sciendo

JAISCR, 2020, Vol. 10, No. 2, pp. 113 – 123
10.2478/jaiscr-2020-0008

FAST IMAGE INDEX FOR DATABASE MANAGEMENT ENGINES

Rafał Grycuk[1,*], Patryk Najgebauer[1], Miroslaw Kordos[2], Magdalena M. Scherer[3], Alina Marchlewska[4,5]

[1]*Czestochowa University of Technology, al. Armii Krajowej 36, 42-200 Częstochowa, Poland*

[2]*Department of Computer Science and Automatics, University of Bielsko-Biala, Poland*

[3]*Faculty of Management, Czestochowa University of Technology al. Armii Krajowej 19, 42-200 Częstochowa, Poland*

[4]*Information Technology Institute, University of Social Sciences, Łódź, Poland*

[5]*Clark University, Worcester, MA 01610, USA*

[*]*E-mail: rafal.grycuk@pcz.pl*

Submitted: 18th July 2020; Accepted: 23th January 2020

Abstract

Large-scale image repositories are challenging to perform queries based on the content of the images. The paper proposes a novel, nested-dictionary data structure for indexing image local features. The method transforms image local feature vectors into two-level hashes and builds an index of the content of the images in the database. The algorithm can be used in database management systems. We implemented it with an example image descriptor and deployed in a relational database. We performed the experiments on two image large benchmark datasets.

Keywords: image descriptors, content-based image retrieval, image indexing

1 Introduction

In the process of image retrieval, users search through databases which consist of thousands, even millions of images. The emergence of content-based image retrieval (CBIR) in the 1990s accelerated extensive research in this area. Generally, CBIR consists in searching for images similar to the query image (or images of a certain class) [8, 11, 12, 14, 15, 24]. The similarity of images is evaluated not by some human description, but by automatically computed image features. In order to retrieve similar images, important query image features are extracted and then, they are matched with those stored in the image database. Identifying features and objects in images is still a challenge as the same objects and scenes can be viewed under different imaging conditions.

In the paper, we propose a system for fast image retrieval based on the nested dictionary of hashes generated from local image features. The method for creating a database index proposed in the paper uses image local features. We use as an example the SURF descriptors to generate a nested dictionary of hashes.

There does not exist a technology for fast and efficient retrieval of images based on their con-

tent in existing relational database management systems. Standard SQL does not contain commands for handling multimedia, large text objects, and spatial data. The algorithm proposed in the paper can be implemented in any modern database management system. We implemented the proposed algorithm and performed experiments in Microsoft SQL Server.

Through this research, we highlight the following features and contributions of the proposed algorithm.

- We present a novel fast database index for database image retrieval.
- Our work provides new insights, showing that the two-level hash-based index provides substantial reduction in the number of vector comparisons needed to retrieve images.
- The method can use nearly any kind of visual descriptors – hand-crafted or trained by deep learning.
- The proposed system is fast and can be used in various relational or non-relational database management systems.

The remainder of the paper is organised as follows. In Section 2, we discuss previously selected works and the SURF image descriptor used as an example. The nested dictionary-based method proposed in the paper is presented in Section 3. Experiments on two large well-known image datasets, showing accuracy and a comparison with the state-of-the-art methods, are shown in Section 4. Finally, conclusions and discussions of the paper are presented in Section 5.

2 Related Work

The proposed method has a similar purpose to the locality-sensitive hashing family [16] but works differently. Other existing solutions such as vantage-point trees (or MVP trees) are based on vantage points [5, 6]. These solutions divide the indexed set of vectors by analyzing distances between data vectors. Each tree node includes a specific multidimensional area of the dataset. The tree root covers the entire area, while its children (child nodes) divide this area into smaller areas. For each node of the tree, n points that are very far apart from each other are selected, around which the points are grouped.

We propose a different approach which does not rely on the tree structure. It is instead a structure of the table with access to data through the index. Grouping the indexed data vectors is not done directly by comparing their mutual distances, but by separately assigning a hash code for each of the data vectors. The hash code is created on the basis of a normalized data vector and encodes the differences between its parts. The groups are created automatically based on common hash codes. A single hash code is a 32-bit integer used to quickly address nodes containing data. The distance between points is included but in a different way. Points which distance indicates that they can be located on the border of several nodes are included in the additional sub-hash code of the second row, which at the same time refers directly to the main-hash code of the main index. Thus, our approach relates somehow to the vantage-point trees (or MVP); however, it works differently.

As an example image descriptor, we use Speeded-Up Robust Features (SURF) to detect and describe local features of an image [4]. SURF is an improved version of SIFT (Scale-invariant feature transform) [18]. It is faster and provides similar functionality. The SURF keypoints are composed of two vectors. The first one provides the following information: position (x,y), scale (detected scale), response (response of the detected feature, strength), orientation (orientation, measured anti-clockwise from +ve x-axis), laplacian (sign of laplacian). The second one is a descriptor which contains 64 numbers. An important advantage of SURF is that it generates less data then SIFT (SIFT has longer 128-element descriptors), which speeds-up further processing. The method has also a parallel implementations [28, 33], thus it generates the results much faster. The algorithm consists of four main stages [3]:

1. Computing Integral Images,
2. Fast-Hessian Detector:
   - The Hessian,
   - Constructing the Scale-Space,
   - Accurate Interest Point Localization,

3. Interest Point Descriptor:
   - Orientation Assignment,
   - Descriptor Components,
4. Generating vectors describing the interest points.

The SURF keypoints are resistant to change of scale and rotation, which can be useful when objects in the image rotate or change scale. Naturally, such immunity is limited; if we change the point of view entirely, the corresponding keypoints will not be detected. In the literature, there are other local image features detectors and descriptors, hand-crafted or deep-learned ones. The presented indexing method can use various local descriptors.

3 Proposed Fast Image Database Index

In this Section, we present a novel, fast two-level image index. The main problem in comparing images by almost any kind of their feature descriptors is the computational complexity. The number of long vector comparisons needed to check the image similarity is often intractable in near real-time. For example, every SURF keypoint descriptor is a 64-element vector. For a medium size, e.g. 1200×720 image, SURF can generate approximately three thousand keypoints, and each of them is described by a 64-element vector. If our dataset has thousands or millions of images, comparing them is time-consuming, and in the case of image retrieval, it is usually unusable because of the long query executions. In the paper, we present a method for reducing the number of descriptor comparisons. The primary purpose of the presented approach is to speed up image retrieval. To this end, we create a dictionary, which allows searching similar and corresponding descriptors fast. Such a dictionary can be used as a new data type in any SQL database. The method can use nearly any kind of visual descriptors – hand-crafted or trained by deep learning. We implemented the SURF descriptors as an example, thus in the rest of the paper, we describe the method using this example.

3.1 Hash Calculation

In the first step, an image descriptor algorithm (SURF in our case) computes a list of keypoints for each image in the dataset. Each keypoint has a descriptor, which is used to calculate descriptor hashes. For each image keypoint, we calculate two 32-bit integer hashes $I1$ and $I2$. The keypoint descriptor is vector $\mathbf{A}$ with elements $A_j \in [-1;1]$. Based on $\mathbf{A}$, we calculate vector $\mathbf{B}$ elements, by the following formula

$$B_j = \sum_{k=0}^{3} A^2_{16j+k},$$

where $j = 0, 1, \ldots, 15$.

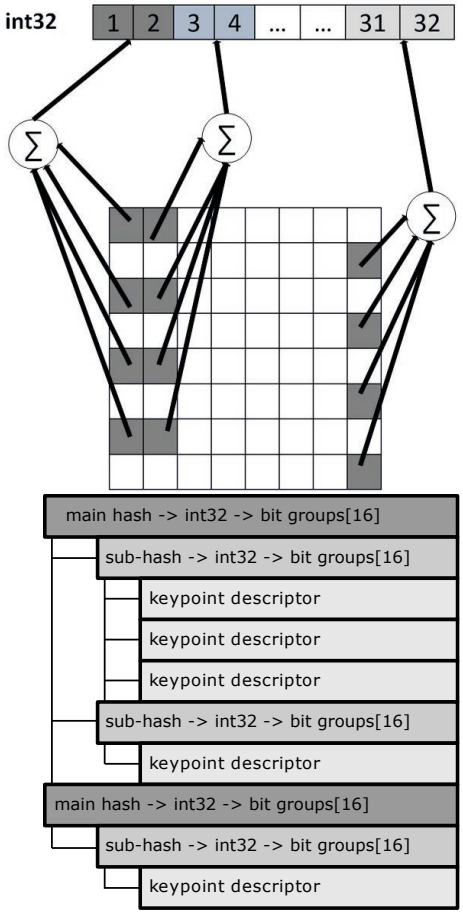

Figure 1. Visual representation of the hash calculation algorithm from the SURF descriptor (see Section 3.1) and the index structure (see Section 3.2).

In the next step we calculate b_{max}, which is the maximal value of vector $\mathbf{B}$ elements

$$b_{max} = \max_{j \in \{0 \ldots 15\}} \{B_j\}.$$

Based on b_{max} we normalize values of elements of **B** within the range $[0;4)$. We calculate vector **C**

$$C_j = \frac{aB_j}{b_{max}},$$

where a is a constant of some value that approaches asymptotically 4. In the next step, the elements of vector **D**1 are calculated by the floor function in order to obtain integer values of **C** elements $D1_j = \lfloor C_j \rfloor$. Then, we fit **D**1 (a vector of sixteen 2-bit numbers) into 32-bit integer. Each value is described by 2 bits in the range [0:3], and computed by $I1 = \sum_{j=0}^{15} D1_j * 4^j$. The previous steps computed the first hash. It is basically a form of an index for a given keypoint descriptor. Now, we calculate the sub-hash (or sub-index), which allows differentiating values positioned in near the boundary. By removing their fractional part we would loss this information. The second hash prevents such loss. Let us calculate vector **T**, which contains only the fractional part of the elements of vector **C**: $T_j = C_j - D1_j$. This vector allows determining vector **D**2 by

$$D2_j = \begin{cases} D1_j - 1 & \text{if } D1_j > 0 \text{ and } T_j < 0.3 \\ D1_1 + 1 & \text{if } T_j > 0.6 \text{ and } D1_j < 1 \\ D1_j & \text{otherwise} \end{cases}$$

Having **D**2 vector, we can calculate the sub-hash $I2 = \sum_{j=0}^{15} D2_j * 4^j$. Finally, we have a pair of hashes $(I1, I2)$ for each descriptor. The main hash and the sub-hash allow to create a nested dictionary described in the next subsection, which significantly reduces the number of needed descriptor comparisons.

3.2 Building Dictionary

After calculating hashes $(I1, I2)$, the image index is created. This step is presented in the form of pseudo-code in Algorithm 1. The index has a structure of a nested dictionary. The regular dictionary is a list of key-value pairs, where the key is unique. Both keys and values can be of any type. Generally, a nested dictionary is a dictionary with another dictionary nested as value (see Figure 2).

In the first step, we initialize (create) a new nested dictionary for each image. Then, we iterate through $ListOfHashes$ and $ListOfDesc$ simultaneously. If our $Index$ does not contain a key equal to $I1$, then we create a new key-value pair with it. In the next step, we check another condition – if the sub-dictionary $Index[I1]$ does not contain $I2$ a new key-value pair in the sub-dictionary is created with the key as $I2$. Next, in the sub-dictionary value $Index[I1][I2]$, a new keypoint descriptor is added. The above steps are repeated for every keypoint descriptor.

INPUT: $ListOfHashes$ - list of hashes, each element contains a pair of hashes ($I1$ and $I2$), $ListOfDesc$ - list of keypoint descriptors
OUTPUT: $Index$ - image index in the form of the nested dictionary
Create a new dictionary $Index$ containing a list of key-value elements, where the key is integer type and the value is a sub-dictionary composed of sub-hash and a list of keypoints (see Figure 2).
foreach $I1, I2 \in ListOfHashes$ **and** $Desc \in ListOfDesc$ **do**
  if $Index$ **not** contains key $I1$ **then**
    Add new key-value pair where key is $I1$ and value is a new sub-dictionary,
  end
  if $Index[I1]$ **not** contains key $I2$ **then**
    Add new key-value pair to the sub-dictionary($Index[I1]$) where key is $I2$ and value is a new list of keypoint descriptors,
  end
  For $Index[I1][I2]$ add new keypoint descriptor $Desc$.
end

Algorithm 1. Index creation algorithm.

Dictionary		
Key	Value	
Main Hash (Integer)	Sub-dictionary	
	Key	Value
	Sub-Hash (Integer)	Keypoint descriptors (List of vectors)

Figure 2. Proposed nested dictionary structure.

The created index structure is presented in Figure 1 (right side). As can be seen, the main hash contains many sub-hashes, and these sub-hashes are composed of a list of keypoint descriptors. Both hashes $I1$ and $I2$ are composed of 16-bit groups,

each group contains 2 bits, which allows using 32-bit integers. The significant advantage of this solution is the application of a dictionary and integers as keys. In most modern languages such as C#, Java, etc., Dictionary (C#) or HashMap (Java equivalent of the dictionary) data types allow searching elements by their keys fast. Moreover, if the key type is a value type (primitive) such as, e.g. integer, the search is even faster. Hashes can also be used to image clustering [13] or image classification [23].

3.3 Executing Query

In a content-based image retrieval system, we can distinguish two steps. The first step is generally an indexation stage where we create a mathematical representation of the images. The second step is executing the query, where we compare the query image with the previously indexed images and retrieve similar ones.

In the presented method, for a given query image the steps presented in Sections 3.1 and 3.2 are executed. When we obtain *index* value for the query image, we can compare it with other previously indexed images in the dataset. The entire process of the query execution is presented in the form of pseudo-code in Algorithm 2.

The algorithm takes query images, distance threshold and list of indexed images as input parameters, and retrieved images as the output. In the first step, we search over every key (main hash) in the query image. If the hashes (keys) are the same, we move on to search in the sub-index and compare sub-hashes in the sub-dictionaries. The crucial aspect of this stage is the comparison using the bitwise XOR operator. We compare sub-hashes $I2$ and check if the value of this operation is less then 16, which means that only the four last bits differ and the sub-hashes are equal. The number of insignificant bits was determined empirically. The presented procedure allows for obtaining a significantly smaller number of comparisons than in the case of comparing every keypoint in the query image to every keypoint in the indexed image database. Next, we compare the remained (narrowed by hashes) descriptors in both indexes by calculating the Euclidean distance. In the next step, we sum up the distances and compare it with the threshold provided as input. If a distance is less or equal then threshold we assume that the image is similar to the query image. The previously described steps are repeated for all the indexed images. In the last step, a list of similar images is returned.

```
INPUT: QueryImage - query image index,
threshold - distance threshold for the query image,
ListOfImages - a list of dataset images with calculated
  indexes.
OUTPUT: RetrievedImages - retrieved images
foreach IndexedImage ∈ ListOfImages do
  sumDistance := 0.0,
  hashesMatched := 0,
  foreach QiIndex ∈ QueryImage.Index do
    if IndexedImage.Index contains key QiIndex.Key then
      foreach QiInnerImgIndex ∈ QiIndex.Value do
        foreach
          InnerImgIndex ∈ IndexedImage.Index[QiIndex.Key]
        do
          if (QiInnerImgIndex.Key xor
            InnerImgIndex.Key) < 16 then
            hashedMatched++,
            Create similairty list ListOfSimilarity
            foreach QiDesc ∈ QiInnerImgIndex.Value do
              foreach IndDesc ∈ InnerImgIndex.Value do
                d := Dist(QiDesc, IndDesc) -calculate the Euclidean
                  distance between two descriptors,
                Add new pair to similarity list where values are
                  distance d and hashedMatched,
              end
            end
            Sort descending the ListOfSimilarity by the
              hashedMatched,
            Take only elements from ListOfSimilarity where values
              of hashedMatched are greater then 4
            Summarize (SimilaritySum) distance d values in
              previously filtered ListOfSimilarity
          end
        end
      end
    end
  end
  if sumDistance <= threshold then
    Add IndexedImage as similar image to the
      RetrievedImages
  end
end
Return RetrievedImages as a list of similar images.
```

Algorithm 2. Query execution algorithm.

The presented algorithm was implemented for experimental purposes in .NET Framework and C# language, and deployed to MS SQL Server as a DLL library with the proposed new type two-factor index.

4 Experimental Results

The experiments were carried out in the proposed method implemented according to the description in Section 3, deployed in the MS SQL Server database. We used two well-known image datasets to check the accuracy and speed, namely the PASCAL Object Recognition Database Collection, Unannotated Database - 101 Object Categories [10], and the COREL Database for Content-based Image Retrieval [27]. We used all the classes in the datasets, and every class was divided into two sets of, respectively, 90 % training images and 10

% query (evaluation) images. The performance of the proposed method was evaluated with *Precision* and *Recall* measures [7, 29]. They are computed using: *DIC* – a set of database images for a given class of objects, *RI* – a set of retrieved images for query, $RRI(TP)$ – a set of relevant retrieved images (true positive), $IRI(FP)$ – irrelevant retrieved images (false positive), $RNRI(FN)$ – relevant not retrieved images (false, negative) and $IRNRI(TN)$ – irrelevant not retrieved images (TN). Table 1 presents relevance measures of the retrieved images for the Pascal VOC dataset. Due to lack of space, we present only random results from the entire dataset. The presented results were compared with the current state-of-the-art methods in Table 3.

Method/Author	Avg.Precis.	Avg.Recall
Proposed method	**0.8780**	**0.9361**
deselaers2008 [9]	0.63	N/A
lin2009 [17]	0.7270	N/A
walia2014 [30]	0.7830	0.1566
mehmood2016 [19]	0.8421	0.1684
ali2016 [2]	0.8627	0.1725
wang2013 [31]	0.6970	0.1300
zeng2016 [32]	0.8057	0.1611
mehmood2018 [20]	0.8785	0.1737
agarwal2012 [1]	0.6020	0.4490
saadatmand2007 [25]	0.6400	0.4160
murala2012 [22]	0.6830	0.5400
sumana2008 [26]	0.7070	0.4900
memon2017 [21]	0.7500	0.5600

Table 3. Comparison of the average precision for the proposed method with the state-of-the-art for the Corel dataset.

Table 2 contains randomly selected experimental results (multi-queries) for the Corel dataset. We narrow the results due to lack of space. Let us take under consideration an example query $71(cat)$. All relevant images were correctly retrieved ($RRI = 180$). Only 9 are improperly retrieved, although they are not from the same class. The *precision* value for this experiment is 95.2 and *recall* equals 100. The average precision for this dataset equals 87.8, which is a very good result on the Pascal dataset.

In Table 3 we compared our method with some state-of-the-art methods. Average *precision* is slightly lower than the highest value ([20]). *Recall*, on the other hand, is much higher than in the case of other methods. One of the essential aspects of our method is the reduction of descriptor comparisons which is presented in Table 4. As can be seen in Table 4, the reduction in the number of comparisons is significant. We took ten images in three resolutions and averaged the number of comparisons needed in their case. For example, in the case of an image of resolution 1200×720 the reduction is over two orders of magnitude.

5 Final Remarks

We proposed a novel database index for content-based image retrieval. For creating the index we used a nested dictionary. By using the index, we can reduce the number of comparisons during the retrieval process. In our implementation, it is embedded into a relational database management system, and it benefits from the SQL commands. The algorithm also provides a new data type for the database. We used Microsoft SQL Server in order to create a CLR UDT type and a CLR Function to implement the algorithm. The proposed solution is open to modifications and can also be ported to other databases or extended to other types of visual feature descriptors. The performed experiments proved the accuracy and effectiveness of our index. We observed a significant reduction in the number of vector comparisons during the retrieval process. We applied the SURF keypoint descriptors to extract local image features as an example, however it is possible to use nearly any local image descriptor, hand-crafted or trained one.

References

[1] Agarwal, M., Maheshwari, R.: Á trous gradient structure descriptor for content based image retrieval. International Journal of Multimedia Information Retrieval 1(2), 129–138 (2012)

[2] Ali, N., Bajwa, K.B., Sablatnig, R., Mehmood, Z.: Image retrieval by addition of spatial information based on histograms of triangular regions. Computers & Electrical Engineering 54, 539–550 (2016)

[3] Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (surf). Computer vision and image understanding 110(3), 346–359 (2008)

[4] Bay, H., Tuytelaars, T., Van Gool, L.: Surf: Speeded up robust features. In: European conference on computer vision, pp. 404–417. Springer (2006)

Id(Class)	RI	DIC	RRI (TP)	IRI (FP)	RNRI (FN)	IRNRI (TN)	Precision	Recall	Accuracy
723(watch)	205	195	174	31	21	6892	84.9	89.2	99.3
582(bonsai)	112	105	99	13	6	7000	88.4	94.3	99.7
590(wheelchair)	51	49	48	3	1	7066	94.1	98.0	99.9
599(brain)	87	81	74	13	7	7024	85.1	91.4	99.7
607(stegosaurus)	52	49	41	11	8	7058	78.8	83.7	99.7
660(stop-sign)	57	53	50	7	3	7058	87.7	94.3	99.9
671(strawberry)	30	29	28	2	1	7087	93.3	96.6	100.0
683(sunflower)	77	70	43	34	27	7014	55.8	61.4	99.1
697(tick)	45	41	37	8	4	7069	82.2	90.2	99.8
701(trilobite)	76	71	65	11	6	7036	85.5	91.5	99.8
708(umbrella)	66	62	60	6	2	7050	90.9	96.8	99.9
714(watch)	210	195	187	23	8	6900	89.0	95.9	99.6
734(water-lilly)	33	31	31	2	0	7085	93.9	100.0	100.0
1(accordion)	47	45	42	5	3	7068	89.4	93.3	99.9
506(beaver)	41	38	28	13	10	7067	68.3	73.7	99.7
514(beaver)	41	38	29	12	9	7068	70.7	76.3	99.7
536(binocular)	29	27	24	5	3	7086	82.8	88.9	99.9
548(bonsai)	113	105	96	17	9	6996	85.0	91.4	99.6
170(dalmatian)	60	55	53	7	2	7056	88.3	96.4	99.9
186(dollar-bill)	46	43	36	10	7	7065	78.3	83.7	99.8
213(dolphin)	58	54	52	6	2	7058	89.7	96.3	99.9
223(dolphin)	58	54	52	6	2	7058	89.7	96.3	99.9
340(elephant)	57	53	52	5	1	7060	91.2	98.1	99.9
406(emu)	48	44	39	9	5	7065	81.2	88.6	99.8
413(emu)	47	44	36	11	8	7063	76.6	81.8	99.7
475(ewer)	75	70	70	5	0	7043	93.3	100.0	99.9
545(flamingo)	59	55	48	11	7	7052	81.4	87.3	99.7
567(flamingo-head)	40	37	34	6	3	7075	85.0	91.9	99.9
581(flamingo-head)	40	37	33	7	4	7074	82.5	89.2	99.8
593(garfield)	30	28	24	6	4	7084	80.0	85.7	99.9
55(helicopter)	78	72	66	12	6	7034	84.6	91.7	99.7
62(ibis)	69	65	60	9	5	7044	87.0	92.3	99.8
267(ketch)	98	93	63	35	30	6990	64.3	67.7	99.1
492(Leopards)	172	162	110	62	52	6894	64.0	67.9	98.4
452(Faces-easy)	371	353	318	53	35	6712	85.7	90.1	98.8
Average							**84.16**	**90.24**	**99.63**

Table 1. Experiment results for the proposed method, performed on the Pascal dataset. Due to lack of space, we present only a part of all queries from various classes, although the average precision is calculated for all the query images in the test set.

Id(Class)	RI	DIC	RRI (TP)	IRI (FP)	RNRI (FN)	IRNRI (TN)	Precision	Recall	Accuracy
638(woman)	520	495	297	223	198	7694	57.1	60.0	95.0
460(train)	316	290	209	107	81	8015	66.1	72.1	97.8
473(train)	310	290	226	84	64	8038	72.9	77.9	98.2
488(deer)	205	190	177	28	13	8194	86.3	93.2	99.5
496(deer)	201	190	184	17	6	8205	91.5	96.8	99.7
511(deer)	204	190	177	27	13	8195	86.8	93.2	99.5
526(deer)	205	190	186	19	4	8203	90.7	97.9	99.7
530(deer)	206	190	179	27	11	8195	86.9	94.2	99.5
537(deer)	201	190	180	21	10	8201	89.6	94.7	99.6
71(cat)	189	180	180	9	0	8223	95.2	100.0	99.9
79(cat)	189	180	176	13	4	8219	93.1	97.8	99.8
95(dog)	305	290	180	125	110	7997	59.0	62.1	97.2
101(dog)	316	290	188	128	102	7994	59.5	64.8	97.3
107(dog)	314	290	186	128	104	7994	59.2	64.1	97.2
18(bus)	96	90	87	9	3	8313	90.6	96.7	99.9
26(bus)	96	90	85	11	5	8311	88.5	94.4	99.8
38(bus)	97	90	82	15	8	8307	84.5	91.1	99.7
47(bus)	93	90	86	7	4	8315	92.5	95.6	99.9
53(bus)	94	90	85	9	5	8313	90.4	94.4	99.8
66(bus)	96	90	86	10	4	8312	89.6	95.6	99.8
106(car)	427	400	375	52	25	7960	87.8	93.8	99.1
121(car)	427	400	391	36	9	7976	91.6	97.8	99.5
135(car)	435	400	383	52	17	7960	88.0	95.8	99.2
141(car)	431	400	383	48	17	7964	88.9	95.8	99.2
151(car)	431	400	387	44	13	7968	89.8	96.8	99.3
156(car)	431	400	395	36	5	7976	91.6	98.8	99.5
162(car)	419	400	387	32	13	7980	92.4	96.8	99.5
181(cards)	98	90	84	14	6	8308	85.7	93.3	99.8
187(cards)	96	90	85	11	5	8311	88.5	94.4	99.8
201(cards)	96	90	84	12	6	8310	87.5	93.3	99.8
205(cards)	94	90	86	8	4	8314	91.5	95.6	99.9
232(cards)	98	90	90	8	0	8314	91.8	100.0	99.9
236(cards)	95	90	84	11	6	8311	88.4	93.3	99.8
246(decoys)	94	90	79	15	11	8307	84.0	87.8	99.7
328(dish)	93	90	89	4	1	8318	95.7	98.9	99.9
396(dish)	94	90	87	7	3	8315	92.6	96.7	99.9
403(doll)	98	90	85	13	5	8309	86.7	94.4	99.8
Average							**87.80**	**93.61**	**99.53**

Table 2. Experiment results for the proposed method performed on the Corel dataset. Due to lack of space, we present only a part of all queries from various classes, although the average precision is calculated for all the query images in the test set.

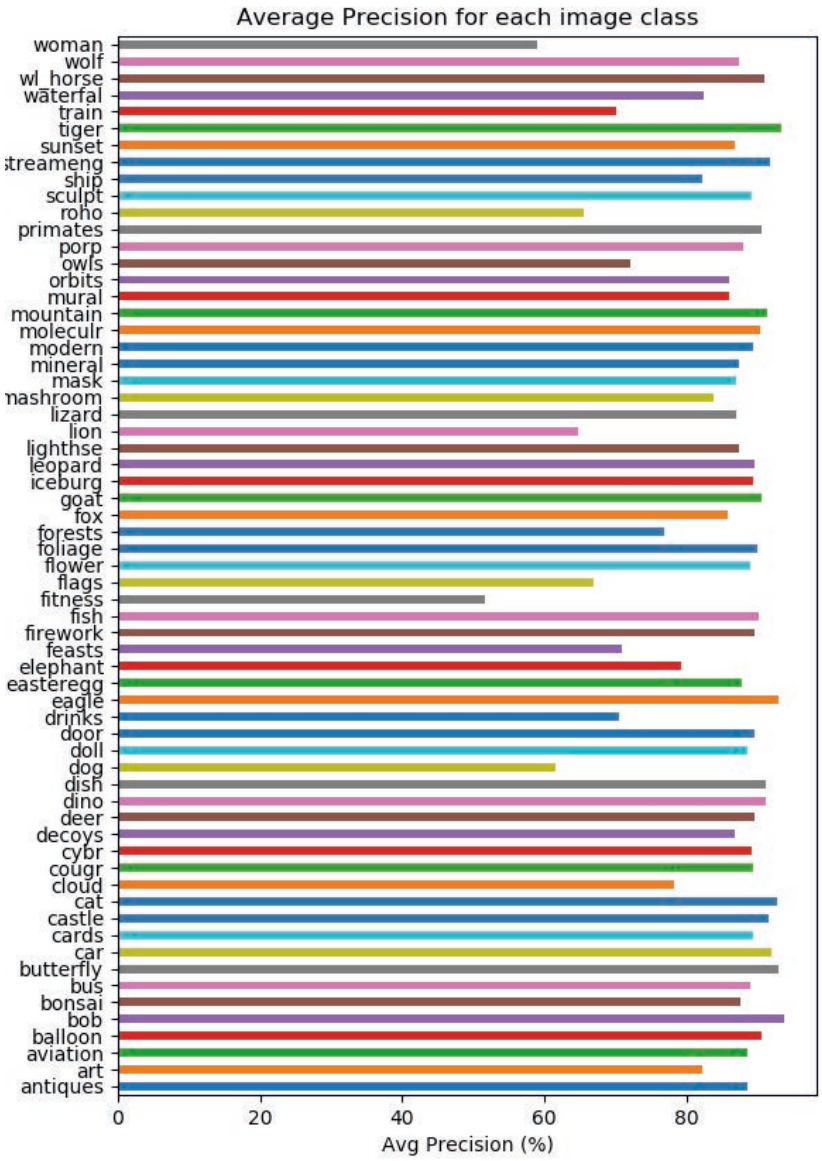

Figure 3. Average precision for each image class for the proposed approach performed on the Corel dataset

Image resolution	Any to any descriptor comparison count	Proposed index comparison count
4500 × 2600px	1.10217E+11	307,931,798
1200 × 720px	60,358,817,792	271,067,676
400 × 138px	191445728	7,160,708

Table 4. Examples of the reduction in the number of vector comparisons needed to compare images by local descriptors with the proposed method. The number of local features depends on the image complexity, and the numbers presented in the table are the average for the image sets.

[5] Bozkaya, T., Ozsoyoglu, M.: Indexing large metric spaces for similarity search queries. ACM Transactions on Database Systems (TODS) 24(3), 361–404 (1999)

[6] Brin, S.: Near neighbor search in large metric spaces. In: Proceedings of the 21th International Conference on Very Large Data Bases, VLDB '95, pp. 574–584. Morgan Kaufmann Publishers Inc. (1995)

[7] Buckland, M., Gey, F.: The relationship between recall and precision. Journal of the American society for information science 45(1), 12 (1994)

[8] Daniel Carlos Guimaraes Pedronette, J.A., da S. Torres, R.: A scalable re-ranking method for content-based image retrieval. Information Sciences 265(0), 91 – 104 (2014). http://dx.doi.org/10.1016/j.ins.2013.12.030

[9] Deselaers, T., Keysers, D., Ney, H.: Features for image retrieval: an experimental comparison. Information retrieval 11(2), 77–107 (2008)

[10] Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. International Journal of Computer Vision 88(2), 303–338 (2010)

[11] Gabryel, M., Korytkowski, M., Scherer, R., Rutkowski, L.: Object detection by simple fuzzy classifiers generated by boosting. In: L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, J. Zurada (eds.) Artificial Intelligence and Soft Computing, Lecture Notes in Computer Science, vol. 7894, pp. 540–547. Springer Berlin Heidelberg (2013)

[12] Karakasis, E., Amanatiadis, A., Gasteratos, A., Chatzichristofis, S.: Image moment invariants as local features for content based image retrieval using the bag-of-visual-words model. Pattern Recognition Letters 55(0), 22 – 27 (2015)

[13] Koren, O., Hallin, C.A., Perel, N., Bendet, D.: Decision-making enhancement in a big data environment: Application of the k-means algorithm to mixed data. Journal of Artificial Intelligence and Soft Computing Research 9(4), 293–302 (2019)

[14] Korytkowski, M., Rutkowski, L., Scherer, R.: Fast image classification by boosting fuzzy classifiers. Information Sciences 327, 175–182 (2016)

[15] Korytkowski, M., Senkerik, R., Scherer, M.M., Angryk, R.A., Kordos, M., Siwocha, A.: Efficient image retrieval by fuzzy rules from boosting and metaheuristic. Journal of Artificial Intelligence and Soft Computing Research 10(1), 57–69 (2020)

[16] Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of massive datasets. Cambridge University Press (2014)

[17] Lin, C.H., Chen, R.T., Chan, Y.K.: A smart content-based image retrieval system based on color and texture feature. Image and Vision Computing 27(6), 658–665 (2009)

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

[19] Mehmood, Z., Anwar, S.M., Ali, N., Habib, H.A., Rashid, M.: A novel image retrieval based on a combination of local and global histograms of visual words. Mathematical Problems in Engineering 2016 (2016)

[20] Mehmood, Z., Mahmood, T., Javid, M.A.: Content-based image retrieval and semantic automatic image annotation based on the weighted average of triangular histograms using support vector machine. Applied Intelligence 48(1), 166–181 (2018)

[21] Memon, M.H., Li, J.P., Memon, I., Arain, Q.A.: Geo matching regions: multiple regions of interests using content based image retrieval based on relative locations. Multimedia Tools and Applications 76(14), 15,377–15,411 (2017)

[22] Murala, S., Maheshwari, R., Balasubramanian, R.: Directional local extrema patterns: a new descriptor for content based image retrieval. International journal of multimedia information retrieval 1(3), 191–203 (2012)

[23] Nobukawa, S., Nishimura, H., Yamanishi, T.: Pattern classification by spiking neural networks combining self-organized and reward-related spike-timing-dependent plasticity. Journal of Artificial Intelligence and Soft Computing Research 9(4), 283–291 (2019)

[24] Rashedi, E., Nezamabadi-pour, H., Saryazdi, S.: A simultaneous feature adaptation and feature selection method for content-based image retrieval systems. Knowledge-Based Systems 39(0), 85 – 94 (2013)

[25] Saadatmand-Tarzjan, M., Moghaddam, H.A.: A novel evolutionary approach for optimizing content-based image indexing algorithms. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 37(1), 139–153 (2007)

[26] Sumana, I.J., Islam, M.M., Zhang, D., Lu, G.: Content based image retrieval using curvelet transform. In: Multimedia Signal Processing, 2008 IEEE 10th Workshop on, pp. 11–16. IEEE (2008)

[27] Tao, D.: The corel database for content based image retrieval (2009)

[28] Terriberry, T.B., French, L.M., Helmsen, J.: Gpu accelerating speeded-up robust features. In: Proceedings of 3DPVT, vol. 8, pp. 355–362. Citeseer (2008)

[29] Ting, K.M.: Precision and recall. In: Encyclopedia of machine learning, pp. 781–781. Springer (2011)

[30] Walia, E., Pal, A.: Fusion framework for effective color image retrieval. Journal of Visual Communication and Image Representation 25(6), 1335–1348 (2014)

[31] Wang, C., Zhang, B., Qin, Z., Xiong, J.: Spatial weighting for bag-of-features based image retrieval. In: International Symposium on Integrated Uncertainty in Knowledge Modelling and Decision Making, pp. 91–100. Springer (2013)

[32] Zeng, S., Huang, R., Wang, H., Kang, Z.: Image retrieval using spatiograms of colors quantized by gaussian mixture models. Neurocomputing 171, 673–684 (2016)

[33] Zhang, N.: Computing optimised parallel speeded-up robust features (p-surf) on multi-core processors. International journal of parallel programming 38(2), 138–158 (2010)

Rafal Grycuk received his M.Sc. and Ph.D. degrees in computer science from Czestochowa University of Technology in 2012 and 2017, respectively. Currently, he is an assistant professor at the Czestochowa University of Technology. His scientific interests cover computer vision, image description, content-based image retrieval and unsupervised learning.

Patryk Najgebauer received M.Sc. (2010) and Ph.D. (2016) degree in computer science from Czestochowa University of Technology. His Ph.D. dissertation focused on visual classification of microscopy samples. Since 2016 he is assistant professor at the Institute of Computational Intelligence at the Czestochowa University of Technology. His work is concerned on developing new methods of image content-based retrieval and indexing, microscopic samples analysis and image enhancement based mainly on deep fully convolutional neural networks combined with heuristic techniques.

Mirosław Kordos obtained his M.Sc. in electrical engineering from the Technical University of Lodz, Poland in 1994 and his Ph.D. in computer science from Silesian University of Technology, Poland in 2005. In years 1994-2005 he worked in the IT industry as a software developer and systems engineer. In 2006-2007 he was a research fellow at the Division of Biomedical Informatics in the Cincinnati Children's Hospital Research Center, USA. In 2008-2009 he was an assistant professor at the Silesian University of Technology, Poland and since 2010 he has been an assistant professor at the University of Bielsko-Biala, Poland. His research interest comprise various aspects of artificial intelligence, especially neural networks and industrial applications of machine learning.

Magdalena Scherer received her M.Sc. degree in computer science from the Czestochowa University of Technology, Poland, in 2008 and her Ph.D. in management in 2016 from the same university. Currently, she is an assistant professor at Czestochowa University of Technology. Her present research interests include machine learning for prediction and classification.

Alina Marchlewska received the M.Sc. degree in mathematics from the Department of Mathematics and Computer Science, University of Łódź, Poland. She obtained the Ph.D. degree in mathematics from the same University. Currently, she is an assistant professor at the Social Academy of Sciences in Lodz. Her research interests include social computing and applications of various artificial intelligence technologies and computing methods in selected IT problems.