Image search engine enhanced by HTML and computer vision annotations

Robert Joscelyne
Completed on 21st April 2023
robjoscelyne@yahoo.co.uk

ABSTRACT

Image search engines play a crucial role in efficiently locating and retrieving relevant visuals from vast image collections. The primary aim was to develop a prototype image search engine based on BM25 and to implement a user interface accessible through a URL. With the help of Octoparse, a web extraction software, images and their annotations were gathered from Pixabay. A Python script was utilized to process the collected data, resulting in a new JSON file containing textual surrogates. You Only Learn One Representation, a state-of-the-art object detection algorithm, was employed to enhance HTML annotations with additional information derived from object detection. The textual surrogates were indexed using the BM25 search engine, and a web-based interface was designed with Streamlit for user access. Upon evaluating the image search engine, high precision rates were observed for specific categories. Nevertheless, attempts to adjust the BM25 model's hyperparameters failed to yield improvements in precision or align with the theory, suggesting possible limitations originating from factors such as image collection composition or model architecture. Misclassifications in YOLOR annotations were detected, indicating a need for further training across a broader range of object classes.

I. INTRODUCTION

Image search has become essential to modern information retrieval systems, enabling users to locate relevant visual content based on keywords or other search criteria. Advancements [1] have been made in computer vision, with technologies such as YOLO (You Only Look Once) gaining prominence for real-time object detection and classification. These developments have paved the way for more sophisticated and accurate image search engines.

Search engines use a web crawler to access various web pages at specific frequencies [2]. Web crawling plays a crucial role in image search, as it involves systematically navigating the internet to gather image data and associated metadata from web pages. This process is vital for populating image search databases, which can then be used to deliver relevant results to users. Octoparse is one popular web scraping tool that can be employed for this purpose, along with other applications such as Scrapy and Beautiful Soup. For this project, a collection of images was crawled from Pixabay. The search engine developed for this project can be accessed through the following link. https://alpaca-search.herokuapp.com/

The primary objectives are:

- Collect a minimum of 1,000 images from Pixabay utilising a web crawler.
- Generate a textual surrogate for each image by extracting information from the HTML content, and enhance the annotations with data from computer vision-based tools.
- Index the textual surrogates of the images using the BM25 search engine developed in assignment 1.
- Create a web-based interface for the image search engine, allowing users to access the system through a URL.

II. ANNOTATION

Image annotation plays a crucial role in various computer vision tasks, such as object recognition, image retrieval and systems for autonomous vehicles. Different approaches to annotation, including HTML tags and computer vision methodologies, have been employed to enhance the quality and relevance of such data. Octoparse is a web extraction software allowing bulk data extraction from websites [3].

Crawler description

Figures 1 and 2 exhibit screenshots of the Octoparse application utilised for extracting image URLs and HTML data from the Pixabay image website. https://pixabay.com/photos/search/?manual_search=1&order=trending&pagi=1 The application was configured to crawl 4316 images and collect image annotations from the site. Figure 2 demonstrates the image URL and annotation pair generated by Octoparse. Subsequently, a custom Python script was created to process the JSON file produced by Octoparse.

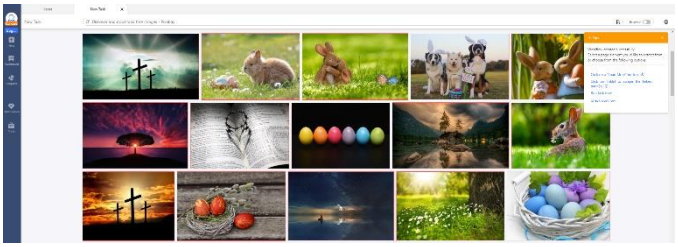


Figure 1. Octoparse tool used for crawling websites.

No.	Image	Info2
1	https://cdn.pixabay.com/photo/2015/04/23/22/00/tree-736885_340.jpg	sunset
2	https://cdn.pixabay.com/photo/2017/04/02/22/36/easter-2197043_340.jpg	easter bunny
3	https://cdn.pixabay.com/photo/2018/01/14/23/12/nature-3082832_340.jpg	waters
4	https://cdn.pixabay.com/photo/2017/09/04/09/38/crosses-2713356_340.jpg	clouds

Figure 2. HTML annotations from Pixabay.

The JSON file acquired from Octoparse was processed using a Python script, which facilitated the downloading of images and the creation of a new JSON file named 'textural.json'. This new file encompasses the image and corresponding annotation under the heading HTML description. The script downloads and saves images to a directory titled 'scraped_images', ensuring they meet a minimum size requirement to filter out potentially irrelevant images such as website navigation icons.

Computer vision

The YOLOR (You Only Learn One Representation) algorithm was utilised to enhance HTML annotations of Pixabay images through object detection. Distinct from YOLOv1-YOLOv5 in terms of authorship, architecture, and model infrastructure, YOLOR provided a unique approach to object detection. The 2021 release paper [4] demonstrated that YOLOR outperformed other state-of-the-art object detection methods, thus justifying its selection for this assignment.

Modifications were made to previously developed code [5] to process all 4316 images, employing the YOLOR algorithm for object detection. The process was executed on a CPU (i7-11700K), with the inference taking 2 hours. Detected classes from the images were added to the textural.json file created previously. A confidence level threshold of 60% was established, indicating that a class would only be detected when the confidence level exceeded 60%. It is worth noting that YOLOR used weights trained on 80 classes from the Microsoft Common Objects in Context (MS COCO) dataset. The inference images were preserved, showcasing the bounding boxes for each class detected, along with the algorithm confidence. Examples from a sample image are presented in figures 3 and 4.


```
"ImageName": "image_4305.jpg",
"HTML_description": "dark",
"yolor_description": "person, handbag, book"
```


Figure 3. Extract from textural.json showing combined HTML and YOLOR annotations for image_4305.

Figure 4. Object detection results using YOLOR for image_4305.

III. INDEXING

An inverted index was developed to improve the efficiency of image querying. Initially, the text data from HTML and YOLOR descriptions underwent preprocessing, which involved the removal of HTML tags, non-alphanumeric characters, and converting the text to lowercase. The text was then tokenised into individual words or subwords, and stemmed to their root form using the PorterStemmer algorithm. Stop words were eliminated from the list of stemmed tokens, resulting in a refined version of the text. As per the findings from assignment 1, a BM25 model was employed.

Research [6] has demonstrated that the Term Frequency-Inverse Document Frequency (TF-IDF) serves as a numerical statistic, signifying the importance of a term to a document within a corpus or collection. Utilising this method, the collection was indexed by iterating through the refined annotations and updating the inverted index for each term encountered. The document frequencies for each term were computed, and the inverse document frequencies (TF- IDFs) were determined. This information, along with the term frequencies and document lengths, were used to compute the BM25 scores for a given query-image pair, thereby enabling efficient and relevant search results.

IV. RETRIEVAL AND USER INTERFACE

A function initially created for assignment 1 has been reused to compute the BM25 scores for the image queries. Figure 5 shows get_bm25_score, which calculates the relevance score for a query and a document that consists of combined HTML-YOLOR text annotations. The BM25 relevance score is a widely used information retrieval metric that measures the importance of a term within a document and a query, considering term frequency and inverse document frequency. It helps to rank images based on their relevance to the query.

```
# Calculate BM25 Relevance Score for Query and Combined HTML-YOLOR Annotations
def get_bm25_score(query, doc, idfs):
    terms = clean_text(doc)
    term_freqs = Counter(terms)
    doc_length = len(terms)
    query_terms = [stemmer.stem(term) for term in query]
    query_freqs = Counter(query_terms)
    score = 0

    for term in query_terms:
        if term not in idfs:
            continue
        term_freq = term_freqs[term] if term in term_freqs else 0
        query_freq = query_freqs[term] if term in query_freqs else 0
        score += idfs[term] * ((term_freq * (K1 + 1)) / (term_freq + K1 * (1 - B + B *
        (doc_length / N)))) *((K3 + 1) * query_freq / (K3 + query_freq))
    return score
```

Figure 5. Function to compute BM25 score for queries.

The function takes three arguments. 1. Query - a list of words representing the search query. 2. Doc - the combined HTML-YOLOR annotations to calculate the score for. 3. Idfs - A dictionary containing the inverse document frequencies of the terms.

The function initially cleanses and tokenises the document text, and then calculates term frequencies. Subsequently, the function processes the query terms through stemming and determines the frequency of each term. The BM25 equation is derived from the work presented in the Encyclopedia of Database Systems [7]. The calculation of BM25 is as follows:

$$\text{score} = \Sigma \left(IDF_t * \left(\left(tf_t * (k1 + 1)\right) / \left(tf_t + k1 * \left(1 - b + b * \left(L / N\right)\right)\right)\right) * \left((k3 + 1) * qf_t / (k3 + qf_t)\right)\right)$$

Where:

- score: The BM25 relevance score.
- IDF_t: The inverse document frequency of term t.
- tf_t: The term frequency of term t in the document.
- k1, b, and k3: Constants used in the BM25 formula, values set to k1=2.6, b=0.85, and k3=0.0.
- L: The length of the document in terms.
- N: The average length of documents in the collection.
- qf_t: The frequency of term t in the query.

The function returns the calculated BM25 score for the given query and document. An example can demonstrate the calculation. The function computes the BM25 scores for the query "bear" and three images (image_3052.jpg, image_1967.jpg, and image_1988.jpg) with their respective combined HTML-YOLOR annotations are returned as shown in figure 6. The output shows the BM25 scores for each image, which can be used to rank them based on relevance to the query.

```
Image Name: image_3052.jpg, BM25 Score: 11.401539848837418
Image Name: image_1967.jpg, BM25 Score: 10.665956632783391
Image Name: image_1988.jpg, BM25 Score: 10.665956632783391
```

Figure 6. BM25 scores for the query "bear".

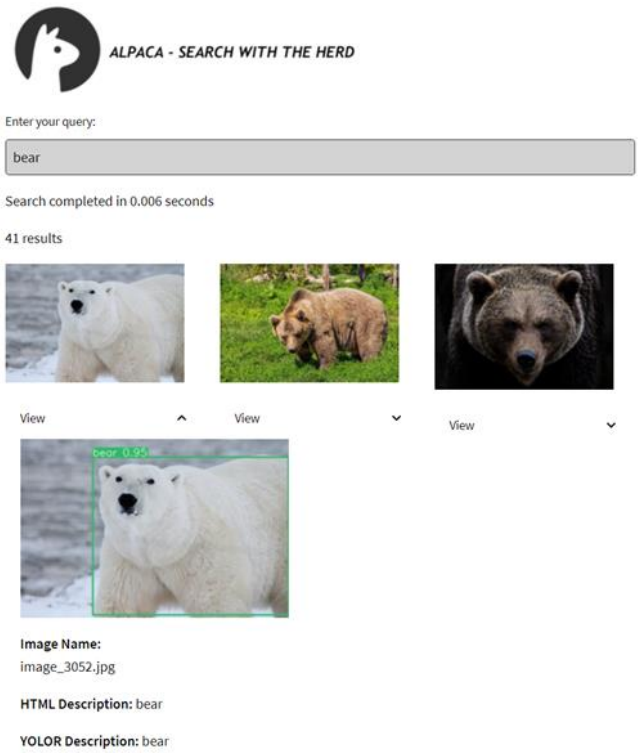


Figure 7. Steamlit interface used for image search engine.

Previous work has shown [8] that Streamlit has been an effective tool for developing user interfaces for search engines. Using Streamlit, a web application called Alpaca, which retains its name from assignment 1, was created. This interface allows users to search for images based on their queries. When a query is submitted, the application calculates the search duration, the number of results found, and displays the top 100 results using the BM25 ranking algorithm in a grid layout. Each grid cell features an image thumbnail, and upon expansion, reveals a larger image along with its name, HTML description, and YOLOR description. Figure 7 illustrates the Streamlit interface developed for this assignment. The Alpaca image search engine was deployed using Heroku and can be accessed via this URL https://alpaca-search.herokuapp.com/ An overview of the Alpaca image search engine is shown in figure 8.

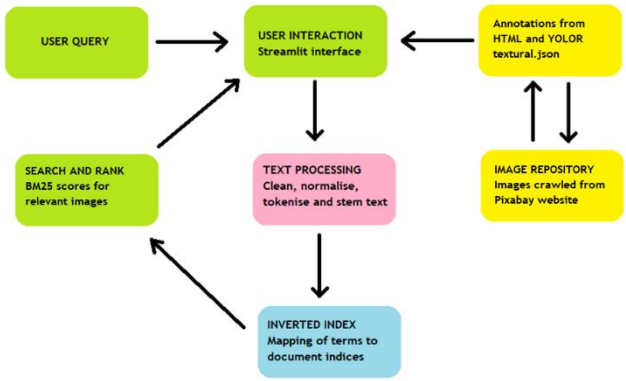


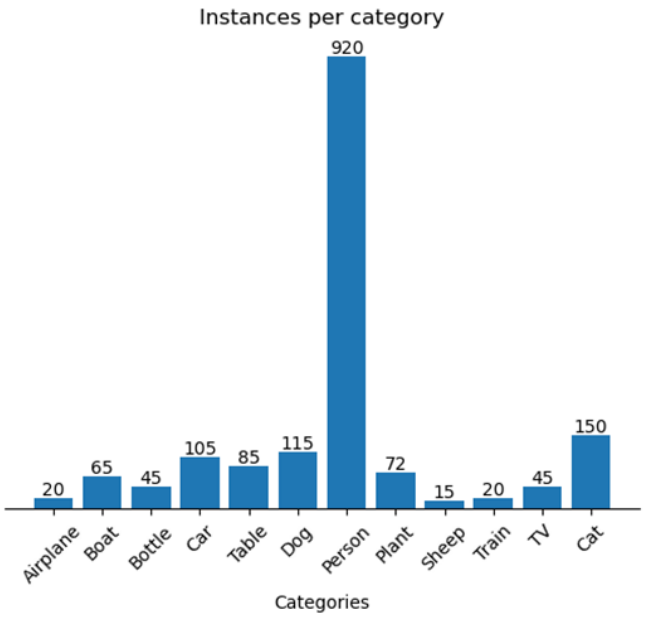
Figure 8. Architecture of the Alpaca image search engine.

V. EVALUATION

Results

A subset of the groups identified by Lin et al. was used to evaluate the image search engine [9]. The categories comprised of Airplane, Boat, Bottle, Car, Table, Dog, Person, Plant, Sheep, Train, TV, and Cat. Counts for each category were ascertained by examining the image repository and tallying the images within each group. The findings are depicted in Graph 1.

Graph 1 displays the distribution of images across the various groups. The most noticeable trend is the high count of images under the 'Person' category, followed by a substantially lower count for 'Cat' and 'Dog'. The remaining categories exhibit relatively lower counts, with 'Airplane' 'Train' and ‘Sheep’ having the lowest counts.



Graph 1: Comparing models using the precision-recall curves.

In an information retrieval system that retrieves a ranked list, the top-n documents refer to the first n entries in the ranking. Precision at n represents the proportion of relevant top-n documents in the list [10]. The calculation for precision is shown in Equation 1.

$$Precision\ @\ n = \frac{Relevant\ images\ in\ top\ n}{n} \qquad \text{Eq. 1}$$

The precision of the search engine was calculated at Precision at 5 and Precision at 10 as shown in Table 1. The Sheep category demonstrates the lowest precision rates at both levels, with 20% at 5 and 30% at 10. Moderate precision rates are observed for Airplane (60% at 5) and TV (80% at 10) categories. Some groups such as Bird, Car, Dog, Person, and Plant maintain high precision rates at both levels, with Bird and Car showing 100% precision at both 5 and 10.

Category	Precision @ 5	Precision @ 10
Airplane	60%	70%
Bird	100%	100%
Bottle	100%	60%
Bottle	100%	90%
Car	100%	100%
Cat	90%	90%
Table	100%	80%
Dog	100%	100%
Person	100%	100%
Plant	100%	100%

Sheep	20%	30%
Train	100%	80%
TV	60%	80%

Table 1: Precision results of image search queries.

Adjusting parameters like b (field-length normalization), k1 (term frequency saturation), and k3 (term frequency in the query) can enhance the performance of the BM25 model, as demonstrated by research [10]. However, despite adjusting hyperparameters k1, b, and k3, no improvement in precision was observed.

Misclassifications were identified in the YOLOR annotations. An example of this is shown in figure 9 where a deer is misclassified as a sheep. Another example includes a lion classified as a dog.

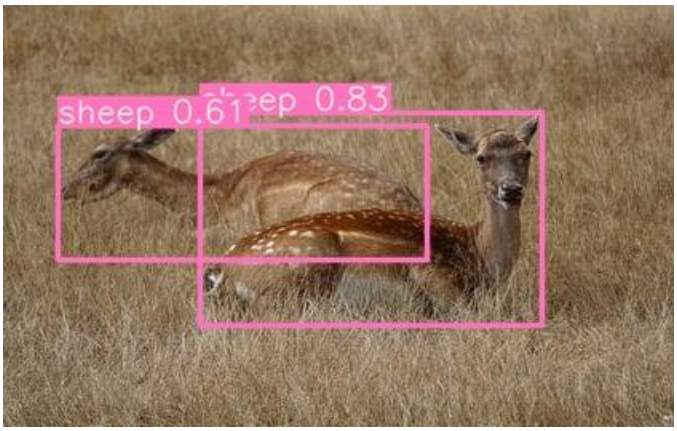


Figure 9. YOLOR misclassifications.

VI. Discussions

A minimum of 1,000 images was collected from Pixabay using a web crawler. Textual surrogates for each image were generated by extracting information from the HTML content and enhancing the annotations with data from YOLOR. The textual surrogates of the images were indexed using the BM25 search engine developed in assignment 1. Finally, a web-based interface for the image search engine was created, allowing users to access the system through a URL.

In Table 1, the observed disparities in precision at 5 and precision at 10 across categories can be ascribed to various factors, including the distribution of image groups in the dataset. As depicted in Graph 1, the dataset is notably imbalanced, with specific categories containing more images than others. This imbalance could be the reason for the improved performance of well-represented categories. For example, the dataset contains 920 images of 'Person' and only 15 images of 'Sheep'. This substantial disparity in the number of images may account for the higher precision observed in the 'Person' category. Additionally, the quality and diversity of the image collection for each category could impact the model's performance. Some categories may possess more diverse and high-quality images, potentially resulting in improved precision outcomes at both levels. Conversely, categories with restricted or less diverse data samples might exhibit lower precision at both levels or inconsistent precision values.

YOLOR may show better performance in adding object detection annotations for some categories compared to others due to inherent challenges in detecting specific objects, overlapping categories or occlusions. These variations in annotation quality could contribute to differences in precision observed across categories. YOLOR was used with the default weights based on the MS COCO dataset containing 80 classes. Consequently, some objects the algorithm encountered were absent in the weights, leading to misclassifications. An example is shown in Figure X, where a deer not present in the MS COCO dataset was misclassified as a sheep.

Adjusting parameters such as b (field-length normalization), k1 (term frequency saturation), and k3 (term frequency in the query) have been shown to enhance the performance of the BM25 model in some research studies [10]. However, despite tweaking these hyperparameters, no significant improvement in precision was observed in this evaluation. This suggests that the model's limitations may stem from other factors, such as the dataset's composition or the model's architecture.

Future work could involve training YOLOR on a more extensive set of classes. By expanding the training data to include a broader range of object classes, the algorithm would have a more comprehensive understanding of various objects, allowing the algorithm to make more accurate predictions. YOLOR is capable of performing annotations on video. As a result, the image search engine could be extended to include video annotation. This would expand the system's capabilities and open up new possibilities for video-based applications, such as video surveillance, automatic tagging of video content, and improved video search functionalities.

VII. CONCLUSIONS

- Successfully collected images from Pixabay using a crawling tool and generated textual surrogates using HTML content and YOLOR annotations.
- Indexed textual surrogates with the BM25 search engine and created a user-friendly web-based interface.
- Observed high precision rates for specific categories, but the imbalanced distribution of image categories influenced the results.
- Identified misclassifications in YOLOR annotations, suggesting a need for further training on a broader range of classes.
- Adjusting hyperparameters did not improve precision, pointing to other factors such as image set composition or model architecture.

References

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, pp. 779-788. DOI: 10.1109/CVPR.2016.91.

[2] Konstantin Avrachenkov, Vivek Borkar, and Ketan Patil. 2021. Deep Reinforcement Learning for Web Crawling. In 2021 Seventh Indian Control Conference (ICC), Mumbai, India, pp. 201-206. DOI: 10.1109/ICC54714.2021.9703160.

[3] Octoparse. (2023). What is Octoparse? Retrieved April 9, 2023, from https://www.octoparse.com/blog/what-is-octoparse

[4] Chih-Yi Wang, I-Hong Yeh, and Homer H.-Y. M. Liao. 2021. You Only Learn One Representation: Unified Network for Multiple Tasks. Institute of Information Science, Academia Sinica, Taiwan; Elan Microelectronics Corporation, Taiwan. https://arxiv.org/abs/2105.04206 [Accessed 11/04/23]

[5] Wong, K. Y. (2022.). YOLOR: You Only Learn One Representation. GitHub. Retrieved from https://github.com/WongKinYiu/yolor [Accessed 11/04/23]

[6] Ankita Gothankar, Lavina Gupta, Nidhi Bisht, Shubham Nehe, and Mayuri Bansode. 2022. Extractive Text and Video Summarization using TF-IDF Algorithm. International Journal for Research in Applied Science & Engineering Technology (IJRASET) 10(3). Retrieved from https://www.ijraset.com/.

[7] Melucci, M. (2009). Vector-Space Model. In: LIU, L., ÖZSU, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_918 [Accessed 11/04/23]

[8] Giuseppe Narciso. 2020. Build a Search Engine for Medium Stories Using Streamlit and Elasticsearch. Better Programming. [online] Available at: https://betterprogramming.pub/build-a-search-engine-for-medium-stories-using-streamlit-and-elasticsearch-b6e717819448. [Accessed 09/04/23].

[9] Tsung-Yi Lin, et al. 2014. Microsoft COCO: Common Objects in Context. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. Springer International Publishing. DOI: 10.1007/978-3-319-10602-1_48.

[10] Nick Craswell. 2009. Precision at n. In: Liu, L., Özsu, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. DOI: 10.1007/978-0-387-39940-9_484.