SOTU in Action

Wan-Lei Zhao Chong-Wah Ngo

Date: 08-Apr.-2012

Version: 1.10 Release

Preface

In the last decade, we have experienced the explosion of multimedia contents in Internet. Due to the easy availability of digital equipment and various ways to propagating media resources, there are large amount of multimedia data (including photos, news videos, speeches and tutorials) proliferating over Internet, TV channels and mobile devices. For example, in every minute, there are *3,320* photos have been uploaded to Flickr. The total number of photos already reaches to *740,880,729*[1]. Similarly, in the most popular video share website YouTube, *200,000* videos are uploaded per day and the total number of videos has reached to *120,000,000* on May 2009. As estimated, to view such a vast number of videos, it would take more than 600 years. Similarly, in traditional media such as TVs, thousand hours of programs have been broadcasted from various channels each day.

The presence of this massive number of photos and videos brings us new challenges. Specifically, we have to re-consider our current way of archiving, sharing and mining multimedia data. After all, we should be able to organize these data properly that allows users to access to the target contents in an efficient manner. This basically comes with the issues such as content based image/video retrieval, image/video annotation, hot events detection, object/landmark recognition and hyper-linking of images/videos. Most of these issues are challenging and have attracted numerous attentions in the research community. A lot of efforts have been devoted to automatically understanding the visual contents. However, due to the gap between low level visual features and human perception, effective approach is yet to occur to solve these challenging issues. Fortunately, the existence of near-duplicate images/videos in large amount arise as a solid step towards handling these timely issues.

On one hand, to detect near-duplicates from such a high volume of images/videos is not a trivial task. In general, the near-duplicates only exist in a small ratio of the total number of possible pair-wise combinations among images/videos. As a consequence, the task of discovering near-duplicates in a corpus is comparable to "finding a (thick) needle in a haystack". In order to enable the algorithm robust to various aforementioned transformations, a sophisticated representation of the images/videos is required. This normally involves much higher computing costs than the traditional representation forms, e.g., global features. However, since we are presented with dataset of large volume and moreover pair-wise checking is required which is exponential to the size of dataset, algorithms must be efficient enough to accommodate such a huge volume computing expense. This subsequently results in two competing requirements: speed efficiency and effectiveness.

[1]Statistics are collected from Flickr on April 28th 2010.

On the other hand, in last ten years we have also witnessed the success of Bag-of-visual Word (BoVW) scheme in different contexts such as content based image retrieval, visual object recognition/classification and semantic concept detection etc. Due to the introduction of keypoint features and BoVW, the framework for content-based image retrieval has been changed profoundly. About ten years ago, researches were still struggling with image retrieval on datasets sized of few thousands. Retrieval performance were suffering from low precision that the image global features can best offer. However in recent years, we have seen the volume of data that BoVW based approaches process has been scaled up to millions. At the same time, high precision can still be maintained.

SOTU is developed as a toolkit for near-duplicate image and video retrieval as well as detection in large scale. It fully adopts BoVW scheme which includes routines for building visual vocabulary, vector quantization and near-duplicate image retrieval/detection. Our aim to release this tool is to make people's works in this area comparable to each other and relieve the efforts of repeating others' work. Techniques integrated with **SOTU** can be found in [9] and our recent work [12, 15]. We also wish this tool become a good reference for the people who are interested in and have been working on this topic.

With **SOTU**, near-duplicate retrieval that requires instant response on a dataset of million level is achievable with a commercial PC. Retrieving near-duplicate videos from a reference set of 400 hours in real time is also possible. However, we do not verdict **SOTU** and the framework it adopts are the final solutions towards this issue. Instead, we might be only open to the first episode of a long but exciting story since we are still witnessing that novel solutions and frameworks arise one after another. If we consider the problem in web-scale, we are really still far away from either what the BoVW allows us to achieve or what we aim to achieve.

In this manual, we briefly review the algorithms and steps that has been integrated in this toolkit. In general, BoVW based approach consists of two steps: offline quantization and online retrieval/detection. For this reason, this manual has been broken into two major chapters, which introduce different quantization methods and retrieval/detection approaches respectively.

SOTU is currently maintained in the author's spare time. For limited time, it is not promised to fix the known bugs timely and work out a new release regularly. However any bug reports or suggestions are appreciated all the time.

Contents

Chapter 1

Preparations

1.1 Extracting Image Local Features

Since **SOTU** fully relies on image local features, before starting with **SOTU**, features must be prepared for images of a given dataset. There are a couple of local feature detectors and descriptors available. The detectors basically locate stable keypoints (and their support regions) which are invariant to kinds of transformations introduced by geometric and photometric changes. Popular detectors include Harris-Affine [2, 16], Hessian-Affine [2, 16], and Difference of Gaussian (DoG) [3, 18]. After localizing the keypoints, a descriptor is required to generate feature for the local interest points. Popular descriptors are SIFT [3], PCA-SIFT [7] and SURF [8]. One can choose **LIP-VIREO** [20] as the extraction tool which integrates Harris-of-Laplacian, Hessian-of-Laplcian, Laplacian of Gaussian and DoG detector etc. along with descriptors: SIFT, PCA-SIFT, F-SIFT, LJet, SPIN and etc. In general, it is able to achieve similar performance as the tools from [16, 18, 19].

Currently, **SOTU** only accepts the file format used by **LIP-VIREO**. If features extracted with the other toolkits (e.g., SIFT from [19]) are used as input, one has to convert them into the format that is recognizable by **SOTU** in advance. Otherwise, the results from **SOTU** could be unpredictable. Please refer to Appendix VI.1 for the valid format of the input feature.

1.2 Building Visual Word Vocabulary

In the version later than V1.04, **SOTU** offers options for automatic visual word generation. One can choose to generate visual word by **SOTU** or generate visual word by himself. For the visual vocabulary generated on one's own, please stick to the format **SOTU** specifies.

Although building visual vocabulary in general follows the same procedure, they are still different in details. In this Section, we present the way that how we produce the visual word vocabulary. In the case that user is not familiar with the procedure, please follow our steps listed below to ensure **SOTU** could achieve stable performance. For those who are already familiar with the procedure, steps listed below can be treated as a reference.

However, in order to make sure the generated visual vocabulary is compatible with **SOTU**, please stick to the format of visual vocabulary that **SOTU** requires (see Appendix VI.4).

Step 1. Sample Randomly on keypoints

Universal visual vocabulary is yet seen, and its possibility is still an open issue to be explored. Under such a circumstance, visual vocabulary is somehow ad hoc and different from dataset to dataset. Traditionally, Visual words are derived from sampled keypoints given an image set. Recent research [9] shows the performance is still quite well even the visual words are derived from an image set which has no overlapping with the one to be processed. Nevertheless, we still suggest user sample images from the dataset to be processed. To make sure the visual words are representative enough, random sampling is recommended. At its first step, user can use **LIP-VIREO** [20] to extract keypoints from images, random sampling on these keypoint files therefore prepares training source for visual words generation. In the version later than v1.06, **SOTU** is able to perform sampling automatically once the keypoint files are given.

First of all, in order to generate the visual vocabulary, sufficient number of keypoints must be prepared. To decide how many keypoints should be sampled is empirical. It is hard to exactly say how many keypoints are enough. According to our experience, the number of keypoints to be sampled is linear to the number of visual words one wants. The number of keypoints to be sampled can be the number of visual words multiplied by a constant in the range from 20 to 80. This ensures we have enough number of keypoints (in the range of [20, 80]) in each cluster to support one visual word.

The output from this step is a matrix **M**. Since we are using *Cosine* similarity as the distance measure in the later stages, each feature vector in **M MUST BE normalized with l_2 norm**. This property is going to be exploited by Repeated Bisecting K-means which we adopted for visual vocabulary generation. Please refer to Appendix VI.2 for the valid format of matrix **M**.

In **SOTU 1.06** and later, we provide function to collect keypoints from files into a matrix. Please refer to Section 1.4 for command line instructions.

Step 2. Cluster on sampled feature matrix M by RB-Kmeans

In the second step, the training matrix is going to be clustered with specified cluster number. This number coincides to the size of visual vocabulary. That is, according to the definition of visual word, one visual word is represented by one cluster of keypoints. In our package, we have our own implementation of Repeated Bisecting K-means (RB K-means) [22]. This algorithm works pretty well in terms of both efficiency and effectiveness ans has been quite popular in the literature.

Alternatively, if one wants to generate visual word on his own, we recommend CLUTO toolkit [22]. It is a powerful toolkit for large-scale clustering in which the original implementation of Repeated Bisecting K-means has been incorporated. With following command line, CLUTO performs clustering on input matrix **M** with RB K-means.

vcluster -clmethod=rb -clustfile=rsltfn matrixfn -cstype=large n

Command above clusters matrix (given by '**matrixfn**') into '**n**' clusters. Clustering results are saved to the resulting file '**rsltfn**'. The '**-clmethod**' option specifies the clustering approach as 'Repeated Bi-secting K-means'. By default, it uses *Cosine* distance as the distance measure. For keypoint features (especially SIFT and its variants), user is suggested to choose *Cosine* distance measure. The option '**-cstype**' is optional. By setting '**-cstype**' to '**large**', the output clusters roughly have similar size. Otherwise, the default setting for '**-cstype**' is '**best**' which could result in extremely unbalanced sizes among clusters.

Note that, in our implementation, we fix the option to 'large' for '-cstype' and 'i2' for 'textbf-crfun'.

Step 3. (Optional) Build two-layer visual vocabulary

Building a two-layer visual word vocabulary could save up quantization time by a lot especially in the case that one decides to use a large vocabulary (e.g. $\geq 10k$) and one has lots of keypoint files to be quantized. According to our observation, there is no significant performance difference (in terms of both accuracy and recall) between the two-layer and one-layer schemes. Although building two-layer BoVW structure requires extra efforts, we still encourage user to do so since it is really awarding. User is referred to Section 1.3 for the details of our way in constructing two-layer visual word vocabulary.

Step 4. Convert clusters into visual words

We choose cluster center of each clustered keypoint group as the visual word. With the output from Step 2, we have clusters set $\mathcal{C} = \{c_0, ..., c_i, ..c_n\}$. And F_{ij} is one feature vector belonging to cluster c_i. We calculate the cluster center as the visual word v_i (as shown in Eqn 1.1).

$$v_i = \frac{1}{|c_i|} \times \sum_{j=1}^{|c_i|} F_{ij} \tag{1.1}$$

1.3 Building Two-layer Visual Word Vocabulary

The aim of constructing a hierarchical visual vocabulary is to speed-up the vector quantization (VQ) process. According to our experiment in near-duplicate image retrieval, the performance differences between one layer visual vocabulary and two-layer visual vocabulary are minor. Figure 1.1 shows a two-layer visual word structure. The root layer acts as an indexing layer. By this way, given a keypoint to be quantized, it is no need to compare it against all visual words (nodes on the leaf layer) of the vocabulary. Instead, we first compare it against all nodes (only several hundreds in normal cases) on the root layer.

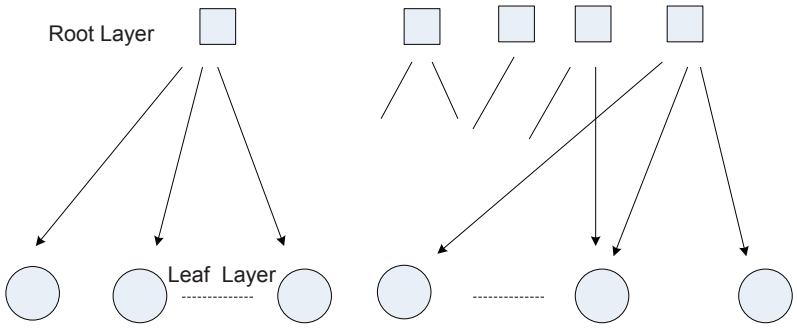


Figure 1.1: Two-layer BoVW mapping structure.

Afterwards, the keypoint is further compared to visual words on the leaf layer which is affiliated to the top-k (k is fixed to ***20*** in **SOTU**) nearest neighbors of the first layer nodes. Constructing vocabulary tree (which usually consists of more than two layers) is possible, however currently **SOTU** only supports two-layer structure.

To construct a two-layer structure, we follow steps listed below.

Step 1. Choose the number of visual words on the root layer

As mentioned above, the root layer acts as an indexing structure, and the number of visual words on the leaf layer is fixed and equals to the visual vocabulary size (n). At this moment, one should choose the number of visual words on the root layer (let's say m) which **MUST BE** smaller than n. Normally, m can be 5 to 15 times smaller than n. The smaller the m, the faster the quantization speed. However, it may have higher chance of missing true nearest neighbor which in turn introduces more quantization errors.

Step 2. Build the root layer visual words

Firstly, we follow Step 1 in Section 1.2 to generate matrix **M**. With RB-Kmeans, matrix M has been clustered into m clusters. Visual words for the root layer can be obtained by Step 4 in Section 1.2.

Step 3. Build the leaf layer visual words

Based on the outputs from the clustering in Step 2, we can partition matrix **M** into m matrices by moving keypoints affliated to the same cluster together. Let's say they are A_1, A_2, ..., A_i, ... A_m. Then we can cluster each of them (A_i) into $\frac{n}{m}$ clusters. Following Step 4 in section 1.2, we are able to obtain $\frac{n}{m}$ visual words from one A_i. Collecting all these visual words together, we get the second layer visual words. Note that, $\frac{n}{m}$ visual words is derived from a certain A_i which corresponds to one visual word on the root layer. As a result, it is required to construct the mapping between the first layer visual words and the second layer visual words so that visual words in the leaf layer can be covered by their nearest neighbors in the first layer visual words. In **SOTU**, to perform VQ with two-layer support, such kind of mapping is required

to supply. One can refer to Appendix VI.3 for the format of the mapping file. If one chooses **SOTU** to generate the two-layer visual vocabulary, this mapping can be automatically generated.

1.4 Commands

1.4.1 Collect Keypoints into Matrix

With **SOTU**, one has been relieved from sweaty efforts in either collecting the keypoints or implementing clustering algorithms. Only two steps are involved in the toolkit to build visual words from keypoint sample files. Command below collects keypoints in 'srcdir' into destine file 'dstfn'.

SOTU **-tc** conf_fn **-s** srcdir **-o** mat **-d** dstfn

Before running above command, one has to provide sufficient keypoint files which are randomly selected and have been organized into one folder (i.e. 'srcdir').

There are two parameters to be specified in the configuration file, '**step**' and '**norm**'. '**step**' is optional. One may write '**step=n**' to specify the sampling rate in each of keypoint source file. For instance, when n equals to '*1*', all the keypoint from a source file will be included. While when n is set to '*2*', one keypoint will be chosen for every two points parsed so far. The default value for '**step**' is set to '1'.

In addition, in the configuration file, one has to specify whether perform l_2 normalization on each feature vector with option '**norm=yes**'. By default, feature will be normalized. However, user is allowed to disable this option if the collected matrix will be used for other purpose by setting '**norm=no**'. **Please keep 'norm' enabled as long as one is going to prepare this matrix to build the visual vocabulary.**

Option '-o' is fixed to '**mat**' and the sampled features will be piled up in 'dstfn' file with special format that has been specified in Appendix VI.2.

1.4.2 Build Visual Vocabulary

Command below shows general options for visual word vocabulary generation both for one-layer and two-layer cases. One is free to choose either one-layer visual vocabulary or two-layer vocabulary. To run following command, a matrix which is composed by sampled keypoints must be prepared.

SOTU **-vc** matrix_fn **-d** destine_fn **-l1** vk_numb [**-l2** sub_nodes]

Option '**-vc**' specifies the location for the keypoint matrix. '**-d**' allows user to specify the destine file name for the visual vocabulary.

Option '**-l1**' specifies the number of visual word on the first layer. If only one layer visual word is going to be built, this number is also the size of visual vocabulary.

Option '**-l2**' is optional, it allows to specify the number of leaf nodes under each first layer node. Once this option is specified, user is going to generate a two-layer visual vocabulary. For this two-layer generation option, **SOTU** outputs three files, they are the first layer visual vocabulary (its size is specified by '**-l1**'), the second layer visual vocabulary (its size is the multiplication of numbers specified by '**-l1**' and '**-l2**') and the mapping between these two visual vocabularies.

It is possible to manually modify the mapping between the first and the second layer visual words in the case that one wants to optimize this mapping. However, it is not suggested since current way already offers good performance. Current setting ensures ***99.94%*** recall comparing with brute-force nearest neighbor search.

Chapter 2

BoVW Quantization

In the BoVW framework, given a predefined visual vocabulary, BoVW quantization is the offline process that projects keypoints into BoVW representation. Such that keypoints from different images no longer compare to each other by point-to-point matching. This is where the BoVW paradigm saves up the overall computing cost. In the stat-of-the-art, BoVW has been widely adopted for two major tasks, namely, image classification and content-based image retrieval. In these two tasks, BoVW has been treated slightly different. In the image classification, on one hand, BoVW is adopted to avoid exhaustive point-to-point matching. From this aspect, it is expected to have a large vocabulary. On the other hand, it is expected that BoVW allows cross-matching between different points as much as possible, which ensures maximal collision between potential matches. From this aspect, the vocabulary size should not be too large. As a trade-off, in practice the size of vocabulary ranges from few hundred to few thousand in the classification task. In the content-based image retrieval, similar awkward exists. Furthermore, the inverted file structure is usually adopted in this context. Due to the concerns about retrieval efficiency, each inverted list cannot be arbitrarily long. Instead BoVW representations are expected to be as sparse as possible. As a consequence, another trade-off has to be made. In practice, the vocabulary size has been seen ranging from 10,000 to several million. In the extremely large-scale dataset, say several billion, it is reported that the size of visual vocabulary has been set to billion level. However, there is no evidence to show that the size of vocabulary should be linear to the size of dataset.

Comparing with direct point-to-point matching, drop in performance becomes inevitable for BoVW approach. The performance degradation is largely attributed to the false visual word matches between the query and reference images. In the literature, these false matches have been alleviated in various ways and at different stages. One type of the false matches has been known as 'burstiness' problem [14]. This issue can be alleviated by re-ranking [14], or by binary quantization [15] for both query and reference images. Besides the burstiness phenomenon, a more popular type of false matches is due to imprecise quantization. The existing solution towards this issue is of filtering out false matches by geometric and visual verification during the online retrieval. One of the most successful verification methods has been demonstrated in [9]. They are known as weak geometric consistency (WGC) and

Table 2.1: General settings for vector quantization

dim=128	required
vq=hard	required
vocab=vocab/ox_l2_sift_20k.txt	required
supcab=vocab/ox_l2_sift_2k.txt	optional
supmap=vocab/ox_l2_sift_map.txt	optional
midx=[**brute**, level]	optional

item in **bold** is the default value.

Hamming Embedding (HE), which are efficient schemes for geometric and visual verification respectively. In order to facilitate the verification, additional information has to be attached with the quantized keypoints. As a consequence, factors pointed out above induce several variations at the quantization stage.

In **SOTU**, these variations are considered and treated as several different quantizaton and retrieval options. User is allowed to perform either basic BoVW retrieval or retrieval involving with sophisticated verification. For this purpose, different quantization options are provided. In this chapter, we explain how to perform quantization with **SOTU** in different manners. **SOTU** supports four types of quantization schemes. They are, the traditional BoVW, BoVW attached with Hamming Embedding signature, BoVW attached with geometric information and BoVW attached with both Hamming Embedding signature and geometric information. The quantization results facilitate different BoVW retrieval/detection strategies that will be explained in **Chapter** 4. In general, classic BoVW retrieval is supported by all BoVW quantizations. Extra affiliated visual or geometric information enables more sophisticated verification, which usually leads to better retrieval/detection performance, and results in higher computing costs in the mean time.

2.1 Command and General Settings

Command below shows the general options for BoVW quantization. To perform quantization, it is assumed that the feature files have been placed under folder '**srcdir**', the feature files are in **LIP-VIREO** format and all file names are suffixed with '**.pkeys**'.

SOTU **-ic** sotu-ic.conf **-v** [**vk**|**he**|**tgc**|**egc**] **-i** itmtab **-s** srcdir

-ic sotu-ic.conf

Option '**-ic**' allows user to provide parameters for the quantization process. An exemplar setting is shown in Table 2.1. "**dim**" specifies the dimension of the input feature vector. Option "**vq**" has to be fixed to 'hard' which indicates that **SOTU** assigns one visual word (instead of assigning multiple visual words) to one feature

vector[1]. Option "**vocab**" is used to set the location of vocabulary matrix. User can refer to Appendix VI.2 about the format of the input matrix.

Parameters mentioned above are always required. While options "**supcab**", "**supmap**" and "**midx**" are optional and they are about choosing appropriate nearest neighbor search approach for visual word quantization. During the quantization stage, **SOTU** provides two nearest-neighbor search methods which can be specified by "**midx**". User can choose among "**brute**" (default option) and "**level**". For "**brute**", NNs search are performed in a brute force manner. That is, each time for an incoming feature vector, the number of comparisons between this feature vector and visual words are equal to the size of visual vocabulary. Option '**level**' allows NNs search undertaken in a more efficient way while it requires to construct a two-layer visual vocabulary in advance (which has been discussed in Section 1.3). For option '**level**', another two parameters ("**supcab**" and "**supmap**") are required. "**supcab**" specifies the location for the root layer visual vocabulary. While "**supmap**" is used to set the file of mapping between root layer visual words and leaf layer visual words. Note that, during the NNs search, **SOTU** employs *Cosine* distance as the similarity measure. The input feature vector is normalized inside **SOTU** before the NN search is undertaken.

-v [**vk**|**he**|**tgc**|**egc**]

Option '**-v**' specifies the quantization scheme. **SOTU** supports fours types of quantization scheme. They are listed below. We explain these quantization schemes in detail from Section 2.2 to Section 2.5.

- **vk** This option allows keypoint feature quantization in traditional way (refer to Section 2.2).
- **he** This option quantizes keypoint file into BoVW representation affiliated with Hamming Embedding signature (refer to Section 2.3).
- **tgc** This option quantizes keypoint file into BoVW representation affiliated with geometric information (refer to Section 2.4).
- **egc** This option quantizes keypoint file into BoVW representation affiliated both with Hamming Embedding signature and geometric information (refer to Section 2.5).

-i itmtab

Option '**-i**' specifies the location for the quantization outputs. **SOTU** outputs three files for one quantization task. They are the name list file (specified by '**imgtab**'), the BoVW file (specified by '**bowtab**') and a weight file (specified by '**wghtab**').

'**itmtab**' is an XML style file which is used to set three options ('**imgtab**', '**bowtab**' and '**wghtab**') for the BoVW quantization (an example is shown in Table 2.3). These three items in '**itmtab**' are bracketed by '<item> </item>' pair. Name list file, which is specified by '**imgtab**', lists all the keypoint file names which have been

[1]In the future version, we will consider to allow multiple assignment which results in better retrieval performance as indicated in [10]

Table 2.2: Format of 'bowtab' file

N1	VK_Unit_1	VK_Unit_2	...	VK_Unit_N1
N2	VK_Unit_1	VK_Unit_2	...	VK_Unit_N2
...	...	...	...	...
Ni	VK_Unit_1	VK_Unit_2	...	VK_Unit_Ni

Table 2.3: Exemplar settings for 'itmtab'

```
<item>
imgtab=databank/ox_build_img.txt
bowtab=databank/ox_build_bow.txt
wghtab=databank/ox_build_wgh.txt
</item>
```

processed in the quantization. The BoVW file which is specified by '**bowtab**' is also a text file. One row in the file keep records for one BoVW vector which corresponds to one image. The format of the '**bowtab**' is shown in Table 2.2. The first number in the line indicates how many BoVW units have been generated for an input feature file. Followed are '**Ni**' units separated with blank character. The format of one BoVW unit is different for different quantization schemes. More detailed explanations are given for each quantization scheme in the sections followed.

Similar to '**bowtab**' file, one row in '**wghtab**' corresponds to one keypoint file in the '**imgtab**'. One row in the file starts with a sequential number and basically records the length of the BoVW vector. It is then followed by the number of keypoints in the original feature file.

Note that, unless you know what you are doing, DO NOT modify the contents of these three files.

-s srcdir

Option '**-s**' allows user to specify the source directory where the source feature files located. In particular, **SOTU** requires the name of feature file to be suffixed with '**.pkeys**'. Otherwise, the feature will be simply ignored. In each feature files, there would be one or more than one d dimensional feature vector. For the detail of file format, please refer to Appendix VI.1.

2.2 Traditional BoVW

In order to quantize feature files under directory '**srcdir**' into traditional BoVW representation, one can specify '**-v**' as **vk**.

Table 2.4: Configuration options for Hamming Embedding Training

medv=vocab/ox_he_medians_32.txt	required
pmat=vocab/pmat_32.txt	required
dim=128	required
vq=hard	required
vocab=vocab/ox_l2_sift_20k.txt	required
supcab=vocab/ox_l2_sift_2k.txt	optional
supmap=vocab/ox_l2_sift_map.txt	optional
midx=[**brute**, level]	optional

item in **bold** is the default value.

SOTU **-ic** sotu-ic.conf **-v vk -i** itmtab **-s** srcdir

The output from the traditional BoVW quantization are three files whose locations have been specified in '**itmtab**'. The BoVW unit is composed by two fields. They are the visual word ID (**unsigned integer**, 32 bits) and its term frequency (**unsigned char**, 8 bits) that appears in the corresponding keypoint file.

2.3 BoVW with Hamming Embedding

If one already trained his Hamming Embedding matrix according to the instructions in **Chapter 3**, the command below allows user to quantize feature vectors into BoVW representation that each keypoint is attached with a binary signature. Note that, in order to generate Hamming Embedding signature, the input keypoint feature vectors **SHOULD NOT** be normalized. In the output file which is specified by option '**bowtab**' in '**itmtab**', a 32-bit Hamming Embedding signature will be generated for each resulting unit.

SOTU **-ic** sotu-ic.conf **-v he -i** itmtab **-s** srcdir

In the configuration file '**sotu-ic.conf**', one has to point out the location of the projection matrix **P** (by '**pmat**') and the trained Hamming Embedding median matrix (by '**medv**'). Please refer to **Chapter 3** for details. In exemplar settings shown in Table 2.4, 'ox_he_medians_32.txt' is the trained Hamming Embedding matrix for medians (see format in Appendix VI.5). 'pmat_32.txt' is an 128×32 projection matrix. The output from '**he**' quantization are three files whose locations have been specified in '**itmtab**'. The BoVW unit for this quantization scheme is composed by two fields. They are the visual word ID (**unsigned integer**, 32 bits) and its Hamming Embedding signature (**unsigned integer**, 32 bits).

2.4 BoVW with Geo-info

Command below produces BoVW representation for feature files under directory '**srcdir**'. In each BoVW unit, it consists geometric information of its corresponding local feature. Note that, to generate such a representation, geometric information including location of local interest point (x, y), scale and dominant orientation (scale and angle) of the local feature must be prepared at the local interest point extraction stage. Otherwise, the output of following command is unpredictable.

SOTU **-ic** sotu-ic.conf **-v tgc -i** itmtab **-s** srcdir

The output from '**tgc**' quantization are three files whose locations have been specified in '**itmtab**'. The BoVW unit is composed by five fields. They are the visual word ID (**unsigned integer**, 32 bits), x and y (**unsigned short**, 16 bits for each), scale (**unsigned short**, 16 bits), dominant orientation information (**unsigned short**, 16 bits).

2.5 BoVW with Hamming Embedding and Geo-info

The following command generates BoVW representation along with Hamming Embedding signature and geometric information. The prerequisites of performing such a quantization are the trained Hamming Embedding matrix and the extracted local features which have been attached with location, characteristic scale and dominant orientation information. Similar to '**he**' option (Section 3.3), user has to specify '**pmat**' and '**medv**' options in the configuration file '**sotu-ic.conf**'. Again similar to '**he**' quantization, the input keypoint feature **MUST NOT** be normalized.

SOTU **-ic** sotu-ic.conf **-v egc -i** itmtab **-s** srcdir

The output from '**egc**' quantization are three files whose locations have been specified in '**itmtab**'. The BoVW unit is a combination on the units of '**he**' and '**tgc**' schemes. As a result, each 'egc' unit is composed by six fields. They are the visual word ID (**unsigned integer**, 32 bits), x and y (**unsigned short**, 16 bits for each), scale (**unsigned short**, 16 bits), dominant orientation information (**unsigned short**, 16 bits) and Hamming Embedding signature (**unsigned integer**, 32 bits).

Chapter 3

Hamming Embedding Training

3.1 Review on Hamming Embedding

Hamming Embedding is a binary signature that is generated for a feature vector [9]. It has been successfully applied in enhancing the performance of BoVW based image retrieval. As shown in the [9, 13], it is able to boost performance of the BoVW based retrieval both in accuracy and speed efficiency. In order to obtain the binary signature, an offline training process is required. For the convenience and completeness, we list the training procedure which has been fully described in [9].

Step 1. Random matrix generation

Generate an orthogonal projection matrix $\mathbf{P}$ $(d_b \times d)$. Draw a matrix of Gaussian $(d \times d$, d is the original feature dimension) values and apply a QR factorization to it. The first d_b rows of the orthogonal matrix obtained by this decomposition form matrix $\mathbf{P}$. d_b equals to the length of the binary signature one prefers.

Step 2. Feature vector projection and assignment

A large set of feature vectors from an independent dataset are randomly sampled. These feature vectors are first assigned to words in the visual vocabulary (generated in the way described in Section 1). Additionally, given a feature vector F_j been assigned to visual word v_i, it is further projected by matrix $\mathbf{P}$. That operation results in a component f, with d_b dimensions.

Step 3. Median values of projected descriptors

For each visual word v_i, compute the median values for d_b dimensions based on the projected components f which fall into visual word v_i. That results in a matrix $\mathbf{M}$ $(n \times d_b)$. And one row of $\mathbf{M}$ corresponds to one visual word in the vocabulary.

Note that $\mathbf{d}$ is the original length of feature vector and d_b is the final length of signature. Take SIFT feature as an example, the feature dimension ($\mathbf{d}$) is 128 and d_b can be typically specified to 32 or 64 [9]. In our program, **SOTU** only supports at most 32 bits signature. As a result, d_b can only be set up-to 32.

Table 3.1: A configuration example for Hamming Embedding Training

dim=128	required
pmat=vocab/pmat_32.txt	required
vq=hard	required
vocab=vocab/ox_l2_sift_20k.txt	required
supcab=vocab/ox_l2_sift_2k.txt	optional
supmap=vocab/ox_sift_map.txt	optional
midx=[**brute**, level]	optional

item in **bold** is the default setting.

3.2 Command

SOTU provides command to generate the target matrix M. Two steps are involved. Firstly, before the training gets started, the input keypoint files has to be prepared in advance (Step 1). Once the training data are ready, following command undertakes training and outputs medians for each projected dimension which covers Step 2 and Step 3 of the Hamming Embedding training process.

SOTU **-tc** sotu-he.conf **-o he -d** itmtab **-s** srcdir

-tc sotu-he.conf

Option '**-tc**' specifies the location for the configuration file. An exemplar configuration file is shown in Table 3.1.

For the settings of these parameters, they are similar to configurations in BoVW quantization, please refer to Section 2 for more details.

-o he

Option '**-o**' allows user to specify the Hamming Embedding training the task. For Hamming Embedding training, please fix it to '**he**'.

-d itmtab

Option '**-d**' specifies the destine files for training outputs.

As shown in Table 3.2, two files must be given which specify destine locations for the training results. The frequency that each visual word appears in the training data will be listed in the file specified by '**embed**'. The aggregated features which have been projected are also saved in this file. This file has no use for later stages, it is intermediate result for verification purpose. File specified by '**medv**' saves trained matrix **M**. This file will be later used for Hamming Embedding based feature vector quantization.

Table 3.2: Setting for destine path of the Hamming Embedding training results

<item>	
embed=hembed/ox_he_embed.hm	required
medv=hembed/ox_he_medians_32.txt	required
</item>	

-s srcdir

Option '**-s**' allows user to specify the source directory where the training feature files located and the input feature **MUST NOT** be normalized. Please refer to Appendix VI.1 for the format of feature file.

Please be reminded that sufficient number of keypoints must be supplied to the training procedure. Such that there are enough number (e.g. few hundred) of keypoints that fall into one visual word. Otherwise, the training process loses its statistical significance. In general, the total number of keypoints that these keypoint files supply should be at least 100 times more than the size of visual vocabulary. For the same reason, user is not suggested to use the same group of the keypoints that have been used in the visual vocabulary construction. If possible, the images from which these keypoints files have been derived should be dissimilar to each other. This is again to ensure the trained medians hold statistical significance.

Chapter 4

BoVW based Near-duplicate Image/Video Retrieval/Detection

In this **Chapter**, the focus has been set on the online retrieval with **SOTU**. To this end, user is supposed to having fulfilled the BoVW quantization with certain options for both query and reference images. We are now ready to compare the BoVW of query images to the BoVW of reference images.

If the vocabulary size is reasonably large (e.g., 10K), the BoVW is going to be sparse. It becomes natural to adopt inverted file structure to index the BoVW of the reference set to avoid linear scanning over the whole reference set. In **SOTU**, the inverted file structure is constructed on the fly given the BoVW vectors of the reference images have been specified. To allow efficient retrieval, **SOTU** loads the whole reference set into the main memory before the retrieval is undertaken. Usually, the memory it takes up is around *95%* of the disc space the BoVW file occupies. No cache is going to be built on the disc. To this point, user is warned to estimate the possible memory consumption. If there is no enough memory to hold the whole inverted file structure, user is suggested to slice the task into several. **SOTU** in deed offers this flexibility and results no performance difference. The time spent on loading the BoVW vectors and constructing the inverted file structure is linear to the size of reference dataset. Usually, it only takes few minutes to load a dataset with one million images. Once the inverted file structure has been built, it will be kept in the memory until all the specified retrieval/detection tasks have been completed. An illustration of the inverted file structure that **SOTU** constructs is shown in Figure 4.1.

BoVW approach has been the classic approach for content-based image retrieval, we also witnessed several of its variants have been proposed in recent years, which are aiming to improve its performance one way or another. This basically results in different quantization schemes and in turn leads to various ways in online retrieval/detection. In **Chapter** 2, we explained four different ways about vector quantization with image local features. Different from the traditional BoVW, the variants of BoVW quantization embed geometric and visual information of each keypoint into the quantized cell. Correspondingly, user is offered with different ways of online retrieval in **SOTU**, which is the main subject of this **Chapter**.

SOTU supports near-duplicate image retrieval in several ways. Namely, they are the

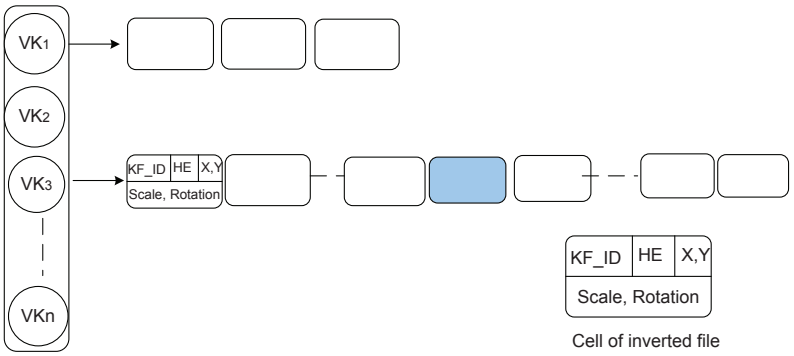


Figure 4.1: Inverted file structure that has been adopted for online BoVW retrieval/detection. An inverted file cell obtained by quantization option 'egc' is shown on bottom-right.

traditional BoVW ('**vk**' option), BoVW with enhanced geometry verification ('**tgc**' option), BoVW with Hamming Embedding filtering (option 'he') and a combination of BoVW, Hamming Embedding plus enhanced geometry verification ('**egc**' option). For detection tasks, only BoVW quantized with '**tgc**' and '**egc**' are enabled with detection option.

4.1 Image/Video Retrieval

4.1.1 General Options

The complete command options for near-duplicate retrieval is shown as follows.

SOTU **-dc** sotu-dc.conf [**-vk**|**-tgc**|**-he**|**-egc**] itmtab **-id**|**-vd** [**vk**|**he**|**tgc**|**vk+**|**egc**] **-d** dstfn

- **-dc sotu-dc.conf**
  Option '-dc' specifies the configuration file.

- **-vk**|**-tgc**|**-he**|**-egc** item
  Option '**-vk**|**-tgc**|**-he**|**-egc**' specifies the BoVW files of the query images. In **SOTU**, retrieval/detection tasks are performed in batch. User is suggested to quantize all the queries into one or several groups of BoVW files.

- **-id**|**-vd** [**vk**|**he**|**tgc**|**vk+**|**egc**]
  This option specifies the retrieval options. '-id' is for image retrieval/detection task, while '-vd' is for video retrieval/detection. Only one of these two options should be selected at once. All the available options are, [**vk**|**he**|**tgc**|**vk+**|**egc**]. However, not all the options are available to every type of quantized BoVW feature.

Table 4.1: A Configuration Example for retrieval tasks

dim=20000	required
cos=0.01	required
refer=etc/itmtab_tgc_ox.txt	required

- **-d** dstfn
  This option specifies the destine file. For different retrieval options, the resulting format could be different.

Detailed explanation can be found in following sections.

4.1.2 Command Option Explained

-dc sotu-dc.conf

Option '**-dc**' specifies the path of configuration file for the retrieval/detection task. An exemplar configuration file is shown in Table 4.1. All parameters listed in the configure file are required for the retrieval task.

Option '**dim**' should be set to the size of visual vocabulary.
Option '**cos**' sets the minimum *Cosine* distance for retrieved items. Retrieved items whose *Cosine* distances are lower than that threshold are pruned and will not put into the final retrieval results.
Option '**refer**' specifies the location for the reference set. The file follows the same format as 'itmtab' used at quantization stage. However, a little bit different from 'itmtab' used in quantization stage, user is allowed list more than one item in the file. Each item specifies one reference set. So that the reference set can be partitioned into sub-sets. User is allowed to operate on one or several sub-sets according to one's needs or the available memory. In terms of the performance, there is no difference between conducting retrieval on the whole reference dataset at once and operating on sub-sets one by one.

[**-vk**|**-he**|**-tgc**|**-egc**] **itmtab**

For these options, only one can be selected at once. Each option basically corresponds to one quantization scheme (introduced in **Chapter** 2). Before launching the retrieval, one has to make sure the BoVW feature from both query and reference sides have been quantized in the same way. Otherwise, exceptions might occur. This option is followed with the path for an item table ('**itmtab**') at where user specifies the item list of BoVW files for the query side. The format of the item table is every bit as the reference item table which is directed by '**refer**' in the configure file. Similar to reference set, user is allowed to enumerate more than one item (one query set) in the file. The retrieval is pulled out in the same order as the items listed in the '**itmtab**' file. For one query set, **SOTU** outputs one retrieval result file.

```
<item>
imgtab=databank/ox_build_img1.txt
bowtab=databank/ox_build_bow1.txt
wghtab=databank/ox_build_wgh1.txt
</item>
<item>
imgtab=databank/ox_build_img2.txt
bowtab=databank/ox_build_bow2.txt
wghtab=databank/ox_build_wgh2.txt
</item>
   .
   .
```

One can also set query item file the same as the reference item file. In this case, **SOTU** performs retrieval/detection within one dataset. The self duplicates will be omitted automatically from the resulting list.

In addition, BoVW features quantized in different ways support different retrieval schemes. The quantization method and its available retrieval options have been listed in Table 4.2.

-id|-vd [vk|he|tgc|vk+|egc]

This option is always required. At each time, only one of them (either '-id' or '-vd') can be chosen. Option '-id' indicates the task is near-duplicate **image** retrieval, while '-vd' tells **SOTU** to perform near-duplicate **video** retrieval instead.

In both image and video retrieval/detection tasks, one is provided with one or several choices depending on what kind of quantized BoVW features one gets prepared. For instance, for BoVW features obtained from 'vk' the quantization, only traditional BoVW retrieval option 'vk' is supported. While for BoVW feature via 'tgc' quantization option, user is provided with three choices, they are 'vk' (traditional BoVW approach), 'vk+' (BoVW retrieval with weak geometric constraint and Hamming embedding verification [9]) and 'tgc' (BoVW retrieval with enhanced Weak Geometric Constraint [12]). The detailed retrieval choices that are supported for the corresponding quantized feature are listed in Table 4.2.

Option '**vk**' undertakes traditional BoVW retrieval and it employs TF/IDF model. Since, according to our experimental results, there is no significant improvement when incorporating IDF, only TF is considered across all schemes integrated in **SOTU**. The output are image pairs each affiliated with a *Cosine* similarity score.

Option '**he**' prunes false visual word matches during the retrieval stage [9]. It checks the Hamming distance between two matched visual words. Comparing to 'vk' option,

Table 4.2: Corresponding quantize option for a given retrieval option.

Retrieval Option	Quantization Option (-v)	Supported Retrieval Choices (-id|-vd)
-vk	vk	vk
-he	he	vk|he
-tgc	tgc	vk|tgc
-egc	egc	vk|vk+|he|egc

'**he**' saves up the retrieval time while achieves much better performance as indicated in [9].

Option '**tgc**' performs BoVW retrieval with enhanced weak geometric verification on the visual word matches. One is referred to [12] to find the technical details. Comparing with WGC, it is a tighter geometric verification on the visual word matches.

Option '**vk+**' performs BoVW retrieval with weak geometric verification (WGC) plus Hamming Embedding (HE) based filtering which is an implementation of approach described in [9]. Comparing with methods in [9], we DO NOT make assumption that the transformation between query image and reference image is in a particular range. In another word, 'prior knowledge' is not injected into WGC histogram in our implementation.

Option '**egc**' prunes false visual word matches in two ways. It first checks the Hamming distance between two matched visual words and then EWGC (as 'tgc' option) is adopted to verify visual word matches. Comparing to 'tgc' option, it saves up the retrieval time while boosts the performance further.

-d dstfn

This option is always required which provides destine file to save up retrieval/detection results. The file format of the retrieval/detection results are explained in detail in Section 4.1.3.

4.1.3 Format of the Output File

In the result file, all the reference images that their similarity scores to a query higher than the specified threshold are kept in the resulting list.

For 'tgc', 'he', 'vk+' and 'egc' options, **SOTU** always generates two columns of weights. The first column, which is viewed as major score, is the *Cosine* distance (of BoVW) between the query and one reference image. For options such as 'vk', 'tgc' in which Hamming Embedding is not involved, this score is exactly *Cosine* distance between two BoVWs. As a result in these cases, this score should be less than or equal to 1.0. However, for options 'he', 'egc' and 'vk+', where the Hamming Embedding is involved, this score has been re-weighted with the Hamming distance between visual words (see details in [9]). In this case, the score could be higher than 1.0. According to our observation, this score is pretty distinctive.

For the second column scores, they are derived from the online verification. For instance, the geometric verification WGC takes min(Max(H_s), Max(H_r)), where H_s denotes the histogram on the difference of characteristic scales, while H_r denotes the histogram on orientation difference (i.e. rotation). So for retrieval option 'vk+', the second column weight is exactly the value of this min(Max(*), Max(*)). Notice that, the average of the histogram bins has been subtracted from this score, which makes it more robust to noisy matches. For 'tgc' option, which represents the enhanced weak geometric constraint (EWGC) in [12], only one histogram is generated. The second column weight is obtained in the following manner. The values from the histogram peak and its two neighboring bins are first summed up. It is then subtracted by the histogram average. For option 'vk+' and 'egc', we do not subtract the histogram average. We find that Hamming Embedding already helps prune large portion of false matches. The true score would have been simply under-estimated if the average had been removed from the peak.

So far as one can see, the second column of the weights are the rough estimation about the number of correct matches between two images. This information however is somehow already incorporated in the first column weights. Sometimes, one might try different way of re-ranking on the retrieval list, these second column weights could supply one with auxiliary hints. In our practice, only the weights from the first column are considered.

For 'he' option, the second column simply shows the number of matches survive over the Hamming embedding verification. So again it is a rough estimation on the number of correct matches. For 'vk' option, the second column weights have been set to zeros.

4.1.4 Video Retrieval or Frame Retrieval?

Essentially, there is no big difference between option '-vd' and '-id' since the video retrieval in **SOTU** is treated as an image retrieval task as demonstrated in [12]. Comparing to '-id', an addition step that **SOTU** takes for '-vd' is that a 2D Hough transform is conducted on the retrieved frame list of each query. The peak in the 2D Hough transform corresponds to the duplicate frames from two video sequences.

To ensure **SOTU** run smoothly with option '-vd', one has to extract and quantize the video frames in advance. The name of each video frame has to be encoded in specified pattern (which is illustrated in Figure 4.2). Otherwise, exceptions might occur and the results become unpredictable.

As shown in Figure 4.2, all digits are unsigned integers. In the 2D Hough transform, the Video ID, frame sequential number and time code will be considered. The frame sequential order is indicated by the third digit. It should range from 1 to the number of frames extracted from one video. While time code tells which is the current frame of all frames in the video. It ranges from 1 to the total number of frames the video has.

Although one is open to option '-vd', user is still suggested to choose '-id' in video retrieval task. In this case, the output is a list of retrieved frames. Such that one has more options to manipulate with the retrieval results. The post-processing such as 2D Hough transform is left to the user. The exemplar code in C++ for 2D Hough transform

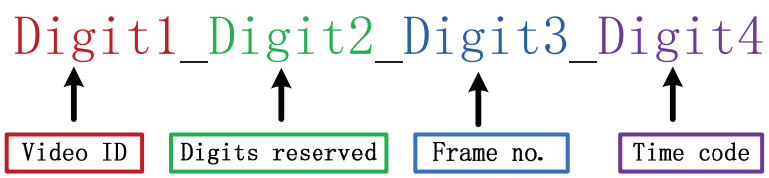


Figure 4.2: Specified format of one video frame name. The name consists of four digital numbers connected by '_', the first digit is the video ID; the second is reserved (a digital number is required); the third one is the frame number; the last one is also an unsigned integer value which indicates where the frame located.

is available upon request[1]. Above explanation applies to video detection tasks which is introduced in Section 4.2.

4.2 Image/Video Detection

In the literature, there are different understandings about the differences between the image retrieval and detection tasks. However, despite this dispute, detection is usually believed to be a more demanding task which requires a true/false judgment over one pair of images/videos. In **SOTU**, the detection algorithm is built upon retrieval algorithms. Namely, given visual word matches between query and a reference image that passed verification through retrieval algorithms, a more tight geometric verification is introduced. By this way, a lot of false matches are expected to be filtered out. Such that we are allowed to judge whether two images are near-duplicate only based on the number of matches that survive through this verification. In this section, before introducing the detection command in **SOTU**, a brief review about the geometric verification that **SOTU** adopts is presented. User is referred to [15] for the complete description.

4.2.1 Review on Reciprocal Validation and Re-ranking

Based on the linear transformation parameters (scale and rotation) that are recovered from different estimation models in SR-PE and EWGC, we can perform reciprocal validation (RV) on visual word matches during the post-processing stage. In [11], affine transformation parameters, namely scale ($\hat{s}$) and rotation ($\hat{\theta}$), are estimated for a matched keypoint pair pq by referring to another matched keypoint pair $p'q'$ (assuming that point p and p' are in image I while q and q' are in image J). The scale $\hat{s}$ between image I and image J can be approximated by the distance ratio between $\overrightarrow{pp'}$ and $\overrightarrow{qq'}$. The rotation $\hat{\theta}$ can be approximated by the angle between line $\overrightarrow{pp'}$ and line $\overrightarrow{qq'}$.

Alternatively, according to EWGC introduced in [12], the transformation parameters scale ($\tilde{s}$) and rotation ($\tilde{\theta}$) for the matched pair pq can also be estimated based on keypoint

[1]Email title: Request for 2D Hough transform code; Email address: stonescx@gmail.com

characteristic scale and dominant orientation.

$$\widetilde{s} = 2^{(s_q - s_p)}, \tag{4.1}$$

$$\widetilde{\theta} = \theta_q - \theta_p, \tag{4.2}$$

where s_p, s_q and θ_p, θ_p are characteristic scales and dominant orientations for keypoints p and p respectively.

For correct matches, $\hat{s}$ must be similar to $\tilde{s}$, while $\hat{\theta}$ must be similar to $\tilde{\theta}$. Comparatively, it is not true for false matches. Therefore, pruning out the noisy visual word matches can be as easy as deleting matches whose discrepancies ($\triangle = max\{|\hat{\theta} - \tilde{\theta}|, |\hat{s} - \tilde{s}|\}$) between estimated parameters are higher than a threshold. As $\hat{s}$, $\hat{\theta}$ and $\tilde{s}$, $\tilde{\theta}$ are estimated independently, such kind of reciprocal validation can be quite effective. Furthermore, the degree of discrepancy: $\triangle$ can be further employed to re-weight the Hamming distance between matched visual words. As demonstrated in [15], the re-ranking helps near-duplicate image pair with only a few correct matches ranked high in the retrieval list.

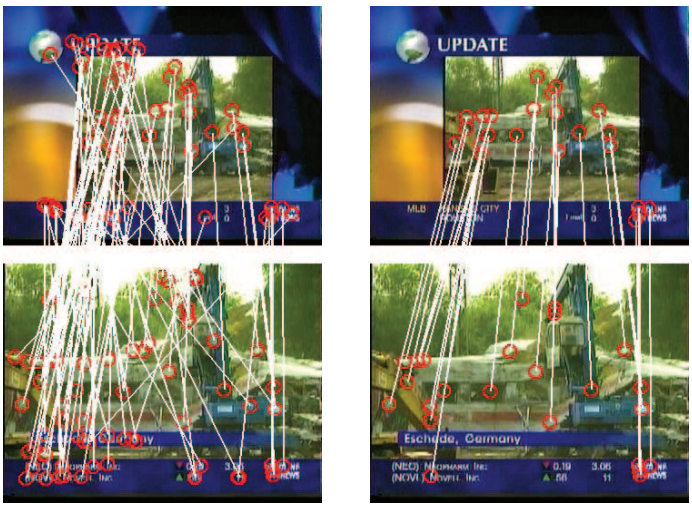
SOTU performs near-duplicate image/video detection based on results of BoVW retrieval. Basically, we inject scale-rotation invariant pattern entropy (SR-PE) proposed in [11] into the retrieval framework. The visual word matches (between one query image and one reference image) obtained during the BoVW retrieval are fed into SR-PE to verify whether the query image is near-duplicate to the reference image. Comparing to the framework proposed in [11], we replace the OOS matching with BoVW matching. Since visual word matches are always mixed false matches, feeding SR-PE with raw visual word matches simply results in low prediction accuracy. This verification process can be 5 to 10 times slower than EWGC but with much higher precision. So it is left to the user to make a choice between retrieval and detection.

As shown in Figure 4.2.1, the visual word matches obtained by 'tgc' are dominated with false matches, however, after this reciprocal geometric verification, many false matches have been filtered while most of the correct matches are preserved. Although it seems geometric verification only already good enough as far as we can learn from this example, there are still many exceptions in which false matches accidentally demonstrate geometrical coherence. This is where the visual verification: Hamming Embedding comes to play.

As a more reliable input to reciprocal verification, visual word matches after Hamming Embedding [9] along with EWGC ('tgc') verification is preferred. That is, only visual word matches which survive through Hamming thresholding and EWGC verification will be fed into SR-PE. In this toolkit, detection option is only open to BoVW features that have been quantized with 'tgc' or 'egc'. Additionally detection on features quantized by 'tgc' is not expected to work perfectly well. The option is offered only for observation or as an ad hoc solution. If one wants to apply **SOTU** to perform near-duplicate detection in practice, quantizing features with 'egc' option is highly recommended.

4.2.2 Command

To ensure reasonable output, near-duplicate detection option is only available for '**egc**' and '**tgc**' quantization schemes. The detection command is shown below.



(a) Matches obtained by 'TGC'

(b) Matches after Reciprocal verification

Figure 4.3: Reciprocal verification on visual word matches obtained by 'TGC' (BoVW+EWGC).

SOTU **-dc** sotu-dc.conf [**-egc**|**-tgc**] itmtab **-id**|**-vd srpe -d** dstfn

The options for near-duplicate image/video detection are similar to that for retrieval tasks except the retrieval option is fixed to 'srpe'. The settings in the configuration file 'sotu-dc.conf' is the same as retrieval tasks.

For options '-id' and '-vd', '-id' indicates near-duplicate image detection task while '-vd' invokes the near-duplicate video detection routine. For near-duplicate video detection task, the name of video frame must follow the specified format illustrated in Figure 4.2. Otherwise, the output from **SOTU** might be unpredictable. The difference '-vd' from '-id' is an extra step of 2D Hough transform on the retrieved video frames, which has been explained in Section 4.1.4.

4.2.3 Command Option Explained

For option '**-tgc**', **SOTU** performs BoVW retrieval with EWGC [12] on the visual word matches at its first step, then the visual word matches which pass EWGC verification are fed into SR-PE routine to undergo further verification.

Option '**-egc**' prunes false visual word matches in two steps. It first checks the Hamming distance between two matched visual words. In the second step EWGC (as '**-tgc**' option) is adopted to remove false matches further. Comparing to '**-tgc**' option, it saves up the retrieval time while reducing the number of false visual word matches. The visual word matches which pass Hamming and EWGC verification are finally fed into SR-PE routine

to check whether one query is near-duplicate to the reference image.

The file format of the detection output is similar to that of retrieval tasks. There are three columns in each row. The first column shows the image pair (one query and one reference image). The second column and the third column list the *Cosine* similarity score and the number of correct matches between the query and the reference images respectively. Although it is expected that only near-duplicate image pairs are kept in the output file, **SOTU** tries to list all likely near-duplicate pairs. So that the scores in the last two columns play an important role to help one judge how near-duplicate that one image is. Although the list might be mixed with many near-duplicates with low score, comparing with the output from retrieval tasks, the detection already helps filter out large portion of unlikely near-duplicate pairs.

Chapter 5

MISC

5.1 A Quick Start with SOTU

In the package, we also provide an example which allows one to get familiar with **SOTU** quickly. We take retrieving near-duplicate (or similar) image in Oxford building dataset [23] as an example. A two-layer visual vocabulary is already generated for the user. The vocabulary is derived from Harris-Laplace point plus SIFT feature by **LIP-VIREO**.

To try **SOTU** with the attached example, one should first extract SIFT feature for images under the "rimg" and "qimg" sub-folders. There are three images (a tiny subset of the Oxford Building) and two of them are near-duplicate (or similar). User is suggested to use **LIP-VIREO** [20] and type following command to extract keypoint for the reference images.

lip-vireo **-dir** rimg/ **-d** harlap -p sift **-dsdir** rdesc/ **-c** lip-vireo.conf

As indicated by the command line, the local interest point feature will be output to the "rdesc" sub-folder. Similarly, one can extract keypoint features for query image under "qimg" sub-folder.

The second step is the off-line quantization, user is suggested to try BoVW quantization with geometry information which doesn't require sweaty Hamming Embedding training in advance. Since the visual vocabulary is ready, user can simply jump to off-line quantization stage. The command line can be as follows.

SOTU **-ic** sotu-ic.conf **-v tgc -i** itemset/ox_itmref.txt **-s** rdesc/

If command runs smoothly, one can find three files ("ox_sift_tgc_ref_img.txt", "ox_sift_tgc_ref_wgh.txt" and "ox_sift_tgc_ref_bow.txt") in the "databank" sub-folder. Similar quantization can be conducted on the keypoint files (under sub-folder "qdesc") derived from query images. The final step is performing near-duplicate retrieval with the BoVW representation generated.

SOTU **-dc** sotu-dc.conf **-tgc** itemset/ox_itmqry.txt **-id tgc -d** ox_tgc_rslt.txt

5.2 Copyright and Usage terms

5.3 Acknowledgements

We would like to express our sincere thanks to Dr. Hervé Jegou from INRIA-LEAR, who discussed with us about the implementation issue of Hamming Embedding. We also feel grateful for generous help from our previous colleague Yang Liu, who is now with Huawei, Beijing.

5.4 Release Notes

5.4.1 Release Note for V1.10

- Several bugs that cause exceptions have been removed.
- This manual has been revised completely.

5.4.2 Release Note for V1.08

- In visual vocabulary generation, user is allowed to choose whether normalize (l_2 norm) the feature. Please check more details in Section 1.4.1.
- Indexing options 'klevel' and 'kdtree' have been disabled. In viewing low speed gaining comparing with high maintenance cost, these two options have been disabled. The NN indexing is no longer supported by the ANN library [17].
- Bugs in V1.06 which cause abnormal visual word vocabulary generation has been fixed.

5.4.3 Release Note for V1.06

- A bug in RB-Kmeans clustering which causes memory leakage has been identified and removed.
- Function for collecting training samples into matrix has been integrated (refer to **Chapter** 1.4 for more details).

5.4.4 Release Note for V1.05

- 2D Hough Transform has been integrated for near-duplicate video retrieval/detection. Check more details from Section 5.2 and Section 6.1.
- Minor modifications have been made on the Hamming Embedding training. The key words have been changed from '**hem**' to '**medv**' and '**hembed**' to '**embed**'. And there is no need to direct another 'freq' file. The frequency information is also kept in the file indicated by 'embed' option.

5.4.5 Release Note for V1.04

- Function for visual word vocabulary generation has been integrated. See more details in **Chapter** 1.4.

Chapter 6

Appendix

VI.1 Format of the keypoint feature file

numkp	m	d		
p_1	p_2	...	p_m	
val_1	val_2	val_3	...	val_k
val_{k+1}	val_{k+2}	val_{k+3}	...	val_{2k}
...	...	...	...	...
val_{d-k+1}	val_{d-k+2}	val_{d-k+3}	...	val_d
p_1	p_2	...	p_m	
val_1	val_2	val_3	...	val_k
val_{k+1}	val_{k+2}	val_{k+3}	...	val_{2k}
...	...	...	...	...
val_{d-k+1}	val_{d-k+2}	val_{d-k+3}	...	val_d

◇ **numkp**: The number of features in that file

◇ **m**: The number affiliated properties, such as x, y, scale and dominant orientation. Especially, for local interest features. These properties should be put in a fixed order. That is x, y, dominant orientation and scale. If there is no affiliated properties, please set **m** to 0.

◇ **d:** dimension of the feature vector. The feature vector can be put in one row or several rows. Values presented appear in same line are separated by blank character. The feature file name **MUST BE** suffixed with '**.pkeys**'.

VI.2 Format of training matrix for visual vocabulary

n	d			
val_{11}	val_{12}	val_{13}	...	val_{1d}
val_{21}	val_{22}	val_{23}	...	val_{2d}
			.	
			.	
			.	
val_{n1}	val_{n2}	val_{n3}	...	val_{nd}

It is a standard matrix with **n** rows and **d** columns. Each row keeps a **d**-dimensional keypoint feature vector.

VI.3 Format of the two-layer visual vocabulary map

$$
\begin{array}{llllll}
\mathrm{l} & & & & \\
n_1 & \mathrm{VID}_1^r & \mathrm{VID}_{11}^b & ... & \mathrm{VID}_{1n_1}^b \\
n_2 & \mathrm{VID}_2^r & \mathrm{VID}_{21}^b & ... & \mathrm{VID}_{2n_2}^b \\
 & & & . & \\
 & & & . & \\
 & & & . & \\
n_l & \mathrm{VID}_l^r & \mathrm{VID}_{l1}^b & ... & \mathrm{VID}_{ln_l}^b
\end{array}
$$

◇ **l**: The number of visual word on the root layer (the first layer).

◇ n_i: Located in the first column of the file. It is the number visual word in the bottom level that affiliated to visual word **i** on the root layer.

◇ $\mathbf{VID}_i$: Located in the second column. It is visual word Id on the root layer. The Id number starts from 0.

◇ $\mathbf{VID}_{ik}$: Located in the remaining columns (except for column 1 and 2). Visual word Id in the bottom level. The Id number starts from 0.

VI.4 Format of the visual vocabulary

$$
\begin{array}{lllll}
n & d & & & \\
V_1^1 & V_2^1 & V_3^1 & ... & V_d^1 \\
V_1^2 & V_2^2 & V_3^2 & ... & V_d^2 \\
 & & & . & \\
 & & & . & \\
 & & & . & \\
V_1^n & V_2^n & V_3^n & ... & V_d^n
\end{array}
$$

It is a standard matrix with **n** rows and **d** columns. Each row writes one d-dimensional visual word.

VI.5 Format of the Hamming Embedding Medians

$$
\begin{array}{lllll}
n & d & & & \\
V_1^1 & V_2^1 & V_3^1 & ... & V_d^1 \\
V_1^2 & V_2^2 & V_3^2 & ... & V_d^2 \\
 & & & . & \\
 & & & . & \\
 & & & . & \\
V_1^n & V_2^n & V_3^n & ... & V_d^n
\end{array}
$$

It is a standard matrix with **n** rows and **d** columns. **n** is the number of visual word, **d** is the size of Hamming Embedding signature.

VI.6 Format of the Retrieval/Detection output

$\text{image}_{q1} \wedge \text{image}_{r1}$	score_1	socre_2
$\text{image}_{q1} \wedge \text{image}_{r2}$	score_1	socre_2
	.	
	.	
	.	
$\text{image}_{qi} \wedge \text{image}_{ri}$	score_1	socre_2
	.	
	.	
	.	
$\text{image}_{qn} \wedge \text{image}_{rj}$	score_1	socre_2

First column shows the retrieved pair consisting of one query image name and one reference image name connected by '$\wedge$'. Second column shows the prior similarity score and third column shows supplementary score. User can treat these two scores in an arbitrary way.

Bibliography

[1] Sivic, J. and Zisserman, A., "Video Google: A Text Retrieval Approach to Object Matching in Videos," *Proceedings of the International Conference on Computer Vision*, Vol. 2, pp. 1470–1477, oct, 2003.

[2] M. K. and C. Schmid, "Scale and affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, pp. 63–86, 2004.

[3] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal on Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[4] Y. Ke, R. Suthankar, and L. Huston, "Efficient near-duplicate detection and sub-image retrieval," in *ACM Multimedia Conference*, 2004, pp. 869–876.

[5] TREC Video Retrieval Evaluation (TRECVID), in http://www-nlpir.nist.gov/projects/trecvid/.

[6] Linderberg, "Feature detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, no. 2, pp. 79–116, 1998.

[7] Y. Ke and R. Sukthanka, "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors," *Conference on Computer Vision and Pattern Recorgnition*, 2004.

[8] H. Bay, and T. Tuytelaars, and Luc V. Gool, "SURF: Speeded Up Robust Features," *European Conference on Computer Vision*, 2006.

[9] H. Jegou, M. Douze, and C. Schmid. "Hamming Embedding and weak geometric consistency for large scale image search," In *ECCV*, pp. 304–317, oct 2008.

[10] H. Jegou, M. Douze, and C. Schmid. "Improving bag-of-features for large scale image search," In *IJCV*, 2010.

[11] W.-L. Zhao and C.-W. Ngo. "Scale-rotation invariant pattern entropy for keypoint-based near-duplicate detection," *IEEE. Trans. on Image Processing*, pp. 412–423, 2009.

[12] W.-L. Zhao, X. Wu and C.-W. Ngo, "On the Annotation of Web Videos by Efficient Near-duplicate Search," *IEEE. Trans. on Multimedia*, vol. 12, no. 5, pp. 448–461, 2010.

[13] M. Douze, A. Gaidon, H. Jegou, M. Marszatke, and C. Schmid, "INRIA-LEAR's video copy detection system," In *TREVCID*, 2008.

[14] H. Jégou, M. Douze, and C. Schmid, "On the burstiness of visual elements," in *IEEE Conf. on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 1169–1176.

[15] W.-L. Zhao and C.-W. Ngo, "Flip-Invariant SIFT for Copy and Object Detection," *IEEE. Trans. on Image Processing*, (to appear), 2012.

[16] A. Zissermanin and etc, "Visual Geomtry Group" in http://www.robots.ox.ac.uk/~vgg/.

[17] David M. Mount and Sunil Arya, "ANN package", in http://www.cs.umd.edu/~mount/ANN/.

[18] Prof. D. Lowe's home page, in http://www.cs.ubc.ca/~lowe/.

[19] Dorko, "Keypoint Detectors & Descriptors," in http://lear.inrialpes.fr/people/dorko/downloads.html.

[20] Wanlei Zhao's projects, in http://www.cs.cityu.edu.hk/~wzhao2/lip-vireo.htm.

[21] "VIREO", in http://vireo.cs.cityu.edu.hk.

[22] G. Karypis, etc. "CLUTO", in http://glaros.dtc.umn.edu/gkhome/views/cluto/.

[23] J. Philbin and A. Zisserman, "Oxford Buidling", in http://www.robots.ox.ac.uk/~vgg/data/oxbuildings/.