The Anatomy of a Web Archive Image Search Engine - Technical Report

ANDRÉ MOURÃO, FCT:Arquivo.pt, Portugal
DANIEL GOMES, FCT:Arquivo.pt, Portugal

This paper presents the research and development of a large-scale web image search engine, implemented as an added-value service to Arquivo.pt's web archive. Users submit a text query and receive as result a list of related archived images and corresponding metadata. Supporting image search over web archives raises new challenges, namely having multiple versions of images and pages referenced by the same URLs over time. This raises problems that live-web search engines do not need to address: how to handle duplication of web-archived images over time and how to rank search results considering the temporal features. At the same time, the volume of temporal data to be processed is considerable, at over 8.5 billion files (2.4 billions images) crawled since 1992. Our approach to information extraction strikes the balance between coverage and efficient representation. This was achieved by storing only metadata that changed over time and by performing content based de-duplication, enabling finding duplicates indexed under different URLs. In addition, we developed a generic algorithm to image caption extraction from HTML pages created over the lifetime of the web. The main contributions of this paper are algorithms to identify relevant textual content in web pages that describe live-web and archived web images, a system architecture and workflow to index web-archived images and a ranking algorithm to order search results over web-archived images. This service is publicly available and fully open source; as of April 2021, it supports image search over 2.4 billion images crawled between 1992 and 2019.

CCS Concepts: • **Information systems → Image search**; **Digital libraries and archives**; **Information retrieval**; *Data mining*.

Additional Key Words and Phrases: image search, web archive, caption extraction, distributed search

1 INTRODUCTION

Back in 2000, a green dress lead to the creation of Google Images [23]. Currently, Google Images is the most popular image search service on the web, answering one quarter of the total web searches [15]. Official figures from 2010 had the number of indexed images over 10 billion [25]. The importance of image search as only increased ever since. Fishkin [15] shows that image search is one of the most commonly used forms of search on the Web. Google Image Search has a share of 22.6% of the searches done on the Internet in the USA amongst the major search engines. However, Google and other large scale search engines provide few details on what features they use for ranking and even fewer information regarding their indexing architecture; most information is presented on high level keynotes and presentations [13].

Commercial search engines are designed to provide users with the most up-to-date information and therefore, the most recent version of web pages and images. They update their *corpora* by the second, replacing old versions of pages and other web files with the newest information available online. However, 80% of web content is not available in its original form (e.g. updated or removed from the web) after only one year [22]. Purging the past and looking only into the present results in a large number of pages and images disappearing from the web, and thus, from commercial search engines. Thus, the need for web archiving initiatives to crawl and preserve this information is clear.

The Wayback Machine and other web archiving initiatives[1] archive the web to contradict the inherent ethereal nature of digital files. Most also provide a way of searching their archived corpus to find relevant pages. However, little advantage is taken from this vast wealth of historical information that is not a web page. This is especially clear for documents that are not text-based, such as images and videos, as there are no dedicated search engines dedicated to

[1]https://en.wikipedia.org/wiki/List_of_Web_archiving_initiatives

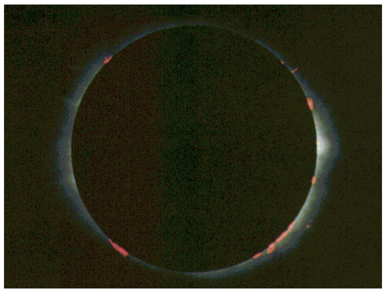

Fig. 1. Oldest archived imaged, dated April 1992

them. Costa [11] user study on information needs of web archive users identified image search as a user need that was not supported at the time of publication.

Web archives complement live-web search engines because they provide a temporal perspective of the web data. However, this fact raises significant challenges not addressed when dealing only with the live-web: how to index web content collected over multiple years and how to deal with the resulting duplicated content (e.g. are pages and images that have not changed over time redundant?).

1.1 The Arquivo.pt web archive

Arquivo.pt is a publicly available research infrastructure that provides access tools over historical web data, mainly focused in the preservation of information of general interest to the Portuguese community. Arquivo.pt is a public service supported by the Ministry of Science of the Portuguese Government and its main focus is to preserve information of general interest to the Portuguese community. Nonetheless, it contains more than 10 billion files collected from 27 million websites in several languages preserved since the early days of the Web and includes collections of international interest such as the Olympic Games, the World War I Centenary Commemoration, European elections and R&D projects[2] or the Geocities archive[3]. Arquivo.pt has been developed since 2007 to incrementally improve access to its collections and respond to the needs of researchers and citizens. The services provided by Arquivo.pt included full-text search, version history listing, advanced search and application programming interfaces (API) that facilitate the automatic process of large-amounts of historical web data or the development of innovative applications. Arquivo.pt has been used to support research & development activities in several areas by preserving and providing access to valuable scientific resources that became unavailable online and are found by researchers (e.g. past news, data sets, grey literature). Since 2016, Arquivo.pt has been investing efforts on image search over web-archived content. The oldest archived image is dated April 1992, Figure 1[4], and is not available in its original URL anymore.

1.2 Research questions and contributions

This article describes the work developed to created the current Arquivo's pt image search service deployed into a production system in April 2021. Figure 2 presents a screenshot of the Arquivo.pt image search user interface. The goal was to develop a system that addresses the challenges raised by the inherent temporal properties of web-archived

[2] https://sobre.arquivo.pt/en/collaborate/colaborative-collections/
[3] https://sobre.arquivo.pt/en/historical-collection-geocities-available-at-arquivo-pt/
[4] https://arquivo.pt/image/search?hitsPerPage=10&query=pt&l=pt&ion-dt-1=1991-01-01&ion-dt-0=1992-08-06&dateStart=01%2F01%2F1991&dateEnd=06%2F08%2F1992

data, while giving users with a familiar look-and-feel similar to a live-web image search engine such as Google images. Arquivo.pt's image search is designed to find images by their associated metadata (e.g. image URL) and information in the HTML pages that reference them (e.g. alternative text, textual caption extracted from HTML).

The research questions that originated this work were the following:

- Which algorithms should be applied in a web archive to identify relevant textual content in web pages that describes web images (RQ1)?
- How to de-duplicate and index a large volume of web-archived images (RQ2)?
- How to rank web-archived images, considering their temporal features (RQ3)?

Arquivo.pt's main novelties arise from the way it deals with temporal web data, volume of information (1.9 billion web images) and openness (open source code, image search API freely available to the public). The main contributions of this work are:

- a set of algorithms to identify relevant textual content in web pages that describes live-web and archived web images;
- a system architecture and workflow to index web-archived images that reduces index sizes by de-duplicating web-archived images across time and space;
- a ranking algorithm to order search results over web-archived images;
- a web user interface to search web-archived images;
- a publicly available running service that can be used to support research studies over historical images published on the web;
- an open-source project that can be deployed to support image-search over other web archives[5].

The structure of this article is as follows: Section 2 gives brief overview of the recent advancements on image search systems on the web and on web archives. Section 3 presents the algorithms applied to identify and associate relevant metadata to textually describe web images and Section 4 describes the algorithms applied to index web-archived images considering the peculiar features of this data set such its high level of duplication. Section 5 describes how the previously described algorithms were combined into the Arquivo.pt image indexing workflow. Section 6 describes the image search system focusing on the chosen ranking features and algorithm, user interface and obtained results. Section 7 exposes the limitations of the presented work and suggests directions for the future of web-archive image search.

2 RELATED WORK

Brin and Page [7] seminal article provided the first detailed glimpse into what became the largest search engine in the world, that spanned numerous other search engines. It describes the initial Google page rank implementation and ranking system, metadata extraction (e.g. anchor text extraction), server architecture and data structures and crawler infrastructure. The article started by presenting the following challenges: "The amount of information on the web is growing rapidly, as well as the number of new users inexperienced in the art of web research." These challenges still exist over twenty years later, as the rate of the growth of data continues to raise and the heterogeneity of media formats widespread on the Web.

However, Google and other commercial search engines provide few details on how their systems work at scale. Existing papers are either outdated [4, 10] or part of a high level presentation [12]. In the broader academic literature,

[5] https://github.com/orgs/arquivo/projects/7)

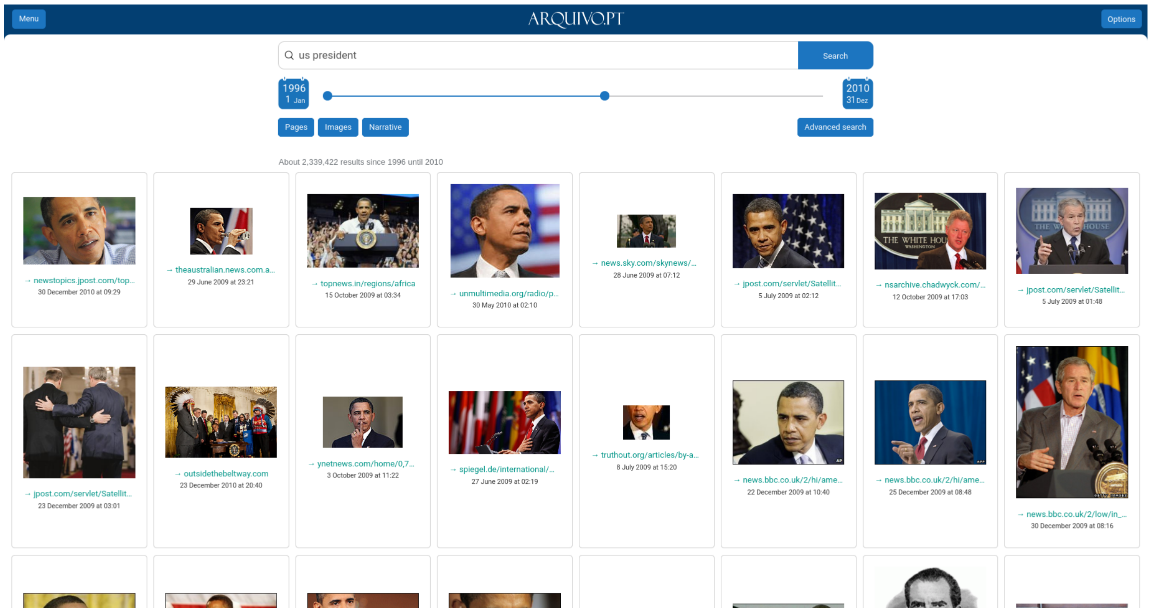
Menu
ARQUIVO.PT
Options
us president
Search
1996 1 Jan
2010 31 Dez
Pages
Images
Narrative
Advanced search
About 2,339,422 results since 1996 until 2010
→ newstopics.jpost.com/top...
30 December 2010 at 09:29
→ theaustralian.news.com.a...
29 June 2009 at 23:21
→ topnews.in/regions/africa
15 October 2009 at 03:34
→ unmultimedia.org/radio/p...
30 May 2010 at 02:10
→ news.sky.com/skynews/...
28 June 2009 at 07:12
→ jpost.com/servlet/Satellit...
5 July 2009 at 02:12
→ nsarchive.chadwyck.com/...
12 October 2009 at 17:03
→ jpost.com/servlet/Satellit...
5 July 2009 at 01:48
→ jpost.com/servlet/Satellit...
23 December 2009 at 03:01
→ outsidethebeltway.com
23 December 2010 at 20:40
→ ynetnews.com/home/0,7...
3 October 2009 at 11:22
→ spiegel.de/international/...
27 June 2009 at 02:19
→ truthout.org/articles/by-a...
8 July 2009 at 15:20
→ news.bbc.co.uk/2/hi/ame...
22 December 2009 at 10:40
→ news.bbc.co.uk/2/hi/ame...
25 December 2009 at 08:48
→ news.bbc.co.uk/2/low/in_...
30 December 2009 at 08:16

Fig. 2. Arquivo.pt image search results for the query "us president" from 1996 to 2010

most papers are focused on lower level systems and techniques, instead of in the deployment of search in a fully accessible online system.

Web archive large scale search and architecture

Fortunately, some web archives provide both large scale page web search and go into detail about their implementations. The Royal Danish Library blog contains a detailed description of the technical usage of SolrCloud at very large scales (16 billion documents and 70 TB of data) [14] to search archived pages. Other examples of scaling Solr to millions and billions of documents come from Solr documentation [26].

SolrWayback 4 [21] provides a system and pipeline, that goes from processing captured data to a fully fledged web search UI. Instead of using the traditional capture indexes (CDX) to replay web archived resources, SolrWayback relies on Apache Solr server(s). ARC/WARC files have been indexed using the British Library WARC-Indexer software[6]. The Hungarian web archive[7] is one of the first archives to use this version of the SolrWayback software. However, none of these systems is designed specifically to find images.

Image search on archives

In 2009, Yahoo announced it was closing down the United States version of the Geocites. It contained more than 38 million user-created web pages. The Internet Archive made a special effort to these web pages before the service was shut down [3]. In 2016, the project *GifCities: The GeoCities Animated GIF Search Engine* was released[8]. It is a text-based animated GIF images search engine, that uses the directory path and image filename as the image information. *Gifcities* contains 4.5 million animated GIF images (1.6 million unique images after de-duplication) extracted from archived Web

[6]https://github.com/ukwa/webarchive-discovery/
[7]http://webadmin.oszk.hu/solrwayback/
[8]https://gifcities.org/

resources. In addition to showing the animated GIF images, it links to the archived version of Web page that originally contained the image.

The Wayback Machine provides a search interface focused on images[9], Instead of providing search in the full set of archived images, search is limited to about 4 million donated images, not the full corpus of archived web data. The Wayback Machine also provides a limited experimental reverse image search API[10], similar to TinEye[11].

Page segmentation and captioning

Arquivo.pt image search goal is to enable finding images that show up in web pages, according to their context and position in page (e.g. which text shows up next to it). There is significant research in web page segmentation [8, 9] and it's application to caption extraction [2]. VIPS [8], [9] perform visual analysis based on HTML structure, vertical and horizontal separators to find cohesive blocks. A recent review on web page segmentation [19] shows how the field evolved over the last years. In our experiments, these techniques are computationally heavy: for example, Alcic and Conrad [2] requires parsing the full DOM tree once to determine the cutoff thresholds between page sections. Thus, we developed a technique that can scale to the billions of pages we need to inspect, Section 3.3.

To the best of our knowledge, little as been published about live-web image search. Even fewer research was performed on how to search images in web archives with the additional challenges raised by the peculiar features of historical web data.

2.1 The Arquivo.pt web archive

In 2018, Arquivo.pt launched a prototype image search system for a sample of the archived data, comprising a total of over 22 million unique images archived from the Web between 1992 and 2017. This paper describes the extension to cover all the collected data, increasing the total number of indexed images to over 1.9 billion. Table 1 shows the statistics for the service, as of April 2021 (available at https://arquivo.pt/image/search?l=en).

Table 1. Statistics for Arquivo.pt as of April 2021

Findable images in 2020	22,881,688
Findable images in 2021	1,862,311,456
Oldest archived image	15/04/1994
Newest archived image	14/11/2020

The main goals of our image search service are to:

- Find and index relevant textual information to associate with archived images;
- Provide a user-friendly interface that enables the exploration of the temporal features of the results (easy to learn, powerful to master);
- Provide search results with an average response time below 1 second.

3 ALGORITHMS TO ASSOCIATE TEXTUAL METADATA TO WEB IMAGES (RQ1)

The main goal of Arquivo.pt's image search is to enable finding images using textual queries. In order to build a such a system for a large web archive, one first needs to find these images and associate related metadata from the archived data.

[9]https://archive.org/details/image
[10]https://archive.readme.io/docs/reverse-image-search-api
[11]https://tineye.com/

Images in HTML pages are used to illustrate the content of the page or subsection of a page, either as an inline image (<img> tag) or as an link (<a> tag). For example, *imgAlt* is a required attribute[12] for inline images that appear in pages. The page creator inputs a textual description of the image to be used it the page cannot be displayed. This is useful if there is a server/connection error that prevents the image from loading or for users that rely on screen readers to browse the web.

Extracting metadata from HTML using data captured for archival purposes poses a specific set of challenges: separating images from the remaining web content, matching HTML pages to images and finding which page text and metadata better represent each page image. The following sections will detail how we approached these challenges, taking into account the billion file scale of your archived data.

3.1 Identify related pages for archived images (Algorithm 1)

In order to create image indexes, one needs to parse WARC and ARC files, to find images and their corresponding metadata extraction. The WARC (Web ARChive) format [17] is the standard way of storing information collected in from the web. It can store any type of web resource content (e.g. web page, image, video, PDF) and related information (e.g. server response codes, headers). Additionally it combines and compresses multiple web resources together into a single WARC file, simplifying the web crawling processes that can reach hundreds of GB of data per crawl. The ARC format is an older revision of the standard. As the differences between ARC and WARC files do not impact most techniques described in this paper, the term WARC will stand in for both ARC and WARC archives, except when diferences are relevant.

Crawler nodes start the archiving process by collecting and parsing HTML pages. Linked resources such as associated images are added to a crawl queue to be archived later. For a longer description behind the challenges behind this process, check Gomes et al. [16], chapter 2. Pages and their linked images are not crawled simultaneously, and can be placed on different WARC files and/or be crawled in different nodes. This leads to our first challenge: matching images to pages.

Images are linked in HTML using their URLs: *<img>* tag *src* attribute, CSS's *background-image* attribute. In WARC records, the image URL is stored as part of the image record or *<a>* tag *href* attribute (e.g. if the *href* ends with an image file extension). For HTML pages, one can find image references by parsing the HTML, extracting the relevant tags (*<img>*, *<a>*, CSS with *background-image*) and extracting the URL from the tags.

Algorithm 1 describes this process. It roughly matches the process in the the ImageInformationExtractor class[13].

ARCs and WARCs go though slightly different pipelines to get the bytes that match the record (*parseArcRecord* vs. *parseWarcRecord*), but the information extraction method is the same for both. For each record, we check whether it is an image record or HTML page record using the mime type provided by the metadata. Other record types are not processed.

All URLs are normalized to SURT[14] (*getSURT*), a canonical URL/URI representation, to match similar URLs with different, equivalent capitalization and formats.

Metadata is extracted for both images and pages. The following section details what information is processed. An important note is we exclude images that are too small (smaller than 50 pixels in height or width) or too large (with

[12] https://html.spec.whatwg.org/multipage/images.html#guidance-for-conformance-checkers
[13] https://github.com/arquivo/image-search-indexing/blob/88fae906631cf14f996aa9b703e85c9af5c264f6/src/main/java/pt/arquivo/imagesearch/indexing/processors/ImageInformationExtractor.java
[14] http://crawler.archive.org/articles/user_manual/glossary.html#surt

Algorithm 1: Matching images to pages

```
Input: Set of WARC files $warcs$;
Output: Map of metadata $metadatas$;
$MIN\_HEIGHT = MIN\_WIDTH = 50$;
$MAX\_HEIGHT = MAX\_WIDTH = 15000$;
$metadatas = \{\}$;
forall WARC warc in warcs do
    forall Record record in warc do
        if record is HTML then // process page
            $imageRef$ = findImageTags($record.html$);
            forall ImageRef imageRef in imageTags do
                $imageSURT$ = getSURT($imageRef.url$);
                $metadatas[imageSURT]$.add(extractPageMetadata($imageRef$));
            end
        else if record is Image then // process image
            $imageSURT$ = getSURT($record.url$);
            $metadata$ = extractImageMetadata($record$);
            if $metadata.width > MIN\_WIDTH\ \&\ metadata.height > MIN\_HEIGHT\ \&$
              $metadata.height * metadata.width < MAX\_HEIGHT * MAX\_WIDTH)$ then
                $metadatas[imageSURT]$.add($metadata$);
            end
        end
    end
end
return metadatas
```

an area larger than 15000*15000 pixels). In our experiments, smaller images are icons or navigational and very large images are usually malformed or result in browser slowdowns when they show up in the search result page.

After processing all WARC files, entries in *imagesMetadata* will consist of a set of image metadatas and page metadatas. As an image may be recorded multiple times, and as images can appear on more than one page, there may be more than one image metadata entries and more than one page metadata entries for the same SURT. Section 4.1 will detail how we merge this records.

3.2 Assign related HTML attributes for each archived image

To find image tags in HTML, we used JSOUP[15] to parse and iterate through the HTML tags. For CSS images, we used an heuristic based on regex, that matches URLs in the form of $url(['"]*(.*?)['"]*)$.

The *extractPageMetadata* function extracts relevant textual metadata for an image in a page. The function behaves differently whether the image is in an **<img>** tag, **<a>** tag or CSS *background-image*. For *<img>* tag, the following information is extracted:

- *imgTitle* - *<img> title* attribute; it is used to provide additional information about the image;
- *imgAlt* - *<img> alt* attribute; it provides alternative information about an image if a user cannot view it;
- *imgCaption* - image caption extracted from the HTML page. More information on this process is available on the following section.

[15]https://jsoup.org/

For *<a>* tag, the following information is extracted:

- *anchorText* - *<a>* tag inner text. This field is used as the *imgCaption* for this type of images.

Images detected using CSS, *background-image*, do not have additional metadata, due to their inherent properties (e.g. it is hard to determine where these images will be placed in the page, as they are used in broad areas of the page using the *style* attribute). Finally, the following information is extracted, regardless of tag type.

- *pageTitle* - Page title attribute; it is used to provide additional information about an HTML page;
- *pageURLTokens* - The keywords of the URL of the HTML page that contains the image;
- *pageCrawlTimestamp* - timestamp (in seconds) for the exact moment the page was crawled. This attribute is present in the WARC record;
- *imgURLTokens* - The keywords of the image URL. The image URL gives an unique path to an image, which often descriptive information such as the image filename.

The *extractImageMetadata* function extracts the following image attributes from it's HTML request headers and file image content:

- *imgWidth* and *imgHeight* - image width and height (in pixels). Extracted by examining the image file content;
- *imgCrawlTimestamp* - timestamp (in seconds) for the exact moment the image was crawled. This attribute is present in the WARC record;
- *imgMimeType* - image mimetype. Although the WARC record contains the mimetype, we manually detected based on image content. This is because the mimetype reported by the original server that sent the image is very often wrong;
- *imgThumbnail* - image thumbnail (max 200x200 pixels), stored in base64.;
- *imgDigest* - digest of the image content bytes generated using SHA-256.

A possible source of image data would be EXIF information. We decided to go against this approach for a number of reasons: only a small subset (1%) have EXIF data, as most image hosting platforms strip this information when making it available to the public. In addition, most of this information is irrelevant for a search engine: focal length, ISO settings and other camera parameters. The information that is potentially relevant (image coordinates and exact capture timestamp) have large privacy concerns that go well above our goal of building a search engine for the images in the web.

Another metadata extraction possibility would be to use deep image classifiers, like Yolo v4 [6]. They can achieve good results when generating consistent image tags and captions, but they are more generic than what HTML pages can provide. Even if the deep system add additional context information, it would be impossible to have it be relevant for the diversity of the crawled data. In addition, the computational power (more specifically GPU computational power) required to process billions of images would greatly increase our total processing time.

Parsing such a set of diverse web data ranging from 1992 to 2020, archived using different formats (ARC and WARC) and using different crawlers (e.g. Heritrix, Browzzler or even created manually from HTML files) resulted in a number of problems and fixes.

- **WARC record chunking and compression** : content in the web can be chunked and compressed in a myriad of combinations. We used *TikaInputStream*, but had to manually wrap it a *BrotlyInputStream* when the (W)ARC record headers reported *br* compressed content;

- **Malformed ARC and WARCS**: some records in (W)ARCs are malformed (e.g. file closed unexpectedly, disk corruption). When possible, we try to skip over malformed records and continue parsing;
- **HTML encoding**: in addition the text encoding reported by the server, we used Tika's text encoding detection function to find the HTML page encoding.

3.3 Extract archived images captions from HTML (Algorithms 2 to 4)

In the previous section, we described how to extract image metadata from HTML pages. But making *imgAlt* a required attribute in the HTML standard is not enough to have page creators add it to their images. In our archive data, we found that only 18% of the images have *imgTitle*, 52% have *imgAlt* and 55% have *imgAlt* or *imgTitle*. Images without specific metadata such as the *imgAlt* and *imgTitle* can only be found using *imgUrl* (which is often not descriptive of image content) or page metadata (*pageTitle* or *pageUrl*).

Page titles and URLs are, at most, descriptive of the main images of a page. For example, in news articles, the HTML page title only describes images that are related to the main story. Images in latest news article lists are left without any metadata that specifically describes them. This lack of metadata gets worse as content creators often only fill the *imgAlt* and *imgTitle* information for the main page images. Where can we find potentially relevant information for all images in a page?

The solution we arrived at was to find text that is close to the image in the page, according to its position in the HTML DOM. This is related to web page segmentation [2, 8, 9, 19] and caption extraction [18] research. Our goal was to find a simple technique that can scale to the billions of pages and images.

Our method applies to *<img>* tags, as we found them to make up about 90% of the images we captured. Images from *<a>* tags have anchor text, which conveys the same type of information as the extract captions and CSS *background-image* lack information about their specific information in the page.

Algorithm 2: Get Parent Node Text

```
Input: HTML <img> node;
Output: imgCaption;

depthMaxNodes = getDepthMaxchild(node)
imgCaption = "";
parent = node.parent();
while imgCaption is empty & parent is not null do
    imgCaption = parent.text();
    parent = parent.parent();
end
return imgCaption
```

The method we used is based on the HTML page tree structure, with *<img>* tags in the leaves. Starting at an *<img>*, as you move up HTML tree (i.e. move to your parent node), the larger the percentage of elements you will have under a node. Our original hypothesis was that, starting at an *<img>* tag, the first ascend node you find that contains *human-readable text* (e.g. by calling the *text()* method using BeautifulSoup[16] or *text()* using JSOUP[17]). If calling *text()*

[16]https://www.crummy.com/software/BeautifulSoup/bs4/doc/#get-text
[17]https://jsoup.org/cookbook/extracting-data/attributes-text-html

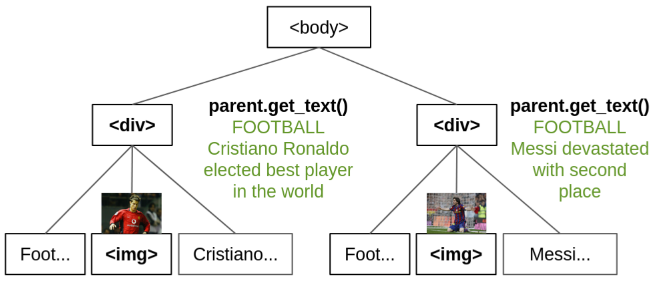

Fig. 3. Example of a HTML DOM tree with image nodes and potential captions

for a node returns an empty string, we replace that node for its parent and check *text()* again. This process is described in Algorithm 2 and is similar to the one described in [18]. The code associated with this algorithm is available here[18]

Figure 3 shows a very simplified example of an HTML page DOM tree with two images as leaf nodes (*<img>*) and potential captions as siblings of the image nodes. Images are encompassed in a *<div>* that also contains an adequate description of what is present in the image (abbreviated as "Foot...", "Cristiano..." and "Foot...", "Messi...").

But the *parent text* technique fails for pages with "flat" structures, where most HTML elements are placed at the same level in the DOM. This is especially prevalent in pages with lists of posts such as blogs. Figure 4 shows an example of the HTML page DOM tree of such a page with two images and potential captions. Image tags are present at the same level as the textual tags and information. If we go up the DOM using the *parent text* technique, the first parent with *human-readable text* will have all the text in the page, losing the desired locality of our caption extractor.

In pages with this structure, the relevant text is present in the *<img> sibling* nodes (next to the tag in the DOM). Thus, if we can detect this structure, we could run the *text()* method on *sibling* nodes to get text that is close to the image in the page. We will call this the *sibling text* technique.

Detecting flat structured pages from across the history the Internet is no simple task. The way HTML pages are structured and developed, from chunky *<table>* based structures to minimal *flex-grid* pages that adapt to all screen sizes. We decided to do a method that can be applied to a broad set of pages.

For each HTML page, we find the level of the DOM that has the highest amount of nodes. This can be computed with reasonable certainty by staring at an image node, and going up the DOM, counting the number of children at each level. We store the depth and number of children of the node with the largest number of children. This is described in Algorithm 3.

The full caption extraction is described in Algorithm 4. The code associated with this algorithm is available here[19] Each image node in a page, we run the *parent text* for each *<img>* tag in the DOM. If the first parent node with text is the one with the largest number of children, the page has a flat structure, and we resort to getting the text from the closest siblings that have *text content.*

This process works effectively for most pages, except for some pages that were causing our Hadoop processed to timeout. This was mostly related to pages containing malformed image galleries (e.g. generated automatically from inefficient HTML templates). Some of these pages contained over 10.000 *<img>* which made the finding parents with

[18]https://github.com/arquivo/image-search-indexing/blob/88fae906631cf14f996aa9b703e85c9af5c264f6/src/main/java/pt/arquivo/imagesearch/indexing/processors/ImageInformationExtractor.java#L543
[19]https://github.com/arquivo/image-search-indexing/blob/88fae906631cf14f996aa9b703e85c9af5c264f6/src/main/java/pt/arquivo/imagesearch/indexing/processors/ImageInformationExtractor.java#L603

Algorithm 3: Get Depth with Max Child Count

```
Input: HTML <img> node;
Output: imgCaption;

depthMaxNodes = -1;
depthMaxNodesCount = -1;
parent = node.parent();
while parent is not null do
    if parent.children().size() > depthMaxNodesCount then
        depthMaxNodes = parent.depth();
        depthMaxNodesCount = parent.children().size();
    end
end
return depthMaxNodes
```

Algorithm 4: Extract captions from HTML pages

```
Input: HTML <img> node;
Output: imgCaption;

depthMaxNodes = getDepthMaxchild(node)
if parent.depth() > depthMaxNodes then
    return getParentNodeText(node);
else
    leftNode = node.leftSibling();
    while leftNode.text() is empty do
        leftNode = leftNode.leftSibling();
    end
    rightNode = node.rightSibling();
    while rightNode.text() is empty do
        rightNode = rightNode.rightSibling();
    end
    return leftNode.text() + rightNode.text() ;
end
```

text algorithm very expensive. As these pages often did not contain useful caption information, we also added a per page time limit on caption extraction (default is 60 seconds per page). If this timeout is reached when extracting captions from a page, the metadata extractor stops running the caption extraction method on the remaining images of that page.

4 INDEXING WEB-ARCHIVED IMAGES (RQ2)

When indexing images from the web, one can find an image that was already found sometime before. Live web search engines are designed so that new images and image metadata always replaces old images, and images that disappear from the web are eventually removed from the index. Web-archived image search engines must provide an historical view of the web and thus, face an additional challenge: how to index images that show up more than once over time (since 1992) and space (on different parts for the web)?

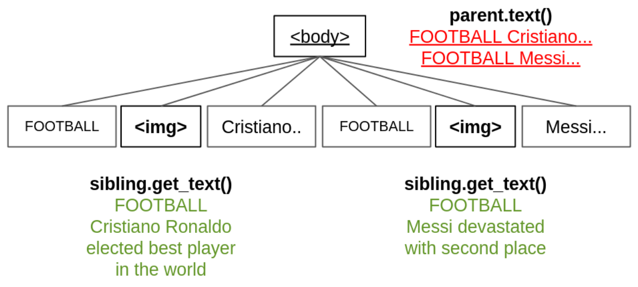

Fig. 4. Example of an unstructured HTML DOM tree with image nodes with wrong parent captions and correct sibling captions

4.1 De-duplicating images across time and space (RQ1, RQ2)

As we examine billions of images and pages collected over more than 30 years, we can have a scenario where we have multiple images and sets of metadata for the same SURT.

In addition to capturing the same image multiple times, the same exact image file may be replicated over multiple SURTs and images under the same SURT may change over time. This means, that, in addition to SURT, one must also look into the image file content when de-duplicating content. This de-duplication problem reaches across two dimensions:

- **Time:** images are archived more than once;
- **Space:** images show up on more than one web page.

A large amount of this information for images captured more than once is redundant (i.e. sets of metadata with the same *imgAlt*, *imgCaption*, ...). In our archived data, we found that 70% of images we collect from the web are duplicated. This leads to an interesting challenge: how to de-duplicate image information, while keeping the most relevant metadata for all images?

To find out what to do with duplicate images, we must go all the way back to the image search page, and consider that users want and do not want to see in their search result pages. After an internal discussion, we arrived at a set of "assumptions" that will guide our de-duplication scenarios:

- Users use Arquivo.pt page search to find pages and image search to find image;
- The link between an image and a page is important. But finding the specific page where the image appears is less important than finding a page;
- When an image appears on more than one page, finding the oldest page best matches the information need of a web archive user;
- Users do not want to see duplicate images on the search results when an image appears in the search results;
- Technical detail information (*imgAlt*, ...) is rarely accessed by users.

After careful examination, we arrived at three de-duplication scenarios that can be applied to images that are duplicated across both time and space:

- **No de-duplication (FULL)**: index every page-image pair. In this scenario, the same image is indexed multiple times, each one with a full set of page metadata matching a different page where it showed up;
- **Oldest page only (LEGACY)**: index image and page metadata for the the oldest page that references the image;

- **Oldest page metadata + full image metadata (COMPACT)**: index oldest page metadata plus unique image metadata (*imgTitle*, *imgAlt*, *imgCaption*) from all pages.

In the initial prototype, the **oldest page scenario** was selected, and de-duplicated images according to a SHA-256 digest of their content. This choice was driven mostly to avoid showing duplicate image search results and reduce the scale of the data to index.

As we progressed with the current version of the image search system, we initially planned to do **no de-duplication**. The same image would be indexed multiple times (one per page). When the user search for an image, we would search in all images and find the best matches (including duplicates). Before returning the final results to the user, we would de-duplicate images by choosing the image whose page better matched the user query.

The problem with this approach is scale: Before de-duplication, we had over 1.9 billion sets of image records. With the continuous growth of data from Arquivo.pt crawls, we'd quickly reach the limits of the servers we are using to provide search.

For the current release of Arquivo.pt web search, we chose the **Oldest page metadata + full image metadata** scenario. It reduces the information to index, while keeping the most important information from the pages. For each image, aggregate the sets of image and page information into a single record (oldest page metadata, all unique values for image metadata). This aggregation is performed by key (SURT or image digest). This process is described in Algorithm 5. It matches the code in the ImageInformationMerger class[20].

The impact of this de-deduplication method is described in Section 5.5.

4.2 Popularity features for images based on duplication (RQ3)

The de-duplication process enables us to collect statistics related to the metadata (e.g. how many times did the *<img>* alt and title entries changed) and use them as features for ranking. [21]. Our reasoning is that images that have many metadata changes are often placeholders (e.g. the default blank person avatar on LinkedIn will have multiple alt tags, representing the names of the users that have the default avatar as their profile picture) or style elements, and thus, less relevant that images that have less metadata.

In addition to features extracted from the HTML textual content page, we compute multiple additional features related to the image popularity and crawl frequency: For each image, the following information is extracted:

- *matchingImages* - number of times the image was crawled;
- *matchingPages* - number of pages that reference the image;
- *imagesInOriginalPage* - number of images in the page (maximum of all pages that reference the image);
- *imageMetadataChanges* - number of times that the image metadata (alt and title) changes;
- *pageMetadataChanges* - number of times that the page metadata (title) changes.

Figure 5 shows the images that appear in the most pages (sorted in descending *matchingPages*). These images are mostly logos or placeholders that show up in a lot of pages, but are not what users would want to see. Thus, very high values in these metrics should downgrade these results in the search result page.

[20] https://github.com/arquivo/image-search-indexing/blob/88fae906631cf14f996aa9b703e85c9af5c264f6/src/main/java/pt/arquivo/imagesearch/indexing/processors/ImageInformationMerger.java

[21] Due to the scale of the data that is being processed, the size of this list is limited to 50 title/alt unique entries per URL. The counter for the number of metadata changes is still increased to all changes.

Algorithm 5: De-duplicate metadata entries

```
Input: Map of metadata $metadatas$ with multiple entries per image (as generated by Algorithm 1;
Output: Map of metadata $metadatasSingle$ with a single page and a single image entry;
$metadatasSingle$ = {};
forall $key$ in $metadatas$ do // the key can be the digest or the SURT
    $metadataSet = metadatas[key]$;
    $oldestPageMetadata$ = {};
    $combinedImageMetadata$ = {};
    forall $imageMetadata$ in $metadataSet$ do
        forall $field$ in $imageMetadata$ do // e.g. imgAlt, imgTitle, ...
            forall $value$ in $imageMetadata[field]$ do // e.g. value of imgAlt, imgTitle, ...
                if $value$ not in $combinedImageMetadata[field]$ then // Add it does not exist already
                    $combinedImageMetadata[field]$ += $value$;
                end
            end
        end
    end
    forall $pageMetadata$ in $metadataSet$ do
        if $pageMetadata.pageCrawlTimestamp < oldestPageMetadata.pageCrawlTimestamp$ then
         // Select oldest page
            $oldestPageMetadata = pageMetadata$;
        end
    end
    $metadatasSingle[key] = oldestPageMetadata + combinedImageMetadata$;
end
return $metadatasSingle$
```

4.3 Assign Not Safe For Work ratings (RQ3)

Arquivo.pt aims at collecting all information for pages in the .pt domain or Lusophone pages in other domains (e.g. .com, .net). The goal is to go towards preserving as much information that may be of interest for the Portuguese community. In that context, some of the images captured may contain pornographic content that users do not want displayed by default. Arquivo.pt enables a default filter in order to exclude pornographic images from the search results, automatically classified as Not Safe for Work (NSFW). This filter can be disabled by the user through the image search interface.

A previous version of the NSFW classifier is documented in [5]. It is based on OpenNSFW[22] deep neural network Caffe models, retrained with additional 8273 NSFW and 9382 SFW examples. As it is based on deprecated technology (Python 2 and Caffe) and converting the model to newer libraries proved not to be feasible, a new techinque had to be devised.

Arquivo.pt uses an NSFW image classifier based on GantMan's model [20], a Deep Neural Network (DNN) solution. It is based on a TensorFlow [1] using a Inception v3 [27] network, trained with over 60 GB of images scrapped from the web. Instead of identifying images as safe or not safe, it returns the probability of an image belonging to one of five categories:

[22]https://github.com/yahoo/open_nsfw

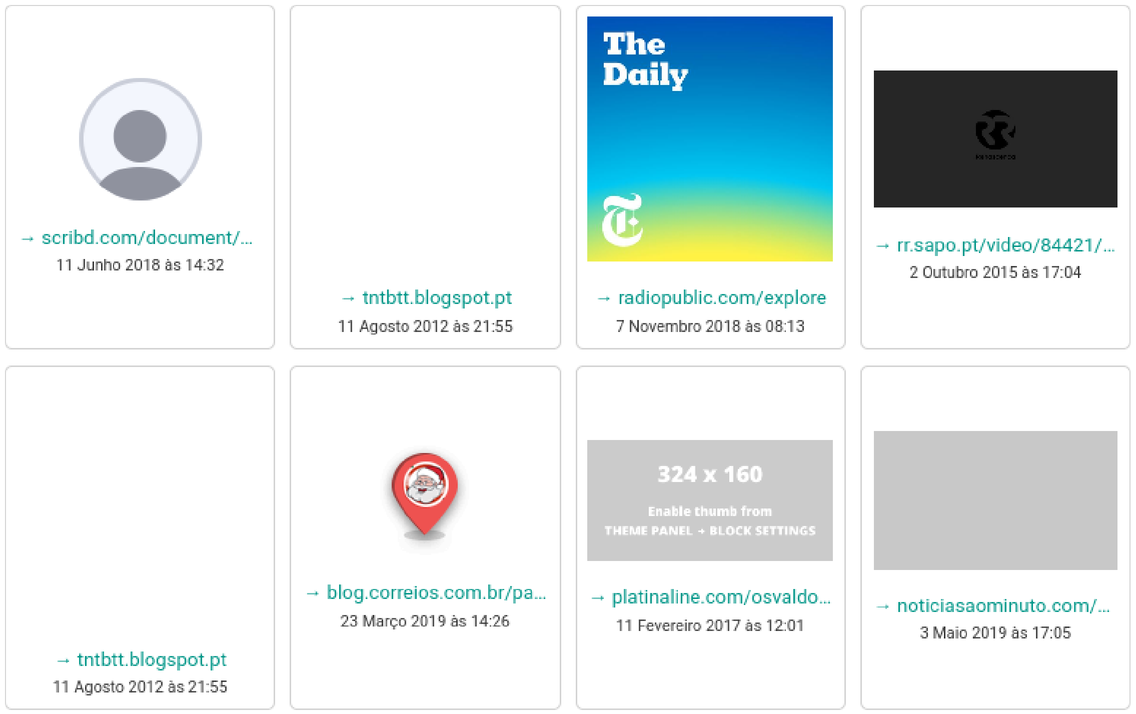

Fig. 5. Images that show up in the most pages

- **drawing**: non-pornographic drawings or illustrations, including anime;
- **hentai**: hentai and pornographic drawings;
- **neutral**: safe for work neutral images;
- **porn**: pornographic images, including images with explicit sexual acts;
- **sexy**: potentially sexually explicit images that are not pornographic (e.g. woman in a magazine cover wearing a bikini).

We also defined a NSFW field defined as:

- **nsfw**: $hentai + porn$

At retrieval time, images are filtered from the search results if $nsfw > 0.5$.

5 IMAGE INDEXING WORKFLOW (RQ3)

The previous sections provide a high level view of the metadata extraction, de-duplication and NSFW classification process. In this section, we will focus on how those processes are executed in our distributed cluster. The workflow of Arquivo.pt image indexing and search pipeline, Figure 6, consists of four main stages running on three separate clusters:

(1) **Hadoop cluster**: Extract and de-duplicate metadata
(2) **NSFW cluster**: NSFW image classification;
(3) **SolrCloud cluster**: Indexing and Search system.

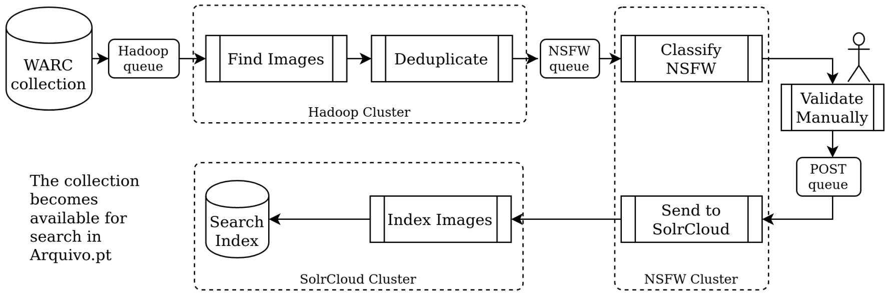

Fig. 6. Arquivo.pt image indexing workflow.

These stages are connected using a set of *Redis* queues, that enable data to move sequentially across all processing stages.

Web crawls are usually limited in time or scope. In Arquivo.pt's case, these crawls can be daily new site crawls, quarterly broad .pt domain crawls or crawls of a specific category of websites (e.g. EU Horizon2020 project sites). Regardless of the type of crawl, the output is a set of WARC files containing the collected resources. Note that, due to wildly varying scopes, craws can generate WARCs sized somewhere between a couple of MB into dozens of TB. Note also that, due to our embargo policy, collections only become available for search two years after they are collected.

The amount of data to process is considerable: 530TB of WARCs, divided into 115 collections. To deal with this scale and simplify our internal tracking on what data is already processed, indexed, etc., processing is performed by collections (i.e. WARCs from the same collection are processed at the same time).

5.1 Extraction of images and metadata

To process data at scale, metadata extraction and de-duplication (Section 3) is executed in an Hadoop cluster. Our algorithm and data partitioning matches the Map-Reduce paradigm very well: WARC record metadata can be extracted and placed in the ideal location in the HDFS <key value> storage (Map stage); and de-duplicated and merged by SURT in the Reduce stage. A similar process can be executed for content-based de-duplication: metadata grouped by SURT can be regrouped according to the SHA-256 image digest, Map stage, and de-duplicated by SHA-256 image digest in the Reduce stage.

Thus, to transform Algorithm 1 into a Hadoop Map-Reduce jobs, we went with two Map-Reduce jobs: Finding images and metadata using Hadoop and Content-based de-duplication.

This task is performed as a Map Reduce process[23], that takes WARCs as its input and outputs image and page metadata entries to HDFS, similar is structure to Algorithm 1 output:

- Map: a map process takes a set of WARCs, extract page and image metadata for all images and page records in WARCs and places them in the HDFS entry matching their SURTs
- Reduce: for each page and image metadata in the *imagesMetadata* set, create a new de-duplicated record (SURT-based de-duplication).

[23]https://github.com/arquivo/image-search-indexing/blob/88fae906631cf14f996aa9b703e85c9af5c264f6/src/main/java/pt/arquivo/imagesearch/indexing/processors/ImageInformationExtractor.java

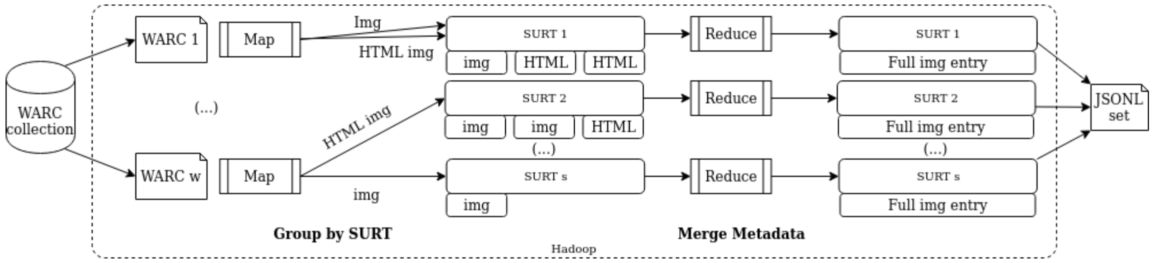

Fig. 7. Data flow for Image Extraction in (W)ARC files

5.2 Content-based de-duplication

This task is performed as a Map Reduce process, that plugs directly into the output of the previous step:

- **Map**: parse a set of (W)ARCs and find images and page *<img>* and pass the extracted data into the Reduce process, grouping image records and metadata *<img>* by SURT;
- **Reduce**: for each set of image and metadata results in a SURT, merge metadata and generate a combined JSON with the image and matching metadata.

Merging images that have different SURT and same digest is straightforward: merge all the image and page metadata records from all the SURTs that have the same image digest into a single record and remove duplicate metadata. But the opposite is more challenging: what happens if there are image records with different digests for the same SURT? In other words, what to do if the content of an image changes without changing its SURT?

To solve this de-duplication issue, we separate records according to their temporal proximity. Consider that, for a certain SURT S, you have an image record with an digest of A crawled in Jan 1st 2018 and an image record with a digest of B crawled in Jan 1st 2020. In addition, you have multiple page metadata records spanning from 2017 from 2020. When we perform content-based de-duplication for SURT S, we'll create two new entries, for digest A and B.

The page metadata records closest to the image with digest A (all up to Jan 1st 2019) are grouped with the image with digest A, with all records closest to digest B (all starting Jan 1st 2019) are grouped with the image with digest B. This division may not be 100% correct: we do not know exactly at what point between Jan 1st 2018 and Jan 1st 2020 the image was changed from an image with digest A to an image with digest B. Thus, we divide this "uncertainty" time equally between image with digest A and image with digest B.

Collections are represented by a list of WARCs files and a name. Our systems supports reading WARCs directly from HDFS or have them downloaded over HTTP. We expected using HDFS WARCs to be faster, but there was no significant difference between the HDFS and downloaded WARCs, This may be due to WARC downloading making a small percentage of computing time or due to the size of our WARC size: due to the way WARCs are split during crawling, most of our WARC are about 100 MB in size. These results may differ using different cluster configurations.

The number of WARCs per Map process can also be manually set. More WARCs per map result in less Map Reduce overhead and faster processing times, but can also lead to memory exhaustion in the Map process. In our experiments, we set it to 5 WARCs per Map, but the optimal value may vary depending on cluster configuration.

The metadata extraction and de-duplication processes are performed per collection. This means that images captured on different collections will have more than one entry, resulting in duplicate images. Section 5.4 describes how we deal with this duplication when indexing the imagess on Solr.

The indexing process was performed in six servers, with slightly different hardware setups:

- server 1 and 2: 2× CPU (Intel(R) Xeon(R) CPU E5-2620 v3, 6 cores, 12 threads), 24 threads in total, 256 GB RAM, HDD only storage (no SSD).
- server 3 to 6: 2× CPU (Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz, 10 cores, 20 threads), 40 threads in total, 256 GB RAM, HDD only storage (no SSD).

Servers were running CentOS 7. Total indexing time for the full 530TB of WARCs, divided into 115 collections was 1346 hours (56 days).

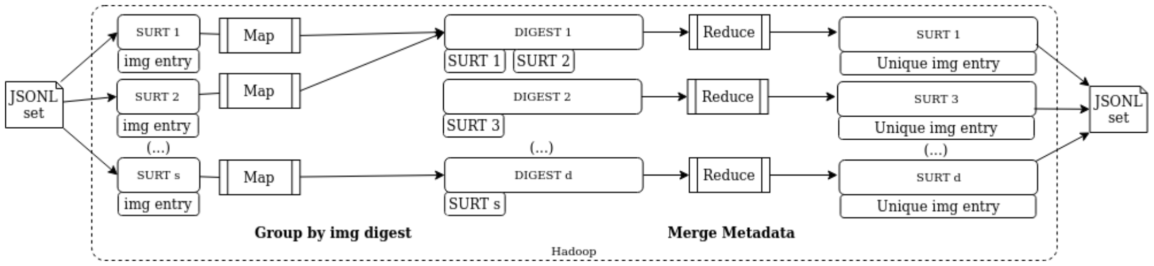

Fig. 8. Data flow for <img> tag extraction in (W)ARC files

5.3 Not Safe For Work classification

After all metadata is extracted and written as JSONL files in HDFS, it is time for the collection to go to the next stage of the pipeline.

Figure 9 shows a simplified representation of how the data goes from the JSONL files to Solr, including NSFW classification. All the images extracted from classified with the classifier described in Section 4.3.

Arquivo.pt is using two servers, each equipped with one Nvidia P40 Graphics Processing Units, in order to process and classify hundreds of images per second. Each of the servers will get a JSONL files from HDFS, process it and write the output to local hard drives. The collection finishes processing when all JSONL files go though the classifier. The dual server architecture is capable of processing 500 images per second (reading and processing JSON line, processing and classifying image and writing output) . As this stage is performed right after Hadoop's information extraction process, meaning that performance is limited by how fast Hadoop runs.

We are planning on how to extract more image metadata on the GPU. A possibility would be to extract dominant colors (to show up on the web UI before images load), scene recognition classifier to extract text categories from images [31], or even an image caption sentence generator [28]. We prototyped a category extractor based in Yolo V4 [6], but decided against using it in release of image search, due to the associated increase in GPU processing time.

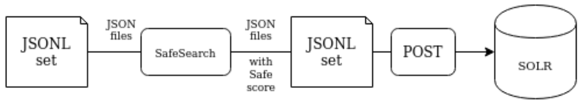

Fig. 9. SafeSearch classification and SOLR index creation

The source code of Arquivo.pt NSFW classifier is open-source and available at GitHub[24].

[24]https://github.com/arquivo/image-gpu-classifier

5.4 Index-based de-duplication

Section 5.2 described our content-based de-duplication process, enabling the de-duplication of image records across time and space. But, due to the way crawl data is organized, WARC data is processed and de-duplicated per collection. Fortunately, we can also perform de-duplication at indexing time using the SolrCloud indexing platform.

The indexing process consists of sending JSONL output files (one image record per line) from Section 4.3 to SolrCloud, which converts these image records to Apache Lucene documents. When images are sent to SolrCloud for indexing, they are identified using an id as a "primary key". SolrCloud also supports adding and updating images after they are indexed, as long as the Solr fields are stored[25]. Thus, what we can do is to perform another round of de-duplication when sending images to Solr, Algorithm 6.

Algorithm 6: De-duplicate Solr images

```
Input: Image metadata $A$ returned from Algorithm 5, where its id is the image digest;
Input: SolrCloud index $I$;
if $A.id$ not in $I$ then
    $I.index(A)$;
else
    $B = I.get(A.id)$;
    if $A.crawlTimestamp > B.crawlTimestamp$ then
        $B.pageMetadata = A.pageMetadata$
    $B.imageMetadata.update(A.imageMetadata)$;
    $I.reindex(B)$;
end
```

Consider a image A; when it is sent to SolrCloud, we check if it already exists a image B with $A.id == B.id$. if B does not exist, it is added as is and indexing is successful. If B exists, we check if A was crawled before B. If so, B's page metadata is replaced with A's page metadata. Then, we update B's image metadata, in a process similar to Algorithm 5, storing only field values that do not exist already in the image. The updated image is then sent to SolrCloud for re-indexing (performed by Lucene under the hood by deleting the image and indexing it again).

This process is performed by collection to ensure that no images are ignored during de-duplication. Due to the way SolrCloud bulk indexer works, de-duplication only works correctly between one image in the index and one new image. If duplicate images are sent to SolrCloud over the same bulk indexing process, only the last of them will be correctly de-duplicated and indexed.

Another important note is that we need to store all our fields in the SolrCloud index. This is due to the way updates are processed in SolrCloud and Lucene: when you update a Lucene document, it deletes the document and created a new one. Thus, at update time, the original metadata is preserved and combined with the new metadata to create the updated image metadata.

Regarding SolrCloud analysers, we are using the Solr's Portuguese language analyser, by using the *text_pt* field type for descriptive textual fields. These fields are *imgAlt*, *imgTitle*, *imgCaption*, *imgUrlTokens*, *pageTitle* and *pageUrlTokens*. The remaining fields are indexed as String (*string*), Float (*pfloat*), Integer (*integer*) or date/time (*pdate*), depending on which type of data they represent[26].

[25] https://solr.apache.org/guide/8_7/updating-parts-of-documents.html#field-storage
[26] https://github.com/arquivo/solr-cloud-scripts/blob/master/images/init/solr-configset/images/conf/managed-schema

As described previously, we used the image digest as the SolrCloud identifier to enable matching images across collections. After indexing is complete, all indexed images become searchable in arquivo.pt/images.

Our initial hypothesis was that such process would reduce the number of images about 20%, but our final experiments show that the reduction in the number of unique images was 39%. The full impact of the distribution process is available in Table 2.

5.5 Indexing obtained results

Table 2 shows the impact of de-duplication at the different stages at the pipeline. The first column contains the stage in which we are at the processing pipeline. The second column shows the percentage of data compared the stage without any de-duplication. The third column shows the reduction in data to index compared to the previous stage.

The effect of de-duplication is apparent at all the stages of the pipeline. The *Image Metadatas* stage would be equivalent to the no de-duplication stage described previously. Staying at this stage would result in having three times more indexing capacity, compared to SURT and content-based de-duplication. This higher indexing capacity would result 70% of these images being the same image with slightly different metadata that would need to be de-duplicated at query time, which would also result in higher processing power requirements.

Merging by SURT has the largest impact in the number of images to index. This is to be expected, as most of the duplicates result from crawling going multiple times to the same pages, or to related pages that have the same images. Matching my digest as a non-negligible impact impact, considering that it is executed on data that was already de-duplicated by SURT.

In our opinion, the most interesting results is the impact of cross-collection de-duplication: Results show that 39% of the images we crawl are duplicates that can be found in previous crawls. A possible explanation is our focus in the "relatively" small .pt domain. Even though the crawling process is non-deterministic (i.e. we do not know exactly what and how much content we will crawl due to temporal constraints), working in a smaller domain means that we will crawl a large amount of images from the same seeds.

Table 2. De-duplication statistics

	Count	% of no de-dup.	Diff. vs. prev.
Total collected files	6,325,224,457	-	-
Total collected images	2,443,485,866	-	-
Image Metadatas (no de-dup.)	1,962,799,850	100%	-
Matching SURT (Section 5.1)	1,170,071,334	60%	-40%
Matching Digest (Section 5.2)	983,373,297	50%	-14%
Collection dedup. (Section 5.4)	595,737,525	30%	-39%

Table 3 shows how many images have the different types of metadata that is extracted from HTML. This stats are computed after all de-duplication stages are performed. This table shows that only 55% of images have either *imgAlt* or *imgTitle*. This images can only be found by words present in their URLs or generic page metadata. Our image caption algorithm was able to extract caption information from about 88% if the images. Combining all types of image specific metadata (*imgAlt*, *imgTitle* or *imgCaption*), we can see that 91% have at least one of the fields.

Table 3. Metadata statistics

	Count	% of total
All images	595,737,525	100%
imgAlt or *imgTitle*	326,175,700	55%
imgCaption	526,081,214	88%
One of *imgAlt/Title/Caption*	541,375,820	91%

6 SEARCHING WEB-ARCHIVED IMAGES (RQ3)

The indexing process left us with 595,737,525 images ready for real-time search. Arquivo.pt image search engine finds images based on the user input text, and retrieves the top results, i.e. the images that better match the user input text. For certain queries, thousands of images are returned with different degrees of relevance to the user from all over the history of the web. Thus, we need to use design a ranking function that takes this into account.

6.1 Ranking features and algorithm

Arquivo.pt is currently ranking the image search results based on the following fields, described in detail in section 5.1: *imgTitle*, *imgAlt*, *imgCaption*, *imgUrlTokens*, *pageTitle* and *pageUrlTokens*.

The Arquivo.pt image search system uses BM25 [24] ranking function for each field. Multiplicative boosts are then given to each field according to their importance. The image search ranking for a single term query is calculated according to Equation (1).

$$\begin{aligned} originalScore = {} & 4 \times imgTitleBM25 + 3 \times imgAltBM25 + \\ & 3 \times imgCaptionBM25 \ + 2 \times imgUrlTokensBM25 + \\ & 1 \times pageTitleBM25 \ + 1 \times pageUrlTokensBM25 \end{aligned} \tag{1}$$

In Equation (1), *imgTitleBM25, imgAltBM25, imgCaptionBM25, imgUrlTokensBM25, pageTitleBM25 and pageUrlTokensBM25* correspond to the BM25 score of the query term for the ranking fields. As one can observe, the most important field for the image search ranking is the image title, followed by the image alt text and caption. These term weighting was performed empirically, by examining the content of the fields and how we expect it to be relevant to a particular image. Image specific fields (e.g. *imgTitle*, *imgAlt*, *imgCaption*) are weighted higher than fields that are more general to the page where the image shows up (*pageTitle*, *pageUrlTokens*). Fields that have to potential to be more descriptive such as *imgTitle* or *imgAlt* are also weight more than less descriptive fields such as *imgUrlTokens*.

Additional ranking scores are given to images that match query terms as phrase queries[27]:

- *Phrase fields (pf)* - boosts the score of an image index when all the terms of a query exist in a given ranking field in close proximity.
- *Phrase slop (ps)* - specifies the distance the indexed search terms can have in the image and still influence relevancy. The amount of slop, i.e. the distance between indexed search terms, is defined by the *ps* field and affects the phrase fields (pf). E.g. if *ps=1* and the input query is *UEFA 2016*, a ranking field containing the text *UEFA Euro 2016*, is considered as a phrase match, because *ps=1* allows up to 1 word of slop between query terms.
- *Phrase bigram fields (pf2)* - similar to *pf* field, but it breaks the input down into word bigrams.

[27] https://solr.apache.org/guide/8_7/the-extended-dismax-query-parser.html

- *Phrase slop 2 (ps2)* - similar to *ps* field, the amount of slop applied to the *pf2* field.
- *Phrase trigram fields (pf3)* - similar to *pf,pf2* fields, but it breaks the input down into word trigrams.
- *Phrase slop 3 (ps3)* - similar to *ps,ps2* fields, the amount of slop applied to the *pf3* field

These boosts are applied exponentially: 1000× boost for ps1, 100× boost for ps2 and 10× boost for ps3. The rationale between choosing this boosting structure is that it ensures that images that have all query terms close together while, penalizing images that have the query terms far apart exponentially. Equation (2) shows how these Arquivo.pt image search ranking boosts for phrase fields (pf), phrase bi-gram fields (pf2) and phrase tri-gram fields (pf3), according to their respective phrase slops (*ps1, ps2, and ps3*).

$$pf(ps) = pf\_aux(4 - ps) \tag{2}$$

$$pf\_aux(x) = 4 \times 10^{x} \times imgTitleBM25\_PSx + 3 \times 10^{x} \times imgAltBM25\_PSx + \tag{3}$$

$$3 \times 10^{x} \times imgCaptionBM25\_PSx + 2 \times 10^{x} \times imgUrlTokensBM25\_PSx + \tag{4}$$

$$1 \times 10^{x} \times pageTitleBM25\_PSx + 1 \times 10^{x} \times pageUrlTokensBM25\_PSx, \tag{5}$$

where the _PSx suffix represents the matching score for the phrase slope query for a slop factor of x.

This process matches the user's expectation and feedback, as most queries consist on a person or institution names [11] and showed good results on our empirical evaluations.

The final image score is as follows:

$$finalScore = \ originalScore + pf(1) + pf(2) + pf(3) \tag{6}$$

For each query, image indexes are ranked, and the image indexes with higher scores are shown to the user. In case of score ties, the following criteria is used:

- finalScore
- if tied, image capture timestamp (oldest first);
- if tied, imgSURT (alphabetical order).

This ensures results are presented in a consistent order across our two SolrCloud instances.

6.2 Implementation of the web-archive image search service

Our 22 million image prototype was tested on an single, over-provisioned server running Apache Solr 6.6. But, as our scale increased about 90-fold, we needed to move to a distributed setup.

We considered going with either SolrCloud[28] or ElasticSearch (ES) for our 595 million image search service. Our offline experiments with 250 million images showed no considerable differences in performance between ElasticSearch and SolrCloud. This is to be expected, as they both are based on Apache Lucene, which is responsible for the (text analysers, posting list inspection and score computation). As both ES and SolrCloud achieved comparable performance and indexing requirements, we decided to go with Apache SolrCloud. It is widely used in the web archiving community (e.g. SolrWayback, Royal Danish Archive) and the ethos and goals of the Solr project (e.g. open-source, non-profit and non-commercial) are more in line with Arquivo.pt's goals.

[28] https://lucene.apache.org/solr/guide/8_7/solrcloud.html

We have eight nodes available for SolrCloud, divided into two equal branches, A and B (four nodes per branch). They have slightly different hardware setups:

- server 1A and 1B: 2× CPU (Intel(R) Xeon(R) CPU E5-2630 v4, 10 cores, 20 threads), 40 threads in total, 512 GB RAM, HDD only storage (no SSD).
- server 2A and 2B: 2× CPU (Intel(R) Xeon(R) CPU E5-2620 v3, 6 cores, 12 threads), 24 threads in total, 256 GB RAM, HDD only storage (no SSD).
- server 3A, 3B, 4A and 4B: 2× CPU (Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz, 10 cores, 20 threads), 40 threads in total, 256 GB RAM, HDD only storage (no SSD).

This set of servers includes the servers used for the indexing process plus two extra servers (of type 1) that were being used as production indexes for the .

Index size is 629.0 GB, divided into 32 shards. Each shard is, has, on average 18.3 million images, and its size on disk is 19.7 GB. Using 4 nodes results in 8 shards per node, and an index size of 157.3 GB per node.

As 157.3 GB is below the amount of RAM we have available per node, we are able to place the full index in RAM. This enables us to keep query response time below one second in most scenarios. Section 6.4 contains more details about the performance of the image search system. To avoid relying on the Operating System file memory pagination and caching, we manually place the directories where the indexes are in memory using *vmtouch*[29]. *vmtouch* is able to manage file system RAM cache, and force files to be placed in memory without the risk of eviction. In addition to enable placing the full index into memory without relying on Solr to warm up cache, it enables the index to stay in memory across SolrCloud restarts.

SolrCloud distribution enables querying any node in the cluster and receive search results from the all the nodes. But nodes have different amounts of RAM and threads available. More specifically, one of the nodes, server 1, has double the amount of RAM (512 GB). To take advantage of this larger amount of memory, we setup a SolrCloud instance without shards. Having a centralized instance without shards enables it to devote more RAM to query cache, and reduce the load from distributed querying and merging from nodes that have to deal with their own shards. We also set an heap size of 31 GB of RAM on all nodes, to benefit from Java's 32 bit pointer compression. Figure 10 gives an overview on how this is organized.

Load balancing is achieved by using hash-based session distribution across our two branches (A and B). This guarantees that the user will always hit the same branch for the same session. This also means that computational resources needs are duplicated, so that we have two branches exactly with the same hardware and data splits.

6.3 Web User Interface and Image Search API

6.3.1 User Interface to search web-archived images. The main goal of Arquivo.pt is to archive web data and make it available and accessible for everyone. Thus, image search must be made available in a user friendly manner, which can be browsed by web users of different levels of expertise.

The user inputs a textual query and a set of image results are displayed in a grid. Similarly to Gifcities, each image search result links to an archived Web page that embeds the image. When clicking on an image result, an image viewer is displayed, showing a larger image together with details about the current image and the web page where it was found.

[29]https://github.com/hoytech/vmtouch

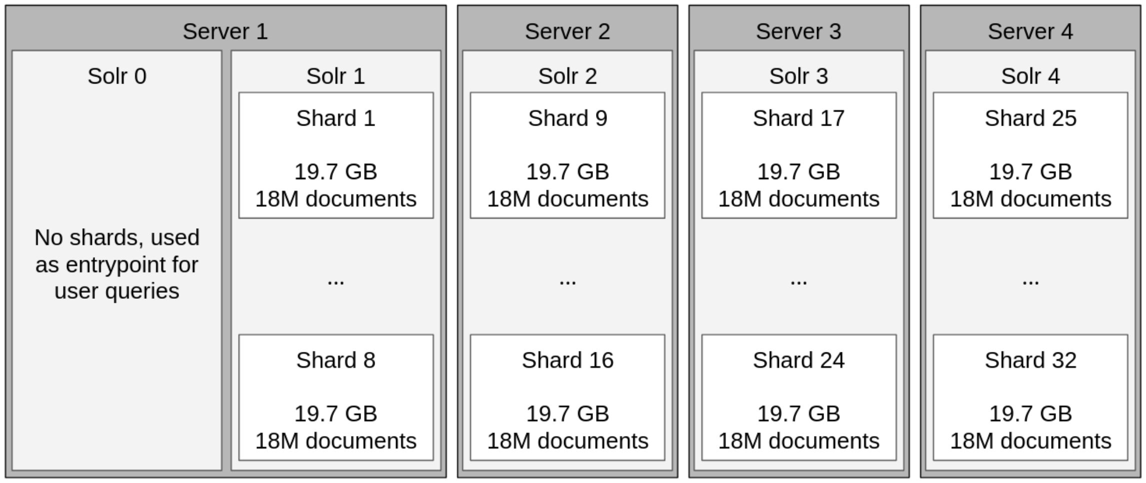

Fig. 10. SolrCloud shard distribution across servers

Arquivo.pt website is built using Express.js. For image search, it queries the API asynchronously to get the search results for the user queries and filters. The source code of Arquivo.pt website is open-source and is freely available at Github[30].

Image Search Filters

Arquivo.pt's image search system supports multiple search result filters that match the filters you can apply in the API:

- Temporal (e.g. find images of *Cristiano Ronaldo* between 2004 and 2008). This is a key filter for users that are looking for images from the past, as it allows focusing on specific time periods;
- Filter by domain (e.g. find all images related with the term *goal* collected from the website portugalnews.com);
- Filter by collection (e.g. find only images which belong to a specific collection[31]
- Filter by mime type (e.g. search for images related with the term *Homer Simpson* in the *GIF* mime type);
- Filter by image size (e.g. search for *large* images of *Lisbon*);

You can find more information about search in the Arquivo.pt's image search FAQ[32][33].

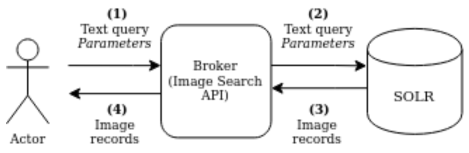

Fig. 11. Image Search API data flow

[30]https://github.com/arquivo/responsive-design/

[31]Collections are set of related web content, captured together. The relation may be temporal (e.g. AWP30 collection: periodic crawl of the PT domain performed on April 2019) or a part of an thematic crawl (e.g. BlogsSapo2018 collection: deep crawl of all sapo.pt blogs). A complete list of Arquivo.PT collections is available here https://docs.google.com/spreadsheets/d/1SjijGAMXgUcwBaH1h7Pov9OLx2YpvR2G39R_CS1B9gs/

[32]https://sobre.arquivo.pt/en/help/image-search/

[33]https://sobre.arquivo.pt/en/help/advanced-image-search/

Table 4. API Performance. q/sec represents the number of queries per second that the system can answer

# requests	Avg.	Med.	$P_{95\%}$	$P_{99\%}$	Throughput
1	115 ms	74 ms	235 ms	769 ms	8 q/sec
3	120 ms	76 ms	259 ms	872 ms	24 q/sec
5	136 ms	85 ms	304 ms	1059 ms	36 q/sec
10	211 ms	128 ms	501 ms	1718 ms	46 q/sec
25	489 ms	266 ms	1297 ms	4334 ms	50 q/sec
50	970 ms	593 ms	2694 ms	6699 ms	50 q/sec

6.3.2 Image Search API. Arquivo.pt developed and open image search API, so that third-party software developers can integrate the Arquivo.pt image search results in their applications, Figure 11. It is also used in Arquivo.pt to build the image search engine, Section 6.3.1

The API is documented on GitHub[34], and the endpoint is located at arquivo.pt/imagesearch.

The response is given in the JSON file format, and is composed by a response header, which contains basic information such as the links to the next and the previous page of results, the total number of images found for the query, and by a list of response items (responseItems) containing the highest scored image indexes. The main search parameter of this API is the *q* parameter (query). For example, one can search for images from the past preserved by arquivo.pt and related with the keyword *Ronaldo* by entering the url arquivo.pt/imagesearch?q=Ronaldo. The API also allows filtering by time, domain, collection (set of pages captured together) and type of image (gif, jpg, png, ...). Filtering by collection is specially important, as it allows to find images captured under special requests (e.g. images from a defunct blogging platform or Portuguese research institution pages).

Each response item contains information such as the image width (*imgWidth*) and height (*imgHeight*), the timestamp when the image was crawled from the Web (*imgTstamp*), how to obtain the image (*imgLinkToArchive*), and how to obtain the page that contains the image (*pageLinkToArchive*). Providing an API that returns image information and the original HTML from where the image was extracted follows one of Google's original goals [7] of building a large scale search engine that can support novel research and is available to the academic and research community.

For more information consult the APIs page[35]. The source code of Arquivo.pt API is open-source and is freely available at Github[36].

6.4 Searching obtained results

Table 4 presents the obtained results for the response times of the API. These experiments where performed using a set of 1000 "two word" queries, extracted and inspired from the query logs. Experiment ran for 5 minutes, meaning that some queries may have been repeated over the experiment, which adequately models how users search. Experiments ran on JMeter and queried the API directly. Results show that the system is able to adequately handle up to 50 concurrent users, while keeping below an average query time below one second.

[34]https://github.com/arquivo/pwa-technologies/wiki/ImageSearch-API-v1.1-(beta)
[35]https://arquivo.pt/apis
[36]https://github.com/arquivo/image-search-api/

7 CONCLUSIONS AND FUTURE WORK

In this article, we described Arquivo.pt image search system. We detailed how we extract relevant image metadata from archived data, how to de-duplicate images (reducing the number of images to archive from 1,800 million to 590 million) and deal with the scale of the archived data. We detail how this information is processed in our cluster: Hadoop metadata extraction and de-duplication, GPU NSFW classification and how our SolrCloud cluster is configured and distributed across nodes. Finally, we show our web interface and API that give access to the indexed data.

Despite the advancements on web archive image search systems described throughout this article, there are still many opportunities for further enhancements. Research possibilities to improve Web archive image search systems include but are not limited to: extracting categories from images using scene recognition classifiers [31], or even generating image captions [28]; retrieving similar images [30]; and retrieving images by dominant color(s) [29], shapes and textures.

Another future goal would be to improve our ranking function, based on real user log information gathered from a running system. We are planning on using our popularity metrics to penalize images that appear in too many pages or have their metadata changed too often. In addition, we are also in the process of creating an annotated image gold collection. This will allows us to fine-tune our ranking function and learn Learning to Rank models.

Finally, improving our API and making our web UI more accessible is fundamental to facilitate the emergence of new applications and research based on images from the past.

8 ACKNOWLEDGEMENTS

The Arquivo.pt web archive assumed the responsibility of preserving the content of the web links cited in this book so that these references may prevail valid over time. Thus, any broken link you may find cited in this book can be recovered from www.arquivo.pt. Submit URLs for crawl as soon as we finish the first version of this paper or try our prototype to archive references automatically[37]. We would like to thank Toke Eskildsen and Thomas Egense from the Royal Danish Web Archive for their inspirational work and valuable discussions.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, and others. 2016. Tensorflow: A system for large-scale machine learning. In *12th ${$USENIX$}$ Symposium on Operating Systems Design and Implementation (${$OSDI$}$ 16)*. 265–283.

[2] Sadet Alcic and Stefan Conrad. 2011. A Clustering-based Approach to Web Image Context Extraction. (2011), 7.

[3] Internet Archive. 2009. Crawldata from Internet Archive from 2009-10-27T18:54:45PDT to 2009-10-27T21:44:06PDT. http://archive.org/details/GEOCITIES-20091027185445-00110-ia400109-c Num Pages: 5293322.

[4] L. A. Barroso, J. Dean, and U. Holzle. 2003. Web search for a planet: The Google cluster architecture. *IEEE Micro* 23, 2 (March 2003), 22–28. https://doi.org/10.1109/MM.2003.1196112 Conference Name: IEEE Micro.

[5] Daniel Bicho. 2019. *Automatic Identification of Not Suitable For Work images.* Master's thesis. IInstituto Superior de Engenharia de Lisboa, Lisboa.

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]* (April 2020). http://arxiv.org/abs/2004.10934 arXiv: 2004.10934.

[7] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30 (1998), 107–117. http://www-db.stanford.edu/backrub/google.html

[8] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. Extracting Content Structure for Web Pages Based on Visual Representation. In *Web Technologies and Applications (Lecture Notes in Computer Science)*, Xiaofang Zhou, Maria E. Orlowska, and Yanchun Zhang (Eds.). Springer, Berlin, Heidelberg, 406–417. https://doi.org/10.1007/3-540-36901-5_42

[9] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. VIPS: A VIsion based Page Segmentation Algorithm. (2003), 32.

[10] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems* 26, 2 (June 2008), 4:1–4:26. https://doi.org/10.1145/1365815.1365816

[37] https://github.com/arquivo/pwa-technologies/issues/1147

[11] Miguel Costa. 2014. *Information search in web archives.* Ph.D. Dissertation. Universidade de Lisboa. https://repositorio.ul.pt/handle/10451/16020?mode=full Accepted: 2015-02-13T17:45:47Z.

[12] Jeffrey Dean. 2009. Challenges in building large-scale information retrieval systems: invited talk. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*. New York, NY, USA, 1–1. http://doi.acm.org/10.1145/1498759.1498761

[13] Jeff Dean. 2009. Designs, lessons and advice from building large distributed systems. *Keynote from LADIS* 1 (2009).

[14] Toke Eskildsen. 2016. 70TB, 16b docs, 4 machines, 1 SolrCloud. https://sbdevel.wordpress.com/2016/11/30/70tb-16b-docs-4-machines-1-solrcloud/

[15] Rand Fishkin. 2018. The Evolution of Search Marketing, Rand Fishkin. https://theinbounder.com/videos/search-marketing-s-evolution-2018-and-beyond.html

[16] Daniel Gomes, Elena Demidova, Jane Winters, and Thomas Risse (Eds.). 2021. *The Past Web: Exploring Web Archives.* Springer International Publishing. https://doi.org/10.1007/978-3-030-63291-5

[17] ISO International Organization for Standardization. 2017. *ISO 28500:2017 Information and documentation — WARC file format.* Standard. International Organization for Standardization, Geneva, CH. https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/80/68004.html

[18] Parag Mulendra Joshi and Sam Liu. 2009. Web document text and images extraction using DOM analysis and natural language processing. In *Proceedings of the 9th ACM symposium on Document engineering (DocEng '09).* Association for Computing Machinery, New York, NY, USA, 218–221. https://doi.org/10.1145/1600193.1600241

[19] Johannes Kiesel, Lars Meyer, Florian Kneist, Benno Stein, and Martin Potthast. 2021. An Empirical Comparison of Web Page Segmentation Algorithms. In *Advances in Information Retrieval. 43rd European Conference on IR Research (ECIR 2021) (Lecture Notes in Computer Science)*, Djoerd Hiemstra, Maria-Francine Moens, Josiane Mothe, Raffaele Perego, Martin Potthast, and Fabrizio Sebastiani (Eds.). Springer, Berlin Heidelberg New York. https://github.com/webis-de/ecir21-an-empirical-comparison-of-web-page-segmentation-algorithms#meier-et-al

[20] Gant Laborde. [n.d.]. *Deep NN for NSFW Detection.* https://github.com/GantMan/nsfw_model Publication Title: GitHub.

[21] Jesper Lauridsen. 2021. SolrWayback 4.0 release! What's it all about? https://sbdevel.wordpress.com/2021/02/12/solrwayback-4-0/

[22] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. 2004. What's new on the web? the evolution of the web from a search engine perspective. In *Proceedings of the 13th international conference on World Wide Web (WWW '04).* Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/988672.988674

[23] Vincenzo Riili. 2019. 18 years after Google Images, the Versace jungle print dress is back. https://blog.google/products/search/18-years-after-google-images-versace-jungle-print-dress-back/

[24] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3, 4 (April 2009), 333–389. https://doi.org/10.1561/1500000019

[25] Nate Smith. 2010. Ooh! Ahh! Google Images presents a nicer way to surf the visual web. https://googleblog.blogspot.com/2010/07/ooh-ahh-google-images-presents-nicer.html

[26] Apache Solr. 2019. SolrPerformanceData - SOLR - Apache Software Foundation. https://cwiki.apache.org/confluence/display/SOLR/SolrPerformanceData

[27] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. *CoRR* abs/1512.00567 (2015). http://arxiv.org/abs/1512.00567 _eprint: 1512.00567.

[28] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, Boston, MA, USA, 3156–3164. https://doi.org/10.1109/CVPR.2015.7298935

[29] Jingdong Wang and Xian-Sheng Hua. 2011. Interactive Image Search by Color Map. *ACM Transactions on Intelligent Systems and Technology* 3, 1 (Oct. 2011), 12:1–12:23. https://doi.org/10.1145/2036264.2036276

[30] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. 2014. Learning Fine-Grained Image Similarity with Deep Ranking. In *2014 IEEE Conference on Computer Vision and Pattern Recognition.* 1386–1393. https://doi.org/10.1109/CVPR.2014.180 ISSN: 1063-6919.

[31] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. 2018. Places: A 10 Million Image Database for Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 6 (June 2018), 1452–1464. https://doi.org/10.1109/TPAMI.2017.2723009 Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.