# An Implementation of Google Image Search Functionality on Wikipedia Picture of the Day Archives

Patrick Travers
Dublin City University
CA6005

## ABSTRACT

This project offers a replication of Google Images functionality from its original 2001 release. Images featured on Wikipedia's Picture of the Day (PotD) during the period 2010–2014 are crawled. Associated HTML is used to create textual image surrogates. These surrogates are enriched by querying Amazon's Rekognition computer vision solution. The resulting surrogates are indexed with a BM25 approach. Ranking employs a term-matching approach. A Flask application is built to integrate these steps and display ranked results on a web-based interface. Without a benchmark for results comparison, comprehensive evaluation is difficult but overall performance can be described as mixed. Performance on a selection of queries is quantified and appraised. Potential improvements are outlined along with suggested directions for future works.

## KEYWORDS

Information retrieval, Image search, Wikipedia, BM25, Flask

## 1 Introduction

September 4 1998 is a well-known date in information technology. In Menlo Park, California a company named Google was founded, offering a textual internet search engine service. Little did anyone realise at the time the heights this company would reach. In the beginning, Google's focus remained on refining their textual search engine. Then, in 2001 their product offerings began to expand. In February 2001, Google Groups was launched, offering a means of hosting discussion forums. This service is still in operation today though it is less widely known than Google Search. Later in 2001, another service was launched which has become almost synonymous with Google Search and enjoys as much popularity in the consumer space: Google Images. The service was allegedly spurred by Google's efforts to satisfy their most popular query of all time to date at that time: Jennifer Lopez's green dress at the 2000 Grammy Awards[1]. Since its release, Google Images has become a behemoth of the search space, providing an essential complement to the original Google Search.

The aim of the current project is to implement the necessary components to replicate Google Images as it first appeared in 2001, or at least a miniature version of it. The project will not attempt to broadly index the Internet's images. Rather, it will index images which have featured on Wikipedia's PotD between 2010 and 2014. On this dataset, appropriate crawling, annotation, indexing, search, retrieval, and serving measures are taken to replicate an outline of Google's initial foray into images. Let us outline presently the project's dataset and the process of web crawling used to obtain it.

### 1.1 Web Crawler

The first step was to crawl relevant web pages, extracting images and their associated HTML text. Wikipedia's PotD Archive page contains links to all previous PotD instalments, organised chronologically by month [2]. Beginning from this root, all subsequent links to PotDs occurring between January 2010 and December 2014 were crawled using Python's BeautifulSoup package. On each month's page, the following steps were taken:

1. Identify all image URLs and download to /images subdirectory. Downloading is necessary within the current project architecture to facilitate computer vision at a later stage.
2. For each identified image, find its caption by navigating to the next nearest <p> tag.
3. Check if the <p> tag has any children with tag <small>. This is done to avoid capturing photograph credits, which are stored in <small> tags.
4. Capture the relevant information within <p> tags, as well as anything within <title> tags, if <title> is present.
5. Save all the obtained text to a nested dictionary of structure {filename: {title:'', caption:''}, {filename: {title:'', caption:'' etc.
6. Export this dictionary as JSON for later use at the indexing stage.

With a subdirectory containing all relevant images and a saved JSON file with a record of related HTML text, sufficient data was collected for the project to proceed.

## 2 Annotation

The net outcome of the crawling process described above was a directory containing 1,850 images (note, the figure does not come to the expected (365x5)-1 = 1,824 because of some inconsistencies in PotD; sometimes PotD has been a collection of images rather than a single image). Among these image files there are 1745 JPEGs, 47 SVGs, 39 PNGs, 11 OGVs and 8 GIFs. As described, each image has an associated key in a JSON file storing title and caption for each image. Title and caption both appear

useful representations of their associated images. Title offers in several words a summation of image contents, while caption provides background information, context and other significant points of interest. However, this project attempted to go one step further in enriching images' textual representations as outlined presently.

Further annotation enrichment was sought by applying a computer vision solution. To these ends, Amazon Web Services's (AWS) Recoknition was used. Of the PotD images obtained, 1,085, or 59% were eligible for this service. This is due to restrictions on file types (JPEG and PNG only) and file size (less than 5MB only). All 1,085 eligible images were uploaded to an AWS S3 bucket. Amazon's boto3 SDK was used to query Rekognition with Python. The relevant Python script loaded the pre-made JSON file of HTML image annotations. In cases where Recoknition could identify image contents, recognised labels were stored as strings using an additional 'labels' key, altering the structure of annotations JSON to the following:

{filename: {title:'', caption:'', labels:''}, filename: {title: …etc.

Rekognition provides an estimated probability of labels being present in a given image. In this case, only labels with 75% confidence or greater were kept while the rest were discarded. After the necessary AWS set-up and configuration steps, the annotations obtained were all achieved within the bounds of AWS Free Tier.

## 3 Indexing

Having obtained PotD images, their HTML surrogates, and additional computer vision annotations where possible, the next step was to index the collection images for later search and ranking. Building on prior investigations, a BM25 indexing solution was selected as it was found to exhibit comparably strong performance in indexing a subset of the CLEF 2009 ad-hoc robust task collection (refered to hereafter as CLEF 2009; further details available on request from author). The steps taken to implement this BM25 solution in Python are outlined presently.

Firstly, the JSON obtained from the crawling/annotation stages was loaded. Then, two passes were needed over the collection to produce the necessary index. A Python dictionary which we will refer to as 'idx' was used to store the index. On the first pass, the following steps were taken for each JSON key:

1. Title, caption and labels (collectively referred to as 'surrogate' hereafter) of each image passed were collapsed to a single string and passed through pre-processing where single-character words and stop-words were removed, along with words containing punctuation and/or numbers.
2. All remaining words were lower-cased, Porter stemmed and split by whitespace to a list.
3. For the resulting list, its length as well as the frequency of each of its constituent terms were stored as a tuple within idx[term][filename].
4. The length of each surrogate was also used to update a counter keeping track of the total number of words in the entire collection, as this information is also required by BM25.

With these steps, the first pass over the collection was completed. The information collected on the first was used to compute average surrogate length (total number of words in the collection / total number of items in the collection). At this juncture it is worth noting that BM25 also requires two hyperparameters termed k and b. These values are set to 1.2 and 0.75 respectively as they were found to perform well on the aforementioned CLEF 2009 task.

With idx now populated with initial data for each document from the first pass, the second pass executed the following steps for idx entry:

1. Calculate the total number of documents containing a word.
2. For each surrogate containing that word, apply the BM25 formula which will give a term-surrogate relevance score. This score is stored in idx in place of the temporary first-pass data.

With this much completed for all idx entries, the full BM25 index was exported as JSON for later use.

## 4 Retrieval

Once again drawing on the approach taken in applying BM25 to CLEF 2009, term-matching ranking was used rather than the more widely used cosine similarity technique. With term-matching surrogate-query relevance is calculated by the sum of a surrogate's individual vector components within the query space, rather than considering the angle between surrogate and query vectors. This approach is appropriate in this instance given that the surrogate and query are of quite different structure and content, meaning it doesn't necessarily do well to force both to exist within the same vector space. For comparing two surrogates, cosine similarity should be essential, but in the case of surrogate-query comparison, term-matching was found to perform well on the same style of comparison for CLEF 2009 and so this approach will be adopted here also,

Measures of query pre-processing are largely the same as those outlined for indexing described above. The only difference is that user queries do not come with 'title', 'caption' and 'labels' keys, rather they are of flat structure, containing only the terms the user has entered. In this way, there is no need to initially flatten the query's structure.

After query pre-processing, each query term's entry in idx (itself a dictionary of structure {filename: BM25 score}), if it exists, is added to a list idx_terms. Once this is complete, if idx_terms is empty the process is finished and no results are returned. If idx_terms contains just one dictionary (meaning only one query term was present as a key in idx), this dictionary is treated as a completed ranking and is returned. Where more than one query term is present as a key in idx, the relevant dictionaries

in idx are combined by summing values having the same key. In linear algebraic terms this is conceptually identical to summing the components of the surrogate vector within the query space. The most relevant document will be that with the largest Manhattan—not Euclidean—magnitude. The ranking process returns a list ordered by ranking score, which the web application will call upon in determining the order in which images should be displayed.

In making the image search functionality available through a web application, Python's Flask package is deployed. At the front-end a user's input is taken via a form field. Once the 'search' button is clicked, the user's query is put through the steps outlined above, returning a list of filenames ranked by relevance. Once the backend receives the images ordered by relevance, Flask's templating engine Jinja is used in a HTML file in order to print filenames and display images in order of relevance. Boostrap is used to style the front-end. The resulting display is somewhat reminiscent of the original 2001 Google Image Search; the most relevant result is displayed in top-left position, with subsequent images displaying from left-to-right and new rows populated as space requires. Images maintain their original shapes and occupy the interface in the nearest available space under the left-to-right, top-to-bottom arrangement mentioned. Previous work by ibrahimokdadov was instrumental in implementing the project's web application component [3].

## 5 Evaluation

For lack of a benchmarked set of pre-labelled queries against which to evaluate search and ranking performance, a somewhat ad-hoc initial system evaluation is given presently. Although absent benchmarking figures makes it difficult to fully contextualise these results, it is hoped we may at least gain some indication as to how the solution performs.

Given the aim of the current solution was to replicate Google Image Search as it first appeared in 2001, let us check the performance of some of the top Google web searches (image search statistics from this time are unavailable) from that time: 'David Beckham', 'Ferrari', 'Eminem', 'The Simpsons', and 'Paris'[4]. Of these only 'Ferrari' and 'Paris' return any results. 'Ferrari' returns a single image of a Ferrari car, and so we will consider its P@5 to be 1. On the other hand, it is difficult to establish whether the results for 'Paris' can be considered 'relevant' or not. It was the author's hope to see a selection of landmarks and/or cityscapes of metropolitan Paris. The top five results relate to various entities associated with or related to Paris, but images of the Eiffel Tower and Notre Dame cathedral appear further down the rankings. Let us tentatively determine this as a P@5 of 0. With this, let us conclude that performance on top Google web searches of 2002 is P@5 of 0.2.

Since the solution is based on Wikipedia PotD between 2010 and 2014, let us consider also how the solution performs on Google's top image searches from a year within this timeframe. The top image searches of 2012 as reported by Google were 'One Direction', 'Selena Gomez', 'iPhone 5', 'Megan Fox', and 'Rihanna'[5]. Of these, only 'One Direction' returns results, but no results have anything to do with the supposed Beatles of our era so we will consider P@5 on this query set to be 0.

Let us consider the strengths and weaknesses of a further ad-hoc selection of queries in order to outline ranking performance more fully. To begin with queries that perform well, the query 'Person' achieves P@5 of 0.8. The returned results are by and large images of individual persons. The driving cause behind this query's success lies in computer vision. When this same query is run ignoring computer vision-generated labels, within the top five results are images of computer keyboards. The fact that Amazon Rekognition returns 'person' when it identifies a human figure in an image is giving a significant performance boost to raw HTML surrogates in this case. When an instance of PotD features 'a person', it is so immediately apparent to a human observer that it is rarely even mentioned within PotD title or caption. Computer vision appears instrumental in returning relevant results where the caption neglects to mention the image's basic constituents. A similar effect is observed with the query 'bread' where the computer vision annotation boosts P@5 from 0 to 0.2—computer vision is successful in unearthing an image which clearly contains bread where the raw HTML surrogate fails to identify it as such. Consulting the title and caption for this image, it is indeed true that bread is not mentioned anywhere in the title or caption. Rather, these textual fields are focused on 'grain' and products of grain.

A query that exhibits unexpectedly strong results with P@5 of 1 is 'helicopter'. However, when one attempts to interpret the meaning of this for ranking performance, it only serves to reinforce the difficulties with assessing non-benchmarked performance as outlined at the beginning of this section. It is difficult to say whether this strong performance on 'helicopter' is simply due to there being many images of helicopters in PotD, or whether there is some particular aspect of the current solution which allows it to rank 'helicopter' with particular quality.

Let us also examine cases where the overall solution is identifiably performing sub-optimally. A transparent way of carrying out this exercise should be to begin with an entry in the PotD archive, to decide a query term by which the image ought to be identifiable, and to trial that query term within the web application to see where that image has rankied among the results. As a first example let us take the PotD from July 9 2012. The image clearly depicts a church. Although the query 'church' achieves P@5 of 1, the particular image in question here does not appear anywhere among the results. In diagnosing why this was so, it becomes clear that the caption gives details of the 'mission' to which the church relates, but never mentions that the image depicts a church, nor did computer vision recognise a church with P>0.75 confidence. Taking another example in this vein, PotD from July 12 2012 is a photograph of a Chicago cityscape. Similarly to 'church', 'city' achieves P@5 of 1, but does not contain anywhere in its output this image of Chicago. The surrogate does not mention that the image depicts a city skyline. Taking another example where not only the surrogate fails to mention a salient feature of the image, but the query fails to return

any results at all is ‘earring’. PotD from August 5 2012 depicts a woman wearing a prominent aural decoration. However, this is not captured by the image surrogate and the query fails to return any results.

## 6 Discussion

Within the results reported, let us attempt to piece apart its essential elements and identify salient trends. Taking the examples of 2002’s searches and 2012’s top image searches, poor performance on these queries is certainly explained by a lack of relevant images in PotD. Irrespective of indexing, searching and ranking quality, these queries will always fail to return results on this dataset. The strong performance of ‘helicopter’ from our ad-hoc queries is most likely due to the same underlying factor: PotD happens to contain many clearly annotated images of helicopters. Let us state then as a general principle, that a greater collection size should improve solution performance given there will be greater likelihood that an image relating to a given query will be present within the collection, independent of annotation richness or indexing, search and ranking quality.

The evaluation of poorly performing queries is perhaps more informative when it comes to identifying potential improvements to the solution. The cases of ‘church’ and ‘city’ being missed due to ‘mission’ and ‘Chicago’ being the necessary search terms immediately suggest a possible improvement. If a pre-trained language model could be applied so that the user need not give exactly the required keyword, but only a similar enough keyword that a word embedding can establish a high degree of similarity, it should have a positive impact on performance. Perhaps a different means to achieve the same ends may be to allow more computer vision annotations to be included in the surrogates. Or more specifically, to lower Recoknition’s required confidence under its current threshold of 0.75. In any case, it seems likely there are many available means to enhance the richness of annotations and improve ranking on these instances of sub-optimal performance.

### 6.1 Potential Improvements

In the context of solution as it has been outlined les us highlight areas that could be improved in each area of the implementation.

*6.1.1 Annotation*. Only images which were eligible for AWS Rekognition benefitted from computer vision annotations—namely, PNGs and JPEGs under 5MB in size. Efforts could be made to facilitate the remainder of files through combinations of compression and filetype conversions. As outlined above, computer vision annotations have measurably improved results for certain queries. Facilitating computer vision on more images may have a significant impact on overall ranking precision.

*6.1.2 Pre-processing*. As the project stands, the approach to text pre-processing is quite harsh. Words containing numbers and/or punctuation are completely ignored. In fact, there is more work required in properly adapting textual pre-processing from that suited to CLEF 2009, to that suiting the current project. Pre-processing is executed in such a way that the final words of sentences are in some cases ignored, simply due to the presence of a punctuation mark. This is caused by the fact that CLEF 2009 is pre-processed so that punctuation marks are mostly already separated from alphanumeric values by whitespace. At the same time, it is not just that the pre-processing used can be simply described as ‘too harsh—there remain gaps in pre-processing where tokens such as ‘--' are capable slipping through the gaps into the processing stage. It appears there is lots of potential to improve IR effectiveness by tweaking the approach to the pre-processing phase in general.

*6.1.3 Indexing & Ranking*. The project in its current form has a notable bug in need of rectification. Since image filenames in their original form have been used as dictionary keys in some data structures mentioned above, not all image files have been successfully recorded throughout the project’s various steps. This is due to Python’s constraints on dictionary key strings. A shift to another means of uniquely identifying images throughout the project should offer a fix.

As an alternative to using the BM25–term-matching approach taken here, there are many available open-source highly optimised search textual search engines. It would be an interesting endeavour to replace the current indexing, search, and ranking solutions with another to see if the current solution can be improved in output quality as well as speed.

Another useful exercise in improving results may be to experiment with various weightings of text importance between title, caption, and labels. It may be reasonable to assume title words are more significant than those elsewhere in the annotation. A better ranking strategy may be to reflect this by assigning different weights to different parts of the surrogate.

Another related attempt may be to incorporate Rekognition’s label confidence into its terms’ contributions. Labels with less than 75% confidence are already being ignored. However, perhaps taking account of the reported level of confidence may facilitate even more precise ranking.

As mentioned above in the outline of BM25, hyperparameters k and b were set at 1.2 and 0.75 respectively for this project. It may be interesting to experiment with these values in a grid-search style, bringing the resultant index through to the search, ranking and evaluation stages to see the effect these indexing hyperparameters may have on overall precision. These values were found to function well on CLEF 2009, but that is not to say better alternatives are not available on this collection.

*6.1.4 Front-end*. Usability may be improved by keeping a clear on-screen record of the query relating to the current results. At present, this is only visible within the page URL.

A corner-case in results retrieval which has not been dealt with properly on the front-end is when a given query does not contain any satisfactory matches in the collection. At present, nothing is displayed on screen. However, a better approach would be that employed by Google among other image search engines:

to display a message explaining that there were no worthwhile matches associated with the query.

The user interface does not offer any avenue for the user to explore where the image has come from or to learn any further information about it (save for the list of filenames atop the display). A significant improvement in usability would be to allow navigation to the relevant PotD archives page. It may also be interesting to display each image's relevance score, giving a more holistic view of rankings.

## 7 Future Works

Now that something of a working proof-of-concept has been implemented, it may be an interesting endeavour to bring the project to a greater scale. For example, implementing the solution on the entire archive of PotD rather than a subset. A key component in such a work would be to avoid downloading images wholesale and integrate the use of image URLs throughout.

At present, Google Image Search offers filtering functionality to narrow results by size, colour, type, time, and usage rights. An interesting extension of this work may be to aim for something like this feature. Allowing keywords such as AND, NOT and OR, and the use of parenthesis as special keywords would permit this useful feature. Minimal additional computation would be required to facilitate query parsing to these ends, while the benefit to the user would be a much more tailored search experience.

## 8 Conclusions

This project has implemented a replication of Google Images as it originally appeared in 2001. Wikipedia's Picture of the Day (PotD) from the period 2010–2014 was crawled, annotated, and indexed. A BM25 ranking system was implemented. A Flask web application was built to accept user queries and display ranked results.

As described above, the solution exhibits mixed performance on a selection of ad-hoc queries. A more standardised means of performance assessment would permit a more comprehensive evaluation and discussion. Notwithstanding the numerous ways the solution may be improved, the project on a whole succeeds in its initial aim of building a miniature Google Images as it was originally in 2001.

## REFERENCES

[1] The tinkerer's apprentice: 2015. *https://www.project-syndicate.org/onpoint/google-european-commission-and-disruptive-technological-change-by-eric-schmidt-2015-01*. Accessed: 2022-04-06.

**[**2**]** Wikipedia:Picture of the day/archive: *https://en.wikipedia.org/w/index.php?title=Wikipedia:Picture_of_the_day/Archive&oldid=1046045330*.

[3] Mokdad, I. *Upload_file_python*. https://github.com/ibrahimokdadov/upload_file_python

[4] El año en búsquedas de Google: *https://trends.google.com/trends/yis/2002/GLOBAL/*. Accessed: 2022-04-06.

[5] El año en búsquedas de Google: *https://trends.google.com/trends/yis/2012/GLOBAL/*. Accessed: 2022-04-06.