

The Cord Image Search - A Specialised Figure Search Engine

VAIBHAV SINGH and MEGHA PUJAMATH, Pennsylvania State University, USA

Multimedia objects, especially figures and tables are essential for the visualization and understanding of research impacts. Although scientific journals and papers are retrieved through various online search services, not all of these systems take advantage of visual information contained in the literature. Consequently, there is a greater demand to develop standardized indexing methods for these multimedia objects, and to address this objective, it is necessary to develop a process for indexing and retrieving these figures and tables. The COVID-19 Open Research Dataset (CORD-19) is a growing resource of scientific papers on COVID-19 and related historical coronavirus research. Papers and prints from a variety of archives are collected, and paper documents are processed through the pipeline to extract full text.[3]

. The aim of this project was to develop a search engine to retrieve the figures and tables from the CORD-19 dataset. To achieve this goal, the papers from the CORD dataset were parsed using Allen AI's API to extract tables and figures and their relevant metadata which enabled efficient indexing possibilities. The extracted JSON data was then indexed into Elastic Search which bundles industry standard search features. Like other search engines, the figure search engine is mostly based on text information associated with images/tables. The search results predominantly includes images, tables and captions.

Additional Key Words and Phrases: CORD-19, bulk indexing, elasticsearch

1 INTRODUCTION

Figures are considered to be an integral part of research and they act as results which convey the objective more effectively when compared to plain text. This motivates search engines that are focused on indexing figures in ETDs. Image search approaches have gained a lot of focus off-late and one such approach that is implemented as part of this project is text-based image retrieval. A text-based retrieval system uses the metadata associated with the image such as the image text and caption to effectively query across the repository of image data. A user's query can be in the form of custom keywords or keyphrases. During the retrieval process, the query is compared with the stored image metadata, and we obtained ranked retrieval based on the backend matching algorithm. Thus, the text-based image retrieval system uses conventional document retrieval techniques [3][4]

1.1 Project Objective

This project aims at implementing effective figure search engine using elastic search to index the references and captions extracted from Electronic Thesis and Dissertation(ETD) figures from CORD dataset and build a search interface that returns ETD tables and figures.

2 LITERATURE REVIEW

2.1 The Importance of Information Retrieval System

Information Retrieval (IR) is the process of finding and fetching information from a collection of data based upon a search request. The data in this collection is referred to as a document. The collection of data is called an index which is optimized. The index can also contain the entirety of the original documents [3] The search request is based on the user query which is called term. These search terms are then converted to search queries by the system. The query is matched against the index and any suitable document is returned as the search result in the order of relevancy. The relevancy score is an indicator of how well a document matched the search query.

2.2 Elasticsearch

Elasticsearch is an open-source analytics engine that can facilitate several use cases. These well-written materials helped us to understand the creation of an inverted index using Elasticsearch, as well as the stop-word list concept and single-pass in-memory indexing. Elasticsearch, which is an open-source, distributed, and RESTful search engine, makes it significantly easier to build a search engine that can analyze a great amount of information effectively and deliver relevant results to users.

Elasticsearch is built to handle large amounts of data while maintaining user-friendly and accessible methods to retrieve data. There is a handful of APIs that allow applications from any language or framework to access an Elasticsearch index. To ease the users' searching processes, multiple filters should be applied on the searching website for users to access the information, and present the most relevant information to the users. To help users to digest the information faster, the design of the interface should be easy and appealing to use, and allow users to effectively find the answers they desire.[4]

2.3 Bulk Indexing

Bulk indexing in Elasticsearch is an API that performs bulk data indexing or delete operations in a single API call which increases the indexing speed. Using the Bulk API is more efficient than sending multiple separate requests. Bulk API is useful when you need to index data streams that can be queued up and indexed in batches of hundreds or thousands, such as logs. The response to a bulk action is a large JSON structure with the individual results of each action that was performed in the same order as the actions that appeared in the request. The failure of a single action does not affect the remaining actions.[4]

2.4 The CORD Dataset

The Covid-19 Open Research Dataset (CORD-19) resource is a large and growing collection of publications and preprints on Covid-19 and related historical coronaviruses such as SARS and MERS. CORD-19 aims to connect the machine learning community with biomedical domain experts and policy makers in the race to identify effective treatments and management policies for Covid-19. The goal is to harness these diverse and complementary pools of expertise to discover relevant information more quickly from the literature. Users of the dataset have leveraged a variety of AI-based techniques in information retrieval and natural language processing to extract useful information.[5]

2.5 Allen AI Pre-processing

The preprocessing of the cord-19 dataset is done using Allen AI api called pdftofigures, which takes as input a scholarly document in PDF form. Its output consists of a figure extracted, a bounding box for both the figure and its caption as well as the identifier of that figure (e.g., "Figure 1" or "Table 3") and the page number that the figure resides on. Here the figure refers to both image and tables. pdftofigures also supports the ability to save images of the extracted figures as images[1][2]

3 DESIGN ANALYSIS AND METHODOLOGY

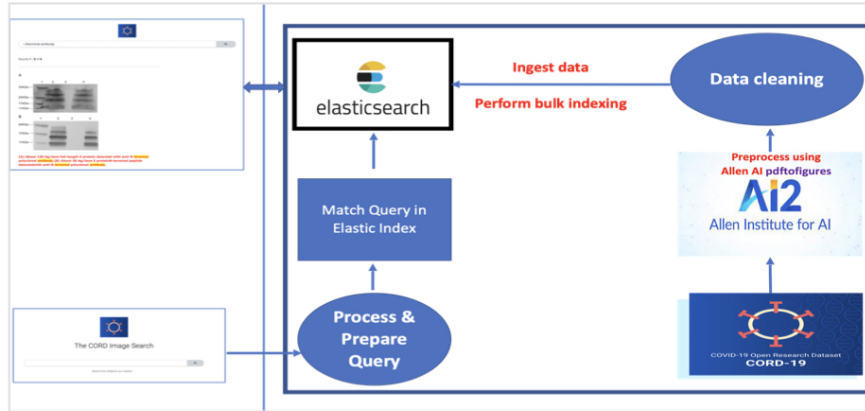


Fig. 1. Design Model of Cord Image Search

The design model in figure 1 can be comprehended in two ways, first, the processing of the cord data that needs to be indexed. Second, the search engine user interface and the back-end queries which fetch the relevant document from the indexed data.

In this project, the dataset used was the CORD-19 dataset. The pdf files in this dataset were processed using Allen AI pdffigures2 [1] repository and the resulting figures and tables along with their image metadata were extracted. The extracted data for any paper was in the following format -

```
{
  "caption": "Table 1. Baseline Demographic Characteristics of the Studied Populations",
  "captionBoundary": {
    "x1": 48.925899505615234,
    "x2": 288.0222473144531,
    "y1": 41.0155029296875,
    "y2": 57.27410888671875
  },
  "figType": "Table",
  "imageText": [{"x": 48, "y": 41, "text": "Table 1"}],
  "name": "1",
  "page": 2,
  "regionBoundary": {
    "x1": 48,
    "x2": 289.44,
    "y1": 66.72,
    "y2": 190.07999999999998
  },
  "renderDpi": 150,
  "renderURL": "/data/sfk5555/results2/fffffd73faaa9e30eddf4a3edede5af10f48a619-Table1-1.png"
}
```

Fig. 2. Design Model of Cord Image Search

The data obtained in the form of a JSON object is further sent through the data cleaning phase where the unnecessary tags are removed and the URL is updated if necessary.

An Elasticsearch index is created using bulk indexing API. A user interface is created using Django to demonstrate the application. The queries are written in python and these queries are run against Elasticsearch using high-level library Elasticsearch DSL. In the search user interface, when a keyword/keyphrase is searched, this query is passed on to the elastic search gateway and a relevant figure or table is extracted and this, in turn, can be viewed in the UI

4 CONSTRUCTION AND RESULT TESTING

4.1 Tools

The tools that we utilized for our implementation were primarily:

1. Elasticsearch - Elasticsearch version used in this project was Elasticsearch 6.2.3
2. JSON - Since elasticsearch utilizes JSON to a much larger extent, in this project JSON objects are considered as input.
3. Python - high level programming language
4. Django Framework

4.2 Dataset Exploration



Fig. 3. Dataset Exploration

As discussed above, the CORD-19 dataset is used for the implementation of search engine. This resource is a large and growing collection of publications and preprints on COVID-19 and related historical coronaviruses such as SARS and MERS. The total files indexed were around 100k, which consisted of 68k figures and 30k tables shown in the figure 2.[5]

4.3 Data Cleaning

```
{
  "caption": "Table 3. Laboratory results of COVID-19 and NON-COVID-19 patients",
  "captionBoundary": {},
  "figType": "Table",
  "imageText": {},
  "name": "3",
  "page": 19,
  "regionBoundary": {
    "x1": 65.75999999999999,
    "x2": 573.6,
    "y1": 91.67999999999999,
    "y2": 215.51999999999998
  },
  "renderDpi": 150,
  "renderURL": "/data/team07/json_tables_figures/bdba0c998d79e576e1829b71001ed7e6e52f9502-Table3-1.png"
}
```

Fig. 4. Json format of the single data

Since the size of data to be indexed was within basic CPU processing potentials, so we didn't eliminate fields from the data as we could gain better intuitions out of it as the project progresses. The renderURL field for all documents had to be updated to match the expected directory structure for our front end application.

4.4 Bulk Indexing

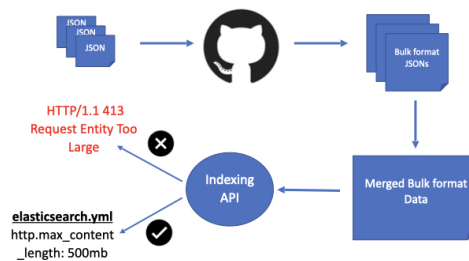


Fig. 5. Bulk Indexing

The JSON data files after the cleanup, are first converted to bulk format using the git API - <https://github.com/mradamlacey/json-to-es-bulk>. This process generates bulk format data which is a prerequisite for bulk indexing. The formatted JSON files are then merged into a single file to harness the potential of BULK API call through a single

request. This also eliminates the need to perform bulk indexing on every file with a loop. Once the merged file is ready, this file is used in the curl command to bulk index all the data.

. The curl command used to bulk index the JSON is

```
curl -v -H'Content-Type : application/json' -XPOST 'http://localhost:9200/cord_index/_bulk' --data-binary @combined.txt
```

4.4.1 Elasticsearch Configuration. We noticed that the elasticsearch has a default max content limit of 100MB beyond which it throws back a 413 Request Entity Too Large error. This is due to elasticsearch.yml configuration which defaults to `http.max_content_length : 100mb`. This was changed to 500MB as per the size of our bulk data file.

4.5 Search Query Implementation

4.5.1 Search API.

. The search API in figure 5, gets all the documents present in an index. Here the model consists of all the attributes that are present in JSON file

```
class SearchResult:
    """
    def __init__(self, resultid, content="No content found", fileurl="No URL found", title="No title found",
                authors="No authors available", description="No description found", affiliation="No location found",
                journal="No journal available", source="", doi=""):
        self.resultid = resultid
        self.content = content
        self.fileurl = fileurl
        self.title = title
        self.authors = authors
        self.description = description
        self.affiliation = affiliation
        self.journal = journal
        self.source = source
        self.doi = doi
    """

    def __init__(self, resultid, caption="No caption found", renderURL="No URL found", description="No description found", image
                self.resultid = resultid
                #self.content = content
                self.description = description
                self.caption = caption
                self.renderURL = renderURL
                self.imageText = imageText
```


Fig. 6. Search API

4.5.2 Multi Match Query and Boosting.

. Basic Match query is implemented to retrieve all documents from an index or a set of indices which match a set of specific criteria. In this query, although both text - caption and image text is considered, the boosting is given to the caption by 10 times. Hence the caption text is always given higher preference [6]

```
term_body = {
  "from": start,
  "size": size,
  "query": {
    "multi_match" : {
      "query" : nquery,
      "fields" : ["caption^10", "imageText"], #boosting here
    }
  },
  'highlight': {
    'fields': {
      'caption': {},
    }
  }
}
```

Fig. 7. Multi Match Query



Covid

Results 1 - 10 of 1254

Variable (normal range)	COVID-19 patients (n=19)	NON-COVID-19 patients (n=15)	p value
WBC (4-10×10 ⁹ /L)	4.92 (1.26-7.63)	6.18(3.37-12.38)	0.30
<4	7/19(36.84%)	4/15(26.67%)	0.72
>10	0 (0%)	2/15(13.33%)	0.19
Leukocytes (1-4)	0.07 (0.00-2.02)	1.11 (0.02-1.05)	0.00

Laboratory results of COVID-19 and NON-COVID-19 patients

Fig. 8. Search Results of Multi Match Query

4.5.3 Fuzziness.

. Fuzzy matching can be enabled on Match and Multi-Match queries to catch spelling errors. The degree of fuzziness is specified based on the distance from the original word, i.e. the number of one-character changes that need to be made to one string to make it the same as another string.[6]

```
term_fuzz_body = {
  "from": start,
  "size": size,
  "query": {
    "multi_match" : {
      "query" : nquery,
      "fields" : ["caption^10", "imageText"], #boosting here
      "fuzziness": "AUTO",
    }
  },
  "highlight": {
    "fields": {
      "caption": {},
    }
  }
}
```

Fig. 9. Fuzziness



GRADE	DESCRIPTION
Clear	0 No lesions to barely noticeable ones; very few scattered comedones and papules
Almost clear	1 Hardly visible from 2.5 meters away; a few scattered comedones and a few small papules; and very few pustules, comedones, and papules
Mild	2 Easily recognizable; less than half of the affected area is involved; many comedones, papules, and pustules
Moderate	3 More than half of the affected area is involved; numerous comedones, papules, and pustules
Severe	4 Entire area is involved; covered with comedones, numerous pustules and papules, a few nodules and cysts
Very severe	5 Highly inflammatory acne covering the affected area, nodules and cysts present

Reproduced with permission from Dr. Jerry Tan*

Comprehensive Acne Severity Scale (CASS)

Fig. 10. Search Result of Fuzziness

4.5.4 Regex Query.

. It matches documents which contain terms that match a regular expression. The below query fetches documents whose name starts with t and ends with y.

```
regex_body = {  
  "from": start,  
  "size": size,  
  "query": {  
    "regexp" : {  
      "caption" : nquery  
    }  
  },  
  'highlight': {  
    'fields': {  
      'caption': {},  
    }  
  }  
}
```

Fig. 11. Regex Query

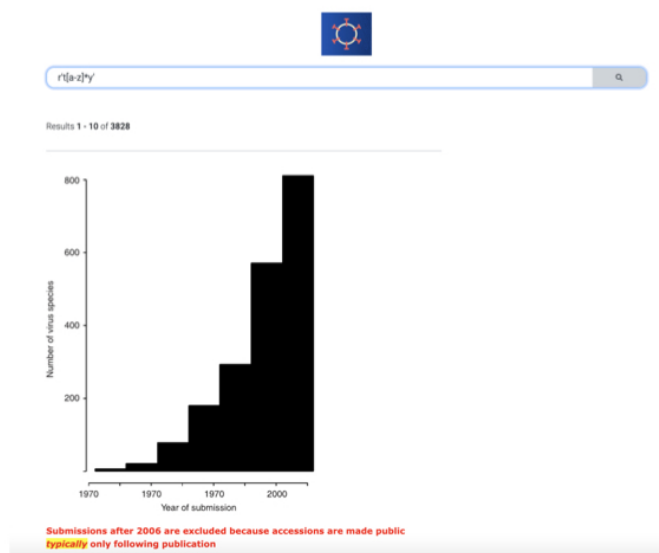


Fig. 12. Search Result of Regex Query

4.5.5 Match Phrase Query.

. The match phrase query requires that all the terms in the query string be present in the document, be in the order specified in the query string and be close to each other. By default, the terms are required to be exactly beside each other but you can specify the slop value which indicates how far apart terms are allowed to be while still considering the document a match.[6]

```
# Matching only in captions
phrase_body = {
  "from": start,
  "size": size,
  "query": {
    "match_phrase": {
      "caption": {
        "query": nquery,
        "slop": phrase_slop
      },
    },
  },
  'highlight': {
    'fields': {
      'caption': {},
    }
  }
}
```

Fig. 13. Match Phrase Query

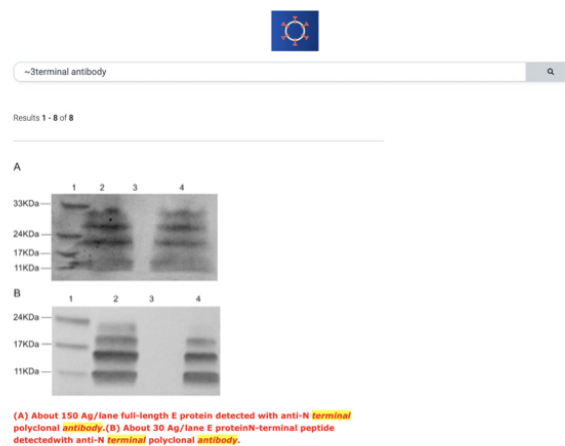


Fig. 14. Search Result of Match Phrase Query

4.5.6 Figure Search Query.

. Figure search query matches the `figType` attribute of the document. This field is constrained to be Figure so that the search results are all figures [6]

```
figure_term_body = {
  "from": start,
  "size": size,
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "figType": "Figure"
          }
        },
        {
          "multi_match": {
            "query": nquery,
            "fields": ["caption", "imageText"], #boosting here
          }
        }
      ]
    }
  },
  "highlight": {
    "fields": {
      "caption": {},
    }
  }
}
```

Fig. 15. Figure Search Query

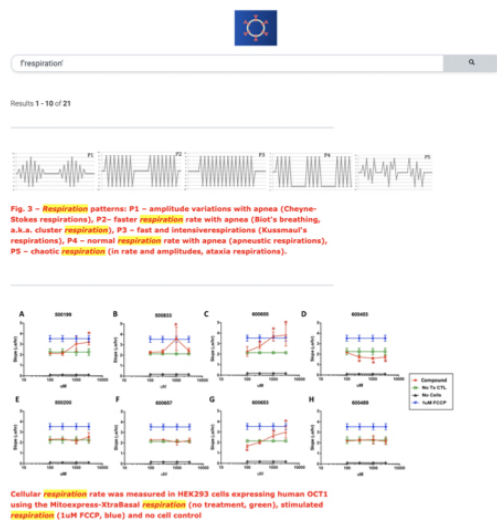


Fig. 16. Search Result of Figure Search Query

4.5.7 Table Search Query.

. Table search query matches the figType attribute to Table. The search results will always result in tables [6]

```
table_term_body = {
  "from": start,
  "size": size,
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "figType": "Table"
          }
        },
        {
          "multi_match": {
            "query": nquery,
            "fields": ["caption", "imageText"], #boosting here
          }
        }
      ]
    }
  },
  "highlight": {
    "fields": {
      "caption": {},
    }
  }
}
```

Fig. 17. Table Search Query



Respiratory

Results 1 - 10 of 3619

	14-23 years		24-43 years		≥44 years		Total number (%)	
	URIs	LRTs	URIs	LRTs	URIs	LRTs	URIs	LRTs
Patients tested	1871	47	2275	129	386	76	3222	274
Total positive	716 (38.3)	11 (23.4)	1087 (47.8)	42 (32.5)	129 (33.4)	28 (36.8)	1857 (57.8)	81 (29.5)
Single respiratory	461 (24.6)	1 (2.1)	691 (30.4)	30 (23.3)	122 (31.6)	25 (32.9)	1264 (39.2)	76 (28.1)
Single respiratory	254 (13.6)	1 (2.1)	361 (15.9)	15 (11.6)	48 (12.4)	11 (14.5)	662 (20.6)	39 (14.3)
RV A	88 (4.7)	1 (2.1)	46 (20.2)	1 (0.8)	7 (1.8)	1 (1.3)	142 (4.4)	4 (1.5)
RV C	18 (0.9)	0 (0)	5 (0.2)	0 (0)	14 (3.6)	1 (1.3)	37 (1.1)	1 (0.4)
RSV	187 (10.0)	4 (8.5)	188 (82.7)	11 (8.5)	28 (7.3)	7 (9.1)	399 (12.3)	16 (5.8)
Flu A	87 (4.6)	2 (4.3)	122 (53.6)	11 (8.5)	28 (7.3)	7 (9.1)	245 (7.6)	26 (9.5)
Flu B	82 (4.4)	2 (4.3)	112 (49.2)	11 (8.5)	4 (1.0)	1 (1.3)	200 (6.2)	4 (1.5)
Flu C	14 (0.7)	0 (0)	14 (0.6)	1 (0.8)	1 (0.3)	0 (0)	29 (0.9)	1 (0.4)
Adenovirus	11 (0.6)	0 (0)	14 (0.6)	1 (0.8)	1 (0.3)	0 (0)	26 (0.8)	1 (0.4)
HRV	1 (0.05)	0 (0)	14 (0.6)	1 (0.8)	1 (0.3)	0 (0)	27 (0.8)	1 (0.4)
HRV	4 (0.2)	0 (0)	14 (0.6)	1 (0.8)	1 (0.3)	0 (0)	29 (0.9)	1 (0.4)

Abbreviations: RV, respiratory virus; URIs, upper respiratory infections; LRTs, lower respiratory tract infections; RSV, human respiratory syncytial virus; HRV, human rhinovirus; RV, respiratory virus; RSV, respiratory syncytial virus.
 *Number in parentheses are percentages.
 †Represents significant differences between the incidence of virus infection by respiratory tract.

© 2009 Institute of Pathogen Biology, Chinese Academy of Medical Sciences, CHN 10, 1146-1152

Respiratory viruses detected in acute upper respiratory tract infections (URTIs) and acute lower respiratory

	Asthma	Bronchitis	Chronic bronchitis	Common cold	Croup	Otitis media	Pneumonia
Respiratory viruses	+++	+	+	+	+	++	+
Influenza	++	+	+	+	+	++	+
RSV	+	+	+++	+	+	++	+
Parainfluenza	+	++	+	+	++	+	+
Coronaviruses	++	+	++	++	+	+	+
Adenoviruses	+	+	++	+	+	+	++

Table 1 Respiratory viruses in upper and lower respiratory disease syndromes

Fig. 18. Search Result of Table Search Query

5 CONCLUSION

We are surrounded with perpetually flowing and infinitesimally growing stream of data. As a result of this, asking the right questions is all that a researcher needs to target in the current information-heavy world. Hence, deriving insights from data seems like the most interesting area of research once the right questions are in place and search engines enable a creation of a bond between a user and a developer as the user enjoys the rapid fast information he obtains and the developer gets a chance to showcase his intuitive brilliance through his backend data handling.

The above premise sets the tone for concluding our experience during this exciting project which gave us the opportunity to play around and derive intuitions from the famous CORD dataset and the even more industry-savvy distributed search engine - Elasticsearch. Despite being mildly superficial in terms of industry standards, this project certainly kindled an interest in the area of search engines which would motivate us to go a step further and dig deeper into what elastic and its peers could offer in terms of performance and features. With the help of our instructor and wonderful teaching assistants, we were able to quickly get a user interface ready to showcase our indexed data in variety of formats. We learnt how to make best use of a single search bar to deliver results in various ways to the end user based on his demands and there's certainly room for extensive improvement.

To sum up, it was a great learning experience both in terms of team work and in terms of technical interactions with the ever so helpful teaching staffs. We as a team would certainly try to take this project to its ultimate completion and we hope for continuous support from our instructor in this undertaking.

ACKNOWLEDGMENTS

We thank Professor Giles, Shaurya Rohatgi, and Ankur Mali for their constant support and guidance throughout the semester

REFERENCES

- [1] Santosh Divvala Christopher Clark. 2015. Looking Beyond Text: Extracting Figures, Tables and Captions from Computer Science Papers.
- [2] Santosh Divvala Christopher Clark. 2016. PDFFigures 2.0: Mining Figures from Research Papers. Allen Institute for Artificial Intelligence University of Washington.
- [3] Hinrich Schütze Christopher D. Manning, Prabhakar Raghavan. Cambridge University Press. 2008. Introduction to Information Retrieval.
- [4] elastic.co. 2021. Elastic Stack and Product Documentation. elastic.co.
- [5] Yoganand Chandrasekhar Russell Reas1 Jiangjiang Yang Darrin Eide Kathryn Funk Lucy Lu Wang, Kyle Lo. 2020. CORD-19: The COVID-19 Open Research Dataset.
- [6] TSvitla. 2021. Elasticsearch DSL for searching and ranking information. TSvitla Team.