Distributed Image Retrieval with Color and Keypoint Features

Michał Łągiewka, Marcin Korytkowski, Rafal Scherer *Member, IEEE*,
Computer Vision and Data Mining Lab, Institute of Computational Intelligence, Czestochowa University of Technology,
Al. Armii Krajowej 36, 42-200 Czestochowa, Poland
Email: {michal.lagiewka; marcin.korytkowski; rafal.scherer}@iisi.pcz.pl

***Abstract*—Big data term refers to different variations of large datasets to complex to be processed by traditional computing methods. The paper presents a system for retrieving images in relational databases in a distributed environment. Content of the query image and images in the database is compared using global color information and local image keypoints. Image keypoints are indexed by fuzzy sets directly in a relational database. To distribute the process to several machines we use the Apache Hadoop software framework with HDFS.**

***Keywords*—Hadoop, HDFS, distributed storage, content-based image retrieval, color histogram, image keypoints, relational databases.**

I. Introduction

Content-based image retrieval (CBIR) has become recently to be well established in the literature. Yet, nearly all of the solutions presented so far are not designed nor suited for relational databases. Relational databases reign supreme in the business world but storing a huge amount of undefined and unstructured binary data and its fast and efficient search and retrieval is a problem for them. Examples of such data are images or video files. One, old solution devoted to storage and retrieval of images in a database is the methodology proposed in [1], where PostgreSQL database server was used to store and compare images by color-based features. There were also attempts to implement CBIR in commercial database systems. An example might be the Oracle database environment called "interMedia", where image retrieval is based on the global color representation, low-level patterns and textures within the image, such as graininess or smoothness, the shapes that appear in the image created by a region of uniform color and their location. It was described in Oracle Database Online Documentation (10g Release 2, Chapter 6, Content-Based Retrieval Concepts) but was abandoned in newer Oracle versions. Thus, the standard SQL language does not have commands for handling multimedia data and image files are stored often directly in database tables which causes low efficiency of the whole system and even time-consuming data backup.

To address these problems, the authors proposed earlier ([2], [3]) a CBIR system that was able to store and index images in a relational database with local interest points. Local invariant features have gained a wide popularity with the most popular local keypoint detectors and descriptors SURF [4], SIFT [5] or ORB [6]. In the previous work and in this paper we use 128-element SIFT descriptors. The system also allowed to query the database about the image content by SQL commands. Information about local visual features was indexed in the relational database by fuzzy sets [7], [8], [9], [10], [11], [12]. and the AdaBoost algorithm [13]. The mechanism used for database image indexing is depicted in details in [2] and [3].

978-1-5090-5795-5/17/$31.00

Nowadays we face constant increase in the need for processing big data [14], [15]. A concept of high-speed solutions in content-based image retrieval is a natural solution for improving efficiency. The algorithms become more and more accurate, but in most cases this fact does not contribute to decreasing time of retrieval. Possible solutions are parallel [16] or distributed computing. In the paper we propose a distributed framework for image retrieval from a relational database based on the system described in the previous paragraph. We decrease the time of searching across the file system and delivering the final results using file management by the Hadoop Distributed File System (HDFS) and Apache Spark (http://spark.apache.org/streaming/). By using the Spark framework we manage file streams to determine the fastest way to cooperate with the Hadoop file system. This brings us to maintain improvement of the algorithm accuracy without occurring processing-time spikes, comparing to traditional mass-storage entities.

Every time a user queries the proposed system with an image, keypoints for this image are generated using a local interest point detection (SIFT in our case). Those results are compared with a set of pre-generated features stored in a relational database engine, e.g. MS SQL Server. As a result, we obtain a list of linked image addresses (given by filenames). We spread this list through the Apache Spark framework to find, map and reduce the final results to receive unique file names and percentage matching to objects found in the query image.

The proposed system is unique as it is a rare example of CBIR in a relational database with image index created by fuzzy sets, boosting metalearning and image local keypoints. Creating the index is fast so is its expanding in the case of adding new image classes. Moreover, thanks to using the Hadoop Distributed File System, it is highly scalable and the performance of the system depends only on the hardware resources. The rest of the paper is organised as follows. In Section II we present the proposed distributed CBIR system with a brief description of the database image indexing method

which was presented in detail in our previous work [2] and [3]. Section III presents issues concerning resultant image list reduction and Section V concludes the paper.

II. Proposed System

Figure 1 presents a general concept of the designed solution. The system has to be initialized by preparing a dataset using our previous work [17] to index each image and generate local interest points with the procedure presented in [2] and [3]. Then, the system is ready to accept a query image, to analyze it, to find the most relevant images from the database and finally to add it to the collection for future queries.

Indexing images by local keypoints was inspired somehow by the idea presented in [13]. We use boosting metalearning to obtain the intervals of visual feature values that will be utilized in the database index. We create weak classifiers in the form of fuzzy rules. Fuzzy rules have an antecedent and a consequent part. The antecedent part has fuzzy sets with Gaussian membership functions whose parameters are adapted during the boosting procedure. The main idea is to find the most representative fuzzy rules for a given image class and to fast classify query images afterwards. The algorithm uses the boosting meta learning to generate suitable number of weak classifiers. In each step we randomly choose one local feature from the set of positive images according to its boosting weight. Then we search for similar feature vectors from all positive images. Using these similar features we construct one fuzzy rule. Undoubtedly, it is impossible to find exactly the same features in all images from the same class, thus we search for feature vectors which are similar to the feature picked randomly in the current step. This is the one of the reasons for using fuzzy sets and fuzzy logic. As aforementioned, the detailed description of creating relational database index is presented in [2] and [3]. Every keypoint structure from the database which meets certain conditions is compared with a blurred input image for lowering demands of further processing and increasing chance of matching.

Generally, to improve the retrieval speed with local features we face a choice between decreasing precision, e.g. by lowering descriptors' detail level and reducing pool of the dataset, e.g. by choosing a group of images as a result of database query for further comparing. We decided to use both of these methods to strengthen effects of reducing the list of possible data to compare with higher precision and creating percentage coverage index in order to sort it out. The literature provides many methods to process images by local features, e.g. [18], [19], [20], [21]. This allowed us to focus on pragmatic research for the hardware and software configuration to further accelerate image retrieval. At this point we decided to reduce hardware deficiencies by virtualization, however, we took a step ahead and moved data storage-dedicated virtual machines into separate hard disk drives to protect searching across the distributed file system from the bottleneck effect. We kept data operating-servers (Master and SQL Server) within one solid state drive to physically increase processing time and maintain better data transfer between

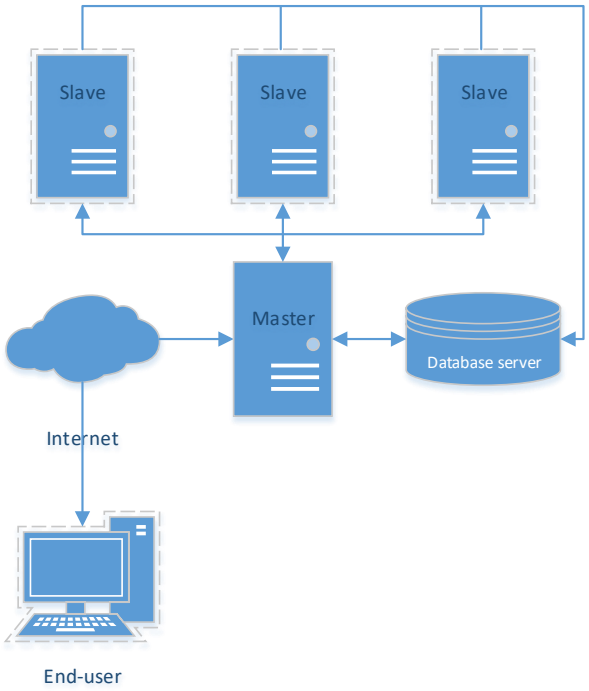

Fig. 1. Diagram of the proposed distributed system for image retrieval based on color and local interest points. Images are stored in the Hadoop Distributed File System distributed among slave machines and are indexed by relational database management system.

software and hardware (for writing operations, e.g. collecting results from HDFS). For better performance of the image retrieval we require our storage to be accessible on all worker nodes. Distributed File System (DFS) is an approach which allows to store expansible varieties of data. The significant aspect of accessing files is a single namespace for the entire system [22]. The most common distributed storage system implementation with high fault-tolerance are Hadoop DFS, Google Cloud, Amazon Simple Storage Service (S3) and Tachyon [23]. The main differences between them are:

- HDFS – file-oriented divided storage with permissions integrated with the operating system,
- S3 – object-oriented, files as objects stored within containers (called buckets),
- Tachyon – memory-centric distributed storage system in which Master operates on permissions and global metadata, data is stored as replicated blocks close to HDFS, because it is based on it,
- Spark and Hadoop within Google Cloud Platform – mixing object-oriented buckets with HDFS.

Each of the listed platforms is paid (either per user account or used hardware), except of the Apache Hadoop, which allows to build a custom setup for better adjustment to the needs of the proposed system. However, building such a system requires an investment in hardware. The fact of hardware limitations is the

TABLE I
VIRTUAL MACHINES SETUP

Name	vCores	RAM	Storage
Server	2	2 GB	SSD
SQL Server	2	3 GB	SSD
Worker 1	1	1,5 GB	HDD
Worker 2	1	1,5 GB	HDD
Worker 3	1	1,5 GB	HDD

most common issue in the retrieval and may cause unexpected delays while processing.

We provided our system with specific hardware configuration in which:

- Server (Master) is a virtual machine managing input, output and database requests, equipped with 2 virtual cores and 2GB of RAM (SSD1),
- SQL Server is a virtual machine handling database (MS SQL Server), equipped with 2 virtual cores and 3GB of RAM (SSD1),
- Worker 1-3 (Slaves) are virtual machine set up for the Hadoop environment with 1 virtual core and 1.5GB of RAM each (HDD1, HDD2, HDD3).

To reduce the bottleneck effect we placed every worker in a separate physical drive and assigned a single virtual core each with a clock speed of 3.5 GHz to avoid hardware limitations. Our solution is based on the Java platform to provide cross-platform communication and input-output interface. Every query (input) image is served by the Master, which generates keypoints and supports transactions with MS SQL Server. After the return of the database query results, the Master searches across the distributed file system and collects (maps) possible results with matching percentage of each image coverage of image keypoints (in the case of multiple objects participating in the input image, percentage coverage is expressed by the average of each matching). Next, the Master reduces the results to create a list of unique file links ordered by similarity to the query image and sends it to the end-user. We can describe the time effort needed to complete the retrieval by

$$T_f = T_p + T_s + T_c + T_l \ , \tag{1}$$

where T_f is the total time from submitting the input image to receiving the final output, T_p is the time of the input image processing (generating keypoints and database querying), T_c is the time of reducing and comparing for the best match, T_l is the time of ordering the final list for output, T_s is the time of searching across the distributed file system and merging the results given by

$$T_s = T_{s1} \vee T_{s2} \vee \ldots \vee T_{sn} \text{ or } T_s = \text{MAX}(T_{s1}, \ldots, T_{sn}) \ . \tag{2}$$

This means that T_p depends on the algorithm speed, T_p is always the longest time from each Worker (Slave), T_c is strictly connected with the input image complexity (e.g. number of objects, generated keypoints to compare within) and T_l depends on the sorting time (i.e. on the list size). Although Slaves work asynchronously, there have to be set a

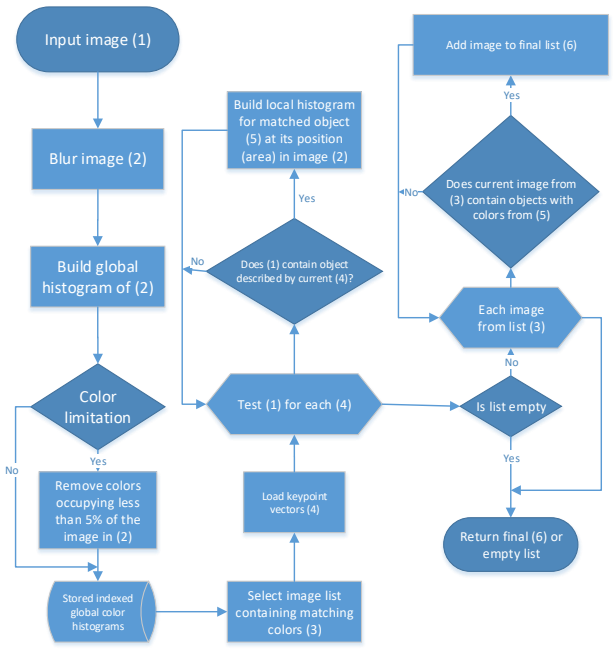

Fig. 2. Workflow of the proposed distributed image retrieval based on global color information (color histogram) and local features (SIFT keypoints).

watch to control if every Slave responded (from Slaves which are currently available in the network). Time T_c can be also represented as

$$T_c = \sum_{i=1}^{n} T_{o_i} + \ T_{r_i} + \sum_{j=1}^{m} T_{v_{i_j}} \ , \tag{3}$$

where n is a number of Slaves, T_o is the time of the network connection and operations (since we use local network, we assume it is approximately constant for every Master-Slave communication), T_r is the time of reducing list of further proceeded images (e.g. names, links, paths), m is the number of objects detected in initial processing (T_p), T_v is the time of calculating percentage participation of keypoints in a single object between input and compared image [20]. For retrieving reasons, every input image at the system entry is blurred with Gaussian blur. At the next step we determine if there is a need of a limitation for color occupying less than certain percentage, e.g. 5%. The reason for strict limitation can be limited resources or limited retrieval (processing) time. The other method is to pick a number of dominant colors by the occupation ratio, e.g. three dominating colors. After reducing the number of colors we work with, we are able to select from the database objects (rows) describing images by keypoints, which were stored with relations to certain groups of colors with their participation percentage. Having indexed images from the dataset stored in the database, we are provided with the list of images related to selected keypoints. At this point we test the input image with each row of the selected keypoints within matching color group. If we were able to find any matching object within or near a color group, we build a local

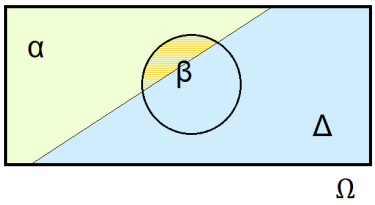

Fig. 3. Scheme of building a subset of images.

histogram for these objects. These matched objects can be related to a certain position in the image to retrieve those structures at a specific position, unless the matched object covers most of the image or it is the image itself. The next step allows to verify if the list of local histograms with related keypoint structures is not empty. If the list does contain at least one object, we can assume there is at least one image in the dataset matched with the input image. This method allows us to check related list of images indexed in the database in relation with keypoint structures to determine if any of these images contain matching objects from the input image. Related images do not have to cover all matched objects, but the more is matched, the better the result is. These results containing any of the matched objects in the input image are added to the final list in preparation for returning to the end-user. An important clue in the retrieval system is setting if we want to return the final list as an ordered or unordered list. Returning unordered objects is faster, as we return matching images as soon as the system notices a match between images.

III. Dataset Reduction

With each processed item in dataset, the probability of finding a match changes. The most recommended way to deal with such problem is to reduce possible operations within the file system and network. In our case, Ω is a dataset of all the labeled data. It is less demanding to split our dataset in two subsets (Fig. 3):

1) items meeting retrieval requirements of matching, e.g. certain color (based on histogram) – α,
2) other items not meeting the requirement – Δ.

Basing on this fragmentation the number of items matching the first requirement is given. In some special cases, when there are more attributes of items stored in the dataset, there always should be chosen the smallest collection meeting the requirements. For example, if the number of items matching color in its histogram is higher than the number of objects meeting certain (scalable) position [24] at the retrieved images in a subset, the smaller one, in this case the position of an object, should be chosen. In Fig. 3 by β we marked all the items in dataset Ω matching remaining criteria (structure described with keypoints). Unlike in the case of the previously listed requirements (i.e. color and position), we are skeptical in the case of detected objects at input image because:

- our system may recognize an unclassified object (class does not exists in database),
- classified structure may be vitiated by an error,
- classified object may not meet other requirements, if given.

The top area of β (Fig. 3) is an intersection of elements meeting two of the given requirements. Retrieving an image with several classified structures and single additional requirement is very similar to a single object case. We are able to retrieve every structure matched with keypoints matching additional requirements as a separate retrieval and after completing it, to combine the list of repeating images in each case. Multidimensional retrieving of a single image with multiple objects and multiple additional requirements for each structure is more complex. There is a significant difference in comparison to retrieving object(-s) with a single additional requirement. In that case we have to consider retrieving keypoint structures before additional requirements to reduce dataset the most. After creating a smaller dataset containing only images with matching structures in each of them, it is possible to reduce the subset by the elements not meeting additional requirements or to return the list of images ordered by the sum of percentage share of meeting additional requirements.

IV. Experimental Results

We performed several tests, passing multiple images from the PASCAL Visual Object Classes (VOC) dataset [25] as input to check if our system is able to handle various object shapes, angles and distortions. We also tested validation of the expected results under some parts of our system being unreachable (e.g. shutting down one of the Workers) and we were able to confirm that a distributed file system (HDFS with Spark in the presented case) can provide meaningful results due to its replication methods. Then, we migrated part of the Slaves to other physical hardware and from analyzing communication times we were able to observe similar time-results (slightly slower). In order to consider all the possibilities, we moved further in optimizing metadata of the dataset in SQL Server. We designed a table structure to handle a simple attribute, color in our case, for each indexed object. We decided to store it as quantized color histogram at each image segment. During empirical tests, initial dataset reduction using pre-indexed metadata improved efficiency and effectiveness of the retrieval due to a smaller dataset and the additional parameter. As a result, the end-user receives better matched results ordered by similarity to the query image. Content-based image retrieval is a complex process requiring hardware resources and a specific knowledge about the expected results. Our aim was to find a balance between the final image list accuracy and performance by reducing the number of colors and ignoring comparison of object sizes between matched objects in the input image and an image being compared to. We tested the system by querying it with various images and an example with returned similar images is shown in Figures 4 and 5. Generally, the accuracy of the retrieval is similar to the one presented in [2] and [3] as the image indexing procedure is the same. The goal of the solution presented in the paper is to achieve speed and scalability. It is hard

Fig. 4. Example query image with the returned list of images.

Fig. 5. Example query image number two with the returned list of images.

to compare the solution to the previous one by the authors as it is very hardware-depended. We recommend to load objects to RAM before retrieving the queue to minimize the latency of the hard drive reading process (creating retrieving buffer). Even if the reduced dataset is small enough to fit it fully to RAM we strongly recommend to split it in buffered queue parts, unless the dataset is smaller than a single buffer queue part. A partial buffer should not be used in the case of parallel requests to the system. This method is inspired by the Tachyon system of storing metadata of last used objects in fast memory (RAM or SSD). If there exists a possibility to add another parameter (no hardware limitation) to store in a SQL database, e.g. color, texture, structure spatial position, we strongly recommend to do so. This is an important step in reducing dataset within certain parameters which translates into faster processing of smaller datasets. Furthermore, using a stored connection between image and contained structures as a set of keypoints in a database table or object can be considered in counting objects of a certain class. This may allow to treat classified structure as first reducing criteria, if the number of objects meeting this requirement is lower than additional requirements.

V. Conclusion

We presented a distributed system for content-based image retrieval where images are stored and indexed by a relational database management system. This is a very important feature as nearly all CBIR systems are not designed to work in a relational database environment. The system has a good scalability, which means that the number of Slaves can be increased, even through the Internet. To optimize transactions between Master and SQL Server we can merge them into one virtual machine to maximally reduce network operations (with migrating Hadoop to Windows environment due to MS SQL Servers). The proposed approach demonstrates the following

advantages partly coming from the original method presented in [2], [3]:

- the indexing method is relatively accurate in terms of visual object classification,
- learning of features and image classification is very fast,
- expanding the system knowledge is efficient as adding new visual classes to the system requires generation of new fuzzy rules whereas in the case of, e.g., bag-of-features it requires new dictionary generation and re-learning of classifiers,
- the system is highly scalable and the performance of the system depends only on the hardware resources.

The accuracy of the retrieval is similar to the one presented in [2] and [3] as the image indexing procedure is the same. The goal of the solution presented in the paper is to achieve speed and scalability. It is hard to compare the solution speed to the previous one by the authors as it is very hardware-depended. Generally, it is faster than a not-distributed version and adding more hardware and Slave machines makes it faster. There can be done some further changes in the processing of the initial input image by incorporating features similar to [17]. This means the proposed system can recognize objects with color parameter given. Such a feature might be able to reduce a subset of compared images, which means faster processing on a smaller amount of objects matching color requirements. Storing additional data such as texture, color or approximate size of objects (proportionally to relevant objects in other stored images and compared to background objects at the same image) can lead to reduce processing time due to a smaller dataset containing only objects referenced by SQL query results. The proposed solution is semi-parallel because only the file system has been distributed. Our work is mostly aimed at eliminating a bottleneck of image retrieval systems designed as single-entity solutions. The list sorting process can be further parallelized but it was not within the scope of the presented work. Performance of the database part of the solution can be also increased through the use of a SQL server cluster, where the process of generating the index in the form of rules can be parallelized and spread across several servers. There is also a possibility to exchange the relational database engine into a distributed database, e.g. Apache HBase or MapR-DB.

REFERENCES

[1] V. E. Ogle and M. Stonebraker, "Chabot: retrieval from a relational database of images," *Computer*, vol. 28, no. 9, pp. 40–48, Sep 1995.

[2] M. Korytkowski, L. Rutkowski, and R. Scherer, "Fast image classification by boosting fuzzy classifiers," *Information Sciences*, vol. 327, pp. 175–182, 2016.

[3] M. Korytkowski, "Novel visual information indexing in relational databases," *Integrated Computer-Aided Engineering*, vol. 24, no. 2, pp. 119–128, 2017.

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[5] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, Nov 2011, pp. 2564–2571.

[7] K. Cpalka, K. Lapa, A. Przybyl, and M. Zalasinski, "A new method for designing neuro-fuzzy systems for nonlinear modelling with interpretability aspects," *Neurocomput.*, vol. 135, no. C, pp. 203–217, Jul. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2013.12.031

[8] I. A. Harmati, A. Bukovics, and L. T. Koczy, "Minkowski's inequality based sensitivity analysis of fuzzy signatures," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 6, no. 4, pp. 219–229, 2016.

[9] K. Łapa, J. Szczypta, and R. Venkatesan, "Aspects of structure and parameters selection of control systems using selected multi-population algorithms," in *International Conference on Artificial Intelligence and Soft Computing*. Springer, 2015, pp. 247–260.

[10] M. Prasad, Y.-T. Liu, D.-L. Li, C.-T. Lin, R. R. Shah, and O. P. Kaiwartya, "A new mechanism for data visualization with tsk-type preprocessed collaborative fuzzy rule based system," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 7, no. 1, pp. 33–46, 2017.

[11] M. Scherer, J. Smolag, and A. Gaweda, "Predicting success of bank direct marketing by neuro-fuzzy systems," in *International Conference on Artificial Intelligence and Soft Computing*. Springer International Publishing, 2016, pp. 570–576.

[12] V. Stanovov, E. Semenkin, and O. Semenkina, "Self-configuring hybrid evolutionary algorithm for fuzzy imbalanced classification with adaptive instance selection," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 6, no. 3, pp. 173–188, 2016.

[13] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–511–I–518 vol.1.

[14] K. P. Nguyen, G. Fujita, and V. N. Dieu, "Cuckoo search algorithm for optimal placement and sizing of static var compensator in large-scale power systems," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 6, no. 2, pp. 59–68, 2016.

[15] A. M. Serdah and W. M. Ashour, "Clustering large-scale data based on modified affinity propagation algorithm," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 6, no. 1, pp. 23–33, 2016.

[16] H. Piech and M. Stys, "Serialization of input and output data transfers in parallel structures," *Scientific Research of the Institute of Mathematics and Computer Science*, vol. 6, no. 1, pp. 241–252, 2007.

[17] M. Lagiewka, R. Scherer, and R. Angryk, "Color-based large-scale image retrieval with limited hardware resources," in *Artificial Intelligence and Soft Computing: 15th International Conference, ICAISC 2016, Zakopane, Poland, June 12-16, 2016, Proceedings, Part II*. Springer International Publishing, 2016, pp. 689–699.

[18] M. Gabryel, "A bag-of-features algorithm for applications using a nosql database," in *Information and Software Technologies*, ser. Communications in Computer and Information Science. Springer, Cham, 2016, vol. 639, pp. 332–343.

[19] ——, *The Bag-of-Features Algorithm for Practical Applications Using the MySQL Database*. Springer International Publishing, 2016, pp. 635–646.

[20] L. Juan and O. Gwun, "A comparison of sift, pca-sift and surf," *International Journal of Image Processing (IJIP)*, vol. 3, no. 4, pp. 143–152, 2009.

[21] R. Oji, "An automatic algorithm for object recognition and detection based on asift keypoints," *arXiv preprint arXiv:1211.5829*, 2012.

[22] S. Salehian and Y. Yan, "Comparison of spark resource managers and distributed file systems," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, Oct 2016, pp. 567–572.

[23] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–15.

[24] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[25] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.