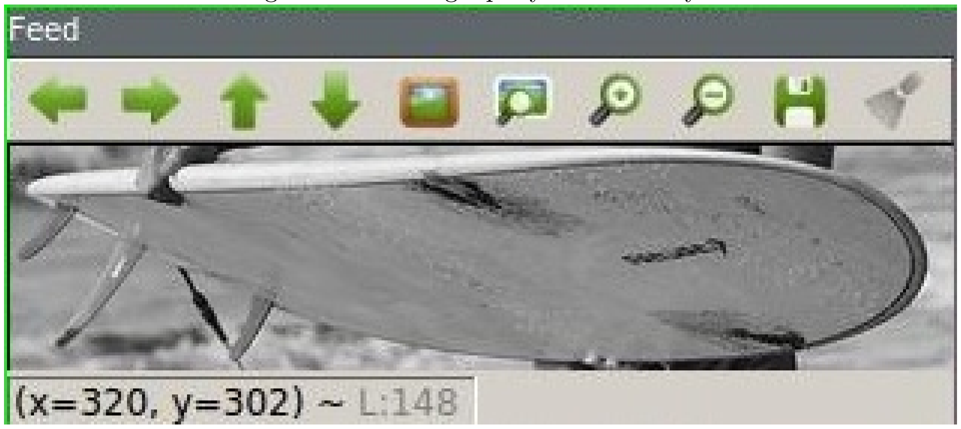


Figure 3: sub image query functionality 2



Considering the query implementation, as the scope of this project was to implement a sub-image search engine/image retrieval technique a function was implemented with the help of OpenCV Lib in order to crop the image using the mouse to draw a rectangle of the preferred region of interest. After the choice of the region is done, the cropped image will be shown where if the user presses any button the query will be fed to the algorithm. After the cropped query is fed to the algorithm it follows the usual procedure of detecting keypoints and producing a BoW histogram which histogram is compared with each indexed histogram with the OpenCV implemented function cv::compareHist() with Pearson Correlation metric (Figure.2) that produces -1 for min similarity and 1 for max similarity. The image names that are selected for the ranked list have a correlation threshold, $t = 0.11$, where $correlation_{value} > t$, and in the ranked list folder the first 20 images of the sorted (descend) ranked List are written. The reason behind the limit of 20 images in the ranked List is that there are some cases like STOP sign or pizza where there are a lot of well shown examples and the algorithm finds all the images that contain these objects although this choice can be easily adjusted. Usually one query takes 30 seconds.

Figure 4: OpenCV correlation

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}}$$

$$\bar{H_k} = \frac{1}{N} \sum_J H_k(J)$$

4.7 Compilation and Run Guide

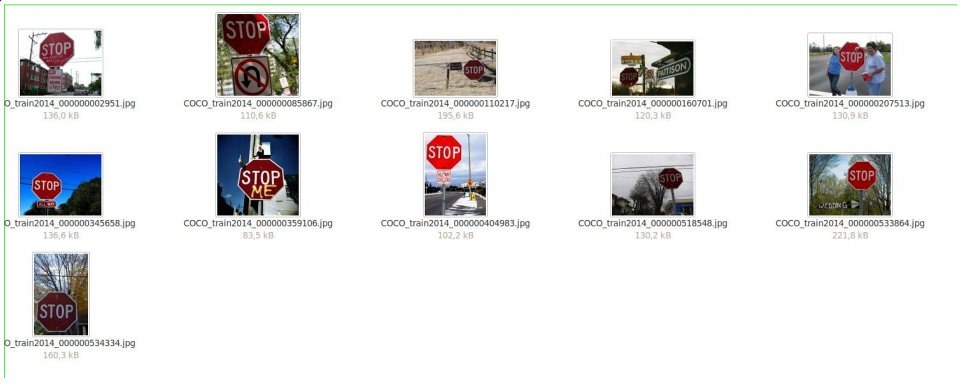
In order for the system to work we need the sample of the segmented photos(sampleLast) and the sample of the whole photos(samplewhole). Different categories can also be chosen to create a different sampling than the one that is already made. In order to do that just run the python 2 script:

python preproccess.py.
It has to be noted that this could take some time and also there is a link provided in the README in order to download a zip file containing these necessary folders in case something goes wrong. For the compilation of the program in C++ nothing is needed to be done just navigate to the main folder (FinalProject) and run "make". Then three executable should be build named: "createBow", "createIndex10k" and "query". What is needed for the system to work is only the query which can be called by:
./query image-name sampleLast/
and produces the results (ranked list) in a folder call rankedList. If one wishes to create a new dictionary with createBow needs also the sampleLast/ folder to use it as:
./createBow sampleLast/
which will produce a yml file called dictionary10k.yml and finally in order to recreate the index one can run:
./createIndex10k sampleLast/
which will produce an index folder named bIndexgg/.

5 Experiments

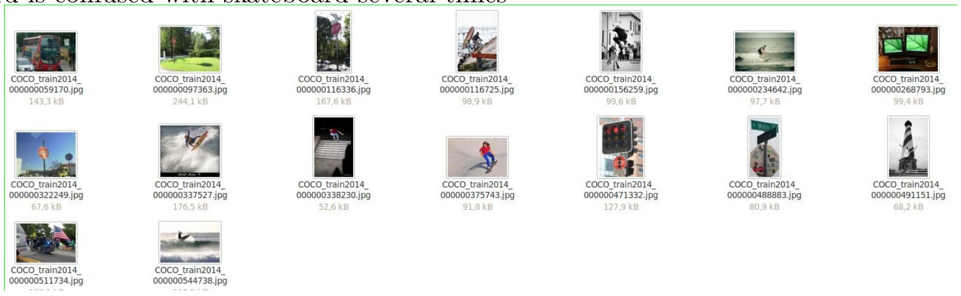
During the research on how to implement this CIRS two main parts were the most time consuming choosing the database and process it towards the needs of the scope of the project and K-means clustering of the millions of descriptors from thousands of images. Firstly for testing if the bag of words approach of OpenCV works as expected an experiment run for a small dataset 4 categories 300 images that can be found here in this tutorial *http://ttic.uchicago.edu/ mostajabi/Tutorial.html* and tested the OpenCV BoW implementation with 1500 centers in a multiclass Support Vector Machine(SVM) implementation of the same library in the test set provided by the tutorial which resulted to an accuracy of 87%. After confirming that BoW worked the experiments were focused on the segmentations of the dataset, the number of descriptors and the number of clusters. The number of descriptors was mentioned because what is needed for the creation of an efficient dictionary is to get as much good information from a picture as possible as each image's group of descriptors that is fed to the bowTrained class is one object we want the SURF detector to detect the most descriptors that is possible this is why the hessianThreshold of SURF was given empirically at a value of 350-500 that produced 10 million descriptors for the 24 thousand whole pictures which were given to the algorithm as 51 thousand unique segmentations. Several runs for different number of clusters were tried in a range $[1500, 80000]$ and the final configuration of the system works with 10.000 centers which resulted an accuracy with the same test used for the tutorial with SVM at 38%. The 80.000 centers configuration was not tested with the SVM due to some bag that was resulting in segmentation fault in the experiment in the same code that run for the 10.000 centers experiment but was tested empirically and did not provide better results than the 10.000 centers.
As it can be seen from the Figures bellow the algorithm works well in the cases that an object has a very constant shape and colour as for example pizza and STOP sign and produce noisy ranked listes in items which vary in scale and transformation like cars or dogs.

Figure 5: STOP sign query: (query was a part of the first image containing only the stop sign)


o_train2014_000000002951.jpg
136,0 kB
COCO_train2014_000000085867.jpg
110,6 kB
COCO_train2014_000000110217.jpg
195,6 kB
COCO_train2014_000000160701.jpg
120,3 kB
COCO_train2014_000000207513.jpg
130,9 kB
o_train2014_000000345658.jpg
136,6 kB
COCO_train2014_000000359106.jpg
83,5 kB
COCO_train2014_000000404983.jpg
102,2 kB
COCO_train2014_000000518548.jpg
130,2 kB
COCO_train2014_000000533864.jpg
221,8 kB
o_train2014_000000534334.jpg
160,3 kB

In Figure.5 a good working example is shown the sub image part was taken as shown in 4.5 and fed the algorithm with a query containing only the rectangle around the STOP sign which produces nice results. The same results is shown in pizza queries also.

Figure 6: SURFBOARD query noisy ranked list. 4 surfboards found also as expected surfboard is confused with skateboard several times



In Figure.6 we can see a bad example of the functionality of the algorithm with the given query to be the image from Figure.3 with the surfboard. Even if it finds 4 surfboard images there are some random pictures and also as expected the confusion with the skateboard images.

6 Conclusion

The approach of Bag of Visual Words can work for large scale image retrieval well with having in mind that clustering needs to be optimized in order to find the best configuration of centers for each dataset while there is no rule of thumb that can be followed to solve this problem and the time that is needed for K means in order to converge to a good configuration can also be optimized by parallelization methods or other faster clustering algorithms. OpenCV is a helpful library but in order to improve the performance a lot of the functions that were used should be implemented by scratch as there are some cases that the automated procedure would not work or as expected, at the best performance possible or it might not provide alterations that could be needed for that scope.

References

[1] P Tirilly, V Claveau, P Gros, *A review of weighting schemes for bag of visual words image retrieval*, Research Report PI 1927-Avril, 2009.

[2] MJ Huiskes, MS Lew, *Performance evaluation of relevance feedback methods*, In Proceedings of the 2008 international conference on Content-based image and video retrieval, pp. 239-248. ACM, 2008.

[3] J Liu, *Image Retrieval based on Bag-of-Words model*,arXiv preprint arXiv:1304.5168, 2013.

[4] H Bay, A Ess, T Tuytelaars, L Van Gool, *Speeded-Up Robust Features*,Computer Vision and Image Understanding, 2008.

[5] Q Tian, N Sebe, MS Lew, E Loupias, TS Huang, *Content-based image retrieval using wavelet-based salient points*, Photonics West 2001-Electronic Imaging, 2001.

[6] J Sivic, A Zisserman, *Video Google: Efficient Visual Search of Videos*, Lecture Notes in Computer Science Volume 4170, 2006.

[7] G Csurka, CR Dance, L Fan, J Willamowski, C Bray, *Visual Categorization with Bags of Keypoints*, ECCV, 2004.