XXXVIII Jornadas de Automática
https://doi.org/10.17979/spudc.9788497497749.0991

Oculus-Crawl, a Software Tool for Building Datasets for Computer Vision Tasks

Iván de Paz Centeno, Eduardo Fidalgo Fernández, Enrique Alegre Gutiérrez, Wesam Al Nabki
Dpto. Ingeniería Eléctrica y de Sistemas y Automática,
Universidad de León, Campus de Vegazana s/n, 24071 León, Spain,
ipazc@unileon.es, efidf@unileon.es, ealeg@unileon.es, mnab@unileon.es

Abstract

Building datasets for Computer Vision tasks require a source of a large number of images, like the ones provided by the Internet search engines, joined with automated scraping tools, to construct them in a reasonable time. In this paper it is presented Oculus-Crawl, a tool designed to crawl and scrape images from the search engines Google and Yahoo Images to build datasets of pictures, that is modular, scalable and portable. It is also discussed a benchmark for this crawler and an internal feature for storing and sharing big datasets, that makes it suitable for Computer Vision and Machine Learning tasks. In our tests we were able to crawl and fetch 11.555 images in less than 14 minutes, including also their meta-data description, showing that it might be well-suited for retrieving large datasets.

Key words: crawler, search engine, dataset, images, computer vision.

1 INTRODUCTION

Nowadays there exist a huge number of search engines that allow us to search content on the web including almost any type of resource, ranging from documents and pictures to sounds and videos. The nature of the web is to link multiple resources as hyper-links among them and, following the analogy, the process of reaching an end resource is done by crawling the interconnected nodes. Historically, the search engines have been fed by multiple web crawlers [4, 6, 9] that automatically track and follow the hyper-links from the content of the web, creating a database of entries that are usually formatted into a human-readable view in order to be presented to humans and to be read by humans. This adds an overhead in the automatic retrieval of content from search engines, as most of the times their results require to be analyzed and parsed from a markup language; in addition, the way to navigate through their content is usually handled dynamically by JavaScript code in form of AJAX calls [5], which requires of a sort of human intervention like scrolling down the content or clicking on certain regions of the view, adding extra layers of complexity to the task of crawling those web sites. Even though most of them are not program-friendly in terms of extracting information, there have existed many successful attempts in retrieving useful information by automatically parsing the results from those search engines, like the framework Scrapy [15], the project icrawler [3] for python, or Apache Nutch [2], which takes advantage of big data tools such as Apache Hadoop [1]. Despite Computer Vision is one of the computer fields that most demand of large numbers of images, commonly required to solve specific classification or detection problems, crawling and scraping tools might be well suited for it. Even though most of the times Computer Vision problems leverage into public datasets, sometimes it exists the need to improve them or create new ones. For those situations where a distributed crawling is required and there is a lack of a distributed infrastructure, we provide an alternative named Oculus-Crawl, a crawler in the form of command line tool for images from the search engines Google and Yahoo, that can follow conveniently a distributed nature [10], which is isolated from underlying Big Data frameworks, and that can be shipped in the form of Docker containers [11]. It does not require to write code and is projected to be used as a source for Computer Vision and Machine Learning datasets.

The paper is structured in 4 sections, being the section 1 dedicated to the introduction; the section 2 is destined to an overview of the architecture of the solution which gives small insights about its main features; the section 3 summarizes the experiments and results we had with the tool applied to different topology configurations and tool options; and the section 4 dedicated to the conclusions and possible future works for this tool.

2 ARCHITECTURE

The tool has three roles out-of-the-box: a factory, a crawler and a client. Each role is performed by its correspondent entry point in the application, and they communicate each other through the factory, which exports an API-REST interface on a

specific port. The generation of a dataset comprises 5 stages:

1. The request to the factory for the generation of a dataset, done by the client.
2. The crawling of images, done by the crawlers.
3. The fetching of crawled images, done by the factory.
4. The packaging of the images into a single zip file with their crawled meta-data, done by the factory.
5. The publication of the dataset into a public directory, done by the factory.

An overview of the stages can be seen in Figure 1.

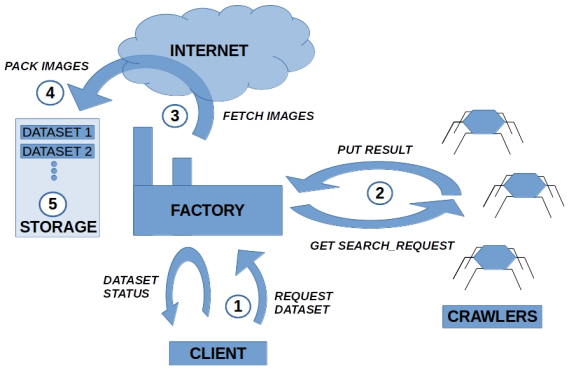

Figure 1: The 5 stages in the generation of a dataset of images.

The factory allows to create requests for generation of datasets within a single HTTP call. Each request for generation of a dataset is formed by a set of search requests for a specific search engine and a search words to be used, among other parameters; a visualization of this scheme can be found in Figure 2.

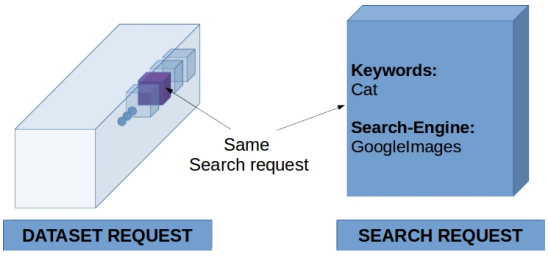

Figure 2: A dataset generation request is an aggregation of search requests. It is handled by the factory and the crawlers.

The crawlers, that might be running on any host, steadily poll the factory for requests of new datasets until one is retrieved, which initiates the crawling process. This process consists of retrieving search requests from the dataset request, crawling their results and returning them back to the factory. The final scenario is a system on where the available crawlers compete to retrieve search requests and process them until all are processed. Then, they jump to the next available dataset request or stay idle waiting for new ones. This crawler's behavior leads to a scalable distributed system, where increasing the number of crawlers reduces proportionally the overall time for crawling. Note that since the factory is an HTTP API-REST server, it can also be scaled up by load balancing it the same way a web server is usually scaled up.

When a dataset is completely crawled, the factory starts fetching all the crawled items in order to generate the final elements of the dataset, each consisting of the content of the image and its associated meta-data in JSON format. Note that the crawlers only gather the meta-data referenced by the search engine, including the URL to the images; and the factory, once the crawler process is finished, fetches their content.

2.1 THE SEARCH SESSION

The search session is the key feature in Oculus-Crawl as it preserves the whole dataset state in form of a serializable JSON structure that can be saved directly to a file. Therefore, each dataset request will have a search session attached that can be managed remotely, through the factory's API-REST. It can be used to backup a process of dataset creation at any time and to restore this process remotely, from the client side.

The proposed key feature in Oculus-Crawl suits perfectly in the creation of Computer Vision datasets, as once this session is filled up, it can be used as a pre-fetching step on the creation of a dataset, avoiding the need to crawl again. Moreover, the serialization of the session state eases the sharing of the dataset over the network, hence, reduces the bandwidth, since its size is several times smaller than the complete fetched dataset, as shown in Figure 3.

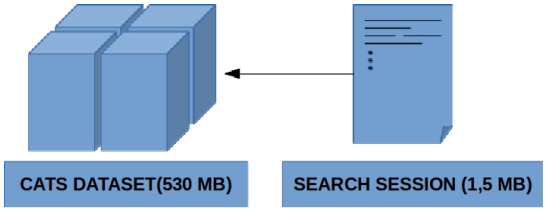

Figure 3: The search session representing an entire dataset. It contains information enough to rebuild the dataset without the need to crawl again.

2.2 THE SEARCH ENGINES

It is a common practice for search engines to associate meta-data to each of the elements they present to the final user. This meta-data might be useful for tagging the resources displayed by a search engine, as it is common to find resources with a descriptive text attached. Oculus-Crawl takes advantage of this behavior, storing the descriptive text with the width, height and file extension in the meta-data file of the final raw dataset. Currently, Google Images and Yahoo Images are officially supported by Oculus-Crawl. During the development phase of the tool, Bing Images was also supported, but a change in their presentation scheme left the search engine currently out of support. It is stated that search engines set limits on the number of elements they display for a single search request, as it is demonstrated that the behavior of most of the users is to look and use only the first entries of the results [16]. This leads the search engines to leave the less accurate elements they display on the least results they show, or to limit the number of results they provide e.g. Google Images is limited to 400 elements. When a search engine is not limited or its limit is too high, like Yahoo Images, the tool establishes a soft limit on approximately 500 results. This soft limit avoids to process an excessive number of pictures from a single Document Object Model (DOM) and also discards the least accurate resources. In Table 1 are shown the limits for each search engine.

Table 1: Search engines limits. Hard limit is imposed by the search engine. Soft limit is imposed by Oculus-Crawl.

Search Engine	Results Limit	Type
Google Images	400	Hard
Yahoo Images	500	Soft

In order to circumvent these limits and at the same time retrieve the most accurate results for a given topic, Oculus-Crawl follows a divide-and-conquer strategy, splitting each search request in multiple search requests, having each slight changes that consist of appending an adjective to the main search words. This task is accomplished by the client role of Oculus-Crawl, which accepts a set of adjectives in addition to the main search words, and automatically combines them, therefore, generating multiple and different search requests that forces the search engines to change the nature and order of the elements displayed for each. Hence, a list of adjectives compatible with the main search words should be manually provided during the invocation of the client.

2.3 USAGE OF ADJECTIVES

The Oculus-Crawl client accepts as input a set of adjectives in order to combine them with the original search words with the goal of increasing the number of pictures retrieved. Each combination leverages into a different set of results but sharing all of them the same inner semantic. A restriction, however, is that the adjectives chosen should be applicable to the search words context, e.g. a chair can be blue, beautiful or small; but can not be angry or thirsty. Even though the search engines always retrieve results regardless of the search keywords used, the results of using incompatible adjectives lead to an unknown or incorrect semantic, where most of the results are probably going to be out of the context of the original search words. This is explained by the fact that search engines associate key words to images, being the origin of these key words in the description that usually users attach next to the images in the HTML documents.

2.3.1 Number of adjectives to use.

The number of adjectives used for crawling affects the number of images retrieved. In order to know how many adjectives are needed to build a certain dataset of N images for a single topic, being L_i the limit for the search engine i, the Eq. (1) approximate it.

$$Adjv(N) = \frac{N}{\sum_{i=1}^{|L|} L_i} \quad (1)$$

Even though the number of images should be proportional to the number of adjectives used, the factory implements a deduplication mechanism of images, during the fetching stage, that may decrease the number of total pictures compared to the results retrieved with Eq. (1), the higher the number of adjectives used.

2.4 THE FETCHING STAGE

When the crawling process is finished, the factory fetches all the resources crawled. The fetching stage consists of a pool of 10 workers that downloads distributed the content of each of the crawled resources, which implies that up to 10 images can be downloaded simultaneously. Having increased this value might have forced the DNS servers to resolve too frequently the addresses of the hosts that contains the resources, which could potentially be blocked due to the Request Response Limit (RRL) of certain Domain Name Servers (DNS) [17], which can lead to a temporal

ban from the DNS resolver, hence, stopping the factory from successfully generating the dataset. Nonetheless, this parameter will eventually become configurable. Lastly, when a resource is requested to a host, the Oculus-Crawl factory sets a timeout of 15 seconds for the host to answer this request before it is marked as invalid and discarded from the dataset.

2.4.1 Deduplication of resources.

It is common for multiple search engines to refer to the same resources in certain number of results. This can be split into two different situations: 1) the same resource is hosted by two different hosts; 2) multiple search engines provide a reference to exactly the same host. In both cases, the end resource is the same, but the description used as meta-data might be different. A way to tackle this problem is to hash the resources in order to discard duplicates. In Oculus-Crawl, the hashing is done by using the MD5Hash [14] algorithm for each resource, which allows to retrieve the links and search engines that point to the same resources and storing them along with the meta-data element for each resource. For this reason, Oculus-Crawl might be also useful to catch hosts with duplication of resources. The reasons for choosing MD5Hash instead of a more secure hashing method is: 1) even though a collision of hashes is possible [18], in the case of a hash collision for different resources in high sized datasets, it is probably not going to cause a big trouble for the end dataset; 2) low sized datasets are not prone to present collisions and 3) because in scaled environments where a high sized dataset is required, the speed in hashing takes importance and MD5Hash is one of the fastest and reliable-enough hashing methods. However, it is common to have the same picture duplicated with different dimensions or formats each, a situation that MD5Hash or most of the hashing methods are not able to tackle. In this case, a more complex hash algorithm can be used like the Perceptual Hashing [12], which Oculus-Crawl will include in the future.

2.4.2 Inferring the extension of the pictures

When the crawling process is finished, the factory fetches all the resources crawled. In order to know the extension of the fetched picture, the name of the URL that points to it can not be trusted, as it does not necessarily point to a file-system file, e.g. a URL that apparently refers to an image because ends with ".jpg" might refer to an HTML document or a binary executable file instead. Hence, finding the correct extension requires of checking at the response headers of an HTTP HEAD call to the remote server that is hosting the picture, and to process the MIME-type header that specifies what kind of resource it is returning. Even though this MIME-type header can not be completely trusted, as not all the web servers return a correct MIME-type header for the resources they send, it is the fastest method for inferring the resource's format in a reliable-enough way. Note that MIME-Type is the most reliable method just after the checking at the resource's content itself, and it is also used by the web browsers to correctly parse the retrieved content for the web pages they render. For this reason, Oculus-crawl follows an extension inferring procedure that, by priority, consists of: 1) retrieve the extension from the MIME-type; 2) use the URL name to inaccurately infer the extension when the first method fails. If none of both methods are able to report a valid picture extension, the file is stored in the dataset without extension.

2.5 TECHNOLOGIES USED

Oculus-Crawl has been built entirely in Python3. The project can be directly executed in any x86_64 architecture by using Docker with the *latest* Docker image for Oculus-Crawl[1], which contains all the dependencies satisfied.

2.5.1 Factory process.

The factory process uses the Python's library *Flask* [7] to expose an API-REST which allows standardized interactions for crawlers and clients with the datasets' sessions. Moreover, this functionality can be easily tested and consumed externally (e.g. using HTTP calls with the UNIX tool cURL) or wrapped and interfaced in a web view. This means that the creation of a dataset can be invoked, tracked, backup-ed or dumped at any time without the need to have explicitly a client; however, Oculus-Crawl bundles a specific client for managing these tasks. The factory is a multi-tasked process which uses the Python's library *urllib2* to fetch the crawled resources whenever the crawling stage has finished, distributed among processes by using the Python's library *multiprocess*. It also uses the Python's library *hashlib* to perform MD5Hash on each fetched resource in order to avoid exact duplications of resources. The final zipped dataset is generated by using the library *shutil* from Python.

[1] https://hub.docker.com/r/dkmivan/oculus-crawl/tags/

2.5.2 Crawler process.

The crawler process takes advantage of the framework *Selenium* and its web-drivers [13] for the web-browser *Firefox*. This framework allows a direct interaction with the elements from the DOM and, at the same time, to perform common user's actions like clicking on buttons, performing scrolls or filling forms on the HTML view. Moreover, this scheme takes advantage of the JavaScript engine from the web-browser since the page gets rendered. This way of crawling through Selenium adds overhead on the processing of the HTML by increasing the load times due to rendering the web-page rather than only parsing the HTML, however this behavior reduces the probability for the crawler of getting detected as a bot. Even though it uses a graphical web-browser instead of direct HTTP calls, it can run in non-graphical environments by using the library *PyVirtualDisplay* to wrap the view in a virtual display. Furthermore, a crawler process can be split in several workers taking advantage of the Python's library *multiprocess*, behaving each as a single crawler instance and increasing the overall speed for crawling the resources within a single host.

2.5.3 Client process.

The client process wraps all the API-REST calls from the factory for the generation and tracking of datasets by using the Python's *requests* library. It is a simple client that generates a dataset request on the factory and steadily polls for its status until it is finished, showing a progress bar for each of the stages in the dataset generation.

3 EXPERIMENTS AND RESULTS

We tested the tool in two dedicated servers Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, one dedicated server Intel(R) Xeon(R) CPU D-1531 @ 2.20GHz and one virtual private server Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz, each of them connected to different networks of 1 Gbps of connectivity. We followed different topologies, running each Oculus-Crawl role in different machines and also combining all the roles together in a single machine to measure the performance impact. In order to help in the measurements of our tests, we defined a measurement variable that we called *adjective_rate*, which represents the ratio of adjectives per crawler. We realized that, for a small *adjective_rate*, a computer with poor performance running a crawler does not improve significantly the overall performance of the crawling process when added to the crawlers pool, as shown in the Table 2 for the cases A_1 and B_1. Nonetheless, the performance increases only on situations where the *adjective_rate* is larger, as shown in the Table 2 for the cases A_2 and B_2. This fact is explained because the search-requests are retrieved by the crawlers whenever they get freed rather than equally distributed among them; leading faster crawlers to process most of the requests, a situation that is best used in the case of a high number of adjectives.

Table 2: Benchmark of crawling same search words with 3 adjectives. A_i for the case of a single and fast crawler and B_i for the case of sharing the crawling process from the same fast crawler with an extra slow crawler.

Crwls	Adjv	Size	Imgs	Time
A_1	3	465 MB	2118	4m 20s
B_1	3	612 MB	2525	4m 17s
A_2	15	2,4 GB	11342	14m 47s
B_2	15	2,4 GB	11555	13m 40s

A_1: 1 crawler x 3 workers
B_1: 1 crawler x 3 workers + 1 slow crawler x 1 worker
A_2: 3 crawlers x 3 workers
B_2: 3 crawlers x 3 workers + 1 slow crawler x 1 worker

During the crawling process, Oculus-Crawl mixes the search words with each adjective in order to generate new search requests, which usually results in different images being displayed by the search engine. The number of images retrieved is proportional to the number of adjectives used for generating the dataset; however, as it can be stated in the Figure 4, the number of images is less than expected the more adjectives are used.

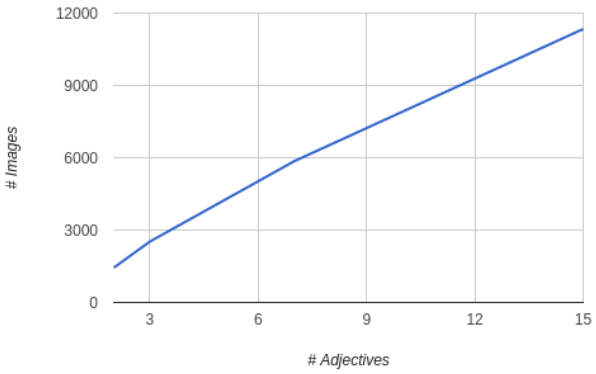

Figure 4: Relation between number of images retrieved and number of adjectives used to crawl.

The reason behind this distribution is that Oculus-Crawl hashes the images by their MD5Hash in order to discard duplicates, and it is more likely to

find more duplications or pictures out of service the more adjectives used for crawling. However, we noticed that when the crawlers were spread among servers located in different countries rather than a single country, the number of images retrieved was higher, as shown in the Table 2 in the case of B_1 and B_2. This is explained by the fact that some search engines, like Google, displays different results for the same search words based on the geographic localization of the IP that made the request [8], which reduces the probability of duplication of images.

We also measured the time spent by Oculus-Crawl to process 4 adjectives using from 1 single-threaded crawler to 4 single-threaded crawlers spread on 4 different machines and networks, as shown in Figure 5.

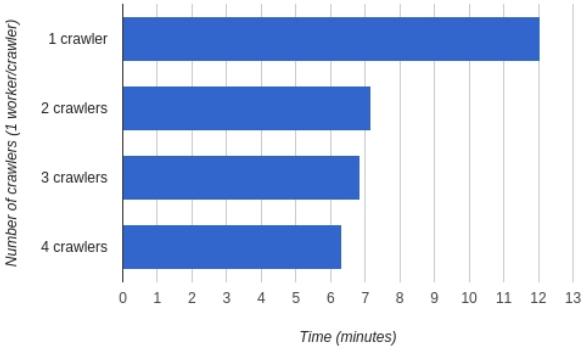

Figure 5: Time benchmark for 4 adjectives in function of the number of crawlers with a single worker each.

The measurement was done by using the UNIX *time* command preceding the invocation of the dataset creation, which gives an exact execution time. The highest increase rate in time performance was achieved by using two crawlers, which passed from 12 minutes to 7. However, the best performance was achieved by using as many crawlers as adjectives. When the crawling process is finished, the factory starts to fetch each resource from the references and finally to compress it in order to be published in a directory, which adds a static time independently of the number of crawlers working in the same pool. This static time depends on: 1) the connection speed of the factory's host and the throughput of the factory's hard disk; 2) the size, availability and connection speed of each crawled resource's host. Under our tests, 4 adjectives leveraged into 3200 images, a total of 800 MB of size and it took 2 minutes and 20 seconds from the start of the fetching stage until the publishing stage.

Finally, we counted the extensions inferred by Oculus-Crawl by checking the MIME-Type from the response headers and the URL names whenever a MIME-Type was missing, showing that the most used picture extensions belong to the JPEG extension group. In ours tests, we realized that there were some files fetched by the factory that were executables, like shown in the Table 3. This implies that search engines for images sometimes might refer to resources that are not images, even though originally they were images, showing that a preprocessing of the files to ensure that they are images is desirable.

Table 3: Extension for images found by crawling 3 adjectives.

Extension	Count	Representation
.jpg	3190	93,91%
.png	117	3,44%
.gif	81	2,38%
.jpeg	4	0,12%
.html	2	0,06%
.jpg c200	1	0,03%
.bin	1	0,03%
.exe	1	0,03%

4 CONCLUSIONS AND FUTURE WORKS

We developed and presented Oculus-Crawl, an stand-alone alternative for existing crawling tools that serves for building Computer Vision datasets by crawling images from Google and Yahoo images. It was discussed its suitability for building large datasets due to its modular and scalable architecture and its capacity to circumvent the search engines limits by combining adjectives with search words. Within our tests, we were able to crawl and fetch 11.555 images in less than 14 minutes. We concluded that the best results are achieved by distributing crawlers' workers among different countries, which leads to a different set of pictures being displayed for the same search words reducing the probability of duplications and increasing the quality and richness of the final dataset; also, the usage of as many crawlers as adjectives gives the best performance. We provided a relation between number of adjectives and number of images retrieved for a single search-word topic and a function to know an approximation of how many adjectives should be used to retrieve a specific number of images for a given topic. We also discussed about one benchmark for performance in function of the number of crawlers used and another benchmark for the impact of different speeds in multiple crawlers, In addition, we found a practice on certain hostings of swapping an original

indexed image with an executable, thus a check of image correctness before fully using the scraped dataset is advisable. During our tests we used high-end machines that vastly satisfied the needs of the tool; a much lesser specifications might be capable of achieving the same results.

The creation of a dataset of images is the first step in building a working model for Computer Vision and Machine Learning, but in some cases it is required to label each of the elements that compose the dataset; for this reason it might be useful to combine the results of this tool with some logic able to take advantage of the extracted metadata for each element in order to infer a correct label for each resource.

Also, future directions point towards retrieving other kind of resources like sounds, music, documents and videos; and to increase the number of supported search engines. Even though this software is non graphical, it might be able to be interfaced as a web page. In addition, another way of improving this tool is to automate the generation of adjectives that are semantically valid with the main search words, e.g. by using Natural Language Processing (NLP) techniques based on the language that the main search words belong to.

Lastly, we propose a session scheme, which is a way to share datasets based on crawling, that contains references instead of the whole crafted dataset's content. This tool is able to use this session scheme to rebuild the same dataset in any other computer, easing the sharing process of a crawled dataset.

4.1 HOW TO CONTRIBUTE

This project is released as open-source under the GNU GPL v3 License. It can be located in a git repository within GitHub[2]. Any contribution can be done by making pull requests to the repository or filling the issues tracker.

Acknowledgements

This research was funded by the framework agreement between the University of León and INCIBE (Spanish National Cybersecurity Institute) under addendum 22.

References

[1] Apache Software Foundation. Hadoop. Version 2.8.0. Mar. 22, 2017. URL: https://hadoop.apache.org.

[2] Apache Software Foundation. Nutch. Version 1.13.0. Apr. 22, 2017. URL: http://nutch.apache.org.

[3] Chen, K. Python icrawler. Version 0.3.6. May. 8, 2017. URL: https://github.com/hellock/icrawler.

[4] Desai Student, K., Devulapalli Student, V., Agrawal Asst, S., Kathiria Asst, P., and Patel Professor, A., (2017). Web Crawler : Review of Different Types of Web Crawler, Its Issues, Applications and Research Opportunities. *International Journal of Advanced Research in Computer Science*, 8(3).

[5] Duda, C., Frey, G., Kossmann, D., Matter, R., and Zhou, C. (2009). AJAX crawl: Making AJAX applications searchable. In *Proceedings - International Conference on Data Engineering*, pages 78–89.

[6] El-Ramly, N., Harb, H., Amin, M., and Tolba, A., (2004). More effective, efficient, and scalable web crawler system architecture. *International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04.*, pages 120–123.

[7] Grinberg, M., (2014). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 1st edition.

[8] Gupta, V., Gomes, B., Lamping, J., McGrath, M., Singhal, A., and Tong, S., (2008). System and method for providing preferred country biasing of search results. US Patent 7,451,130.

[9] Hafri, Y. and Djeraba, C., (2004). High performance crawling system. *Proceedings of the 6th ACM SIGMM International Workshop on Multimedia Information Retrieval*, pages 299–306.

[10] Kausar, M. A., Dhaka, V. S., and Singh, S. K., (2013). Web Crawler: A Review. *International Journal of Computer Applications*, 63(2):975–8887.

[11] Merkel, D., (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2.

[12] Niu, X.-m. and Jiao, Y.-h., (2008). An overview of perceptual hashing. *Acta Electronica Sinica*, 36(7):1405–1411.

[13] Razak, R. A. and Fahrurazi, F. R. (2011). Agile testing with selenium. In *Software Engineering (MySEC), 2011 5th Malaysian Conference in*, pages 217–219. IEEE.

[14] Rivest, R., (1992). The md5 message-digest algorithm. *IETF Network Working Group, RFC 1321*.

[2] https://github.com/ipazc/oculus-crawl

[15] Scrapy, A., (2016). Fast and powerful scraping and web crawling framework. *Scrapy. org. Np*.

[16] Silverstein, C., Marais, H., Henzinger, M., and Moricz, M., (1999). Analysis of a very large web search engine query log. *ACM SIGIR Forum*, 33(1):6–12.

[17] Vixie, P., (2014). Rate-limiting state. *Communications of the ACM: ACM Queue*, 12(2):10.

[18] Wang, X., Feng, D., Lai, X., and Yu, H., (2004). Collisions for hash functions md4, md5, haval-128 and ripemd. *IACR Cryptology ePrint Archive*, 2004:199.