A Reverse Image Search Engine for Tracking Image Usage Across the Internet

Jake Grogan[1], Connor Mulready[1], James McDermott[1], Martynas Urbanavicius[1]

[1]School of Computing, Dublin City University, Ireland

jake.grogan8@mail.dcu.ie, connor.mulready2@mail.dcu.ie, martynas.urbanavicius2@mail.dcu.ie, james.mcdermott7@mail.dcu.ie

Abstract. This report details a specification for a reverse image search system for tracking the usage of images across the internet. The purpose of this system is to allow the potential user to upload images to an interface which performs reverse image search and returns a range of links to web pages which have been identified to contain an exact or modified copy of the original image. This allows the user to analyse the usage of images across the web. Within this report, we describe and compare possible algorithms for indexing and searching images, along with detailing both the benefits and limitations of sub-components which the system will be comprised of. We have included mockups which detail a proposed user interface, along with a set of architectural diagrams, implementation descriptions, and evaluation plans.

Keywords: Reverse Image Search, Vantage-Point Trees, Feature Extraction, Indexing, Deep Ranking

1. Introduction

Reverse image search is a content-based information retrieval technique incorporating the use of images that a system can base its search upon. This type of search involves a distinct lack of keywords/terms in the search query, removing the need for a user to continuously guess, a process less efficient in returning a correct result. This type of search also allows users to discover metadata related to an image, such as its popularity, or whether manipulated/compressed versions exist.

Our motivation in designing this system is centered around the idea of copyright, and the widespread immoral plundering of image content from across the internet. Tracking the piracy of images is a large scale problem, and though some defense mechanisms exist there are ways in which such devices may be bypassed.

To satisfy this motivation, we aim to address this problem by designing a system which tracks the usage of images across the web and can identify cases of online copyright infringement. We can discern that the objective and purpose of this system is to allow users to search via sample image and retrieve similar or matching images, which in turn could lead to the identification of images which are in breach of copyright.

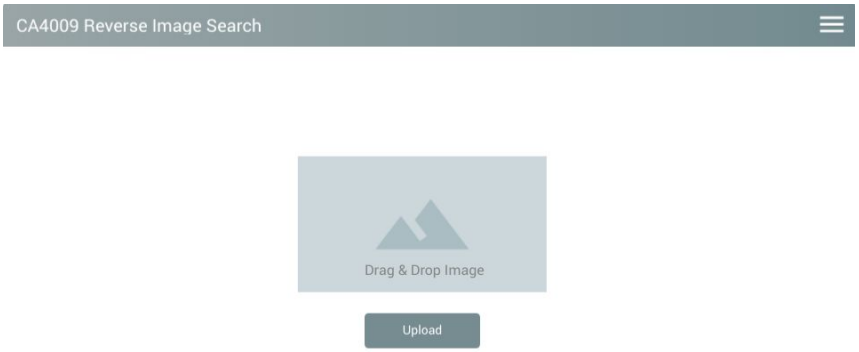
Our system will take a sample image provided by the user and extract metadata which will later be used for ranking. This image will be converted to grey-scale as removing colour means objects will not be defined by colour. The image is downscaled to speed up feature extraction, which is carried out to identify unique characteristics of the image. These characteristics are hashed and indexed in a Vantage-point tree, a structure with the ability to efficiently partition data in n-dimensional metric space, along with associated metadata. This tree can be searched to a given depth for similar images. Images retrieved are ranked by metadata stored in the tree along with the Hamming Distance between the hashes of the sample image provided by the user and the hash of each respective image retrieved from the Vantage-point Tree.

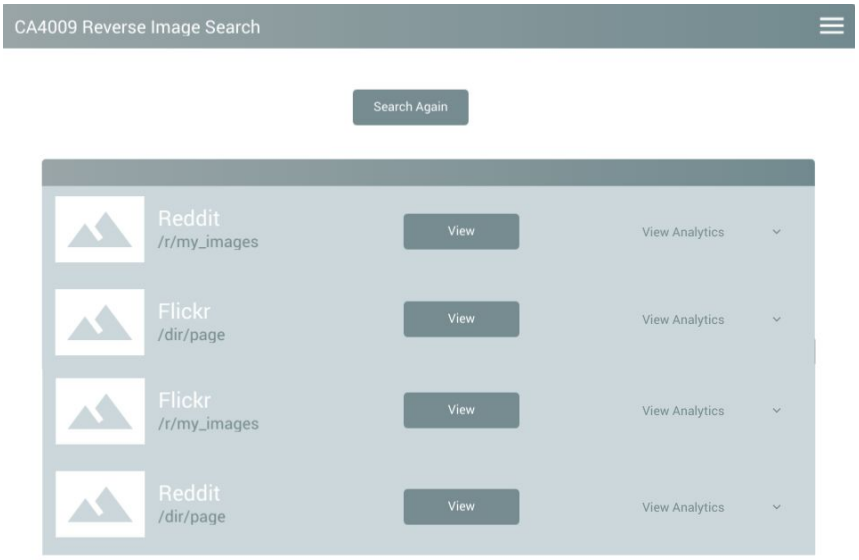
One limitation with this system comes in the form of ranking during search. How information is indexed should strike a balance between precision and recall when searching, so all relevant images are retrieved, even if they have been transformed. Precision in our indexing will yield less false negatives whilst retrieving less results. Recall is an inverse of precision, more false negatives are yielded as a consequence of more results being retrieved in total. Striking this balance will prove to be a limiting factor in the quality of how images are retrieved and ranked.

2. User Analysis

Our system will be used by anyone wanting to track usage of an image across the internet. While we are aware that this is a large catchment, we can drill down to specific user types based on why they are performing reverse image lookup. As previously mentioned, our system recognizes cases of copyright infringement. This means that the likes of artists, photographers, and general creators should be able to utilise the system. Obviously we can expect users outside of these groups, but for the purposes of this report we want to zone in on user’s that this system would prove beneficial to.

Ideally little knowledge is needed to use our system. We expect that users have prior knowledge on uploading files online. The UI below is designed to be simple and straight-forward, it shouldn't seem intimidating or unfamiliar to anyone. Therefore, it is fair to state that our system can be used by everyone, from those with little knowledge of search engines to well experienced search engine users. As previously stated, mockups of the UI for uploading a query image and the resulting search results UI can be found below.





We expect query types to fall into one basic category - image based queries. Queries made to our system will not require textual input from the user, instead it requires the user to upload a simple sample image, which will then be processed by the system. Outside of this, no other types of queries will be made.

Following our data flow diagram below, the user starts by uploading a query image through the web interface. From this point, the system begins the processing of the image and the retrieval of results is hidden from the user. Retrieved results are then presented to the user in an affable way, such that any user should be able to interpret the results with ease. A detailed explanation of the processing and retrieval stages can be found in *Section 3.1 - Architecture*.

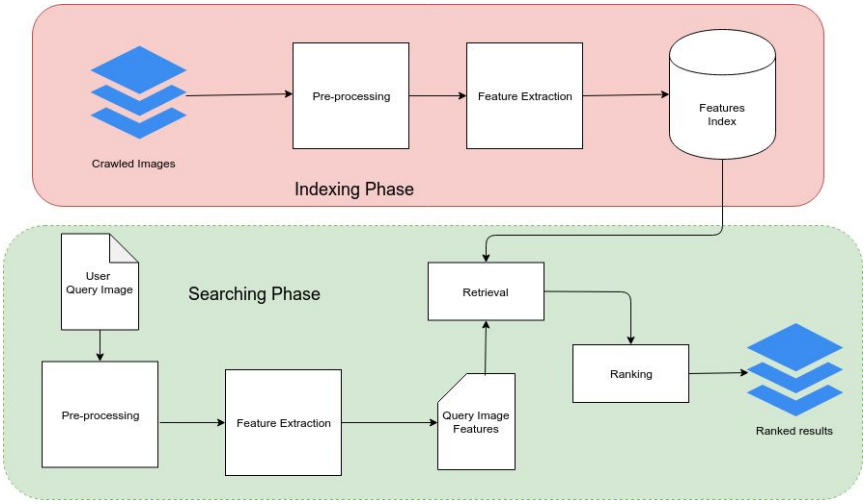
3. Functional Description

3.1. Architecture

Below is an architecture diagram which details the two separate phases of our system. These can be seen as two separate flows of data in the system.

The upper half of the diagram , the *Indexing Phase*, shows the path to the Index which crawled images follow. First they are to be pre-processed, which includes processes such as grayscaling and downscaling, the latter of which takes place only if the image ‘s dimensions are greater than a set value. These pre-processing steps are taken in order to speed up edge detection as these same crawled images undergo feature extraction, which is carried out to identify unique characteristics of every crawled image. From here crawled images are indexed in the Feature Index which is a VP tree, which will be utilised for image retrieval in the *Searching Phase*.

The lower half of the diagram details the aforementioned *Searching Phase*, which begins when a user uploads a sample image. This again undergoes pre-processing, which as detailed above applies the processes of grayscaling and downscaling (where appropriate). This is done to speed up edge detection which will be used to query the images features against that in the Feature Index, which was built during the *Indexing Phase.* This Feature Index consists of hashes generated using a perceptual hashing function which are stored in a Vantage-point Tree along with other information such as the URL the image originated from. Images retrieved are then ranked and displayed to the end user.



In the above diagram, we talk about the processes of building the *Feature Index, Feature Extraction*, and *Ranking* of retrieved images. In the next section we expand upon our decision making process when it came to selecting the appropriate algorithms and data structures that will power these operations.

3.2 Algorithms

3.2.1. Feature Extraction

Features are extracted by means of edge detection. While there is no one-size-fits-all algorithm for such extraction, we discovered two detection algorithms known as Canny edge[2] and Marr-Hildreth edge [8] which can be used for such a purpose. Canny edge is based on extrema of the first derivative of the Gaussian operator, where Marr-Hildreth edge is based on the zero-crossings of the Laplacian of the Gaussian operator, which in both cases, are applied to the image for various values of sigma, the standard derivative of the Guassian.

The Canny edge algorithm extends the Sobel edge algorithm. That is to say that the input of Canny edge is the output of Sobel edge. Canny edge will receive all discovered edges from Sobel edge and proceed to adjust the width of the edges to a single pixel. This is done by first finding the orientation of the edge and checking neighbours for the strongest/brightest point.

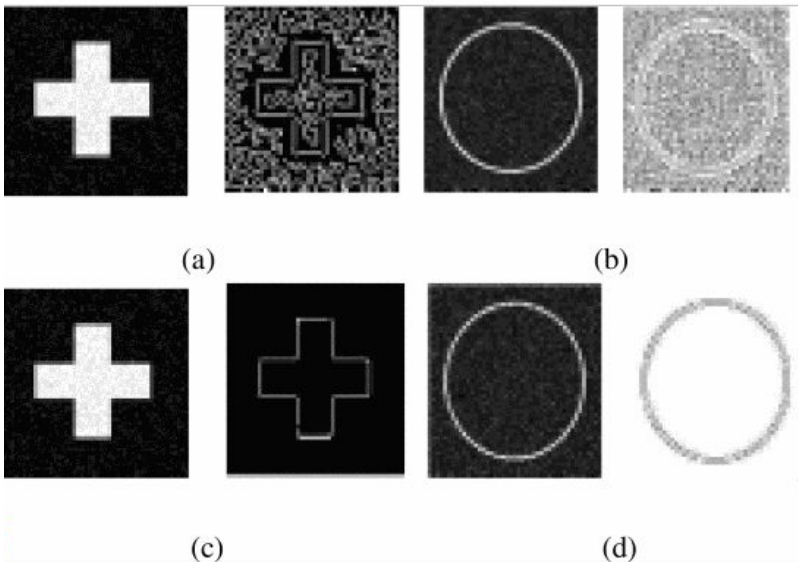


Figure 1. A comparison between Canny and Marr-Hildreth edge detectors on noisy image (a) and (b) using Marr-Hildreth, (c) and (d) using canny

The image above compares both Canny and Marr-Hildreth edge detectors, showing that Canny detectors do a better job on localization, though edge contours become fragmented. This means that if we were concerned with preserving said contours, the Marr-Hildreth edge algorithm would be better suited. If using Canny edge, a technique called Gausian smoothing can be utilised to alleviate any errors that would occur due to noisy pixels. This is done all the while preserving strong edges, which is why we chose to use it.

3.2.2. Generating Perceptual Hash

Extracted features are then used to generate the fingerprint of an image as a perceptual hash. Perceptual hash functions are analogous producing similar hash values when input features are similar, whereas cryptographic hashing relies on the avalanche effect where small change of input results in drastic change in resulting output.
For our system we chose discrete cosine transform as the perceptual hash function since it performs the best when images are modified due to DCT coefficients staying stable under image modifications e.g applying blur, crop, compression, grayscale or rotation.

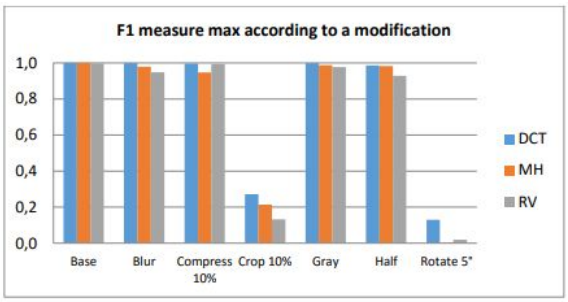


Figure 2. Maximum F-measure of DCT, MH and RV perceptual hash function against modifications

The system will scale down the image to 32x32 pixels and apply a discrete cosine transformation per row and per column. Pixels with the highest frequencies will all be presented in the top left corner 8x8 in size which we use to calculate the hash. Next we will calculate the median of the greyscale values from the top left corner and generate the hash by turning all values below median to 1 and values above median to 0 resulting in a 64bit hash.

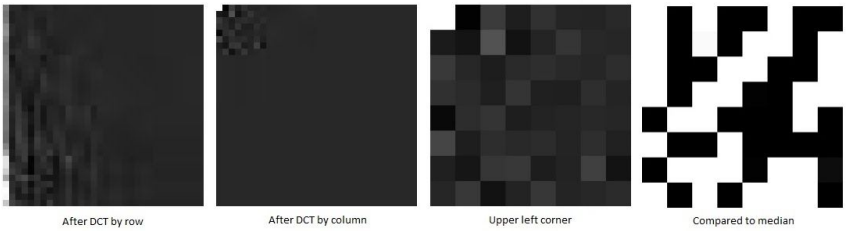


Figure 3. Generating perceptual hash

3.3. Vantage-Point-Trees

A Vantage Point tree [4], more commonly referred to as a VP-tree is a type of metric tree [5]. Metric trees have the ability to efficiently partition data in n-dimensional metric space.

One advantage of VP-trees is that we can perform range queries on a dataset, for example, doing nearest neighbour search. These are quite similar to KD-trees commonly used in K-Nearest-Neighbour search.

VP-trees work by selecting a position in space known as the vantage point and partitioning the data into two sets

1. Points near the vantage point
2. Points far from the vantage point

This process is then applied recursively, partitioning the space into smaller and smaller sets whereby neighbours in the tree have smaller distances to each other. An illustration of this may be found below:

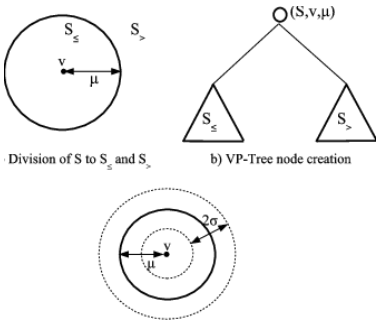


Figure 4.

The inside and outside subtrees both contain exactly half the nodes of the parent. At each node we must calculate the median of the nodes data. This yields a value ***mu*** (μ). This must be calculated at each level and it takes ***O(n)*** time.

The tree makes binary splits on each iteration, therefore the tree has a depth of ***O(log n)***. The overall construction complexity of the tree is ***O(nlogn)***.

Although we pay a slight penalty in constructing the tree, we get very efficient search. Typically, when searching a VP-tree, we search for the nearest neighbour to some point ***x***, however we can extend this functionality to find the ***k*** nearest neighbours of ***x***. In other words, we are searching for the ***k*** most similar images to a query image ***x***.

Any given node in the tree has a vantage point ***v***, and a threshold distance ***t***. Our query image ***x*** will be some distance ***d*** from ***v***. If the distance ***d*** is less than the threshold distance ***t***, then we know we must search the left subtree (i.e. the metric space that is within the threshold distance), otherwise, search the right subtree (i.e. the metric space outside the threshold distance). If, during this recursive process some image ***s*** is discovered such that its distance to the query image ***x*** is less than ***t - d***, it is retrieved as a similar image. We continue this process until we have ***k*** similar images retrieved. If however there are only ***k'*** similar images within the vantage point i.e. ***k'*** images that satisfy distance ***t - d*** from the query image (where ***k'*** < ***k***), then other images must be retrieved from the right subtree.

The space complexity of the VP-tree is approximately ***n***.

There are some things to acknowledge however when employing a VP-tree. Our main concern is the index degradation[4]. As the database grows and new images are added to the index, the tree will become more and more unbalanced. This can cause a complete loss of index efficiency up to the point where the tree must undergo complete reconstruction. Determining when the reconstruction should take place will be outlined in the evaluation section.

4. Retrieval and Ranking

The retrieval process is outlined in the above section. To continue on, when the ***k*** most similar images are retrieved, they are not guaranteed to be ordered by similarity. In order to rank the retrieved images by similarity we must look again at the feature vector hash. As the hashing function is a perceptual hashing function, we can use the hamming distance to rank images by similarity.

The hamming distance is essentially an ***XOR*** operation performed on the hash of each retrieved image and the hash of the query image feature vector. To rank we simply count the number of 1's in the resulting binary string. This will give us our ranking score. Images whose hashes have the smallest hamming distance from the query image hash are the most similar. We therefore rank images from the lowest to the highest hamming distance.

Consider the following images

Figures 5, 6, 7 respectively.

Assume the image on the far left is the query image and the other two images are images retrieved during retrieval. The hamming distance between the query image and the red flowers is 39 while the hamming distance between the query image and the far right butterfly is only 8. From this experiment, and others, it is clear that ranking by hamming distance of the hashes works as an efficient similarity ranking scheme.

5. Evaluation

Evaluation of the proposed system not only requires an evaluation of its effectiveness but it also requires an analysis of certain components in order to determine the best suited values for variables. The first of which is the optimal value for the number of similar images to retrieve from the VP-tree. It is important to run various experiments to determine this value as we want to retrieve as many relevant images as possible and the least amount of irrelevant images.

Secondly, we must run experiments to determine when the VP-tree should be completely rebuilt. This is important as we do not want to do this too often as it would have an enormous performance hit to the system but also not leave it too long as we would lose the index efficiency.

Here we layout a test and evaluation plan to measure the effectiveness of the system. Firstly we must build a large collection of images to index in the search engine. For a large portion of these images we also want to manipulate them in some way. Manipulation of images includes blurring, cropping, rotation, compression, changing the images colour profile, and adding watermarks.

From these base and manipulated images we propose to build a collection of sets. Each set will contain a base (unmodified) image and modifications of that base image. We will then index each image in each set and search using the base image. In order to determine the effectiveness of image retrieval we have outlined some measures below.

The systems precision [10] is the fraction of retrieved documents that are relevant.

$$Precision = \frac{\#(relevant\ items\ retrieved)}{\#(retrieved\ items)}$$

The systems recall [10] is the fraction of relevant documents that are retrieved.

$$Recall\ (R) = \frac{\#(relevant\ items\ retrieved)}{\#(relevant\ items)}$$

A single measure that trades off precision versus recall is the F measure [10], which is the weighted harmonic mean of precision and recall:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

$$F_{\beta=1} = \frac{2PR}{P + R}$$

It is possible to change the weights in the harmonic mean of the F measure in order to tune it. This is done by changing the β parameter. Values of $\beta < 1$ emphasize precision, while values of $\beta > 1$ emphasize recall. It is important to experiment with the value of β and select its appropriate value in order to benchmark the system in accordance with its purpose.

Although not covered here, one possible extension to this is using machine learning to benchmark the systems effectiveness. Deep Ranking [7] has proven to be effective in learning fine-grained image similarity and looks to be a promising technique that could be incorporated for system evaluation.

6. Discussion and Future Work

In this report we have proposed a system for reverse image search. It's motivation, an analysis of it's expected users and their expected skill levels, it's architecture, and underlying algorithms and data structures have also been laid out along with an evaluation plan and mockup user interfaces. We have looked at some key components of the system in greater detail and drew comparisons between different methods for their implementations, giving pros and cons and why we believe they would be best suited to this system.

To the development team, we advise the following steps:

1. The understanding of this proposed system can be concretised by taking time to implement further in-depth architecture component analysis based upon the implementation details and diagrams we have designed.
2. The implementation of the hashing and feature extraction algorithms can be aided by utilising open-source libraries such as pHash and OpenCV.
3. Structures, such as Vantage-point trees, can be implemented through the use of open-source libraries, such as 'vptree' for Python.
4. In finding optimal values for variables (such as K for the number of similar images to retrieve, values of σ and the threshold **t** for canny edge, and when to reconstruct the VP-tree) during integration tests, it's imperative to frequently conduct experiments, as values for these can differ from a theoretical standpoint and a practical implementation.

7. Appendix

Figure 1 - From https://ieeexplore.ieee.org/abstract/document/977981

Figure 2 - From Mathieu Gaillard, Elod Egyed-Ssigmond (2017)*, Large Scale reverse image search; A method of comparison for almost identical image retrieval* [3]

Figure 3 - From https://content-blockchain.org/research/testing-different-image-hash-functions/

Figure 4 - Michail Vlachos*, Illustration of the VP-tree partitioning method and hierarchical structure.*

Figure 5 - Photo by Krzysztof Niewolny on Unsplash

Figure 6 - Photo by Brooke Davis on Unsplash

Figure 7 - Modified photo by Krzysztof Niewolny on Unsplash

8. References

[1] Lad Ami D. et al. (2017), *Reverse Image Search Image retrieval using various techniques*, International Journal of Advance Engineering and Research Development Volume 4, Issue 4:

[2] John Canny (1986), *A Computational Approach to Edge Detection*, IEEE transactions on pattern analysis and machine learning intelligence, Vol. PAMI-8, No. 6:

[3] Mathieu Gaillard, Elod Egyed-Ssigmond (2017)*, Large Scale reverse image search; A method of comparison for almost identical image retrieval,* Université de Lyon.

[4] Il'ya Markov, *VP-tree: Content-Based Image Indexing*, Saint-Petersburg State University -

[5] P. N. Yianilos (1992), *Data Structures and Algorithms for Nearest Neighbour Search in General Metric Spaces*, Proceedings of the Fourth ACM-SIAM Symposium on Discrete Algorithms -

[6] L. Vaughan (2004), *New measurements for search engine evaluation proposed and tested,* Faculty of Information and Media Studies, University of Western Ontario, London, Ont., Canada N6A 5B7

[7] J. Wang *et al.* (2014), *Learning Fine-grained Image Similarity with Deep Ranking,* Google Inc, California Tech, CVPR'2014, IEEE

[8] D. Marr, E. Hildreth (1979), *Theory of edge detection*, M.I.T. Psychology Department and Artificial Intelligence Laboratory, Cambridge, Massachusetts 02139

[9] Van Rijsbergen, C.J. (1979), *Information Retrieval* (2nd ed.), Chapter 7, Butterworth-Heinemann.

[10] L. Derczynski (2016), *Complementarity, F-score, and NLP Evaluation*, Proceedings of the International Conference on Language Resources and Evaluation.