An Efficient Parallel Strategy for Matching Visual Self-similarities in Large Image Databases

Katharina Schwarz, Tobias Häußler, and Hendrik P.A. Lensch

Computer Graphics, Tübingen University
72076 Tübingen, Germany

Abstract. Due to high interest of social online systems, there exists a huge and still increasing amount of image data in the web. In order to handle this massive amount of visual information, algorithms often need to be redesigned. In this work, we developed an efficient approach to find visual similarities between images that runs completely on GPU and is applicable to large image databases. Based on local self-similarity descriptors, the approach finds similarities even across modalities. Given a set of images, a database is created by storing all descriptors in an arrangement suitable for parallel GPU-based comparison. A novel voting-scheme further considers the spatial layout of descriptors with hardly any overhead. Thousands of images are searched in only a few seconds. We apply our algorithm to cluster a set of image responses to identify various senses of ambiguous words and re-tag similar images with missing tags.

1 Introduction

Finding similarities between images is a computational intensive task that is necessary in many computer vision applications, e.g., image retrieval and organization, object detection or recognition. Typically, the comparison is based on extracted features representing important image properties. Mostly, it is assumed that multiple similar images share the same properties as well as the extracted features. A major challenge is the extraction of suitable features because they should be classified as looking similar if captured under varying lighting conditions, from slightly different viewpoints, or with partially occluded objects.

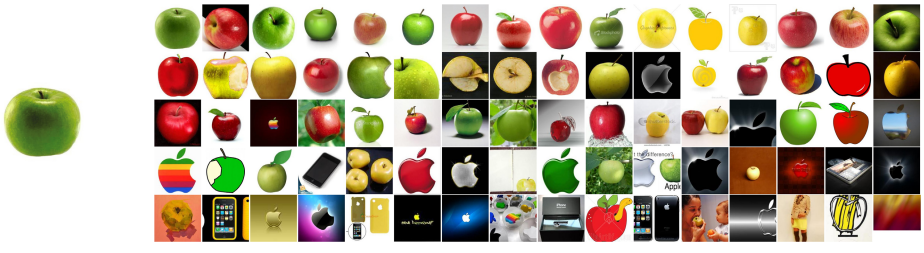


Fig. 1. Images retrieved from Google Images querying "apple" sorted by similarity to the template (left). Decreasing similarity from top left to bottom right.

A. Fusiello et al. (Eds.): ECCV 2012 Ws/Demos, Part I, LNCS 7583, pp. 281–290, 2012.

The features have to account for changes in rotation, scale, illumination, color, texture, etc. Moreover, the set of common properties can vary drastically when taking images of various domains (photographs, drawings, sketches) into account.

In order to search for similar images, large image databases such as Flickr or search engines like Google Images typically use meta-data, tags, and textual search queries specified by users while ignoring the visual content. Flickr, e.g., contains millions of images and a simple search often yields millions of results. The quality of the results is largely based on the search term and the quality of the meta-data. Improvement on the quality of answers can only be achieved by taking, besides textual data, also the visual appearance of images into account. The local self-similarity descriptor introduced by Shechtman and Irani [1] encodes local similarities within an image region and successfully finds templates in other images. Unfortunately, since the computation of this descriptor is very expensive, applying it to large databases is a big challenge.

We present a variation of a self-similarity algorithm that makes it applicable to huge image databases. Descriptor generation and matching run completely on a modern GPU using CUDA. Due to our suitable representation of the descriptor database as well as a new voting-scheme considering the spatial arrangement with hardly any overhead, our implementation scales to databases with thousands of images that can be searched in only a few seconds. Further, no additional pre-processing steps like learning or quantization are needed. Evaluation is performed with ETHZ, Caltech 101 and MIRFLICKR datasets as well as over one million images downloaded from Flickr. Even for matching a template with the large Flickr sets, we can compute over 1400 full image comparisons per second. We apply our algorithm to cluster a given set of image responses to identify various meanings of ambiguous words and re-tag images with similar shapes but missing tag. It also could be used for real-time analysis of video streams.

2 Related Work and Background

Descriptors on Large Databases. Different descriptors and matching strategies have been used for large-scale image retrieval and similarity matching. Local descriptors such as SIFT [2] were used in the bag-of-visual-words (BOV) approach [3], ignoring the global shape and spatial arrangement of an image. Zhang et al. [4] group multiple visual words to encode spatial arrangement in the inverted file structure. Another approach is based on global descriptors such as GIST [5]. Because of its low memory requirements it scales up to very large databases [6]. Johnson et al. [7] used GIST to organize large photo collections on the GPU with a SIFT-based geometric verification to further refine the ranking generated by the global descriptor.

Whereas images of similar scenes do not necessarily show the same objects with similar geometric layout, a certain combination of features is typical. So, learning and classification methods have been used in combination with local and global descriptors. Xiao et al. evaluated such descriptors on a large database ("SUN database") [8]. Shrivastava et al. [9] proposed a computationally intensive

method to find visual similar images over different domains learning features that are most important for a particular image. In contrast, we aim at efficiently finding similar images across various domains without any prior learning steps.

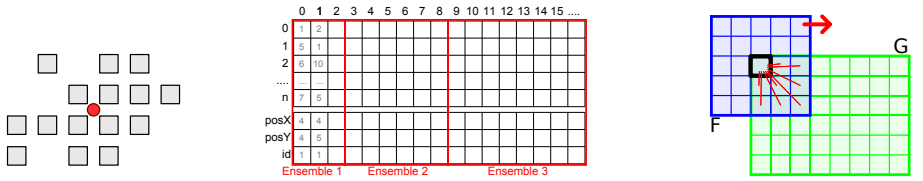
Self-similarities. The local self-similarity descriptor, on which our work is based, was introduced by Shechtman and Irani [1]. It was developed on the observation that similar images do not necessarily share properties like colors, textures, or edges. So, measuring them is not always sufficient for comparison. These images are similar because their local intensity pattern is repeated in nearby image locations in a similar relative geometric layout which is captured in this descriptor.

The self-similarity descriptor is generated by measuring the similarity of an image patch within its surrounding region. The sum of squared differences (SSD) between a 5×5 image patch centered at an image pixel and all 5×5 patches in the surrounding region is calculated and normalized, leading to a correlation surface that is subsequently transformed into a binned log-polar representation with 80 bins. Some of the descriptors are non-informative, because they do not capture any local self-similarity or they capture too much. Non-informative descriptors are discarded and the remaining descriptors of an image form a global ensemble. Ensembles are similar if the distance between the descriptor values is small and the spatial arrangement of the descriptors is similar. Boiman and Irani used the self-similarity descriptor to detect objects in images based on both freehand sketches and real images with an optimized ensemble matching strategy [10]. Their elimination of comparison calculations at locations where the similarity is probably very low leads to a scattered memory access pattern that does not fit well onto the GPU. Therefore, we developed a different GPU-optimized strategy.

Chatfield et al. [11] use the self-similarity descriptor to retrieve deformable shapes. They refine the sparsification of descriptors and study the influence of quantization on matching performance for large-scale retrieval using a BOV approach. In contrast, we do not need any quantization and, thus, avoid errors therefrom. Moreover, time-consuming generation of vocabulary is not necessary.

3 Approach and Implementation

Based on the self-similarity descriptor [1], we developed a simple approach that enables searching for similar images to a given template or calculate similarity values for an image set (e.g. Fig. 1) also in huge databases. Our system first creates the self-similarity descriptors of an image that form an ensemble (Fig. 2(a)) and stores the ensembles of all images in a database suitable for parallel GPU-based comparison (Fig. 2(b)). This database of ensembles is only created once and matching then operates directly on the database without any further pre-processing steps by comparing ensembles (Fig. 2(c)). Our efficient implementation is able to compare more than 1400 images in only one second.



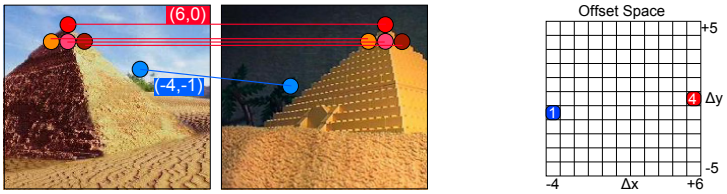
(a) Ensemble (b) Database layout (c) Ensemble matching

Fig. 2. Approach. (a) Ensemble is formed by informative descriptors (rectangles) capturing the spatial arrangement around the center (red dot). (b) Layout for storage of ensembles in device memory. (c) All descriptors in ensemble F are compared to all descriptors in the database. Matching descriptors cast a vote on the spatial arrangement.

Database Handling. The first step is to compute the spatial ensembles as given in [1]. This step can be trivially parallelized on the GPU using CUDA. The ensemble of the descriptors of a 256×265 image occupy about 600 KB, typically containing 440 descriptors. In order to handle large databases a caching strategy is proposed. Ensembles residing in GPU device memory are swapped out to host memory if the device memory becomes exhausted. As the same strategy is used for host memory, ensembles are finally swapped out to disk which is determined by a simple LRU (least recently used) strategy. The ensembles are stored in a 2D array with one descriptor per column (Fig. 2(b)) ensuring fast access to same values in different descriptors in subsequent CUDA threads. As the ensemble a descriptor belongs to as well as its position within the ensemble have to be known in the matching stage, a second array stores additional information.

Matching Ensembles. In the matching stage, the ensemble of a template is compared to all ensembles in the database, yielding a similarity measure. The detailed comparison of ensembles is described in the following. We can either compare one query image to all images stored in the database or compute a weighted similarity graph between each pair of images in a cluster. In each case, multiple scales are supported. As already mentioned, the approach presented by Boiman and Irani [10] does not fit well onto the GPU. Thus, we decided to use a simple brute-force voting of first identifying the potential center and then calculating the score of an ensemble. Our approach is similar to a 2D cross-correlation (Fig. 2(c)). Every descriptor in the query ensemble is compared to the descriptors of all ensembles in the database. For small distance, the template descriptor casts a vote for a certain central position in a database ensemble. Votes indicate how many template descriptors are similar to database descriptors in the same spatial arrangement. Then, the votes are weighted by a number denoting how scattered the voting template descriptors are.

Details of the Matching Strategy. When querying for an image, the template ensemble I is compared to the whole database with ensembles I'_k ($k = 1..K$, with K images in database). Thus, at first the squared distance t between descriptor d_i

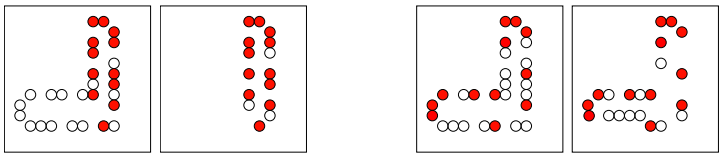


(a) Similar descriptors in different images (b) Offset Space

Fig. 3. Voting in offset space. Lines connect similar descriptors (a). Red descriptors share same offsets and vote in offset space at corresponding positions. Blue descriptors vote for another offset. Maximum in offset space indicates best matching positions (b).

of I and descriptor d'_{kj} of I'_k ($i, j = 1..80$ bins) is calculated by one CUDA thread per combination of template and database descriptor. If t is below some threshold T, the spatial offset between d_i and d'_{kj} is used to vote for an ensemble offset in offset space $S^s_k(\Delta x, \Delta y)$ (Fig. 3). This offset space exists for each ensemble in the database. In order to account for small deformations and variations in scale, each bin in offset space contains 3×3 offsets. As soft-weighting would require many memory accesses, we only increase $S^s_k(\Delta x, \Delta y)$ by 1 if distance t is below T. Because this is rarely the case, the number of memory accesses is very small. Thus, an atomic instruction is used that operates directly on global memory. The offset for which most descriptors voted indicates the displacement where the template image fits best to the database image.

However, the number of votes does not contain any information about the arrangement of the voting descriptors in the template ensemble. If only descriptors in a small region of I cast a vote, then the similarity is smaller than if descriptors were uniformly distributed over I (Fig. 4). In order to incorporate the spatial arrangement, we decided to include the position of the voting descriptors into the matching results. Therefore, we partition I into rectangular regions R_m (e.g. 5×5) and assign each descriptor to such a region by its position in the ensemble. In addition to $S^s_k(\Delta x, \Delta y)$, a second offset space $S^r_k(\Delta x, \Delta y)$ stores information about the regions where the voting descriptors are located. $S^r_k(\Delta x, \Delta y)$ is organized as 2D 32-bit integer array. Bits b_0 to b_{24} are connected



(a) Comparing I with I'_1 (b) Comparing I with I'_2

Fig. 4. Regions in voting. Both times the same number of descriptors cast a vote (red). This would result in the same similarity, although shape of I is more similar to I'_2 than to I'_1. We increase the comparison score by taking the position into account (b).

Algorithm 1. Voting in offset space (pseudo-code)

```
for all i, j, k in parallel do
  if (t ← ‖d_i − d'_kj‖²) < T then
    Δx ← x_{d'_kj} − x_{d_i}
    S^s_k(Δx) ← S^s_k(Δx) + 1
    S^r_k(Δx) ← S^r_k(Δx) | (1 << ν)
  end if
end for
```

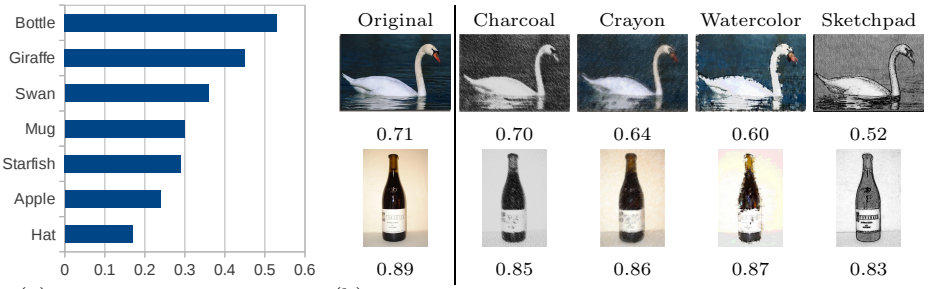
to a region. The bit b_ν is set if a descriptor in region R_ν casts a vote (Alg. 1). Then, offset $\Delta\mathbf{x} = \arg\max_{\Delta x \Delta y} m(\Delta x, \Delta y)$ and similarity $s = \frac{\max(m(\Delta x, \Delta y))}{r \cdot \max(c_I, c_{I'_k})}$ are calculated with $m(\Delta x, \Delta y) = S^s_k(\Delta x, \Delta y) \cdot \mathrm{popcnt}(S^r_k(\Delta x, \Delta y))$. While r describes the number of informative descriptors in the template ensemble I, c_I and $c_{I'_k}$ is the number of descriptors in I and I'_k, respectively. The number of set bits in x is counted with $\mathrm{popcnt}(x)$.

For comparison on various scales, the query is scaled to different sizes before comparing it with the database. Matching all ensembles in the database with each other needs K^2 comparisons. For L scales, the database must contain ensembles in L scales. Consequently, $(LK)^2$ comparisons have to be performed. Due to redundant comparisons, the complexity can be reduced to $(L+1)K^2$.

4 Evaluation

In order to analyze accuracy, speed, and memory requirements of our approach, we performed experiments on the datasets ETHZ Extended Shape Classes [12] (383 images in 7 categories) and Caltech 101 [13] (9145 images in 101 categories, we removed "BACKGROUND" and "Faces_easy"). For testing on larger image counts, we used MIRFLICKR [14] containing 1 M Flickr images and additionally downloaded over a million images from Flickr with some random categories.

Visual Results. The self-similarity descriptor already works well for finding similar forms over various domains. To validate our changes in the algorithm, we performed tests with the ETHZ and Caltech datasets. Thus, a database is searched for each image yielding a similarity value between the query and all images. The results are sorted by their similarity and the average precision (AP) is calculated from this list. The mean average precision (mAP) is calculated for each category. The results for ETHZ (Fig. 5(a)) as well as for Caltech dataset vary a lot depending on the different categories. Categories with images sharing a distinct shape work best (e.g. Caltech: Airplanes (mAP=0.84), Motorbike (0.71), Faces (0.58)). Other categories (Pyramid (0.17)) contain images with very cluttered background that are not well suited as a template image. So, they have negative impact on the mAP of a category. Our results for ETHZ are similar to [11]. For measuring cross-domain matching, we applied several effects to the templates (Fig. 5(b)) without changing the images in the database. As expected, the AP remains for every effect nearly the same as for the original image.



(a) mAPs for ETHZ categories (b) APs for various domains (Effects applied to templates only)

Fig. 5. Visual evaluation on ETHZ dataset. Approach works well for different domains.

Performance. We measured performance on a NVIDIA GeForce GTX 580 with 1,5 GB VRAM. The host system uses Intel Xeon X5660 CPU and 48 GB RAM. First, we only consider the generation of all descriptors of a single image, comparing our GPU version against the OpenCV implementation on CPU (Fig. 6). The CPU test was performed on an AMD Phenom II X4 965 CPU with an OpenMP-optimized version. For every image size, our GPU descriptor generation algorithm performs about ten times faster than the CPU implementation.

As already mentioned, we created the database such that our matching is performed very fast and can also be applied to very large image datasets without waiting for hours or even days. Results are shown in Table 1. Images were downscaled to 256×256 pixels. Times for creating the database as well as for matching vary depending on size of the image set and number of descriptors. For the Caltech set we even retrieved 2849 full image comparisons in only 1 sec. (load DB: 0.89 sec., match: 2 sec.). In order to compensate for variations, we also tested two larger datasets with images of varying sizes and content from Flickr: MIRFLICKR and images we downloaded for random categories. Both times, matching an image still resulted in over 1400 comparisons per sec.

This indicates a great speedup contrasted to [11], where comparing a single pair of VGA images with quantized descriptors took at least 20 sec. on a 2.4 GHz Pentium. Further, compared to [7], our method promises performance benefits while performing not only geometric verification, but also advanced shape

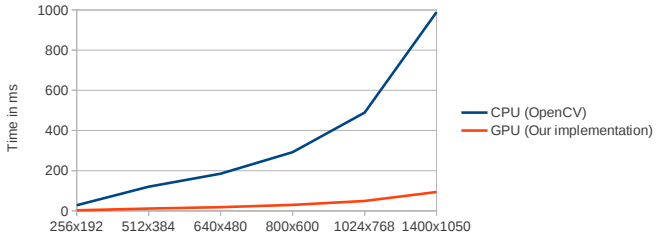


Fig. 6. Runtime comparison: OpenCV vs. our GPU version for descriptor generation

Table 1. Performance measurements for ETHZ, Caltech and two larger datasets

	Images	Descriptors		Create DB	Matching one image	
		Total	Informative	[h:min:s.ms]	[h:min:s.ms]	Img. Comp./ sec.
ETHZ	383	533,280	165,784	00:00:06.11	00:00:00.23	1644
Caltech	8,242	10,675,016	3,936,554	00:01:13.71	00:00:02.89	2849
MIRFLICKR	999,997	1,331,604,604	446,570,589	09:06:30.17	00:11:51.48	1407
Flickr Images	1,087,007	1,430,601,920	486,668,007	07:38:55.78	00:12:52.31	1406

matching. In their work, verifying about 4,000 clusters of 30,000 images takes 40 minutes. Our implementation is also faster than various methods based on locality sensitive hashing functions, implemented for CPU [15]. As, on average, only about 30% descriptors in an image are informative, more than half are discarded. The memory required to store the informative ones was 48MB for ETHZ, 1.1GB for Caltech 101 and even 139GB for our Flickr dataset. Thus, memory requirements are larger than with a quantized approach.

5 Applications

Large-scale image databases normally allow to search by textual queries which often are ambiguous and lead to images with different visual appearance. Due to our efficient implementation, the search results can be improved by taking the shape of the objects shown in images into account and clustering many similar images (around 800 images in about 10 min) or even re-tagging a large database.

Clustering. Based on the calculated similarity values of our implementation, we generate clusters of similar looking images. The database is created with images retrieved from Google Images by searching for a single word. Then, comparing all images with each other results in a distance graph. The nodes represent images, the edge weight corresponds to the distance ($1 - similarity$). In this graph, cluster centers are found by searching for the node that has the most neighbors with a distance smaller than a threshold. This threshold is based on the average distance of all nodes in the graph. Finally, a cluster consists of the center and all its neighbors in a certain range. By repeating this process multiple times

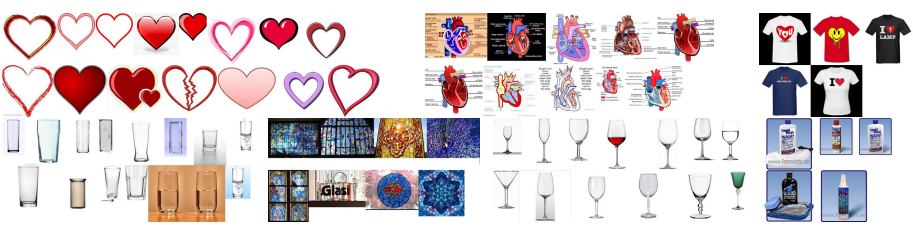


Fig. 7. Detected clusters in search results for “heart”, “glass” (Google Images)

Fig. 8. Images (attached: tag-lists) retrieved in our Flickr set containing similar shape as template (framed) but not according tag. Images not necessarily show same object.

while removing the previously found centers and neighbors, several clusters are extracted. It is amazing what different clusters are found within some categories showing the ambiguity of the words. For example various meanings of "glass": different forms of glasses for drinking, windows or even glass wash liquids (Fig. 7).

Re-tagging. As our implementation works very efficient, it can be used to quickly find multiple images containing a shape similar to a template image in a large photo database. The test photo collection with about one million images we downloaded from Flickr is enriched with a tag-list we also obtained from Flickr (textual information describing the image and added by Flickr users). Searching for a template image in the photo database may lead to a number of images that contain visual similarity to the template although not containing the according query in their tag-list (Fig. 8). In this case, an additional tag is added to the list which leads to a more complete tag-list. If the photo collection is then searched in a textual way, more true positive images are returned.

6 Conclusion

We present an efficient approach to find similar images in large datasets based on the local self-similarity descriptor and an ensemble matching strategy that runs completely on GPU using CUDA. We made some effort such that the descriptor database only has to be generated once for all images and, afterwards, enables efficient matching that works directly on it. New images can be directly added. Based on a novel voting-scheme to compare the spatial arrangement of descriptors, our GPU implementation searches the content of thousands of images in only a few seconds without any further pre-processing steps. Thus, images can

be searched nearly instantly. Evaluation is performed with several datasets. Depending on the image size and content, on average about 1800 image comparisons are carried out per second. Applying our implementation to clustering of a given set of images retrieved for a textual query leads to fascinating identifications of various meanings of ambiguous words and extending tag-lists of similar images can further improve retrieval based on textual search.

Acknowledgments. This work has been partially funded by the DFG Emmy Noether fellowship (Le 1341/1-1) and an NVIDIA Professor Partnership Award.

References

1. Shechtman, E., Irani, M.: Matching Local Self-Similarities across Images and Videos. In: IEEE Conf. on Comp. Vis. and Pat. Recogn, CVPR (2007)
2. Lowe, D.G.: Object Recognition from Local Scale-Invariant Features. In: Proc. of Int. Conf. on Comp. Vis., ICCV (1999)
3. Sivic, J., Zisserman, A.: Video Google: A Text Retrieval Approach to Object Matching in Videos. In: Proc. of Int. Conf. on Comp. Vis (ICCV), vol. 2, pp. 1470–1477 (2003)
4. Zhang, Y., Jia, Z., Chen, T.: Image Retrieval with Geometry-Preserving Visual Phrases. In: IEEE Conf. on Comp. Vis. and Pat. Recogn (CVPR), pp. 809–816 (2011)
5. Oliva, A., Torralba, A.: Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. Int. Journal of Comp. Vis. 42, 145–175 (2001)
6. Douze, M., Jégou, H., Sandhawalia, H., Amsaleg, L., Schmid, C.: Evaluation of GIST descriptors for web-scale image search. In: Proc. of ACM Int. Conf. on Image and Video Retrieval, CIVR (2009)
7. Johnson, T., Georgel, P., Raguram, R., Frahm, J.M.: Fast Organization of Large Photo Collections using CUDA. In: Wksp. on Comp. Vis. on GPUs, ECCV (2010)
8. Xiao, J., Hays, J., Ehinger, K.A., Oliva, A., Torralba, A.: SUN database: Large-scale scene recognition from abbey to zoo. In: IEEE Conf. on Comp. Vis. and Pat. Recogn, CVPR (2010)
9. Shrivastava, A., Malisiewicz, T., Gupta, A., Efros, A.A.: Data-driven Visual Similarity for Cross-domain Image Matching. ACM Trans. Graph. 30 (2011)
10. Boiman, O., Irani, M.: Detecting Irregularities in Images and in Video. Int. Journal of Comp. Vis. 74, 17–31 (2007)
11. Chatfield, K., Philbin, J., Zisserman, A.: Efficient Retrieval of Deformable Shape Classes using Local Self-Similarities. In: Wksp. on Non-rigid Shape Analysis and Deformable Image Alignment, ICCV, pp. 264–271 (2009)
12. Schindler, K., Suter, D.: Object Detection by Global Contour Shape. Pattern Recogn. 41, 3736–3748 (2008)
13. Fei-Fei, L., Fergus, R., Perona, P.: Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories, vol. 12, p. 178. IEEE Computer Society, Los Alamitos (2004)
14. Mark, J., Huiskes, B.T., Lew, M.S.: New Trends and Ideas in Visual Concept Detection: The MIR Flickr Retrieval Evaluation Initiative. In: MIR 2010: Proc. of the 2010 ACM Int. Conf. on Multimedia Information Retrieval, pp. 527–536. ACM, New York (2010)
15. Aly, M., Munich, M., Perona, P.: Indexing in Large Scale Image Collections: Scaling Properties and Benchmark. In: IEEE Wksp. on Applications of Comp. Vis., WACV (2011)