Leiden Institute of Advanced Computer Science (LIACS)

Application of Text Mining on Spatial Visual Sentences

Graduation Master Thesis 2015, Leiden University
Prof. M. Lew

Xiwen Cheng

xcheng@liacs.nl

May 17, 2015

Abstract

Domestic photography has been booming since the introduction of personal devices equipped with cameras. End-users struggle in finding relevant pictures in these ever growing photo collections. Content-based image retrieval (CBIR) solves this and takes away burdens like manual tagging. CBIR methods are often inspired by text mining techniques and concepts. In this project the gap of image and text document semantics analogy is closed with the introduction of *Spatial Visual Sentences*. These visual sentences uses the Bag-of-Words model to construct semantically meaningful word sequences based on segmentation techniques like Superpixels. To illustrate its effectiveness Latent Semantic Indexing among other text retrieval techniques are applied to visual sentences against two datasets: UKBench and MIRFLICKR. Results on MIRFLICKR were not great due to the content variations however recall of objects in UKBench looks promising. Spatial Visual Sentences which is very similar to Visual Phrases in terms of creating word groups has proven to be quite effective for object recalls.

Keywords: Image retrieval, CBIR, text-based search algorithms, text retrieval, information retrieval, Latent Semantic Indexing, Latent Semantic Analysis, K-Means, Self Organizing Map, SURF, SIFT, Superpixels, Watershed, Canny

Acknowledgements

I would like to thank my supervisor Prof. M. Lew at LIACS for his guidance throughout this long thesis journey. I am also grateful for all the support I received in the past few years from family, friends and colleagues at Mendix. There were some tough routes along the way.

Finally I am very thankful to reviewers for their constructive feedback. They are ...

After countless hours of studying, discussions, contemplation, self-reflections and code rewrites my goal has finally been fulfilled.

To quote Henry Ford, a great innovator:

“There are no big problems, there are just a lot of little problems.”

Contents

List of Algorithms

List of Figures

List of Tables

Listings

1. Introduction

The amount of personal pictures is growing rapidly over the past few decades. Flickr.com is a popular image hosting website among the photography community. Figure 1 shows since the inception of Flickr how many public photos were uploaded per month [1]. In the year 2014 1.84 million photos were uploaded on average per day. By May 2015 Flickr had over 10 billion [2] images. Not just the younger generation but also elders possess portable devices like smart-phones, tablets and dedicated cameras which enables them to take photographs within a hand reach. As a result users end up with gigantic collections of photographs.

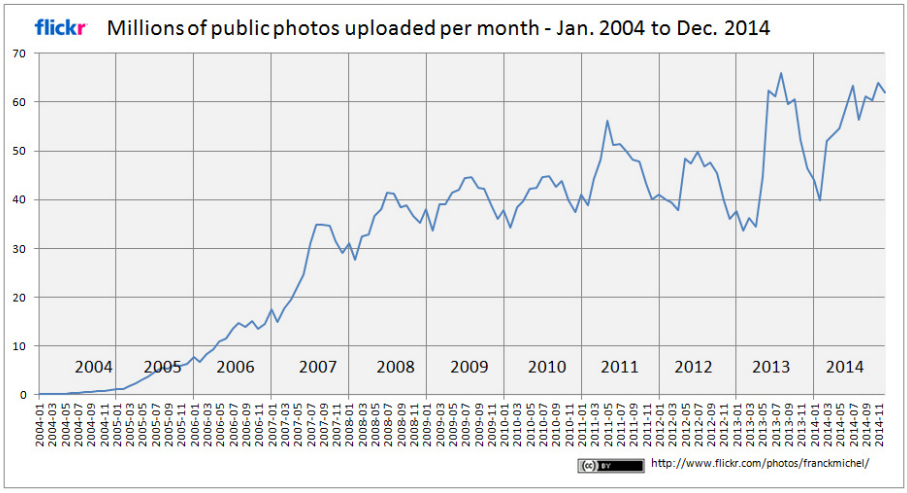


Figure 1: Millions of public photos uploaded per month to Flickr between 2004 and 2014

Due to the sheer amount of digital media most users do not bother with manual tagging or cataloging. There is therefore great demand for systems that can aid users in retrieving relevant images from big image collections with least possible input from the end-user. This research focuses on the former: retrieve relevant images based on a sample input image. Though this area has been researched extensively the Bag-of-Words (BoW) model as basis has been proven to be successful [61, 37, 65]. Just like BoW most derivatives [54, 30, 68, 64, 70, 56] including our method are inspired by text retrieval methods. Our main contribution is to complete the semantic analogy between an image and a text document by introducing *spatial visual sentences*.

The spatial visual sentence concept introduced here fits in the analogy (Figure 2) between image and text document in semantic granularity drawn by Zheng et al [70]. Images are encoded as a 2-dimensional array of *pixels*. This primitive is analog to *letters*

[1] `https://www.flickr.com/photos/franckmichel/6855169886/in/photostream/`
[2] `http://blog.flickr.net/en/2015/05/07/flickr-unified-search/`

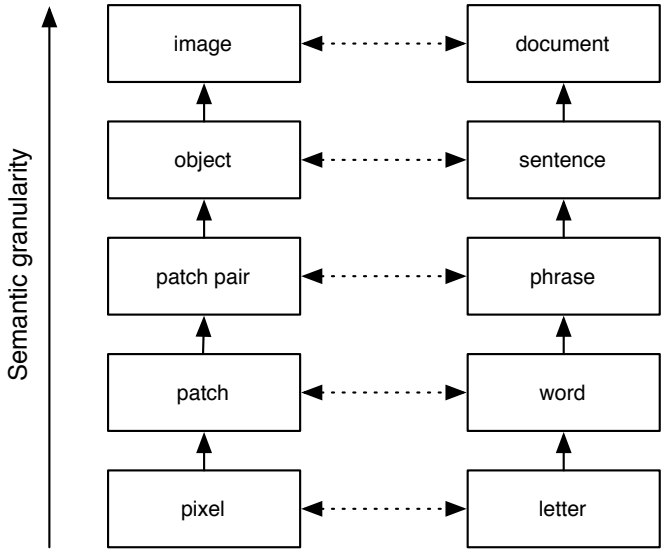


Figure 2: Analogy between image and text document in semantic granularity [70]

of an alphabet. First level of abstraction is feature descriptor [13] like SIFT and SURF (Section 3.1) are examples of methods which can represent a *patch* (pixels region) as a feature vector. Because such vector is computed based on a collection of pixels adjacent to each other it is similar to composing a *word* using letters. *Phrases* can be constructed by grouping words based on some criteria like visual word co-occurrences [64] or Euclidean distances. The final semantic gap is closed with *sentence* which in images context corresponds to an *object* or group of words with emergent visual semantics. This grouping method relies on image segments that are inspired by Gestalt principles [24, 35, 26]. We coin this approach as *Spatial Visual Sentences* because it decomposes an image into meaningful group of words with respect to visual recognition of figures.

This thesis is organized as follow:

- Section 2 presents an overview of relevant text mining methods and concepts inspired by *Natural Language Processing*. Some of them are applied during the different phases of the proposed Spatial Visual Sentences algorithm.
- Section 3 outlines state of the art image processing techniques that are also used by our algorithm. Of each technique two or more variations are discussed for comparison in section 5.
- Section 4 dives into the details on how to compute Spatial Visual Sentences from pixels. It also addresses ways to compare visual sentences or images with each other using three similarity measures.

- Section 5 presents how the proposed algorithm performs in terms of recall and precision against two popular datasets and a mixture of algorithm configurations.
- Section 6 summarizes this thesis and draws some conclusions based on the results presented in section 5.
- Finally section 7 sheds some light on possible future work that could improve the proposed algorithm in terms of speed and accuracy.

Full source code including subsets of the datasets are publicly available [3].

[3] `https://github.com/xiwenc/cbir-invenio`

2. Text Mining

Text mining or *knowledge discovery from text* deals with computer assisted text analysis. Hotho et al summarized three possible definitions [19] of *text mining* in the research areas: Information Retrieval, Natural Language Processing and Information Extraction. In this article we will consider the following definition:

> Text mining is the application of algorithms and methods from the fields machine learning and statistics to texts with the goal of finding useful patterns.

The mining process can be simplified into two main phases:

- Preprocessing: Transform documents into structured data taking into account the syntax and semantics of natural languages where possible.
- Classification: Group documents or parts of it based on a similarity measure to speed up retrieval of similar content.

The rest of this section will describe these phases in more detail. Most text mining approaches apply the *Bag-of-Words* model to represent text documents. It extracts keywords from a document. The importance of a word within a document can be quantified using several known models like *probabilistic model*, *logical model* and *vector space model*.

2.1. Text preprocessing

A text document consists of one or more *pages*. Each page is composed of an ordered list of *phrases* separated by a *punctuation mark*. Multi-page documents can be easily transformed to single-page by concatenating all pages. For the sake of simplicity we will only consider single-page documents from now on forward. Each phrase is in turn an ordered list of *words* separated by *white spaces*. A word is an ordered list of terminals also known as characters or in more general: *symbols*.
The set of terminals specific to language is called the *alphabet*. And the set of words that can be composed using the alphabet based on grammar and spelling rules of a language is coined the *vocabulary*. We will use the *English* language as reference to illustrate some concepts in the rest of this section.

2.1.1. Stop Words Removal

Stop words is a group of words used to filter out before or after processing of natural language data. The idea is to reduce complexity of subsequent computational tasks by reducing the vocabulary size and discarding *non-important* data. Examples in English are *the*, *a*, *an* and *is*. It's important to find a balance in what to discard in order to improve performance because it could severe accuracy of phrase searches. Often the most frequently used words are chosen as they do not contribute to distinctiveness of query results.

2.1.2. Synonym, Stemming and Lemmatization

In linguistic there are several ways to group/summarize multiple words into a single term based on their semantics, syntax and context in which it is used. The word *better* is a *synonym* of *good* and vice versa. These two words have no commonalities in their construction at all yet they have similar meaning. Hence synonyms are language specific and is often presented as a static lookup table.

Words with similar composition like *cats* and *catty* can be mapped to the root *cat*. This process for reducing inflected or sometimes derived words to their stem word, base form or root is called *stemming*. Most algorithms that determine the stem word of a given target word uses lookup table that maps to inflicted forms or more flexible rules like *suffix-stripping algorithm*.

A more complex form of determining the stem of a word is *lemmatization*. The *part of speech* is first detected using the context in which the word occurs. Next the stemming rules are applied depending on the word category. For instance the word *meeting* can be a base form of a noun or the verb *to meet* depending on the context:

- *During last meeting*
- *We are meeting again tomorrow*

Additional linguistic preprocessing specifics to semantics add limited value to Bag-of-Words because co-occurence of terms serve as an automatic disambiguation for classification purposes[28]. Nonetheless progress[20] is being made to exploit these concepts:

- **Part-of-Speech tagging**: classify the category of the word based on the context it is used in e.g. noun, verb, adjective, etc.
- **Text chunking**: create groups of adjacent words in a sentence e.g. *blue balloon*.
- **Word Sense Disambiguation**: Resolve ambiguity using the context in which a word is used. For instance *bank* could mean *financial institution* or *border of a lake or river*.

2.1.3. Keyword Selection

The words which make up the vocabulary can further be reduced by applying selection algorithms. One way of doing this is to select keywords based on their *entropy*[31]. The entropy of a word t is defined as:

$$W(t) = 1 + \frac{1}{\log_2|D|} \sum_{d \in D} P(d,t) \log_2 P(d,t) \quad \text{with} \quad P(d,t) = \frac{\text{tf}(d,t)}{\sum_{l=1}^{n} \text{tf}(d_l,t)} \tag{1}$$

Where:

$$
\begin{aligned}
D &= \text{the set of documents} \\
T &= \text{the set of all different terms } \{t_1, ..., t_m\} \text{ occuring in } D \\
\text{tf}(d,t) &= \text{frequency of term } t \text{ in document } d
\end{aligned}
$$

Words that occur in many documents get low entropy value. In an ideal situation all the values are computed and sorted. The top n can then be selected to be the index terms. Greedy approach[19] can also be applied to reduce required computational resources for large document collections.

2.1.4. Vector Space Model

Given a computed set of index terms the original document collection can be transformed to a more efficient data structure suitable for searching. The vector space model can represent documents as numerical feature vector with m-dimensions: $w(d) = (x(d,t_1), ..., x(d,t_m))$. Where m is the number of distinct terms and $x(d,t)$ is the weighting function. The simplest form to encode the vector is using *booleans*:

$$
x_b(d,t) = \begin{cases} true & \text{if } t \text{ occurs in } d \\ false & \text{if } t \text{ does not occur in } d \end{cases} \tag{2}
$$

Salton et al[52] proposed a weighting scheme $x_s(d,t)$ which combines term frequency $tf(d,t)$, inverse document frequency $idf(t)$ and length normalization factor. By normalizing the weights against the document length each document has equal chance of being retrieved.

$$
x_s(d,t) = \frac{\text{tf}(d,t) \times \text{idf(t)}}{\sqrt{\sum_{j=1}^{m} \text{tf}(d,t_j)^2 (\text{log}(N/n_{t_j}))^2}} \tag{3}
$$

Where:

$$
\begin{aligned}
\text{log}(N/n_t) &= \text{inverse document frequency idf(t)} \\
N &= \text{size of document collection D} \\
n_t &= \text{number of documents in D containing term t}
\end{aligned}
$$

The similarity between two documents d_1 and d_2 can be computed using Euclidean distance:

$$
S(d_1, d_2) = \sqrt{\sum_{k=1}^{m} |x(d_1,t_k) - x(d_2,t_k)|^2} \tag{4}
$$

As the name already suggests: large distance value means less similar. While identical documents would yield a distance of 0. This function can be used to retrieve similar

documents based on a query encoded as another document d_q. Thus a search action equals to computing $S(d, d_q)$ for all $d \in D$.

2.2. Clustering

In data mining there are two main categories of structuring data: by means of classification and clustering. The former method requires a predefined set of classes and optionally example data that belongs to each class. Sebastiani [53] discussed a plethora of approaches using machine learning. Examples are Naïve Bayes Classifier, Nearest Neighbor Classifier, Decision Trees and Support Vector Machines. Clustering algorithms tackles the issue from the opposite direction: partition the input data into k clusters based on similarity measures. Given the set of feature vectors described in section 2.1.4 it is possible to automatically cluster the data. The rest of this section outlines two well known and effective clustering algorithms namely *k-means* and *Self Organizing Map*.

2.2.1. k-means

In the field of statistics and data mining, k-means is the most popular clustering method. There are several variations employing heuristics in order to improve the original computational complexity of NP-hard in finding the global optimum. The standard k-means algorithm starts by randomly initialize the centroids using the input dataset to be clustered. Next it iteratively assigns elements to partitions based on nearest centroid and recomputing the centroids using partition centers (Algorithm 1). The loop ends when the centroids doesn't change or after a fixed number of iterations.

Algorithm 1 Standard k-means

Input: Set D, distance measure $dist$, number k clusters
Output: Partition P of k disjoint subsets of D such that $\cup_{p \in P} = D$ and $\cap_{p \in P} = \emptyset$

1: Initialize by randomly selecting k elements from D as starting centroids $C = \{c_1 ... c_k\}$
2: **repeat**
3: Assign $d \in D$ to p_i if closest to centroid c_i with respect to $dist$
4: Recalculate centroids C
5: **until** The centroids C are stable
6: **return** P = $\{p_1, ..., p_k\}$

2.2.2. Self Organizing Map

Self Organizing Map (SOM), also known as Kohonen map [25] is a special architecture of neural networks that clusters high-dimensional data vectors. The clusters are arranged in a low-dimensional topology such that clusters nearby each other are more

similar to each other than those further away. Unsupervised training of the network is possible because the learning process is aided by a similarity measure. The network structure has two layers (Figure 3).

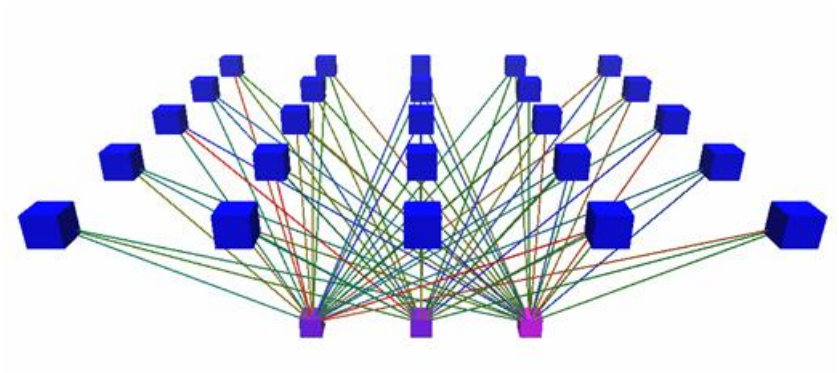


Figure 3: Kohonen network structure: output layer (top) and input layer (bottom) [59]

Neurons in the input layer correspond to the input dimensions n of a document d. This input is presented as $\vec{I} = [t_1, ..., t_n]$ where t_i is a term in d. The output layer consists of k nodes each corresponding to a cluster center. All neurons in the input layer are connected to all nodes in the output layer. Thus each node W contains n weights $\vec{W} = [w_1, ..., w_n]$. Algorithm 2 depicts how the learning process works.

SOM starts by randomly assigning small initial values to each $\vec{W_i} \in W$ (Line 1). For every iteration a random input vector $\vec{I}$ selected from the training set D (Line 6). The Best Matching Unit (BMU) γ is determined by computing all the Euclidean distances between $\vec{I}$ and each $\vec{W_i}$ then select the node W_i with the lowest distance (Lines 7-10). Next a radius σ is computed based on the iteration number y and a time constant λ which starts with σ_0 and converges to 1 over time (Line 11). Lines 12-16 updates neighbors of BMU γ within radius σ based on influence rate Θ and learning rate L which decays exponentially. The algorithm finishes the training when the given number of iterations m has been reached.

A trained Kohonen network classifies a new input vector $\vec{I'}$ the same way BMU is computed.

2.3. Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an indexing and retrieval method that uses *singular value decomposition* to identify patterns in term relationships and concepts in unstructured text. It is based on the principle that words used in the same contexts tend to have similar meanings (semantics). A good indication of the effectiveness of LSI is Google [38, 27, 10, 5] relying on it for information retrieval. Because of its mathematical approach it is independent of language and is therefore interesting to apply it to images. For details and analysis of LSI please refer to other publications [60, 42].

Algorithm 2 Self Organizing Map: Learning process

Input: W nodes, training set D, max iterations m, map radius σ_0, learning rate L_0
Output: trained nodes W

Randomly initialize each $\vec{W}_i$
Iteration counter $y = 0$
Time constant $\lambda = m/log(\sigma_0)$
repeat
    Increase $y = y + 1$
    input vector $\vec{I}$ = select_random(D)
    for all Nodes W_i **do**
        $\delta_i = S(\vec{I}, \vec{W}_i)$ (See equation 4)
    end for
    Best Matching Unit $\gamma = W_i$ with lowest δ_i
    Neighborhood radius $\sigma = \sigma_0 e^{-y/\lambda}$
    for all Nodes W_i within radius σ **do**
        Influence rate $\Theta = e^{-S(\vec{\gamma}, \vec{W}_i)/2\sigma^2}$
        Learning rate $L = L_0 e^{-y/\lambda}$
        Update $\vec{W}_i = \vec{W}_i + \Theta L(\vec{I} - \vec{W}_i)$
    end for
until $y >= m$
return W

3. Content-Based Image Retrieval

Content-Based Image Retrieval (CBIR) is the application of computer vision techniques to image retrieval problems. Unlike *text-based image retrieval* which exploits the availability of textual information, CBIR purely relies on the ability to make sense of the actual content. Visual perception of the human brain is extremely sophisticated which enables it to identify visuals with semantics like recognizing written text or *kids playing with a beach ball*. There is a known saying *"A picture is worth a thousand words"* that clearly highlights the short coming of annotated images. Mainly because the perception of one person may not be the same as someone else's. Also nowadays the sheer amount of personal digital imaging makes it an impossible task to properly annotate all the data. Hence there is a great demand for good CBIR systems.

A basic CBIR system (Figure 4) composes of two phases. First an image collection is preprocessed and transformed into a data structure suitable for searching. Then a user can present to the system a query image in which it will return a list of similar images using this data structure.

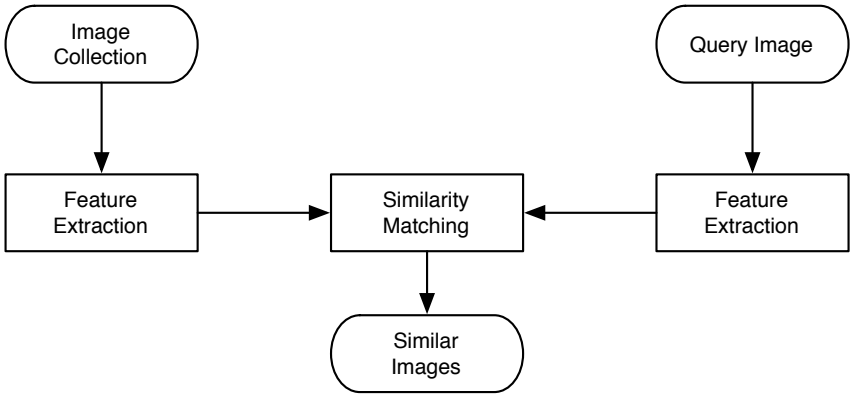


Figure 4: Generic Content-Based Image Retrieval system

In this section we will dive into methods used during the preprocessing phase and possible data structure representation. The concepts and methods described were selected based on their relevance to be building blocks of the algorithm outlined in section 4. Please consult surveys [58, 57, 30, 29] on CBIR for more complete overview of existing research on this topic.

3.1. Image Features and Detectors

Images are colored pixels arranged in a two-dimensional plane. Examining these pixels individually to make "sense" of them is extremely computationally expensive. Therefore raw image information is often transformed into an abstract and compressed representation known as *image features*. Image feature defined as interesting part of an image

is the basis of many computer vision algorithms. The performance of these algorithms greatly depend on how good the features are. Feature effectiveness can be expressed in terms of the following properties[58]:

- *Repeatability*: High probability of identifying similar parts between two images taken of the same scene with different viewing conditions like angle and illumination.
- *Distinctiveness/informativeness*: Has to be descriptive enough to distinguish detected features from each other.
- *Locality*: Describes an image patch taking into account the size to mitigate potential occlusion problems.
- *Quantity*: Number of detected features should be sufficiently large to enable object detection.
- *Accuracy*: The same feature must be detected during location or scale change.
- *Efficiency*: Feature must be detected and computed within reasonable time crucial for real-time applications.

The most basic image features are based on color, texture, shape and location. *Global features* try to describe the image using a single vector [65] while *local features* computed per interest point in an image are capable of recognizing objects. Based on comprehensive surveys on CBIR [29, 30, 9] we will focus only on more state-of-the-art local feature descriptors which are affine invariant. This allows matching of feature vectors under different transformations like scale and rotations.

Prior to extracting descriptors the area or point of interest must be identified using a *detector*. There are four main categories of detectors: *Edges*, *interest points*, *regions of interest* and *ridges*. Canny[8] and Sobel[55] are well-known examples of edge detector operators which find sets of points with strong gradient magnitude in the image. Blob detectors[15, 12] focus on regions of interest in images which might be too smooth for corner detectors to find points of interest[50, 49].

Local image patches extracted from the detected features are known as *feature vector* or *feature descriptor*. SIFT[33, 32] and SURF[4] are known to perform well in terms of repeatability[17, 23, 51, 36, 57]. Hence the rest of this section is devoted to give a quick overview on how these two methods work. Please refer to the original publications [32, 4] for full details. Juan et al [23] concluded SIFT is slow and doesn't perform well with illumination changes while it is invariant to rotation, scale changes and affine transformations. SURF is fast and performs as good as SIFT but is not stable to rotation and illumination changes.

3.1.1. Scale-Invariant Feature Transform

This section briefly outlines the *Scale-Invariant Feature Transform* (SIFT) algorithm introduced by Lowe[32]. It starts by identifying candidate locations and its corresponding scale that are invariant to scale change. The detection uses *scale-space extrema* in the *Difference-of-Gaussians* (DoG) function convolved with the image. Each sampled point is compared to its eight neighbors in the current scale and nine neighbors of both scale-up and scale-down (Figure 5). It is considered to be an candidate if it is a local extrema.

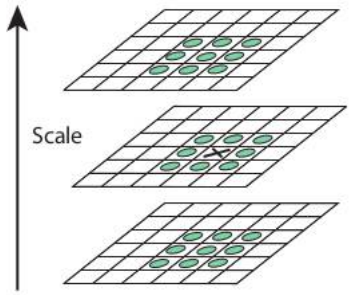


Figure 5: Finding local extrema by comparing neighboring pixels across scales [44]

The set of found keypoints are then refined by rejecting low contrast extrema and edge keypoints because the DoG function has strong responses along edges.

A region around a keypoint is chosen based on the scale found in previous steps. Of this region an orientation histogram with 36 bins is created weighting the gradient magnitude and Gaussian-weighted circular window. Top 80% highest peaks of the histogram are selected yielding multiple keypoints of the same location and scale but different directions.

A keypoint descriptor is represented by a $4 \times 4 \times 8 = 128$ dimensional feature vector. The 16×16 array around the keypoint is divided into 16 sub-regions of 4×4. For each sub-region an 8 bin orientation histogram is created (Figure 6).

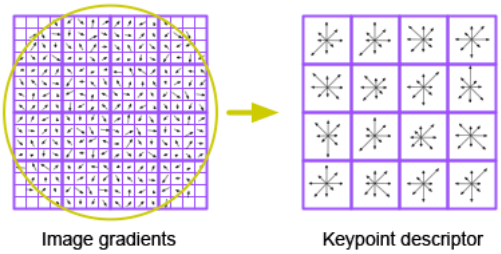


Figure 6: 128 dimensional feature vector computed from image gradients [3]

3.1.2. Speeded Up Robust Features

Bay et al [4] presented a high performance scale- and rotation-invariant interest point detector and descriptor named *Speeded Up Robust Features* (SURF). SURF approximates Difference of Gaussian with *Box filter*. Figure 7 shows an example of the approximation. Convolution with box filter can be calculated with integral images. Therefore it is possible to parallelize the computation for different scales. Furthermore SURF relies on determinant of Hessian matrix for both scale and location.

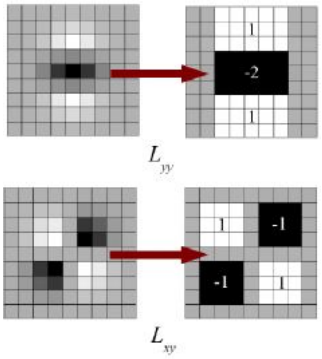


Figure 7: Difference of Gaussian approximation with Box Filter [45]

Orientation assignment is computed using wavelet responses in horizontal and vertical directions for a neighborhood of size $6s$ where s is the *scale*. Next the wavelet responses are weighted with a Gaussian then plotted as illustrated in figure 8. The dominant orientation is estimated by the sum of all responses within a window of angle 60 degrees. The longest vector over all windows defines the orientation of the interest point.

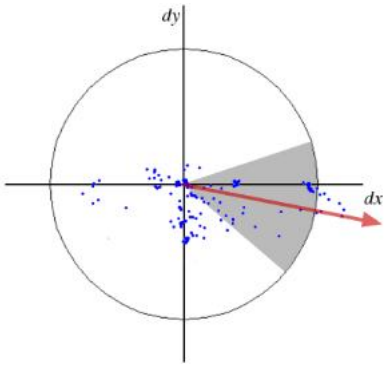


Figure 8: Sliding orientation window to detect dominant orientation [45]

The SURF descriptor describes an interest area of $20s$ around the detected keypoint. This area is divided into 4×4 subregions. For each subregion a vector $v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$ is calculated based on 5×5 samples. Where dx and dy

are wavelet responses in horizontal and vertical directions respectively. And $|dx|$ and $|dy|$ are the absolute values of the responses. By concatenating v of all 4×4 a single descriptor vector is represented with 64 dimensions.

3.2. Image Segmentation

Image segmentation is the process of partitioning an image into multiple segments. Segments are groups of pixels which *belong* together based on some characteristics like colors, textures, position, intensities, etc. Ideally each segment represents a real-world object. There are numerous surveys [62, 43, 34, 16, 63, 66] on image segmentation that give very thorough overview of existing methods. There are two main type of methods: boundary-based use the discontinuity property of pixels in relation to its neighbors while region-based apply the similarity property of nearby pixels. Recent proposals [63] in improving segmentation performance suggest in combining both methods for more accurate segments.

In the rest of this section the methods *SLIC Superpixels* and *Watershed Transform* are discussed. Watershed requires the aid of markers to compute the segments. These methods compared by Al-Kubati et al [2] are *Canny edge detector* and *Otsu Threshold*. To conclude the section the mentioned methods are applied on two sample images. The code used to are based on implementations of segmentation methods in *OpenCV* and *Scikit-image*.

Other good segmentation alternatives are Graph Cut [63, 11] and Felzenszwalb's algorithm [14] based on pairwise region comparison. Due to the scope of this project we will not discuss all of them.

3.2.1. SLIC Superpixels

Achanta et al [1] introduced *Simple Linear Iterative Clustering* (SLIC) which can generate compact nearly uniform superpixels. Superpixel is a concept originally developed by Xiaofeng Ren and Jitendra Malik [47]. Each superpixel is a segment obtained from low-level grouping process.

According to Ren et al [46] a superpixel map has many desired properties:

- Computationally efficient: reduces complexity of images by using the pixel groups instead of individual pixels.
- Representationally efficient: a single pixel has at most 8 adjacent neighbors while superpixels can have many more which is more efficient to represent in models as n relations.
- Perceptually meaningful: all pixels in a superpixel most likely has some uniform properties in common like color and texture.

The SLIC algorithm is summarized in algorithm 3. It is somewhat similar to *k-means* with regard to the body-construction of the algorithm. It starts by initializing k cluster centers then iteratively assign pixels to the best matching centroid and recompute cluster centers until some acceptable residue error E is reached. The distance measure takes into account the color and pixel position. For full details please refer to the original publication [1].

Algorithm 3 SLIC Superpixels segmentation [1]

1: Initialize cluster centers $C_k = [l_k, a_k, b_k, x_k, y_k]^T$ by sampling pixels at regular grid steps S.
2: Perturb cluster centers in an $n \times n$ neighborhood, to the lowest gradient position.
3: **repeat**
4: **for all** cluster center C_k **do**
5: Assign the best matching pixels from a $2S \times 2S$ square neighborhood around the cluster center according to the distance measure.
6: **end for**
7: Compute new cluster centers and residual error E {$L1$ distance between previous centers and recomputed centers}
8: **until** $E \leq$ threshold
9: Enforce connectivity.

3.2.2. Watershed Transform

The watershed transform method [6] belongs to the region-based class. It was inspired by geography. Consider gray-scale image to be a landscape which is flooded with water. At points where different water areas meet each other a dam is built forming watershed lines. In order for the algorithm to perform well the *marker*-based variation was introduced. Markers essentially guide the algorithm in finding the segments otherwise it would over-segment.
Listing 1 shows how watershed can be used in *OpenCV*. The function prototype is $watershed(image, grayed, edges, min_ratio, max\_count)$:

1. $image$ is the original input image.
2. $grayed$ is the gray scale of the input image.
3. $edges$ is an image of the same dimension as input image of which is used as basis for the markers
4. min_ratio is the minimal ratio of a contour in $edges$ which must be exceeded by $\frac{contour\_area}{total\_image\_area}$. A value less or equal to 0 yields all detected segments indifferent of the contour area.
5. max_count denotes the maximum number of segments to return from $image$ giving bigger segments more priority.

Sections 3.2.3 and 3.2.4 show how $edges$ can be constructed. A markers mask is constructed by tracing the contours in $edges$ and drawing them on a black backgrounds. Finally the watershed algorithm is applied to the original image together with markers.

Roerdink et al [48] looked into speeding up watershed transform by means of parallelization. They concluded the speedup to be achieved is pretty modest because of a global operation being the bottleneck.

3.2.3. Canny Edge Detector

The Canny edge detector operator uses multiple stages to detect a wide range of edges in images. We will not dive into the details of the algorithm but only highlights the parameters. Interested readers are advised to read the original publication [8] instead.

The function prototype of Canny edges in listing 1 is $canny(image, gaussian\_ksize, threshold1, threshold2)$. It accepts four parameters:

1. $image$ is the input image.
2. $guassian\_ksize$ is the filter size of the Gaussian kernel.
3. $threshold1$ is a threshold of hysteresis.
4. $threshold2$ is also a threshold of hysteresis.

Setting the thresholds too low will miss details and too high will miss important information.

3.2.4. Otsu Threshold

Thresholding itself can be considered to the simplest form of image segmentation. For every pixel in the input image if the intensity is lower than a given threshold τ it converts the pixel to black otherwise white. Otsu's [41, 18] method finds a threshold τ where the sum of the foreground and background spreads is at its minimum.

The code in listing 1 also function prototype $otsu(image, gaussian\_ksize)$. It accepts two arguments:

1. $image$ is the input image.
2. $gaussian\_ksize$ is the filter size of the Gaussian kernel.

3.2.5. Comparison

Figures 9 and 10 show the different results of two example images by applying SLIC Superpixels; Canny and Otsu together with the watershed algorithm. For these examples

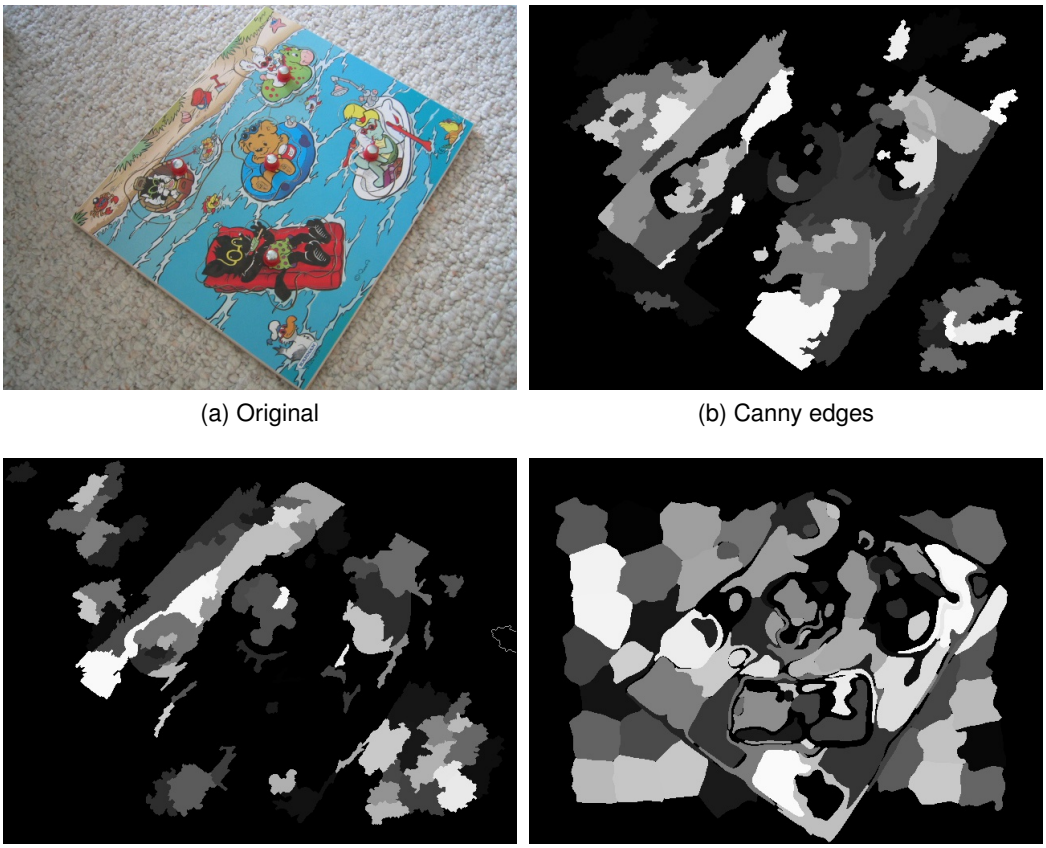
the following parameter values were used:

$gaussian\_ksize$	$=$	$(7, 7)$	Brush size used by Gaussian blur
$threshold1$	$=$	20	First hysteresis threhold for Canny
$threshold2$	$=$	100	Second hysteresis threhold for Canny
min_ratio	$=$	0.0001	Minimal ratio of the segment size (Section 3.2.2)
max_count	$=$	100	Maximum number of segments to derive

The values were chosen purely based on human trial-and-error evaluation. Each segment is filled with a unique gray scale color (at most 255 variations). Without quantitative or reference data it's hard to conclude which method performs best. Hence we'll try to judge them based on intuition instead. In figure 9 the better results are computed by SLIC and the worst with Otsu. Same ranking applies to figure 10. It is very unlikely there is a single set of parameters that works perfectly for all possible input images. Section 5 will present results of different parameter values and elaborate on what values to choose and why.

3.3. Spatial Visual Phrases

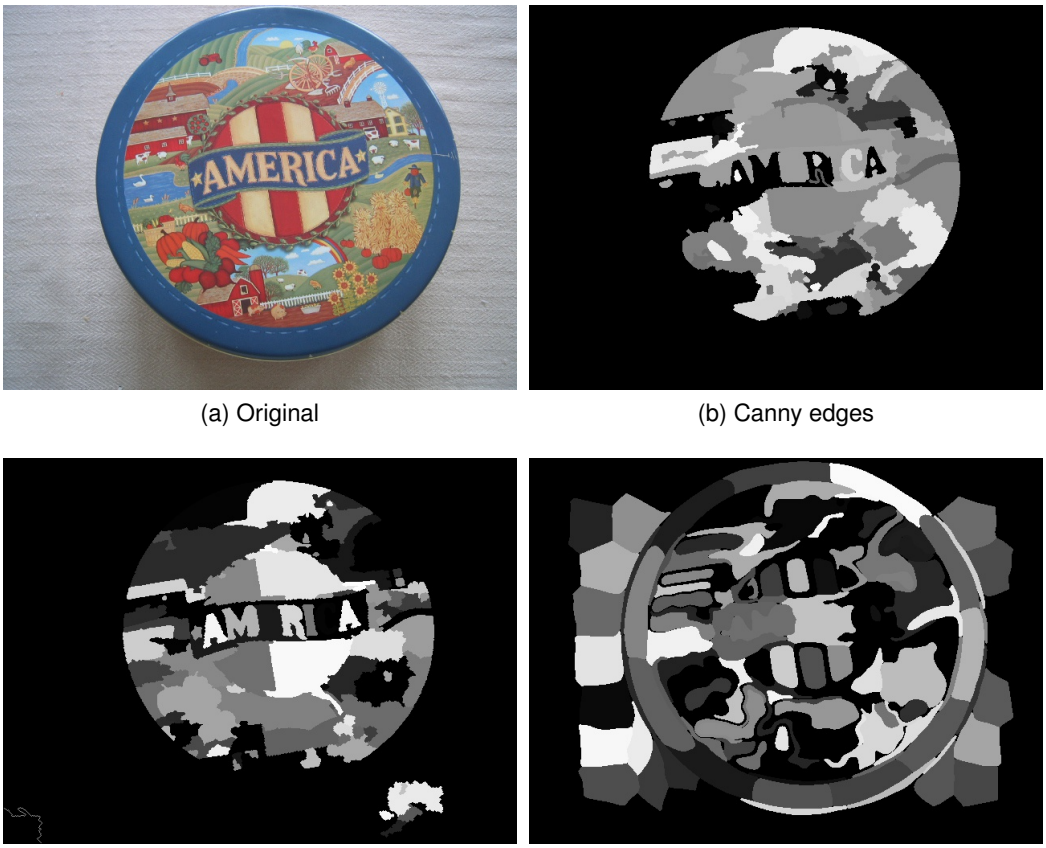
If we consider a *pixel* of an image to be a *letter* in an alphabet then an *image patch* (being a set of pixels) is equivalent to a *word*. In computer vision it's commonly referred to as *visual word*. The Bag-of-words (BoW) model using the analogy of visual words has shown to perform pretty well as image retrieval method. It can be implemented using image features as described in section 3.1.
A major drawback of BoW is the lack of spatial relationship information of the words in the original image. Recent research [69, 67, 68] has been conducted to close this *semantic* gap and therefore improve search accuracy by introducing *spatial visual phrases*. Spatial visual phrases are collections of k visual words. The phrases are constructed based on the co-occurrence of two visual words. In section 4 we show how to push the text analogy further by introducing *visual sentences*.



(a) Original (b) Canny edges

(c) Otsu Threshold (d) SLIC Superpixels

Figure 9: Segmentation of a board game



(a) Original (b) Canny edges

(c) Otsu Threshold (d) SLIC Superpixels

Figure 10: Segmentation of a bottle cap

4. Spatial Visual Sentences

To apply text retrieval algorithms on images based purely on content the concept of *spatial visual sentences* is introduced. A sentence can be defined[4] as:

"a sequence of words capable of standing alone to make an assertion, ask a question, or give a command, usually consisting of a subject and a predicate containing a finite verb "

Each spatial visual sentence is an ordered sequence of features (words) that belong together if they reside (spatial) in an image segment (visual semantics). Such sentence could be interpreted as representing an object with a sequence (1-dimensional) of visual features. The process of computing these sentences is inspired by concepts from text retrieval (Section 2) and CBIR (Section 3). The main goal of spatial visual sentences is to capture the semantics of images in a concise representation to allow for fast online querying. This is achieved by reducing the vast amount of mid-level descriptors (low-level being pixels themselves) to a higher level semantically meaningful groups.

Given a collection C of images all the points of interests and their descriptors are extracted using a feature detector and extractor like SURF and SIFT discussed in section 3.1. Then a sample of p descriptors are used as input for a clustering algorithm like k-means or SOM. The resulting k cluster centers are used as the vocabulary B. For each image d in C segments are computed using SLIC superpixels, Otsu-watershed or Canny-watershed (Section 3.2). Each sentence is an ordered sequence of terms from vocabulary. A term (or word) is selected if the term is within the boundaries of the segment. The order of the terms is determined using the position the term was found in the original image. Algorithm 4 shows the full algorithm how an album A is computed given C as input. The rest of this section dives into the details of the main components/phases.

4.1. Visual Word Vocabulary

In order to reduce the complexity of searching through groups the descriptors in a group is mapped to a close representation mimicking the concept of having a vocabulary of a language. Given a collection of images all their features are detected and extracted. Feeding these features into a clustering algorithm like k-means or SOM yields k clusters (Figure 11). Each cluster center is considered to be a word (or term) in the vocabulary. The vocabulary size depicts the discriminative level and thus the effectiveness of the library to be searched: $0 \leq k \leq |C|$ where $|C|$ is the total number of features detected in the collection C. For $k = 0$ means no discriminative and thus equals to random search. And $k = |C|$ would be too discriminative resulting in zero speed-up. k should be chosen

[4] `http://www.wordreference.com/definition/sentence`

Algorithm 4 Spatial Visual sentences

Input: Collection C containing n images
Output: Album A

Create album A
for all Image in C **do**
    Compute and extract descriptors
end for
Create list of descriptors Q by randomly selecting p descriptors from the images
Compute the vocabulary B with size k by clustering Q into k clusters
Compute and flag noisy words in B
for all Image d in C **do**
    Compute the *words* in d using B
    Identify the set of *segments* S in d
    for all Segment s in S **do**
        Compute sentence v by selecting word w if its position is in s and w is not noisy
        v is sorted by position (Starts from top-left and ends in bottom-right)
    end for
    Add d with augmented sentences to A
end for
return A

such that there is a good trade-off between accuracy and speed. We propose the following equation:

$$k = |C| \cdot \gamma \tag{5}$$

Where γ is the desired speed up expressed as a ratio of the original $|C|$ count.

For large collections the clustering process could be slow. Existing publications [22, 40] suggest random sampling strategy is simple yet quite effective. Sampling can be done in two phases: images or descriptors. Image sampling is faster because a fraction of computational resources for descriptor extraction can be omitted and thus more *efficient*. However descriptors sampling is better in terms of descriptiveness of the resulting vocabulary because all descriptors have equal probability of being selected.

4.2. Visual Word Construction

Depending on the chosen feature descriptor each descriptor is represented as a vector $\vec{v}$ of undefined dimensions (64 for SURF and 128 for SIFT). During the visual word mapping phase each descriptor is matched against the provided vocabulary B to find the best matching word $w \in B$. w_i is stored as a natural number being the index of the word in B. Thus each image at this stage is represented as a vector $\vec{d} = (w_0, ..., w_n)$ allowing duplicates of w_i in $\vec{d}$ because a word can occur multiple times in a document.

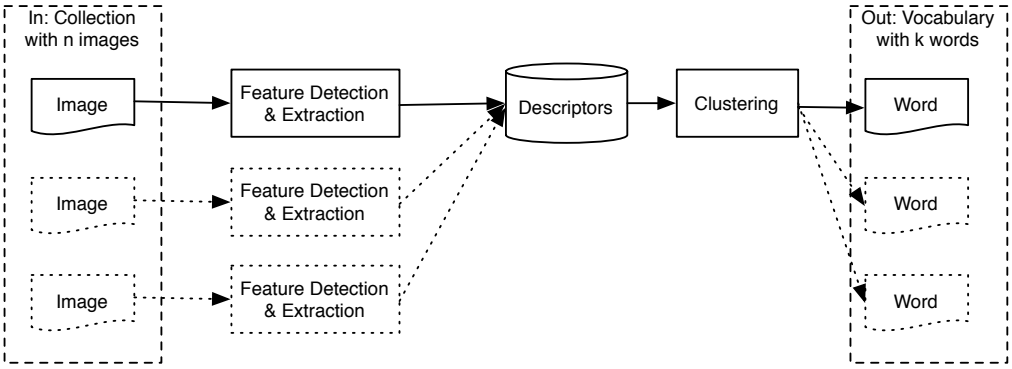


Figure 11: Computing a vocabulary from a collection of n images

After all images have been mapped to their respective $\vec{d}$ each word in B is counted how often it occurs in the collection C. Words with high occurrences are considered to be stop-words (Section 2.1.1) and thus discarded from all $\vec{d} \in C$. Rare words (low occurrence) are also discarded because they can be considered as outliers and thus barely improve retrieval effectiveness. In the implementation these words are flagged as *noise* and kept in the vocabulary for future mappings. Efficiency is improved by skipping noisy words during construction of visual sentences.

4.3. Visual Sentence Construction

Each image in collection C is segmented into F fragments. Each fragment $f_s \in F$ is defined as a set of vectors which together make up a contour. A segment is a vector $\vec{s} = (w_0, ..., w_n)$ for all $w_i \in B$ if w_i is within the boundaries of f_s. The elements in $\vec{s}$ are sorted by the position of w_i found in the original image. It begins with top-left and ends with bottom-right. This representation does not work well with transformed image like rotation or mirroring because they would yield different ordering. Depending on the application rotation can be fixed during preprocessing of scenery photographs using the horizon as reference. Or alternatively align with direction of main light source if there are shadows present. Figure 12 shows how an image is transformed into a list of sentences.

4.4. Similarity Measures

Using the computed album A where each image d is represented as a list of sentences $S = (\vec{s_0}, ..., \vec{s_n})$ it is possible to apply generic sequence matching algorithm and other language independent methods on two sequences. We will start with *difflib.SequenceMatcher*[5] that implements *Ratcliff/Obershelp pattern recognition* algorithm. Finally TF-IDF (Section 2.1.4) and LSI (Section 2.3) implementations from *gensim* [6] are

[5] `https://docs.python.org/2/library/difflib.html`
[6] `https://radimrehurek.com/gensim/tut3.html`

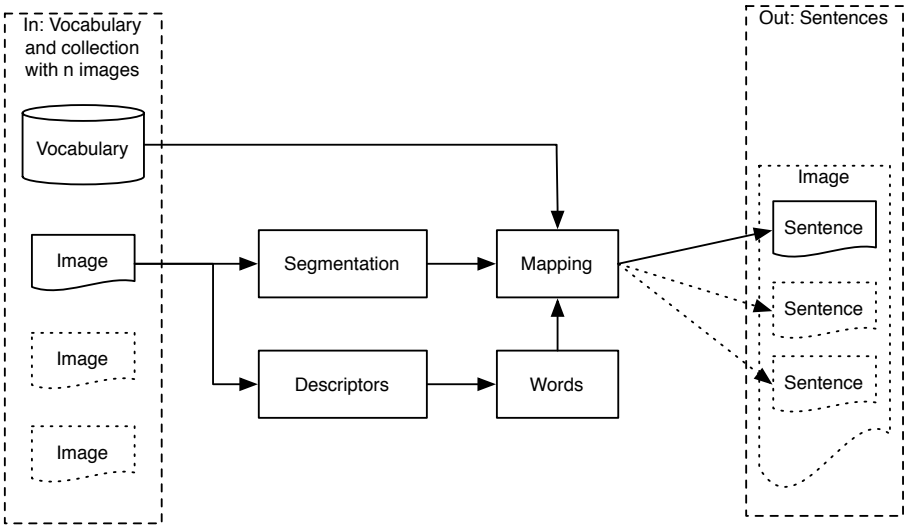


Figure 12: Sentences of an image are computed using a global vocabulary, feature descriptors and segments derived from the image

applied to verify the effectiveness. Similarity values are floats in the range of $[0...1]$. 0 being nothing in common and 1 means identical sequences.

4.4.1. Ratcliff/Obershelp

Ratcliff/Obershelp algorithm [7] computes the similarity of two sequences. The ratio ρ is computed with

$$\rho = \frac{2M}{T}$$

where M is the number of matches and T the total number of elements in both sequences. Matching characters are those in the longest common subsequence plus, recursively, matching characters in the unmatched region on either side of the longest common subsequence. Listing 2 shows how this algorithm is used to calculate the similarity of two images $image1$ and $image2$. The ratio ρ_s of a sentence $s \in image1$ is the maximum ratio of s against each sentence $s' \in image2$. The effective ratio ρ' is the average of all the sentence ratios ρ_i of $image1$.

4.4.2. Gensim: TF-IDF and LSI

Gensim [7] models work with a corpus. Corpus is a collection of vectors representing a document collection. An album A as computed in section 4 can be transformed to a corpus in two ways:

1. CorporaOfImages: all words in the image are considered to part of a single vector

[7] `https://radimrehurek.com/gensim/index.html`

2. CorporaOfSentences: all words in the sentence of a particular image are considered to part of a single vector

Both methods have been implemented in listing 3. The methods implement the function *similarity(model, image)* which compute the similarity of a target image I^T against the corpus. *CorporaOfImages* is simple because each image is equivalent to a vector. However *CorporaOfSentences* is slightly more complicated because the computation is done on sentences level. It keeps a mapping of $\vec{v} \rightarrow d$ where $\vec{v}$ is a vector in the corpus and d is the image $\vec{v}$ belongs to. The effective ratio $\rho'(d)$ where $d \in A$ is the best model M similarity value of all sentences $s \in I^T$ compared against all sentences $s' \in d$. Note that short $s \in I^T$ (lower than *min_length*, default is 5) are ignored. Listing 3 also includes wrapper implementations of two similarity models: TF-IDF and LSI. These models combined with a corpus can be used to compute the similarity of each image in A against I^T.

5. Results and Evaluation

This section outlines how *Spatial Visual Sentences* are tested. We will evaluate the effectiveness with *precision* and *recall* of 4 randomly chosen images from 2 datasets. These two measures are defined as:

$$Precision = \frac{|\{relevant\} \cap \{retrieved\}|}{|\{retrieved\}|}$$

is the fraction of retrieved documents that are relevant to the search. And

$$Recall = \frac{|\{relevant\} \cap \{retrieved\}|}{|\{relevant\}|}$$

is the fraction of documents that are relevant to the search has been retrieved.

$$F_1 = 2 * \frac{precision \cdot recall}{precision + recall}$$

combines both *precision* and *recall* as the *harmonic mean* called F-measure or F-score. In order to compare the performance between configurations we also introduced *Average F-Score* which is the average F-Score of the 4 target images.

The rest of this section presents the possible algorithm configurations, datasets and finally the results.

5.1. Datasets

To verify the effectiveness we will run the algorithm against 2 datasets:

1. UKBench [39]: A collection of 10200 images. All images have dimensions 640×480. The set contains $\frac{10200}{4} = 2550$ groups. Each group of 4 images (Figure 13) show the same object with different lighting, orientation and/or scale. Results are evaluation by checking if the retrieved images belong to the same group as the target image.

2. MIRFLICKR [21]: Dataset of 25000 Flickr[8] images. Each image comes with a list of human annotated *tags*. Evaluation of the results is done by checking the tags of the target image against those of the retrieved images. Figure 14 shows a few samples from the set. Unlike UKBench the image sizes, illumination, sceneries and compositions in MIRFLICKR are not uniform so it is very representative for real-world scenarios.

The results presented does not utilize the full set of the original images. Due to high complexity only the first 400 images of each set were considered. Also note that the target image is selected from the same indexing set. Thus the target image itself should

[8] `https://www.flickr.com/explore`

always be the best match. To keep both *precision* and *recall* values within reasonable range the number of retrieved documents is always 4. This equals to selecting the top 4 from the results because the algorithm returns a list of all images in the collection sorted by similarity. Worst case scenario precision equals $\frac{0}{4} = 0$ and recall equals $\frac{0}{0} = undef$ but we present it as 0.

The ground truths per data set is computed based on their characteristics: Relevant images in UKBench are groups of 4 and an individual image in MIRFLICKR has user-defined tags. Sections 5.3 and 5.4 discussed these in more detail together with the acquired results.

(a) ukbench00016.jpg (b) ukbench00017.jpg (c) ukbench00018.jpg (d) ukbench00019.jpg

Figure 13: A group of 4 images showing the same subject from UKBench

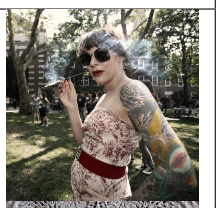
Name	Tags	Image
im1.jpg	governorsisland, cigarette, tattoos, smoke, red, lipstick, dress, sunglasses, shades, belt	
im10.jpg	hemlock, ring, blanket, brooklyn, tweed, giant, doily, eco, wool, cascade	
im100.jpg	farmer, cisauk, ricefield, d300, 18200vr, nikkor, teeje, serpong, westjava, indonesia, platinumphoto, abigfave	
im101.jpg	anawesomeshot, theperfectphotographer, lightning, thunder, flash, spiritofphotography, damniwishidtakenthat	

Figure 14: Four random images from MIRFLICKR with their respective tags

5.2. Algorithm Configurations

Most phases of the algorithm presented in section 4 could have multiple implementations. In this section we summarize the variations creating a list of configurations of the algorithm to test against the datasets. Table 1 lists per phase and parameter which variations are available or interesting to evaluate. Strictly there are $2 \cdot 2 \cdot 3 \cdot 3 \cdot 2 \cdot 3 \cdot 3 = 648$ possible configurations. 20 candidates were handpicked and ran on a test machine described in Table 2.

Phase/Parameter	1	2	3
Feature Descriptors	surf	sift	
Vocabulary clustering	kmeans	som	
Segmentation	otsu	canny	slic
Similarity measure	ratcliff	tfidf	lsi
Corpus mode	images	segments	
Vocabulary size	500	1000	5000
Noise ratio (words discarded)	0.10	0.2	0.4

Table 1: Parameter value and implementation variations

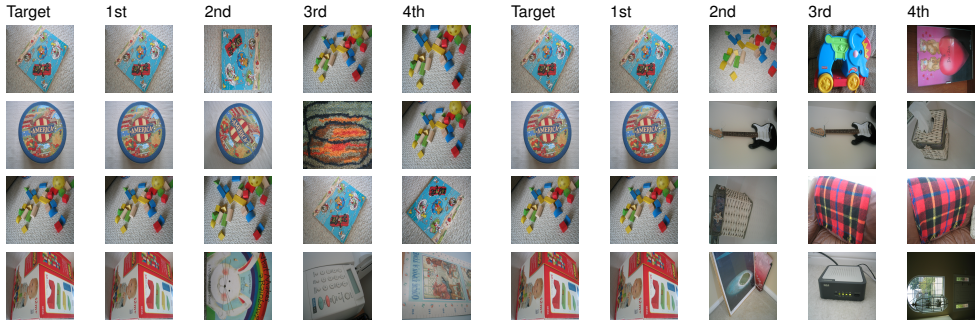
Hardware	**Description**
CPU	Intel i7 920 clocked at 3.6Ghz
Number of logical cores	8
Memory	12 GB
Software	**Version**
Operating System	Debian Jessie (3.16.7-ckt4-3)
Python	2.7.9
scikit-image	0.11.3
scipy	0.15.1
numpy	1.9.2
gensim	0.11.1-1
OpenCV	2.4.8

Table 2: Test machine specifications

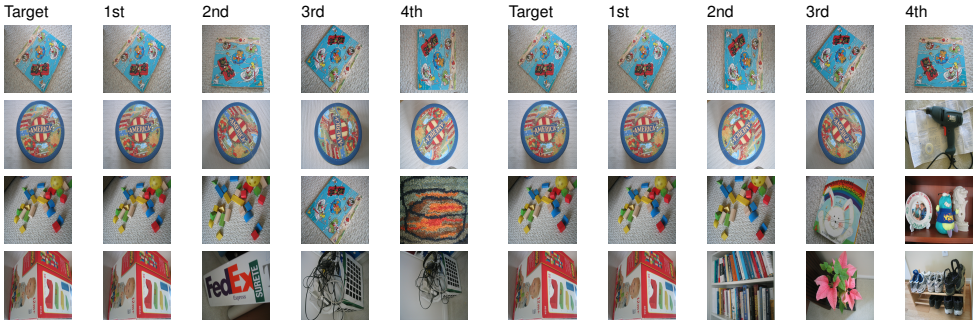
5.3. Retrieval from UKBench

Four random images were selected from four different groups to validate the effectiveness of the preselected configurations. The target images are shown in figure 15 together with few retrieved images. The set of *relevant* images is derived from the filename: every 4 images of the sorted dataset is a sample of the same object. Given the relevant and retrieved images the three measures can be computed and shown in figure 16. Because F-score is the harmonic mean of both precision and recall we will discuss the former only. Overall the results are pretty good. The best performing configuration is *surf-kmeams-slic-lsi-images-5000-02*. It seems the bigger the vocabulary size the better the F-score. It makes sense because a large vocabulary increases de-

scriptiveness however it requires more computational resources.



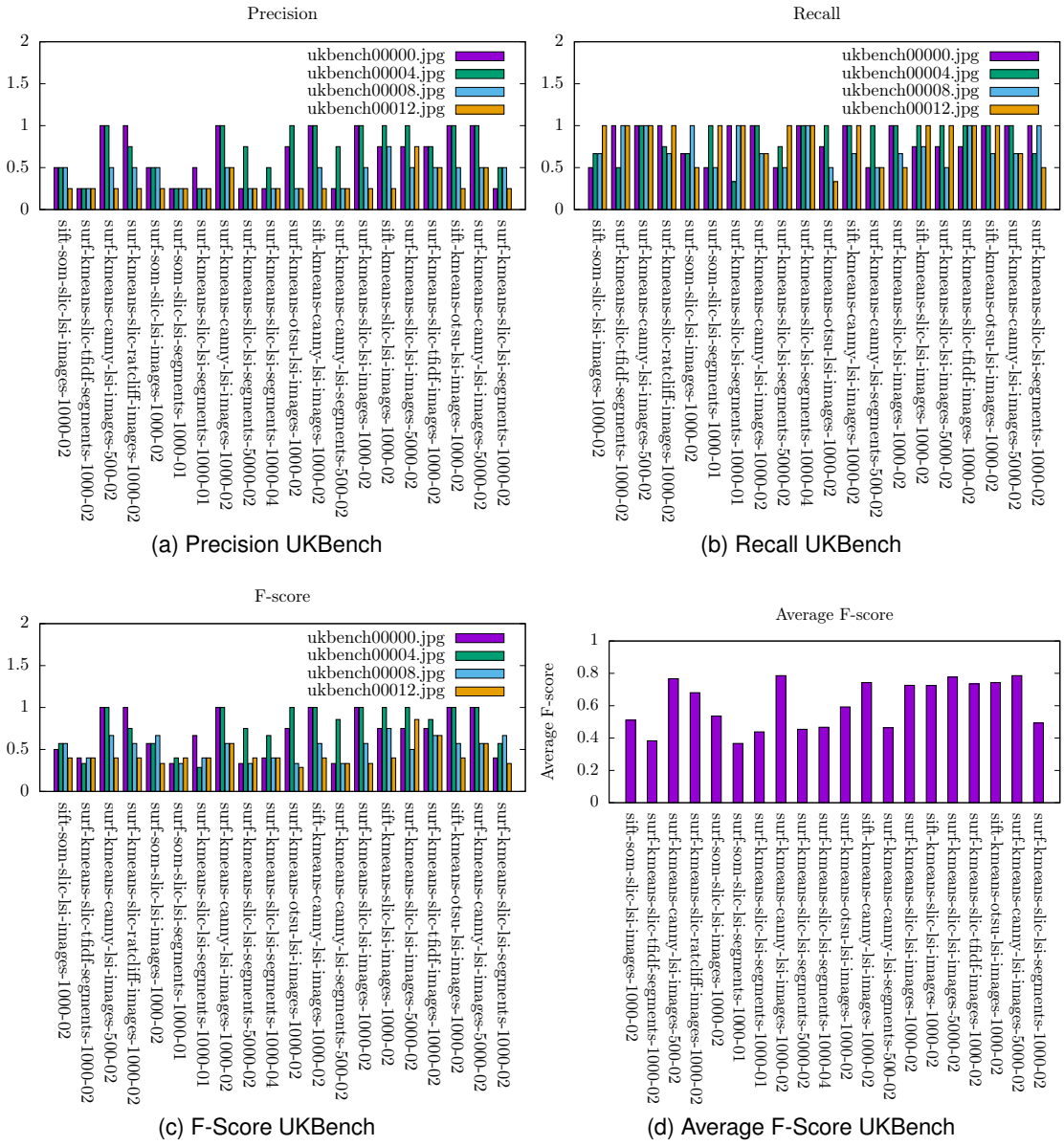
(a) sift-som-slic-lsi-images-1000-02 (b) surf-kmeans-slic-tfidf-segments-1000-02



(c) surf-kmeans-canny-lsi-images-500-02 (d) surf-kmeans-slic-ratcliff-images-1000-02

Figure 15: Top 4 retrieved images of 4 configurations from UKBench

The *segments* corpus mode is less accurate than full image summarization (surf-kmeans-slic-tfidf-images-1000-02 vs surf-kmeans-slic-tfidf-segments-1000-02). Table 3 compares the similarity averages of the retrieved images per target image. *Segments* tend to yield higher averages. Other configurations with *segments* also do not perform well. We suspect this could be related to how the similarity measure is computed (Listing 3). Because the similarity value is based on the *best* matches per visual sentence it could be misled by short sentences together with small vocabulary. The averaged F-Score graph (Figure 16d) somewhat confirms this observation as the lowest averages uses *segments* together with mixture of the configuration options.



(a) Precision UKBench

(b) Recall UKBench

(c) F-Score UKBench

(d) Average F-Score UKBench

Figure 16: Precision, recall, F-Score and average F-Score of 4 images from the UKBench dataset

Image	surf-kmeans-slic-tfidf-images-1000-02	surf-kmeans-slic-tfidf-segments-1000-02
ukbench00000.jpg	0.505602302379	0.500583332847
ukbench00004.jpg	0.357798258541	0.712458482012
ukbench00008.jpg	0.484299128875	0.537217399455
ukbench00012.jpg	0.296052606019	0.622530572814

Table 3: Average similarities of *surf-kmeans-slic-tfidf-images-1000-02* and *surf-kmeans-slic-tfidf-segments-1000-02*

5.4. Retrieval from MIRFLICKR

The MIRFLICKR dataset is a mixture of amateur and professional photographs. Unlike UKBench with its groups of 4 samples of the same subject this dataset is very unpredictable. Each sample in MIRFLICKR accompanies a list of user defined tags. Using the tags as metadata we can classify whether two pictures are related. For instance if the target image contains the tags *explore* and *flower* any other image with either *explore* or *flower* as tag is considered to be a candidate. This method is used to compute the set of *relevant* images. Instead of blindly selecting some images from the dataset as targets we select 4 targets from the top 4 most frequent tags. Figure 17 shows for our subset 20 most frequent tags. The selected target images are shown in figure 18 and cover the respective tags: *explore*, *blue*, *flower* and *red*.

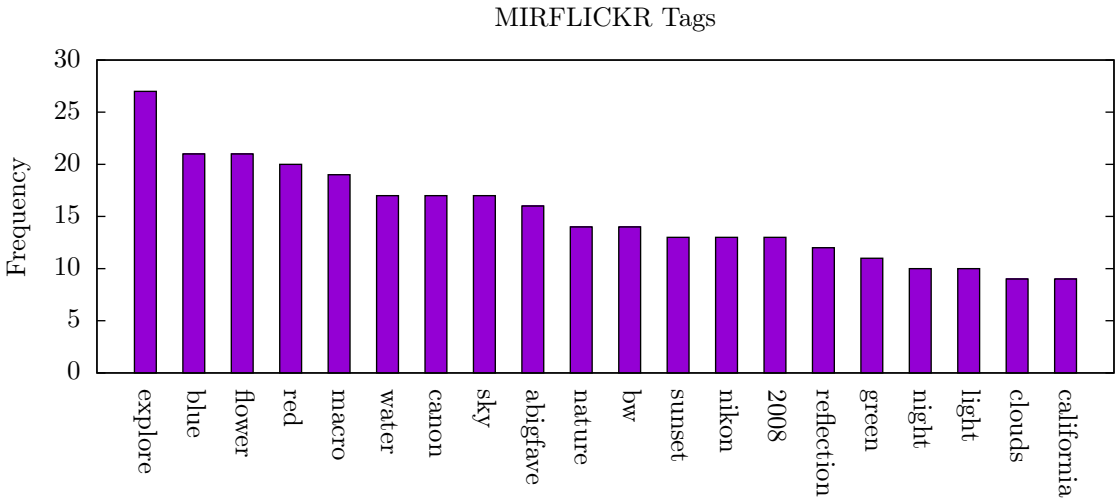


Figure 17: Top 20 tags and their frequencies from a MIRFLICKR subset

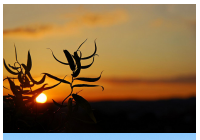
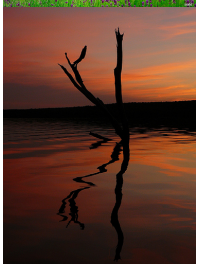
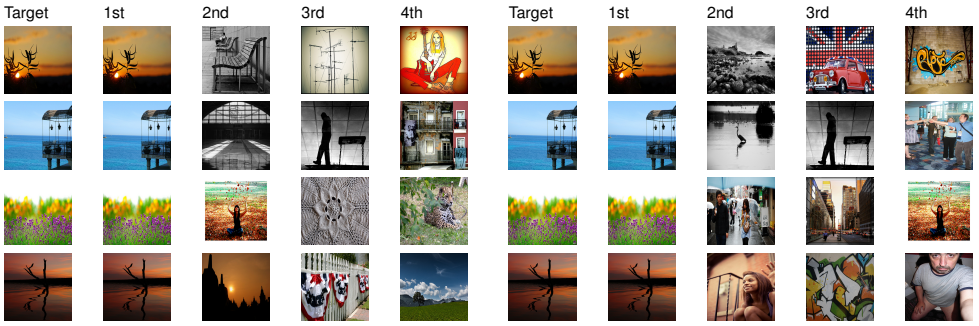
Name	Tags	Image
im102.jpg	nature, athome, d80, flora, sunset, explore, goldstaraward, goldsealofquality, ...	
im109.jpg	water, travelerphotos, travel, sightseeing, photographer, panoramas, panorama, pano, nikonstunninggallery, nature, mediterranean, landscape, island, hot, downtown, blue, ...	
im125.jpg	lavenderfield, lavender, herb, flower, rainyseason, redhot-pokerbokeh, torchlilyboke	
im79.jpg	okmulgee, oklahoma, sunset, red, drippingspringslake, tree, water, reflection, blueribbonwinner, abigfave, explore	

Figure 18: Four target images (explore, blue, flower, red) from MIRFLICKR with their respective tags

Figure 19 shows for four configurations the top 4 retrieved images per target image. And figure 20 shows the performance of all selected configurations. Most notably like UKBench, the target image itself was the most similar. This implies all configurations are capable of recalling the original image. Though this might sound trivial it is nice to have this affirmation.

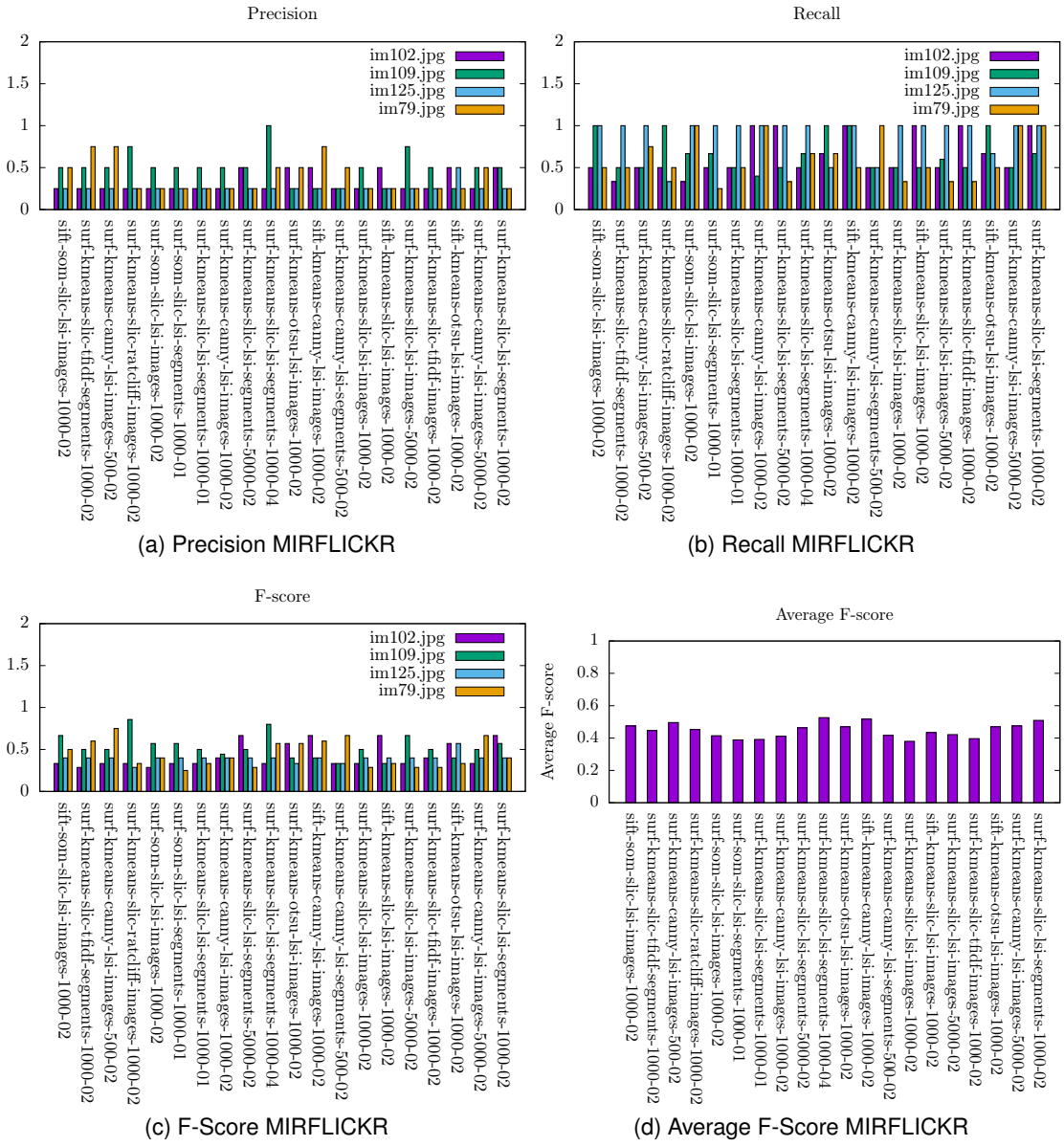
(a) sift-som-slic-lsi-images-1000-02 (b) surf-kmeans-slic-tfidf-segments-1000-02



(c) surf-kmeans-canny-lsi-images-500-02 (d) surf-kmeans-slic-ratcliff-images-1000-02

Figure 19: Top 4 retrieved images of 4 configurations from MIRFLICKR

Taking a closer look at figure 20 all configurations were able to find at least another relevant image. Some of them even found two. *surf-kmeans-slic-lsi-segments-1000-04* even managed to find at least 3 (there could be more because we capped the results to the top 4 retrieved images; the first being the self-image). That still holds for the averaged F-Score (Figure 20d). Other good performers are *surf-kmeans-canny-lsi-images-500-02*, *sift-kmeans-canny-lsi-images-1000-02* and *surf-kmeans-slic-lsi-segments-1000-02*. This is remarkable because *surf-kmeans-canny-lsi-images-500-02* only uses half the vocabulary size as the other ones with fairly little performance degradation.



(a) Precision MIRFLICKR

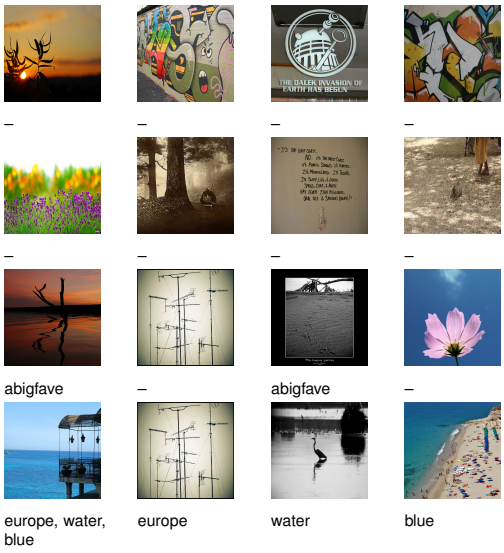
(b) Recall MIRFLICKR

(c) F-Score MIRFLICKR

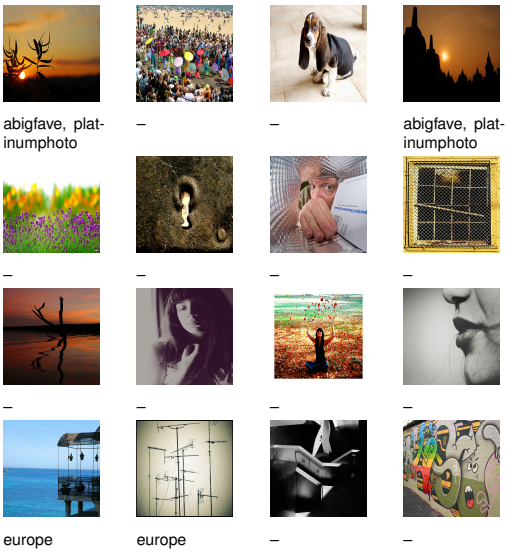
(d) Average F-Score MIRFLICKR

Figure 20: Precision, recall, F-Score and average F-Score of 4 images from the MIRFLICKR dataset

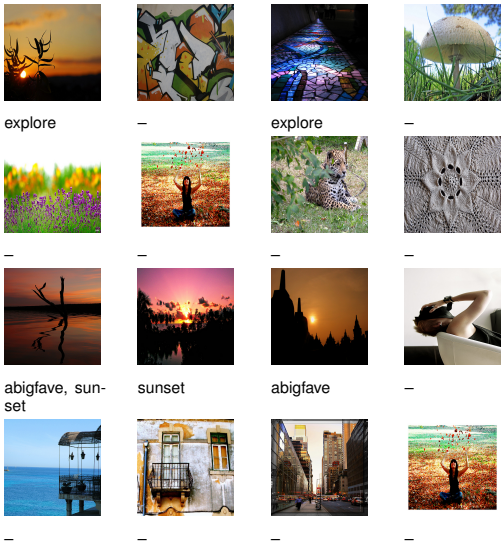
Lets investigate the results of best four configuration in more detail. The retrieved images and their matching tags are shown in figure 21. Out of the lot only the images of *abigfave*, *blue* and *sunset* seem to make sense. Besides the ability to recall the original image it somewhat fails to retrieve semantically similar candidates in which we could automatically confirm with user-defined tags.



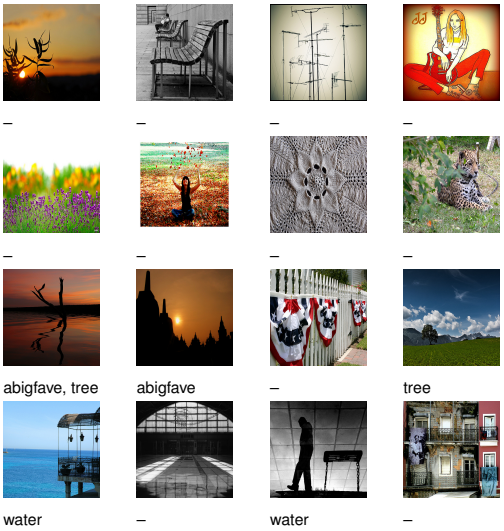
(a) surf-kmeans-slic-lsi-segments-1000-04



(b) surf-kmeans-slic-lsi-segments-1000-02



(c) sift-kmeans-canny-lsi-images-1000-02



(d) surf-kmeans-canny-lsi-images-500-02

Figure 21: Retrieved images with matching tags per configuration from MIRFLICKR

6. Conclusions

In this thesis we outline methods related to text mining and image retrieval followed by a proposal in applying these methods to collectively implement *Spatial Visual Sentences*. This method aims to fill in the analogy of sentences in the context of text documents. While words equal to visual descriptors we derive a limited set of dictionary from the global word list applying concepts like stop words. Using this vocabulary sentences are composed using proximity of visual features (vocabulary words) within an image segment. We looked into several methods in partitioning an image into semantically meaningful segments. Having closed this semantic gap in the analogy between images and text documents we can leverage existing text algorithms to improve accuracy and hopefully also speed of offline retrieval of visually identical fragments between images.

The method has been tested against two popular datasets namely UKBench and MIRFLICKR in combination with a wide variety of configurations of the proposed algorithm. There are several configurations available because the algorithm is split up into different phases and each phase cane be implemented differently. Overall we got promising results with UKBench because the dataset is focused on retrieval of the same object. MIRFLICKR composed of user-taken photo's from all over the world posed to be quite a challenge to retrieve relevant images. This is mainly because the photographs share very little visual feature similarities. Therefore we can conclude that our algorithm is best suited to recall objects from a large collection. Although the recall was not always 100% accurate it was able to retrieve the candidates taken from different angles and scales just fine.

In a nutshell we are pretty satisfied with the results. Because the scope of this project was purely closing this semantic gap hence we did not focus on delivering a high-performance implementation[9]. Therefore speed has not been investigated thoroughly. In section 7 we describe possible continuations of this project.

[9] https://github.com/xiwenc/cbir-invenio

7. Future work

This project's focus was an attempt in closing the semantic gap of the analogy between text document and image. The results look promising for object recall however in the future we would like to look into the following questions and possible improvements:

1. **Augment word/sentence representation with metadata**: Words in our vocabulary are defined by clustering feature descriptors. Accuracy could be improved further by extending the word vector or sentence vector with extra information like color histogram, geolocation of the picture if present and time period when the photo was shot. These would increase descriptiveness and potentially reduce time complexity if this meta information is used as a quick filter. For instance if the target picture was taken during a road-trip through Europe in June 2014 the algorithm can narrow down the search space within that time period and location.

2. **Family photo album as dataset**: We tested the algorithm against UKBench and MIRFLICKR datasets of which fairly good results are achieved in the former but it pretty much failed with the latter. This implies it is best suited for object recalls. However to confirm this hypothesis we need a different kind of dataset: A perfect candidate would be a family photo album. The balance between objects (family members, cars, clothes, etc...) and scenery (beach, mountains, winter, house interior, etc...) would yield better matches because of more feature descriptor similarities.

3. **Similarity measure**: The current proposed similarity measure is very optimistic because it uses similarities of best matches. As a result short sentences are more likely to yield high similarity values. We have tackled this issue by specifying a minimal sentence length in the algorithm. However a more weighted measure based on for example length of the sentence could give more accurate results.

4. **Relevance feedback application**: Spatial visual sentences can recall objects pretty accurately from large collections. To take this to our advantage users can query the system by means of sub-images. For instance the user Jim is looking for a picture of Melissa and him at the beach. A simple relevance feedback application would be: Present 10 random images from the collection to the user. If Jim recognizes a fragment of relevance like *Melissa* he marks that particular region of interest. In the next iteration more relevant results would be presented to Jim; hopefully more picture rank up with Melissa in it. After a few iterations Jim would be able to identify the picture he sought. This example shows it is possible to construct complex queries and the sentences model fits pretty well in it.

A. Appendix: Code

```
import cv2
import numpy
import operator

from logger import logger


def watershed(image, grayed, edges, min_ratio, max_count):
    """ Applies watershed algorithm to 'image' with markers derived
        from 'edges'
    Args:
        image: original image
        grayed: grayed and optionally blurred version of 'image'
        edges: a binary image
        min_ratio: only contours in 'edges' with an area bigger are used as
                   markers
        max_count: maximum number of segments to derive
    Returns segments, markers, count
    """

    markers = edges.copy()
    _, markers1, _ = extract_segments(
        grayed,
        markers,
        min_ratio=min_ratio,
        max_count=max_count
    )
    markers32 = numpy.int32(markers1)
    cv2.watershed(image, markers32)
    watersheded = cv2.convertScaleAbs(markers32)
    _, edges = cv2.threshold(
        watersheded,
        1,
        255,
        cv2.THRESH_BINARY_INV
    )
    segments, markers, count = extract_segments(
        grayed,
        edges
    )
    return segments, markers, count


def canny(image, gaussian_ksize=(7, 7), threshold1=20, threshold2=100):
    """ Computes Gaussian blurred grayscale and Canny edges
    Args:
        image = image array; use cv2.imread(...) to load from file
        gaussian_ksize = filter size e.g. (5, 5)
        threshold1 = first threshold of the hysteresis procedure
        threshold2 = second threshold of the hysteresis procedure
    Returns (grayscale, edges)
```

```
    """
    if len(image.shape) == 3:
        grayed = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    elif len(image.shape) == 2:
        grayed = image
    else:
        raise Exception("Unsupported input image")

    blurred = cv2.GaussianBlur(grayed, gaussian_ksize, 0)
    edges = cv2.Canny(blurred, threshold1, threshold2)
    return (grayed, edges)


def otsu(image, gaussian_ksize=(7, 7)):
    """ Computes Gaussian blurred grayscale and Otsu threshold edges
    Args:
        image = image array; use cv2.imread(...) to load from file
        gaussian_ksize = filter size e.g. (5, 5)
    Returns (grayscale, edges)
    """
    grayed = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(grayed, gaussian_ksize, 0)
    ret, edges = cv2.threshold(
        blurred,
        0,
        255,
        cv2.THRESH_BINARY + cv2.THRESH_OTSU
    )
    return (grayed, edges)


def seg_otsu_watershed(image, min_ratio=0.0001, max_count=100):
    grayed, edges = otsu(image)
    return watershed(image, grayed, edges, min_ratio, max_count)


def seg_canny_watershed(image, min_ratio=0.0001, max_count=100):
    grayed, edges = canny(image)
    return watershed(image, grayed, edges, min_ratio, max_count)


def seg_slic(image, min_ratio=0, max_count=100):
    from skimage.segmentation import slic
    from skimage.segmentation import mark_boundaries
    from skimage import img_as_ubyte

    grayed = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    segments = slic(image, sigma=5)
    black = numpy.zeros(image.shape, numpy.uint8)
    edges_sci = mark_boundaries(black, segments, color=(1, 1, 1))
    edges_gray = cv2.cvtColor(img_as_ubyte(edges_sci), cv2.COLOR_BGR2GRAY)
    ret, edges = cv2.threshold(
        edges_gray,
        80,
```

```
106         255,
107         cv2.THRESH_BINARY
108     )
109     segments, markers, count = extract_segments(
110         grayed,
111         edges,
112         min_ratio,
113         max_count
114     )
115     return segments, markers, count
116
117
118 def extract_segments(grayed, edges, min_ratio=0, max_count=100):
119     contours, hierarchy = cv2.findContours(
120         edges,
121         cv2.cv.CV_RETR_TREE,
122         cv2.cv.CV_CHAIN_APPROX_SIMPLE
123     )
124     markers = numpy.zeros(grayed.shape, numpy.uint8)
125     total_area = edges.shape[0] * edges.shape[1]
126     contours_total = len(contours)
127
128     contours_no_child = []
129     for i in range(contours_total):
130         h = hierarchy[0][i]
131         if h[2] == -1:
132             contours_no_child.append(contours[i])
133         else:
134             logger.debug('Skipped child contour {i}'.format(i=i))
135
136     contours_filtered = []
137     for contour in contours_no_child:
138         area = float(cv2.contourArea(contour))
139         ratio = area / total_area
140         if ratio >= min_ratio:
141             contours_filtered.append((contour, area))
142         else:
143             logger.debug('Skipped contour with ratio %f' % ratio)
144
145     contours_filtered_sorted = sorted(
146         contours_filtered,
147         key=operator.itemgetter(1),
148         reverse=True
149     )
150
151     assert(max_count < 255 - 2)
152     contours_count = min(len(contours_filtered_sorted), max_count)
153     contours_capped = contours_filtered_sorted[:contours_count]
154     contours_no_tuple = [c for c, v in contours_capped]
155
156     # color 1: border
157     colors = range(2, 255)
158     import random
159     random.shuffle(colors)
```

```
    for i in range(contours_count):
        cv2.drawContours(markers, contours_no_tuple, i, colors[i], -1)
    logger.info(
        'Segments found {total}, {count} satisfied min_ratio'.format(
            total=contours_total,
            count=contours_count
        ))
    segments = contours_no_tuple
    return segments, markers, contours_count
```

Listing 1: segmentation.py

```
class SentenceDiff(object):

    @staticmethod
    def distance(image1, image2):
        assert len(image1.sentences) > 0
        assert len(image2.sentences) > 0

        ratios = []
        for s in image1.sentences:
            ratios.append(max(map(
                lambda x: SentenceDiff.distance_sentence(s, x),
                image2.sentences
            )))

        ratio_total = sum(ratios)
        count = len(ratios)
        return ratio_total / float(count)

    @staticmethod
    def distance_sentence(sentence1, sentence2):
        sm = difflib.SequenceMatcher(
            None,
            [word.value for word in sentence1.words],
            [word.value for word in sentence2.words],
            autojunk=False
        )
        return sm.ratio()
```

Listing 2: similarity.py

```
from gensim import models, similarities


def document(sequence):
    counts = {}
    for item in sequence:
        if item in counts:
            counts[item] = counts[item] + 1
        else:
            counts[item] = 1

    result = []
```

```
    for k, v in counts.iteritems():
        result.append((k, v))
    return result


class LSI(object):
    def __init__(self, corpus, num_features):
        self.lsi = models.LsiModel(corpus, num_topics=num_features)
        self.index = similarities.SparseMatrixSimilarity(
            self.lsi[corpus],
            num_features=num_features
        )

    def similarity(self, doc):
        sims = self.index[self.lsi[doc]]
        return sims


class TFIDF(object):
    def __init__(self, corpus, num_features):
        self.tfidf = models.TfidfModel(corpus)
        self.index = similarities.SparseMatrixSimilarity(
            self.tfidf[corpus],
            num_features=num_features
        )

    def similarity(self, doc):
        sims = self.index[self.tfidf[doc]]
        return sims


class CorporaOfImages(object):
    def __init__(self, album):
        self.album = album
        self.index = []
        self.corpus = []

    def get_corpus(self):
        if len(self.corpus) > 0:
            return self.corpus

        for image in self.album.images:
            sequences = []
            for sentence in image.sentences:
                sequence = sentence.export()
                sequences.extend(sequence)
            doc = document(sequences)
            self.corpus.append(doc)
            self.index.append(image)

        return self.corpus

    def similarity(self, model, image):
        sequences = []
```

```
        for sentence in image.sentences:
            sequence = sentence.export()
            sequences.extend(sequence)
        doc = document(sequences)
        sims = model.similarity(doc)

        fitness = {}
        for i, j in list(enumerate(sims)):
            filename = self.index[i].filename
            fitness[filename] = j
        return fitness


class CorporaOfSentences(object):
    def __init__(self, album, min_length=5):
        self.album = album
        self.index = []
        self.corpus = []
        self.min_length = min_length

    def get_corpus(self):
        if len(self.corpus) > 0:
            return self.corpus

        for image in self.album.images:
            for sentence in image.sentences:
                sequence = sentence.export()
                doc = document(sequence)
                if len(sequence) >= self.min_length:
                    self.corpus.append(doc)
                    self.index.append(image)

        return self.corpus

    def similarity(self, model, image):
        result = None
        for sentence in image.sentences:
            sequence = sentence.export()
            doc = document(sequence)
            if len(sequence) >= self.min_length:
                sims = model.similarity(doc)
                if result is None:
                    result = sims
                else:
                    result = [max(i, j) for i, j in zip(result, sims)]

        fitness = {}
        for i, j in list(enumerate(result)):
            filename = self.index[i].filename
            if filename not in fitness:
                fitness[filename] = j
            else:
                fitness[filename] = max(fitness[filename], j)
```

```
120         return fitness
```

Listing 3: gensim.py

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. Technical report, 2010.

[2] Ali Abdo Mohammed Al-Kubati, Jamil AM Saif, and Murad AA Taher. Evaluation of canny and otsu image segmentation. In *International conference on emerging trends in computer and electronics engineering (ICETCEE'2012)*, pages 24–25, 2012.

[3] Ravimal Bandara. Bag-of-features descriptor on sift features with opencv (bof-sift), February 2015. `http://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT-Features-with-O`.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[5] Clifford A Behrens, Dennis E Egan, Yu-Yun Ho, Carol Lochbaum, and Mark Rosenstein. Automatic recommendation of products using latent semantic indexing of content, September 2 2003. US Patent 6,615,208.

[6] Serge Beucher and Christian Lantuéjoul. Use of watersheds in contour detection. In *International workshop on image processing, real-time edge and motion detection*, 1979.

[7] Paul E Black. Ratcliff/obershelp pattern recognition. *Dictionary of Algorithms and Data Structures*, 2004.

[8] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[9] Ritendra Datta, Jia Li, and James Z Wang. Content-based image retrieval: approaches and trends of the new age. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 253–262. ACM, 2005.

[10] Scott C Deerwester, Susan T Dumais, George W Furnas, Richard A Harshman, Thomas K Landauer, Karen E Lochbaum, and Lynn A Streeter. Computer information retrieval using latent semantic structure, June 13 1989. US Patent 4,839,853.

[11] Andrew Delong, Anton Osokin, Hossam N Isack, and Yuri Boykov. Fast approximate energy minimization with label costs. *International journal of computer vision*, 96(1):1–27, 2012.

[12] Hongli Deng, Wei Zhang, E. Mortensen, T. Dietterich, and L. Shapiro. Principal curvature-based region detector for object recognition. pages 1–8, June 2007.

[13] Thomas Deselaers, Daniel Keysers, and Hermann Ney. Features for image retrieval: an experimental comparison. *Information Retrieval*, 11(2):77–107, 2008.

[14] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[15] P.-E. Forssen. Maximally stable colour regions for recognition and matching. pages 1–8, June 2007.

[16] Jordi Freixenet, Xavier Muñoz, David Raba, Joan Martí, and Xavier Cufí. Yet another survey on image segmentation: Region and boundary information integration. In *Computer Vision—ECCV 2002*, pages 408–422. Springer, 2002.

[17] Arturo Gil, OscarMartinez Mozos, Monica Ballesta, and Oscar Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, 21(6):905–920, 2010.

[18] Andrew Greensted. Otsu thresholding, June 2010. `http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html`.

[19] Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. In *Ldv Forum*, volume 20, pages 19–62, 2005.

[20] Andreas Hotho, Steffen Staab, and Gerd Stumme. Ontologies improve text document clustering. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 541–544. IEEE, 2003.

[21] Mark J. Huiskes and Michael S. Lew. The mir flickr retrieval evaluation. In *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*, New York, NY, USA, 2008. ACM.

[22] Jiwoon Jeon, Victor Lavrenko, and Raghavan Manmatha. Automatic image annotation and retrieval using cross-media relevance models. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 119–126. ACM, 2003.

[23] Luo Juan and Oubong Gwun. A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3(4):143–152, 2009.

[24] Kurt Koffka. *Principles of Gestalt psychology*. Routledge, 2013.

[25] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

[26] Gert Kootstra, Niklas Bergström, and Danica Kragic. Gestalt principles for attention and segmentation in natural and artificial vision systems. In *Semantic Perception, Mapping and Exploration (SPME), ICRA 2011 Workshop*. eSMCs, 2011.

[27] Thomas K Landauer and Michael L Littman. Computerized cross-language document retrieval using latent semantic indexing, April 5 1994. US Patent 5,301,109.

[28] Edda Leopold and Jörg Kindermann. Text categorization with support vector machines. how to represent texts in input space? *Machine Learning*, 46(1-3):423–444, 2002.

[29] Michael S Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 2(1):1–19, 2006.

[30] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. A survey of content-based image retrieval with high-level semantics. *Pattern Recognition*, 40(1):262–282, 2007.

[31] Karen E Lochbaum and Lynn A Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information Processing & Management*, 25(6):665–676, 1989.

[32] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[33] D.G. Lowe. Object recognition from local scale-invariant features. 2:1150–1157 vol.2, 1999.

[34] L Luccheseyz and SK Mitray. Color image segmentation: A state-of-the-art survey. *Proceedings of the Indian National Science Academy (INSA-A)*, 67(2):207–221, 2001.

[35] Jitendra Malik, Serge Belongie, Thomas Leung, and Jianbo Shi. Contour and texture analysis for image segmentation. *International journal of computer vision*, 43(1):7–27, 2001.

[36] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2(2):438–469, 2009.

[37] Henning Müller, Paul Clough, Thomas Deselaers, Barbara Caputo, and Image CLEF. Experimental evaluation in visual information retrieval. *The Information Retrieval Series*, 32, 2010.

[38] Rajesh Namase. Latent semantic indexing: How does the lsi algorithm work?, April 2014. `http://www.techlila.com/latent-semantic-indexing/`.

[39] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161–2168, June 2006. **oral presentation**.

[40] Eric Nowak, Frédéric Jurie, and Bill Triggs. Sampling strategies for bag-of-features image classification. In *Computer Vision–ECCV 2006*, pages 490–503. Springer, 2006.

[41] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[42] Christos H Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168. ACM, 1998.

[43] Bo Peng, Lei Zhang, and David Zhang. A survey of graph theoretical approaches to image segmentation. *Pattern Recognition*, 46(3):1020–1038, 2013.

[44] OpenCV Project. Introduction to sift (scale-invariant feature transform), February 2015. `http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html`.

[45] OpenCV Project. Introduction to surf (speeded-up robust features), February 2015. `http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html`.

[46] X Ren. Superpixel: Empirical studies and applications.

[47] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 10–17. IEEE, 2003.

[48] Jos BTM Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta informaticae*, 41(1):187–228, 2000.

[49] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. 2006.

[50] E. Rosten, R. Porter, and T. Drummond. Faster and better: a machine learning approach to corner detection. oct 2008.

[51] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[52] Gerard Salton, James Allan, and Chris Buckley. Automatic structuring and retrieval of large text files. *Communications of the ACM*, 37(2):97–108, 1994.

[53] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[54] Arnold WM Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. Content-based image retrieval at the end of the early years. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(12):1349–1380, 2000.

[55] I. Sobel. An isotropic 3x3 image gradient operator. *Machine Vision for three-demensional Sciences*, 1990.

[56] Bart Thomee, Erwin M Bakker, and Michael S Lew. Top-surf: a visual words toolkit. In *Proceedings of the international conference on Multimedia*, pages 1473–1476. ACM, 2010.

[57] Tinne Tuytelaars, Christoph H Lampert, Matthew B Blaschko, and Wray Buntine. Unsupervised object discovery: A comparison. *International journal of computer vision*, 88(2):284–302, 2010.

[58] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends® in Computer Graphics and Vision*, 3(3):177–280, 2008.

[59] Unknown. Untitled, February 2015. `http://johannes.lampel.net/bll137-Dateien/image044.jpg`.

[60] Peter Wiemer-Hastings, K Wiemer-Hastings, and A Graesser. Latent semantic analysis. In *Proceedings of the 16th international joint conference on Artificial intelligence*, pages 1–14. Citeseer, 2004.

[61] Jun Yang, Yu-Gang Jiang, Alexander G Hauptmann, and Chong-Wah Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 197–206. ACM, 2007.

[62] NC Yeo, KH Lee, YV Venkatesh, and Sim Heng Ong. Colour image segmentation using the self-organizing map and adaptive resonance theory. *Image and Vision Computing*, 23(12):1060–1079, 2005.

[63] Faliu Yi and Inkyu Moon. Image segmentation: A survey of graph-cut methods. In *Systems and Informatics (ICSAI), 2012 International Conference on*, pages 1936–1941. IEEE, 2012.

[64] Junsong Yuan, Ying Wu, and Ming Yang. Discovery of collocation patterns: from visual words to visual phrases. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[65] Konstantinos Zagoris, Savvas A Chatzichristofis, and Avi Arampatzis. Bag-of-visual-words vs global image descriptors on two-stage multimodal retrieval. In *Proceedings of the 34th international ACM SIGIR conference on research and development in information Retrieval*, pages 1251–1252. ACM, 2011.

[66] Hui Zhang, Jason E Fritts, and Sally A Goldman. Image segmentation evaluation: A survey of unsupervised methods. *computer vision and image understanding*, 110(2):260–280, 2008.

[67] Shiliang Zhang, Qi Tian, Gang Hua, Qingming Huang, and Shipeng Li. Descriptive visual words and visual phrases for image applications. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 75–84. ACM, 2009.

[68] Yimeng Zhang, Zhaoyin Jia, and Tsuhan Chen. Image retrieval with geometry-preserving visual phrases. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 809–816. IEEE, 2011.

[69] Qing-Fang Zheng and Wen Gao. Constructing visual phrases for effective and efficient object-based image retrieval. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 5(1):7, 2008.

[70] Qing-Fang Zheng, Wei-Qiang Wang, and Wen Gao. Effective and efficient object-based image retrieval using visual phrases. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 77–80. ACM, 2006.