DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch123

Image Similarity Search using a Negative Selection Algorithm

Stein Keijzers[1], Peter Maandag[1], Elena Marchiori[1], Ida Sprinkhuizen-Kuyper[2]

[1]Department of Computer Science, Faculty of Sciences, Radboud University Nijmegen, The Netherlands
[2]Radboud University Nijmegen, Donders Institute for Brain, Cognition and Behaviour, Centre for Cognition, The Netherlands
skeijzers@home.nl, pmaandag@upcmail.nl, elenam@cs.ru.nl, i.kuyper@donders.ru.nl

Abstract

The Negative Selection Algorithm is an immune inspired algorithm that can be used for different purposes such as fault detection, data integrity protection and virus detection. In this paper we show how the Negative Selection Algorithm can be adapted to tackle the similar image search problem: given a target image, images from a large database similar to the query have to be detected. Results of our experimental analysis indicate that the proposed algorithm is capable of detecting images similar to a target (self) image, given the right detectors. Source code and data used in the experiments are available on request.

Introduction

The increasing storage capacity of modern disk devices allows to collect and distribute large-scale image data efficiently. As a consequence, an enormous amount of images is generated and made publicly available. This phenomenon has boosted research on search for similar images. Search by image as implemented for example by Google (2013) allows one to discover all sorts of content that is related to a specific image. Search results include similar images, webpages and even sites that include that picture. The main challenge is to develop efficient methods for achieving high image retrieval quality. Many techniques have been proposed for this task (see for instance Smeulders et al. (2000)).

A crucial step in image similarity search is the comparison of two images, typically a target image and an image from the database. Image distance measures are used for this task. An image distance measure compares the similarity of two images in various dimensions such as color, texture, shape, and others.

Our goal is to investigate whether a negative selection algorithm with simple techniques for comparing images is suitable for tackling the Image Similarity Search problem.

To this aim, we designate the target image as self data and then create detectors for anything that is not self. We consider a simple setting where color is used as the only feature to define distances between images. Color does not depend on image size or orientation, hence is easy to handle.

In order to analyze whether NSA is applicable to Image Similarity Search, two sets of experiments are conducted. First, we consider a publicly available dataset of holiday images and assess manually how good the results of NSA are. Second, we construct a specific dataset consisting of three classes and manually associate a class to each image. We perform a leave-one-out cross validation on the dataset. Specifically for each image, we consider it as target image and use the rest of the data to search for similar images. Then average precision of each target is computed and the mean results are analyzed.

Results of experiments indicate the suitability of negative selection for this task, considering the fact that we use only color as feature. We would like to stress that the goal of this paper is not to try to compete with algorithms such as those used by Google, but to show that NSA can be applied to search by image.

Background

The original negative selection algorithm is inspired by the way that natural immune systems distinguish self from other (Forrest et al., 1994). When the body encounters a virus or any cells that do not belong to the body's own cells, white blood cells are sent out to react to this and to destroy the foreign cells. A specific type of these blood cells are the so-called T-Cells. One of the things these might do is destroy cells infected by viruses. However, one interesting property of these cells is that they somehow know how to discriminate between foreign tissue and your own body. The process that leaves only these T-cells alive is called negative selection (Dasgupta et al., 2011).

The T-cells are formed in the thymus. The thymus is a small organ that is located in front of the heart and near the sternum. In this organ there are a lot of proteins that belong to the body itself. When a T-cell is just formed it might attack these proteins. These immature T-cells that strongly bind to these self-antigens undergo a controlled programmed cell death, referred to as apoptosis (Stibor et al., 2005). We will not go into the details of this mechanism. What is important to observe is that generally only T-cells that attack

foreign antigens leave the Thymus. Thus, the T-cells that survive this process should be nonreactive to self-antigens, but attack antigens they don't know instead.

The Negative Selection Algorithm (NSA) (Forrest et al., 1994) is inspired by a main mechanism in the thymus that produces a set of mature T-cells capable of binding only to non-self antigens. Many variants of this mechanism exist, often exploiting other natural elements, see for instance Gao et al. (2008) and Shapiro et al. (2005). NSA has many applications, most notably in the field of fault and intrusion detection (Dasgupta and Forrest, 1995; Kim and Bentley, 2001; Taylor and Corne, 2003; Dasgupta et al., 2004). Another typical application is anomaly detection (Dasgupta and Majumdar, 2002; González and Dasgupta, 2003).

NSA basically consists of two steps, as shown schematically in Figure 1 and 2 for the scenario where the integrity of a data file or string has to be protected (Forrest et al., 1994). Using NSA, the first step then is to generate a set of detectors. Each detector is a string that does not match a predetermined substring of the protected data. For matching, usually a partial matching rule is defined, because it can be extremely rare that random strings that are generated exactly match the source data, even if these strings are small.

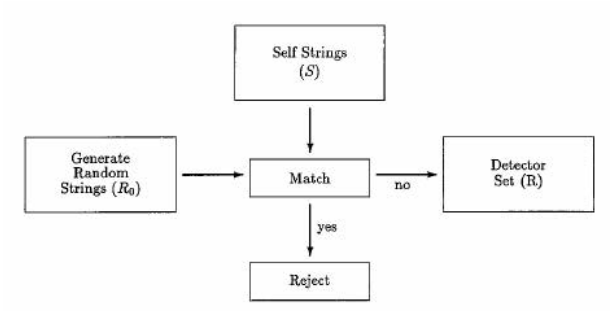

Figure 1: Detector Generation, taken from Forrest et al. (1994)

The second step is to continually monitor the data by comparing them with the detectors. If a detector is ever activated, a change is known to have occurred.

Although this approach might seem too simple to work, it is rather effective: a fairly small set of detector strings has a very high probability of noticing a random change to the original data (Forrest et al., 1994).

NSA for image similarity search

We want to apply the NSA to image similarity search. To this aim, we designate the target image as self data and then create detectors for anything that is not self. For this to work we try to match pixels or pixel groups on each other and use the fraction of matching groups as a similarity measure. A match can be determined via a direct pixel to pixel comparison, but also via a similarity measure, like the one that will

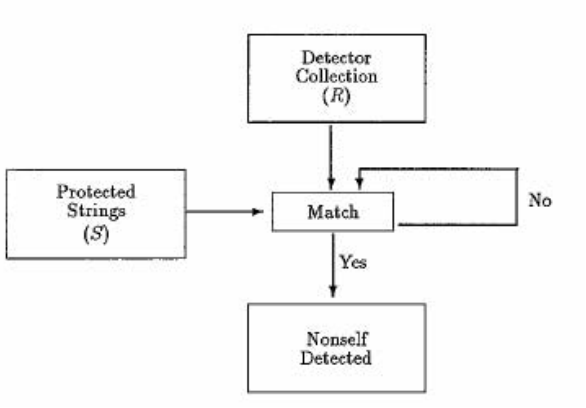

Figure 2: Monitor protected data, taken from Forrest et al. (1994)

be explained later.

Once we have generated detectors implementing these methods we can match them on the data set and retrieve dissimilarity values, which can then be inverted and used to identify images that are very similar to the target image.

In effect, we're trying an alternative approach to achieve what Google does with reverse image search. You can upload a picture to Google and Google returns a list of similar images on the web (Google, 2013). In our case we adapt the negative selection algorithm to accomplish this.

Framework

The NSA algorithm is implemented using C++, because we needed to load and clear many Gigabytes of images in the RAM and this can be done efficiently in C++. It's also object-oriented, so producing a framework was easier.

To apply the algorithm we wrote a framework that handles file input and output. It includes a Detector super-class, which has initialization and detection functions by default. Any detector we implemented was able to provide implementations and extensions to this detector class, while maintaining the basic functionality needed for the algorithm.

Figure 3 illustrates the proposed method. The NSA algorithm (in the center) receives images (the target image and the database of images for searching) from an image-reader class. It then applies its detectors to those images to get a similarity measure.

Detectors

The algorithm is strongly dependent on the detectors that are created. Each of the detectors implements a basic measure of similarity between two images. The idea here is that, with enough of these detectors, we can get a reasonable approximation for the actual similarity by combining the advantages of each detector and by canceling out the downsides of each single similarity measure. The advantage is then that running these simple detectors, possibly in parallel, would

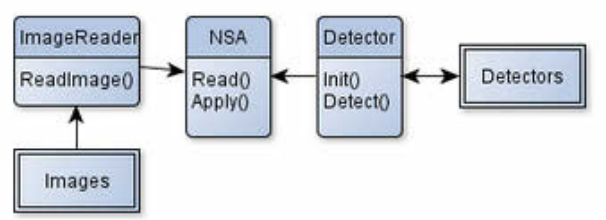

Figure 3: NSA framework

be much cheaper than running more complicated measures, such as those based on compact data structures and Earth Movers Distance (see e.g. Lv et al. (2004)).

Direct pixel similarity (DPS) One of the first ideas that comes to mind when implementing a detector is the direct pixel similarity (DPS). In this case we are working at the pixel level. So the most simple and obvious similarity measure is a direct RGB comparison of each pixel. If the pixels are the same they match, if they are different they do not match. All the matches are counted and normalized to one.

Obviously this is not a very good detector since it is rare that many pixels in a picture match exactly with the target picture. There might be subtle changes in brightness or other minor differences that are barely visible and that will negatively influence such similarity measures.

Therefore we weaken the similarity match by considering a range for each pixel value v, such that we still agree on a match if the value v of each color component is in the range $[v - r, v + r]$, for a given paramter r.

Furthermore, we define a match on a group of n by n pixels if for a certain threshold t, t pixels in a group of the target image match the foreign image. In the end we count all matches and divide by the total amount of pixel groups that were compared to get the final similarity measure. In Figure 4 we can see how the proposed matching pixel group comparison works with $n = 3$, $t = 5$, and $r = 0$. The pixels in row 1 up to 3 and column 1 up to 3 are directly compared. Every pixel in this group has the same value so the match count is 9. This is bigger than the threshold $t = 5$ so these pixel groups match.

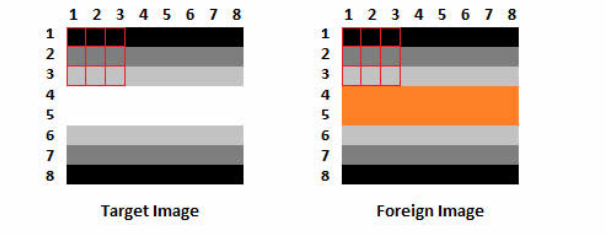

Figure 4: Matching pixel group comparison with $n = 3$, $t = 5$

Figure 5 shows an example of a pixel group that does not match. The pixels in row 4 up to 6 and column 1 up to 3 are directly compared. But the pixels in row 4 and 5 have different values so there are only 3 pixels that match. Since $3 < t$, these pixel groups do not match.

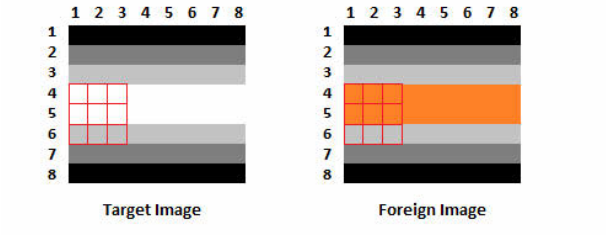

Figure 5: Non-matching pixel group comparison with $n = 3$, $t = 5$

Average color difference similarity (ACDS) Another measure we consider incorporates the summed color difference of each color component. By dividing this summed difference by the absolute maximum difference possible, we get a similarity measure between 0 and 1: the average color difference similarity (ACDS).

The same matching rule as in the previous method can then be used on a group of pixels, but in this case we define the threshold t to be a value between 0 and 1. In this way we are basically saying that a pixel group of the foreign image has to match for $100t$ percent with the target image.

This measure has as advantage that its a lot more flexible: if a picture is similar in part to the target image, but different in some others, it might still get a high score. A disadvantage is that, since comparing colors is not that easy, you can get some results, where colors are considered similar even though to our eyes they might not be. For instance, brown is very similar to grey from pure values, but to the human eye it would be different.

Application

In our application we use the holiday data set (Jegou et al., 2008) that consist of 812 pictures that were made by people on their holidays, see Figure 6 for a snapshot. The collection is applicable to the problem at hand because it contains clusters of similar images, for example for outdoor environments, while maintaining a very broad category, in contrast to other datasets which are either too specific or too general (Toet et al., 2001; Deng et al., 2009; Chen et al., 2009). Indoor surroundings with similar characteristics can also be found, so we were able to identify similar images from this set. A downside of this dataset is that the original images vary in width and height so additional preprocessing on the data set had to be done to improve performance in both runtime and quality. We have resized all images to a smaller, equal size before the algorithm was run. Manual inspection of the results was used to assess their quality.

Figure 6: Snapshot of image dataset

We experimented with the length of detector bit strings as suggested in Forrest et al. (1994), namely powers of 2 ranging from 32 to 256 for this value, which in our case is equal to n^2. Forrest et al. (1994) used a partial matching function that measures an amount of contiguous matches. Since our implemented initial detectors only use direct pixel comparisons, we could not take over their values directly. Instead for DPS we have performed some experiments with values between 0.5 and 1 for the threshold t. A too high threshold causes bad results because it means that every pixel in a group has to be equal to the other group, while a too low threshold causes that every picture is more likely to be highly similar to another. A threshold t of $0.75n^2$ in combination with $n = 4$ yields good results for DPS.

For ACDS the threshold was also manually tuned. In this case a too high treshold results into every picture being similar, while a too low threshold returns a similarity of at most 0. A threshold of 0.1 in combination with $n = 16$ seemed to be nicely in the middle for ACDS.

The optimal value for the color range r was also determined by experimenting. Small values quickly start to give back topmost results which have 0% similarity. If r is set too high then everything is being returned as similar. A value of $r = 30$ gives good results for most pictures. An exception is the city target image (see below) for which a value of $t = 15$ gives better results.

Parameters

After conducting many experiments we settled for $n = 4$, $t = 0.75 * n^2$, $r = 30$ for DPS and $n = 16$, $t = 0.1$ for ACDS.

In the following, we will show results for three target images in different types of environments: city, scenery and an anemone in sea.

City

The city target image is shown in Figure 7. It is characterised by a high diversity in the image.

Figure 7: City Target Image

Figure 8: Best Results for City Direct Pixel Similarity with $n = 16, t = 0.75 * n^2, r = 15$

Figure 9: Best Results for City Direct Pixel Similarity with $n = 16, t = 0.75 * n^2, r = 30$

Figure 8 and Figure 9 show the results for the DPS detector with $r = 15$ and $r = 30$, respectively. These results clearly demonstrate the effects of the range value. If the range is set too high then different colors are considered similar. In the case of $r = 30$ the first result can be explained by the characteristics of the target image. There is the blue sky and a dark part of the city. The piramid photo shows exactly this pattern. If the range is set to a narrower value $r = 15$, better results are obtained.

The results of the ACDS detector for the city image are similar to those in Figure 9. This detector clearly has problems with the diversity in the city view.

Scenery

A scenery target image considered is shown in Figure 10. It is characterised by a bright sky in the upper half and a darker soil in the lower half.

Figure 11 shows the resulting images for Direct Pixel Similarity. We can see that the most similar images that are returned resemble the characteristics of a bright sky in the upper half and a darker substance in the lower part, which corresponds with core characteristics of the target image.

Figure 12 shows the resulting images for Average Color Difference Similarity. Here the resemblance between pictures is still pretty high, but less distinct. For example in the output images number 8 and 3 a less clear distinction between the upper half and the lower half can be observed.

Figure 10: Scenery Target Image

Anemone

An Anemone target image in a sea environment is shown in Figure 13. It is characterised by shades of green over the whole image without a clear separation between the sky and the ground as in the scenery setting.

Figure 14 shows the resulting images for Direct Pixel Similarity. The top three results are exactly what we want to see. The rest of the images also contain similar shades

Figure 11: Best results for Scenery Direct Pixel Similarity with $n = 4, t = 0.75 * n^2, r = 30$

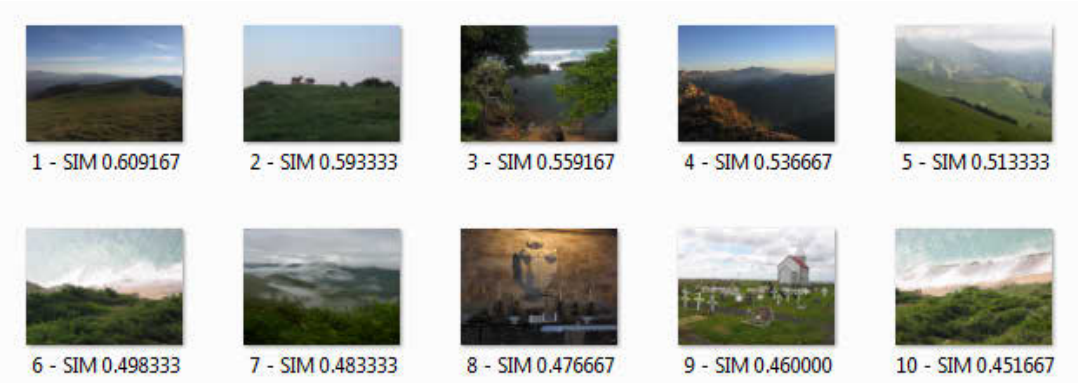

Figure 12: Best results for Scenery Average Color Difference Similarity with $n = 16, t = 0.1$

Figure 13: Anemone Target Image

of green as we would expect. However there are also less obvious hits such as 6, 8 and 10. This might be explained by the fact that the target picture has a lot of variation in pixel colors in each group.

The Average Color Difference yields even worse results, see Figure 15, because it focusses on the average color in a group. This means that pictures which have many different pixel colors within each group will have a higher probability of being similar to each other when they are averaged, which can yield strange results.

Validation on labeled data

In order to assess the quality of the results automatically we consider a small dataset containing 60 images manually labeled according to three classes.

For each image, we remove it from the dataset and compute the average precision (see e.g. Müller et al. (2001)) over the entire ranking. Mean results over all images are reported in Table 1. In order to analyze the significance of these results, the average precision of each target image is compared to that of 1000 random rankings. Empirical p-values are then computed, as the fraction of times the average precision on random ranking was better than that on the ranking generated using NSA. The resulting p-values are 0.004 for ACDS and 0.001 for DPS. Results indicate that NSA is better than a (random) baseline method, that DPS and ACDS have similar variance, and that DPS performs better than ACDS.

Figure 14: Best results for Anemone Direct Pixel Similarity with $n = 4, t = 0.75 * n^2, r = 30$

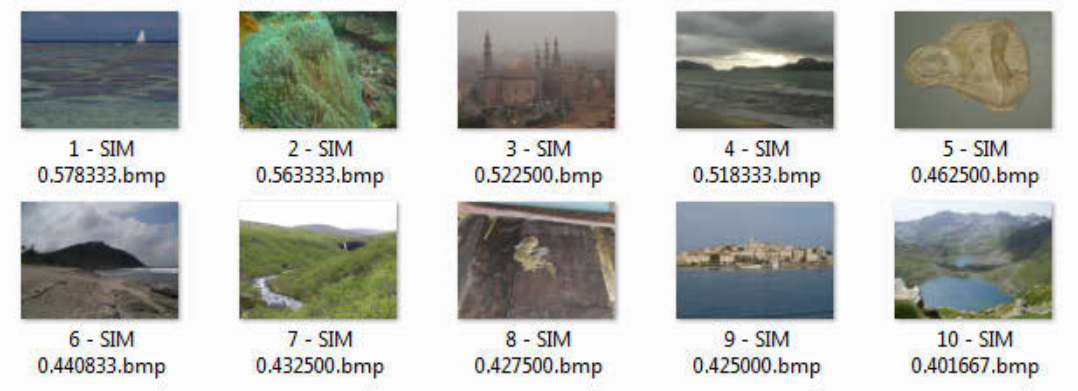

Figure 15: Best results for Anemone Average Color Difference Similarity with $n = 16, t = 0.1$

mean average precision ACDS (std)	mean average precision DPS (std)
35.65% (3.39)	41.82% (3.43)

Table 1: Leave-one-out results on a manually labeled dataset with three classes. Mean average precision across the images; std denotes standard deviation.

Discussion

This paper investigated the use of NSA for content-based image retrieval. We have introduced a simple method based on NSA and showed that it can achieve promising similarity search results on a collection of general-purpose images as assessed by human inspection.

In future work we plan to extend the proposed method by incorporating also detectors defined on dimensions such as texture and shape. This will allow to handle image collections of general-purpose images from various given categories. In this way, for each image, the sets of images from its category can be used as the ground truth in the evaluation. Hence effectiveness measures such as average precision can be used to formally assess the performance of a method. This will allow us to perform a comparative assessment of the proposed method using state-of-the-art algorithms for this task.

The two detectors we have implemented are just two examples of what can be done to measure similarity. There are many more possibilities notably measures based on Earth Mover's Distance, which is a similarity measure between multidimensional distributions (Lv et al., 2004). Other similarity measures include contrast measurements, color matrices or even completely different approaches such as edge detection. If we can use these detectors together, possibly in parallel, the performance and resulting similarity might be improved even further.

Nevertheless the current results based on our simple method are already fairly similar to (manual) human eye selection, which is usually what a user of such a system would want. Our conclusion is therefore that Negative Selection Algorithms can indeed aid in creating a search system for image similarity search.

A number of aspects of this research remain to be investigated. For instance, adding non-determinism and random elements to detectors, thereby applying the algorithm more literally, could give more varied results, but also likely less precise. Detector coverage in such a non-deterministic scenario could then be improved by applying analytic methods (Ji and Dasgupta, 2005). Another possibility is increasing performance of this algorithm, either by better pre-processing of the data (see e.g. the methods described in the survey by Smeulders et al. (2000)), or by applying detectors in parallel.

References

Chen, M., Dhingra, K., Wu, W., Yang, L., Sukthankar, R., and Yang, J. (2009). PFID: Pittsburgh fast-food image dataset. In *Proceedings of the 16th IEEE international conference on Image processing*, ICIP'09, pages 289–292, Piscataway, NJ, USA. IEEE Press.

Dasgupta, D. and Forrest, S. (1995). Tool breakage detection in milling operations using a negative-selection algorithm. Technical Report CS95-5, Department of Computer Science, University of New Mexico, Albuquerque, NM.

Dasgupta, D., KrishnaKumar, K., Wong, D., and Berry, M. (2004). Negative selection algorithm for aircraft fault detection. In *Proceedings of the 3rd International Conference on Artificial Immune Systems, ICARIS*, volume 3239 of *Lecture Notes in Computer Science*, pages 1–13, Berlin Heidelberg. Springer.

Dasgupta, D. and Majumdar, N. S. (2002). Anomaly detection in multidimensional data using negative selection algorithm. In *Proceedings of the 2002 Congress on Evolutionary Computation*, volume 2 of *CEC'02*, pages 1039–1044. IEEE.

Dasgupta, D., Yu, S., and Nino, F. (2011). Recent advances in artificial immune systems: models and applications. *Applied Soft Computing*, 11(2):1574–1587.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2009, pages 248–255. IEEE.

Forrest, S., Perelson, A. S., Allen, L., and Cherukuri, R. (1994). Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Computer Society Press.

Gao, X.-Z., Ovaska, S. J., and Wang, X. (2008). A GA-based negative selection algorithm. *International Journal of Innovative Computing, Information and Control*, 4(4):971–979.

González, F. A. and Dasgupta, D. (2003). Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4):383–403.

Google (2013). Inside search - google. `http://www.google.com/insidesearch/features/images/searchbyimage.html`.

Jegou, H., Douze, M., and Schmid, C. (2008). Hamming embedding and weak geometry consistency for large scale image search. *Proceedings of the 10th European conference on Computer vision*. INRIA Holidays dataset available at `http://lear.inrialpes.fr/~jegou/data.php#holidays`.

Ji, Z. and Dasgupta, D. (2005). Estimating the detector coverage in a negative selection algorithm. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 281–288, New York, NY, USA. ACM.

Kim, J. and Bentley, P. J. (2001). An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO-2001, pages 1330–1337, San Francisco, US. Morgan Kaufmann.

Lv, Q., Charikar, M., and Li, K. (2004). Image similarity search with compact data structures. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 208–217, New York, NY, USA. ACM.

Müller, H., Müller, W., Squire, D. M., Marchand-Maillet, S., and Pun, T. (2001). Performance evaluation in content-based image retrieval: overview and proposals. *Pattern Recogn. Lett.*, 22(5):593–601.

Shapiro, J. M., Lamont, G. B., and Peterson, G. L. (2005). An evolutionary algorithm to generate hyper-ellipsoid detectors for negative selection. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 337–344, New York, NY, USA. ACM.

Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., and Jain, R. (2000). Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380.

Stibor, T., Mohr, P., Timmis, J., and Eckert, C. (2005). Is negative selection appropriate for anomaly detection? In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 321–328, New York, NY, USA. ACM.

Taylor, D. W. and Corne, D. W. (2003). An investigation of the negative selection algorithm for fault detection in refrigeration systems. In *Proceeding of Second International Conference on Artificial Immune Systems, ICARIS*, volume 2787 of *Lecture Notes Computer Science*, pages 34–45. Springer, Berlin Heidelberg.

Toet, A., Bijl, P., and Valeton, J. M. (2001). Image dataset for testing search and detection models. *Optical Engineering*, 40(9):1760–1767.